Minerit de Masiv Seturi de date

Jure Leskovec
Universitatea Stanford
Anand Rajaraman
Rocketship Ventures
Jeffrey D. Ullman
Universitatea Stanford
Copyright cG 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, Anand Rajaraman, Jure Leskovec și Jeffrey D. Ullman

Prefață

Această carte a evoluat din materialul dezvoltat de mai mulți ani de Anand Rajaraman și Jeff Ullman pentru un curs de un sfert la Stanford. Cursul CS345A, intitulat „Web Mining”, a fost conceput ca un curs avansat de absolvire, deși a devenit accesibil și interesant pentru studenții avansați.
Când Jure Leskovec s-a alăturat facultății din Stanford, am reorganizat materialul considerabil. A introdus un nou curs CS224W privind analiza rețelei și a adăugat material la CS345A, care a fost renumerotat CS246. Cei trei autori a introdus, de asemenea, un curs de proiect pe scară largă de minerit de date, CS341. Cartea acum conține material predat în toate cele trei cursuri.

Despre ce este cartea

La cel mai înalt nivel de descriere, această carte este despre extragerea datelor. In orice caz, se concentrează pe extragerea datelor de cantități foarte mari de date, adică date atât de mari nu se încadrează în memoria principală. Datorită accentului pus pe dimensiune, multe dintre noi exemple sunt despre Web sau date derivate de pe Web. Mai departe, cartea ia un punct de vedere algoritmic: extragerea datelor este despre aplicarea algoritmilor la date, mai degrabă decât folosind date pentru a „antrena” un motor de învățare automată a unora fel. Principalele subiecte tratate sunt:

1. Sisteme de fișiere distribuite și map-reduce ca instrument pentru crearea paralelelor algoritmi care reușesc pe cantități foarte mari de date.
2. Căutarea similarității, inclusiv tehnicile cheie ale minhashingului și localității-hashing sensibil.
3. Prelucrarea fluxului de date și algoritmi specializați pentru tratarea datelor care ajunge atât de repede încât trebuie procesat imediat sau pierdut.
4. Tehnologia motoarelor de căutare, inclusiv PageRank Google, link-spam detecție și abordarea centrelor și autorităților.
5. Exploatarea de obiecte frecvente, inclusiv reguli de asociere, coșuri de piață, Algoritmul A-Priori și îmbunătățirile sale.
6. Algoritmi pentru gruparea unor seturi de date foarte mari, de înaltă dimensiune.

7. Două probleme cheie pentru aplicațiile Web: gestionarea publicității și sisteme de recomandare.
8. Algoritmi pentru analiza și extragerea structurii graficelor foarte mari, în special graficele rețelelor sociale.
9. Tehnici pentru obținerea proprietăților importante ale unui set de date mare de către reducerea dimensionalității, inclusiv descompunerea cu valoare singulară și indexarea semantică a cortului.
10. Algoritmi de învățare automată care pot fi aplicați unor date foarte mari, de exemplu ca perceptroni, mașini vector-suport, descendență în gradient și decizie copaci.
11. Rețele neuronale și învățare profundă, inclusiv cele mai importante cazuri speciale: rețele neuronale convoluționale și recurente și memorie pe termen scurt rețele.

Condiții prealabile

Pentru a aprecia pe deplin materialul din această carte, vă recomandăm următoarele premise:
1. O introducere în sistemele de baze de date, care acoperă SQL și programele conexe sisteme ming.
2. Un curs de nivel secundar în structuri de date, algoritmi și discrete matematica.
3. Un curs de nivel secundar în sisteme software, inginerie software și limbaje de programare.

Exerciții

Cartea conține exerciții ample, cu unele pentru aproape fiecare secțiune. Noi indicați exerciții mai grele sau părți de exerciții cu punct de exclamare. The cele mai grele exerciții au un dublu punct de exclamare.

Asistență pe web

Accesați http://www.mmds.org pentru diapozitive, teme pentru teme, cerințe de proiect-mentele și examenele de la cursurile legate de această carte.

Pagina 5

Gradiance Automated Homework

Există exerciții automatizate bazate pe această carte, folosind rădăcina Gradiance-tehnologia întrebărilor, disponibilă la www.gradiance.com/services. Elevii pot introduceți o clasă publică creând un cont la acel site și intrând în clasă cu cod 1EDD8A1D. Instructorii pot folosi site-ul făcându-și un cont acolo și apoi prin e-mail asistență la gradiance dot com cu numele lor de conectare, numele școlii lor și o cerere de utilizare a materialelor MMDS.

Mulțumiri

Coperta este de Scott Ullman.
Dorim să mulțumim Foto Afrati, Arun Marathe și Rok Sosic pentru critici lecturile unui proiect al acestui manuscris.
Erori au fost raportate și de Rajiv Abraham, Ruslan Aduk, Apoorv Agarwal, Aris Anagnostopoulos, Yokila Arora, Stefanie Anna Baby, Atilla Soner Balkir, Arnaud Belletoile, Robin Bennett, Susan Biancani, Richard Boyd, Amitabh Chaudhary, Leland Chen, Hua Feng, Marcus Gemeinder, Anastasios Gounaris, Clark Grubb, Shrey Gupta, Waleed Hameid, Saman Haratizadeh, Julien Hoachuck, Przemyslaw Horban, Hsiu-Hsuan Huang, Jeff Hwang, Rafi Kamal, Lachlan Kang, Ed Knorr, Haewoon Kwak, Ellis Lau, Greg Lee, David Z. Liu, Ethan Lozano, Yunan Luo, Michael Mahoney, Sergio Matos, Justin Meyer, Bryant Moscon, Brad Penoff, John Phillips, Philips Kokoh Prasetyo, Qi Ge, Harizo Rajaona, Timon Ruban, Rich Seiter, Hitesh Shetty, Angad Singh, Sandeep Sripada, Dennis Sidharta, Krzysztof Stencel, Mark Storus, Roshan Sumbaly, Zack Taylor, Tim Triche Jr., Wang Bin, Weng Zhen-Bin, Robert West, Steven Euijong Whang, Oscar Wu, Xie Ke, Christopher T.-R. Da, Nicolas Zhao și Zhou Jingbo. Erorile rămase sunt ale noastre, desigur.
JL
AR
JDU
Palo Alto, CA
Iulie, 2019

Cuprins

1 Data Mining
1
1.1 Ce este Data Mining? 1
1.1.1 Modelare. 1
1.1.2 Modelare statistică. 2
1.1.3 Învățarea automată. 2
1.1.4 Abordări computaționale ale modelării. 3
1.1.5 Rezumare. 4
1.1.6 Extragerea caracteristicilor. 5
1.2 Limite statistice privind exploatarea datelor. 5
1.2.1 Conștientizarea informațiilor totale. 6
1.2.2 Principiul lui Bonferroni. 6
1.2.3 Un exemplu al principiului lui Bonferroni. 7
1.2.4 Exerciții pentru secțiunea 1.2. 8
1.3 Lucruri utile de știut. 8
1.3.1 Importanța cuvintelor în documente. 9
1.3.2 Funcții Hash. 10
1.3.3 Indici. 11
1.3.4 Depozitare secundară. 13
1.3.5 Baza logaritmilor naturali. 13
1.3.6 Legile puterii. 14
1.3.7 Exerciții pentru secțiunea 1.3. 16
1.4 Schița cărții. 17
1.5 Rezumatul capitolului 1. 19
1.6 Referințe pentru capitolul 1. 19
2 MapReduce și noua pila de software
21
2.1 Sisteme de fișiere distribuite. 22
2.1.1 Organizarea fizică a nodurilor de calcul. 22
2.1.2 Organizarea sistemului de fișiere pe scară largă. 24
2.2 MapReduce. 25
2.2.1 Sarcinile hărții. 25
2.2.2 Gruparea după cheie. 26
2.2.3 Sarcinile de reducere. 27

2.2.4 Combinatoare. 27
2.2.5 Detalii despre executarea MapReduce. 28
2.2.6 Tratarea defecțiunilor nodului. 30
2.2.7 Exerciții pentru secțiunea 2.2. 30
2.3 Algoritmi folosind MapReduce. 30
2.3.1 Multiplicarea Matricii-Vector prin MapReduce. 31
2.3.2 Dacă Vectorul v nu se poate potrivi în memoria principală. 32
2.3.3 Operații relaționale-algebră. 33
2.3.4 Selecții de calcul prin MapReduce. 35
2.3.5 Calcularea proiecțiilor prin MapReduce. 36
2.3.6 Unire, intersecție și diferență prin MapReduce. . . . 36
2.3.7 Calcularea Natural Join prin MapReduce. 37
2.3.8 Gruparea și agregarea după MapReduce. 38
2.3.9 Multiplicarea matricei. 38
2.3.10 Multiplicarea matricei cu un singur pas MapReduce. 39
2.3.11 Exerciții pentru secțiunea 2.3. 40
2.4 Extensii la MapReduce. 41

2.4.1 Sisteme de flux de lucru. 42
2.4.2 Scânteie. 43
2.4.3 Implementarea scânteii. 46
2.4.4 TensorFlow. 48
2.4.5 Extensii recursive la MapReduce. 49
2.4.6 Sisteme sincrone în vrac. 51
2.4.7 Exerciții pentru secțiunea 2.4. 53
2.5 Modelul de comunicare-cost. 53
2.5.1 Costul comunicării pentru rețelele de activități. 53
2.5.2 Ora ceasului de perete. 55
2.5.3 Îmbinări multiple. 56
2.5.4 Exerciții pentru secțiunea 2.5. 59
2.6 Teoria complexității pentru MapReduce. 61
2.6.1 Dimensiunea reductorului și rata de replicare. 61
2.6.2 Un exemplu: asocieri similare. 62
2.6.3 Un model de grafic pentru problemele MapReduce. 64
2.6.4 Scheme de cartografiere. 65
2.6.5 Când nu sunt prezente toate intrările. 67
2.6.6 Limite inferioare la rata de replicare. 68
2.6.7 Studiu de caz: Multiplicarea matricei. 69
2.6.8 Exerciții pentru secțiunea 2.6. 73
2.7 Rezumatul capitolului 2. 74
2.8 Referințe pentru capitolul 2. 77

CUPRINS
ix
3 Găsirea unor articole similare
81
3.1 Aplicații ale asemănării seturilor. 82
3.1.1 Similitudinea seturilor Jaccard. 82
3.1.2 Asemănarea documentelor. 82
3.1.3 Filtrarea colaborativă ca o problemă similară. 84
3.1.4 Exerciții pentru secțiunea 3.1. 86
3.2 Shingling de documente. 86
3.2.1 k-zona zoster. 86
3.2.2 Alegerea dimensiunii șindrilelor. 87
3.2.3 Hinging Shinging. 87
3.2.4 Zoster construit din cuvinte. 88
3.2.5 Exerciții pentru secțiunea 3.2. 88
3.3 Rezumate ale seturilor care păstrează similitudinea. 89
3.3.1 Reprezentarea matricială a seturilor. 89
3.3.2 Minhashing. 90
3.3.3 Minhashing și similaritatea Jaccard. 91
3.3.4 Semnături Minhash. 91
3.3.5 Calcularea semnăturilor Minhash în practică. 92
3.3.6 Accelerarea Minhashing-ului. 94
3.3.7 Accelerarea utilizând funcțiile Hash. 95
3.3.8 Exerciții pentru secțiunea 3.3. 98
3.4 Hashing sensibil la localitate pentru documente. 99
3.4.1 LSH pentru semnături Minhash. 100
3.4.2 Analiza tehnicii de bandare. 101
3.4.3 Combinarea tehnicilor. 103
3.4.4 Exerciții pentru secțiunea 3.4. 104
3.5 Măsuri la distanță. 104
3.5.1 Definiția unei măsuri la distanță. 105
3.5.2 Distanțe euclidiene. 105
3.5.3 Distanța Jaccard. 106
3.5.4 Distanța cosinusului. 107
3.5.5 Editare distanță. 108
3.5.6 Distanța de lovire. 109
3.5.7 Exerciții pentru secțiunea 3.5. 109
3.6 Teoria funcțiilor sensibile la localitate. 111
3.6.1 Funcții sensibile la localitate. 112
3.6.2 Familii sensibile la localitate pentru distanța Jaccard. 112
3.6.3 Amplificarea unei familii sensibile la localitate. 113
3.6.4 Exerciții pentru secțiunea 3.6. 116
3.7 Familii LSH pentru alte măsuri la distanță. 116

3.7.1 Familii LSH pentru distanță de lovire. 117
3.7.2 Hiperavioane aleatorii și distanța cosinusului. 117
3.7.3 Schițe. 119
3.7.4 Familii LSH pentru distanța euclidiană. 119
3.7.5 Mai multe familii LSH pentru spații euclidiene. 121

Pagina 10

3.7.6 Exerciții pentru secțiunea 3.7. 121
3.8 Aplicații de Hashing sensibil la localitate. 122
3.8.1 Rezoluția entității. 123
3.8.2 Un exemplu de rezoluție a entității. 123
3.8.3 Validarea meciurilor de înregistrare. 124
3.8.4 Potrivirea amprentelor digitale. 125
3.8.5 O familie LSH pentru potrivirea amprentelor digitale. 126
3.8.6 Articole de știri similare. 128
3.8.7 Exerciții pentru secțiunea 3.8. 129
3.9 Metode pentru grade ridicate de similitudine. 130
3.9.1 Găsirea articolelor identice. 130
3.9.2 Reprezentarea seturilor ca șiruri. 131
3.9.3 Filtrare bazată pe lungime. 131
3.9.4 Indexarea prefixelor. 132
3.9.5 Utilizarea informațiilor de poziție. 133
3.9.6 Utilizarea poziției și lungimii în indexuri. 135
3.9.7 Exerciții pentru secțiunea 3.9. 137
3.10 Rezumatul capitolului 3. 138
3.11 Referințe pentru capitolul 3. 141
4 Fluxuri de date miniere
143
4.1 Modelul de date al fluxului. 143
4.1.1 Un sistem de gestionare a fluxului de date. 144
4.1.2 Exemple de surse de flux. 145
4.1.3 Interogări de flux. 146
4.1.4 Probleme în procesarea fluxului. 147
4.2 Eșantionarea datelor într-un flux. 148
4.2.1 Un exemplu motivant. 148
4.2.2 Obținerea unui eșantion reprezentativ. 149
4.2.3 Problema generală de eșantionare. 149
4.2.4 Variația dimensiunii probei. 150
4.2.5 Exerciții pentru secțiunea 4.2. 150
4.3 Filtrarea fluxurilor. 151
4.3.1 Un exemplu motivant. 151
4.3.2 Filtrul Bloom. 152
4.3.3 Analiza filtrării Bloom. 152
4.3.4 Exerciții pentru secțiunea 4.3. 153
4.4 Numărarea elementelor distincte într-un flux. 154
4.4.1 Problema distincției. 154
4.4.2 Algoritmul Flajolet-Martin. 155
4.4.3 Combinarea estimărilor. 156
4.4.4 Cerințe de spațiu. 156
4.4.5 Exerciții pentru secțiunea 4.4. 157
4.5 Estimarea momentelor. 157
4.5.1 Definiția momentelor. 157

Pagina 11

4.5.2 Algoritmul Alon-Matias-Szegedy pentru al doilea
Momente. 158
4.5.3 De ce funcționează algoritmul Alon-Matias-Szegedy. 159
4.5.4 Momente de ordin superior. 160
4.5.5 Tratarea fluxurilor infinite. 160
4.5.6 Exerciții pentru secțiunea 4.5. 161
4.6 Numărarea celor într-o fereastră. 162
4.6.1 Costul numărărilor exacte. 163

4.6.2 Algoritmul Datar-Gionis-Indyk-Motwani. 163
4.6.3 Cerințe de stocare pentru algoritmul DGIM. 165
4.6.4 Răspunsul la interogare în algoritmul DGIM. 165
4.6.5 Menținerea condițiilor DGIM. 166
4.6.6 Reducerea erorii. 167
4.6.7 Extensii la numărarea celor. 168
4.6.8 Exerciții pentru secțiunea 4.6. 169
4.7 Windows în descompunere. 169
4.7.1 Problema elementelor cele mai comune. 169
4.7.2 Definiția ferestrei în descompunere. 170
4.7.3 Găsirea celor mai populare elemente. 171
4.8 Rezumatul capitolului 4. 172
4.9 Referințe pentru capitolul 4. 173
5 Analiza legăturilor
175
5.1 PageRank. 175
5.1.1 Motoare de căutare timpurie și spam de termeni. 176
5.1.2 Definiția PageRank. 177
5.1.3 Structura Web-ului. 181
5.1.4 Evitarea impasurilor. 182
5.1.5 Capcane pentru păianjen și impozitare. 185
5.1.6 Utilizarea PageRank într-un motor de căutare. 187
5.1.7 Exerciții pentru secțiunea 5.1. 187
5.2 Calcul eficient al PageRank. 189
5.2.1 Reprezentarea matricilor de tranziție. 190
5.2.2 Iterarea PageRank folosind MapReduce. 191
5.2.3 Utilizarea combinatorilor pentru a consolida vectorul de rezultate. . . . 191
5.2.4 Reprezentarea blocurilor matricei de tranziție. 192
5.2.5 Alte abordări eficiente ale iterației PageRank. . . . 193
5.2.6 Exerciții pentru secțiunea 5.2. 195
5.3 PageRank sensibil la subiect. 195
5.3.1 Motivația pentru Page Rank-ul sensibil la subiect. 195
5.3.2 Plimbări aleatorii părtinitoare. 196
5.3.3 Utilizarea PageRank sensibil la subiect. 197
5.3.4 deducerea subiectelor din cuvinte. 198
5.3.5 Exerciții pentru secțiunea 5.3. 199
5.4 Link spam. 199

Pagina 12

5.4.1 Arhitectura unei ferme de spam. 199
5.4.2 Analiza unei ferme de spam. 201
5.4.3 Combaterea spamului prin link. 202
5.4.4 TrustRank. 202
5.4.5 Masa spam. 203
5.4.6 Exerciții pentru secțiunea 5.4. 203
5.5 Huburi și autorități. 204
5.5.1 Intuitia din spatele HITS. 204
5.5.2 Formalizarea interesului și autorității. 205
5.5.3 Exerciții pentru secțiunea 5.5. 208
5.6 Rezumatul capitolului 5. 208
5.7 Referințe pentru capitolul 5. 212
6 elemente frecvente
213
6.1 Modelul Market-Basket. 214
6.1.1 Definiția seturilor de articole frecvente. 214
6.1.2 Aplicații de obiecte frecvente. 216
6.1.3 Reguli de asociere. 217
6.1.4 Găsirea regulilor de asociere cu mare încredere. 219
6.1.5 Exerciții pentru secțiunea 6.1. 219
6.2 Coșuri de piață și algoritmul A-Priori. 221
6.2.1 Reprezentarea datelor din coșul de piață. 221
6.2.2 Utilizarea memoriei principale pentru numărarea seturilor de articole. 222
6.2.3 Monotonicitatea seturilor de articole. 224
6.2.4 Tirania numărării perechilor. 225
6.2.5 Algoritmul A-Priori. 225
6.2.6 A-Priori pentru toate seturile obiecte frecvente. 226

6.2.7 Exerciții pentru secțiunea 6.2. 229
6.3 Gestionarea seturilor de date mai mari în memoria principală. 230
6.3.1 Algoritmul lui Park, Chen și Yu. 230
6.3.2 Algoritmul cu mai multe etape. 232
6.3.3 Algoritmul Multihash. 234
6.3.4 Exerciții pentru secțiunea 6.3. 236
6.4 Algoritmi de trecere limitată. 238
6.4.1 Algoritmul simplu, randomizat. 238
6.4.2 Evitarea erorilor în algoritmii de eșantionare. 239
6.4.3 Algoritmul lui Savasere, Omiecinski și
Navathe. 240
6.4.4 Algoritmul SON și MapReduce. 241
6.4.5 Algoritmul lui Toivonen. 242
6.4.6 De ce funcționează algoritmul lui Toivonen. 243
6.4.7 Exerciții pentru secțiunea 6.4. 244
6.5 Numărarea elementelor frecvente într-un flux. 244
6.5.1 Metode de eșantionare pentru fluxuri. 245
6.5.2 Seturi de articole frecvente în Windows în descompunere. 246

CUPRINS
xiii
6.5.3 Metode hibride. 247
6.5.4 Exerciții pentru secțiunea 6.5. 247
6.6 Rezumatul capitolului 6. 248
6.7 Referințe pentru capitolul 6. 250
7 Clustering
253
7.1 Introducere în tehnicile de clusterizare. 253
7.1.1 Puncte, spații și distanțe. 253
7.1.2 Strategii de clusterizare. 255
7.1.3 Blestemul dimensionalității. 256
7.1.4 Exerciții pentru secțiunea 7.1. 257
7.2 Clusterizarea ierarhică. 257
7.2.1 Clusterizarea ierarhică într-un spațiu euclidian. 258
7.2.2 Eficiența clusterizării ierarhice. 260
7.2.3 Reguli alternative pentru controlul ierarhic
Clustering. 261
7.2.4 Clusterizarea ierarhică în spații neuclidiene. 264
7.2.5 Exerciții pentru secțiunea 7.2. 265
7.3 K-înseamnă algoritmi. 266
7.3.1 Bazele K-Means. 267
7.3.2 Inițializarea clusterelor pentru K-Means. 267
7.3.3 Alegerea valorii corecte a lui k. 268
7.3.4 Algoritmul lui Bradley, Fayyad și Reina. 269
7.3.5 Prelucrarea datelor în algoritmul BFR. 271
7.3.6 Exerciții pentru secțiunea 7.3. 274
7.4 Algoritmul CURE. 274
7.4.1 Inițializare în CURE. 275
7.4.2 Finalizarea algoritmului CURE. 276
7.4.3 Exerciții pentru secțiunea 7.4. 277
7.5 Clustering în spații non-euclidiene. 278
7.5.1 Reprezentarea clusterelor în algoritmul GRGPF. 278
7.5.2 Inițializarea Arborelui Clusterului. 279
7.5.3 Adăugarea de puncte în algoritmul GRGPF. 280
7.5.4 Divizarea și fuzionarea clusterelor. 281
7.5.5 Exerciții pentru secțiunea 7.5. 282
7.6 Clustering pentru fluxuri și paralelism. 282
7.6.1 Modelul de calcul al fluxului. 283
7.6.2 Un algoritm de grupare a fluxului. 283
7.6.3 Inițializarea găleților. 284
7.6.4 Fuzionarea găleților. 284
7.6.5 Răspunsul la întrebări. 287
7.6.6 Clusterizarea într-un mediu paralel. 287
7.6.7 Exerciții pentru secțiunea 7.6. 288
7.7 Rezumatul capitolului 7. 288
7.8 Referințe pentru capitolul 7. 292

8 Publicitate pe web
293
8.1 Probleme în publicitatea online. 293
8.1.1 Oportunități de publicitate. 293
8.1.2 Plasarea directă a anunțurilor. 294
8.1.3 Probleme pentru anunțurile grafice. 295
8.2 Algoritmi on-line. 296
8.2.1 Algoritmi on-line și off-line. 296
8.2.2 Algoritmi lacomi. 297
8.2.3 Raportul competitiv. 298
8.2.4 Exerciții pentru secțiunea 8.2. 298
8.3 Problema potrivirii. 299
8.3.1 Meciuri și Meciuri Perfecte. 299
8.3.2 Algoritmul lacom pentru potrivirea maximă. 300
8.3.3 Raportul competitiv pentru potrivirea lacomă. 301
8.3.4 Exerciții pentru secțiunea 8.3. 302
8.4 Problema AdWords. 302
8.4.1 Istoricul publicității în căutare. 303
8.4.2 Definiția problemei AdWords. 303
8.4.3 Abordarea lacomă a problemei AdWords. 304
8.4.4 Algoritmul de echilibru. 305
8.4.5 O limită inferioară a raportului competitiv pentru echilibru. . . . 306
8.4.6 Algoritmul echilibrului cu mulți ofertanți. 308
8.4.7 Algoritmul de echilibru generalizat. 309
8.4.8 Observații finale despre problema AdWords. 310
8.4.9 Exerciții pentru secțiunea 8.4. 311
8.5 Implementarea AdWords. 311
8.5.1 Sumele potrivite și interogările de căutare. 312
8.5.2 Probleme de potrivire mai complexe. 312
8.5.3 Un algoritm de potrivire pentru documente și oferte. 313
8.6 Rezumatul capitolului 8. 315
8.7 Referințe pentru capitolul 8. 317
9 Sisteme de recomandare
319
9.1 Un model pentru sistemele de recomandare. 319
9.1.1 Matricea utilitară. 320
9.1.2 Coada lungă. 321
9.1.3 Aplicații ale sistemelor de recomandare. 321
9.1.4 Popularea matricei de utilități. 323
9.2 Recomandări bazate pe conținut. 324
9.2.1 Profiluri de articole. 324
9.2.2 Descoperirea caracteristicilor documentelor. 325
9.2.3 Obținerea caracteristicilor articolelor din etichete. 326
9.2.4 Reprezentarea profilurilor articolelor. 327
9.2.5 Profiluri de utilizator. 328
9.2.6 Recomandarea de articole utilizatorilor pe baza conținutului. 329

9.2.7 Algoritmi de clasificare. 330
9.2.8 Exerciții pentru secțiunea 9.2. 332
9.3 Filtrare colaborativă. 333
9.3.1 Măsurarea similitudinii. 334
9.3.2 Dualitatea asemănării. 336
9.3.3 Clusterizarea utilizatorilor și a articolelor. 337
9.3.4 Exerciții pentru secțiunea 9.3. 339
9.4 Reducerea dimensiunii. 340
9.4.1 Descompunerea UV. 340
9.4.2 Eroare Root-Mean-Square. 341
9.4.3 Calculul incremental al unei descompuneri UV. . . . 342
9.4.4 Optimizarea unui element arbitrar. 344
9.4.5 Construirea unui algoritm complet de descompunere UV. . . . 346
9.4.6 Exerciții pentru secțiunea 9.4. 348
9.5 Provocarea Netflix. 349

9.6 Rezumatul capitolului 9. 350
9.7 Referințe pentru capitolul 9. 352
10 miniere grafice de rețea socială
355
10.1 Rețelele sociale ca grafice. 356
10.1.1 Ce este o rețea socială? 356
10.1.2 Rețelele sociale ca grafice. 356
10.1.3 Soiuri de rețele sociale. 358
10.1.4 Grafice cu mai multe tipuri de noduri. 359
10.1.5 Exerciții pentru secțiunea 10.1. 360
10.2 Clusterizarea graficelor din rețelele sociale. 361
10.2.1 Măsuri la distanță pentru graficele de rețea socială. 361
10.2.2 Aplicarea metodelor standard de clusterizare. 361
10.2.3 Interval. 363
10.2.4 Algoritmul Girvan-Newman. 363
10.2.5 Utilizarea Betweenness pentru a găsi comunități. 366
10.2.6 Exerciții pentru secțiunea 10.2. 367
10.3 Descoperirea directă a comunităților. 368
10.3.1 Găsirea clicurilor. 369
10.3.2 Completarea graficelor bipartite. 369
10.3.3 Găsirea subgrafelor bipartite complete. 370
10.3.4 De ce trebuie să existe grafice complete bipartite. 371
10.3.5 Exerciții pentru secțiunea 10.3. 373
10.4 Partiționarea graficelor. 374
10.4.1 Ce face o partiție bună? 374
10.4.2 Tăieturi normalizate. 375
10.4.3 Câteva matrice care descriu grafice. 375
10.4.4 Valori proprii ale matricei laplaciene. 377
10.4.5 Metode alternative de partiționare. 379
10.4.6 Exerciții pentru secțiunea 10.4. 380

Pagina 16

10.5 Găsirea comunităților suprapuse. 381
10.5.1 Natura comunităților. 381
10.5.2 Estimarea probabilității maxime. 381
10.5.3 Modelul grafic de afiliere. 383
10.5.4 Optimizarea discretă a sarcinilor comunitare. . . . 385
10.5.5 Evitarea utilizării modificărilor discrete de membru. . . . 386
10.5.6 Exerciții pentru secțiunea 10.5. 388
10.6 Simrank. 389
10.6.1 Random Walkers pe un grafic social. 390
10.6.2 Plimbări aleatorii cu repornire. 391
10.6.3 Simrank aproximativ. 393
10.6.4 De ce funcționează aproximativ Simrank. 395
10.6.5 Aplicarea Simrank la găsirea comunităților. 396
10.6.6 Exerciții pentru secțiunea 10.6. 397
10.7 Numărarea triunghiurilor. 399
10.7.1 De ce să numărăm triunghiurile? 399
10.7.2 Un algoritm pentru găsirea triunghiurilor. 400
10.7.3 Optimitatea algoritmului de găsire a triunghiului. 401
10.7.4 Găsirea triunghiurilor folosind MapReduce. 401
10.7.5 Utilizarea mai puținor sarcini de reducere. 403
10.7.6 Exerciții pentru secțiunea 10.7. 404
10.8 Proprietăți de vecinătate ale graficelor. 405
10.8.1 Grafice și vecinătăți direcționate. 405
10.8.2 Diametrul unui grafic. 406
10.8.3 Închidere tranzitorie și accesibilitate. 407
10.8.4 Accesibilitate prin MapReduce. 408
10.8.5 Evaluare seminaivă. 409
10.8.6 Închidere tranzitivă liniară. 410
10.8.7 Închiderea tranzitivă prin dublare recursivă. 411
10.8.8 Închidere tranzitivă inteligentă. 413
10.8.9 Compararea metodelor. 415
10.8.10 Închiderea tranzitivă prin reducerea graficului. 417
10.8.11 Aproximarea dimensiunilor vecinătăților. 418
10.8.12 Exerciții pentru secțiunea 10.8. 420

10.9 Rezumatul capitolului 10. 422
10.10 Referințe pentru capitolul 10. 425
11 Reducerea dimensionalității
429
11.1 Valori proprii și vectori proprii ai matricilor simetrice. 430
11.1.1 Definiții. 430
11.1.2 Calcularea valorilor proprii și a vectorilor proprii. 431
11.1.3 Găsirea perechilor proprii prin iterație de putere. 432
11.1.4 Matricea vectorilor proprii. 435
11.1.5 Exerciții pentru secțiunea 11.1. 435
11.2 Analiza componentelor principale. 436

Pagina 17

11.2.1 Un exemplu ilustrativ. 437
11.2.2 Utilizarea vectorilor proprii pentru reducerea dimensiunii. 440
11.2.3 Matricea distanțelor. 441
11.2.4 Exerciții pentru secțiunea 11.2. 442
11.3 Descompunerea valorii singulare. 442
11.3.1 Definiția SVD. 442
11.3.2 Interpretarea SVD. 444
11.3.3 Reducerea dimensiunii utilizând SVD. 446
11.3.4 De ce funcționează reducerea la zero a valorilor singulare scăzute. 447
11.3.5 Interogarea utilizând concepte. 449
11.3.6 Calcularea SVD-ului unei matrice. 450
11.3.7 Exerciții pentru secțiunea 11.3. 451
11.4 Descompunerea CUR. 452
11.4.1 Definiția CUR. 453
11.4.2 Alegerea corectă a rândurilor și coloanelor. 454
11.4.3 Construirea matricei de mijloc. 455
11.4.4 Descompunerea completă a CUR. 456
11.4.5 Eliminarea rândurilor și coloanelor duplicate. 457
11.4.6 Exerciții pentru secțiunea 11.4. 458
11.5 Rezumatul capitolului 11. 458
11.6 Referințe pentru capitolul 11. 460
12 Învățare automată pe scară largă
463
12.1 Modelul de învățare automată. 464
12.1.1 Seturi de antrenament. 464
12.1.2 Câteva exemple ilustrative. 464
12.1.3 Abordări ale învățării automate. 467
12.1.4 Arhitectura de învățare automată. 468
12.1.5 Exerciții pentru secțiunea 12.1. 471
12.2 Perceptroni. 471
12.2.1 Antrenarea unui Perceptron cu Prag zero. 471
12.2.2 Convergența perceptronelor. 475
12.2.3 Algoritmul Winnow. 475
12.2.4 Permiterea pragului să varieze. 477
12.2.5 Perceptroni multiclase. 479
12.2.6 Transformarea setului de antrenament. 480
12.2.7 Probleme cu perceptronii. 481
12.2.8 Implementarea paralelă a perceptronelor. 482
12.2.9 Exerciții pentru secțiunea 12.2. 484
12.3 Suport-Mașini Vector. 485
12.3.1 Mecanica unui SVM. 486
12.3.2 Normalizarea hiperavionului. 487
12.3.3 Găsirea separatorilor aproximativi optimi. 489
12.3.4 Soluții SVM după coborârea gradientului. 492
12.3.5 Coborârea în gradient stochastic. 495

Pagina 18

12.3.6 Implementarea paralelă a SVM. 496
12.3.7 Exerciții pentru secțiunea 12.3. 497

12.4 Învățarea de la vecinii cei mai apropiați. 497
12.4.1 Cadrul pentru calculele cele mai apropiate. . . . 498
12.4.2 Învățarea cu cel mai apropiat vecin. 498
12.4.3 Învățarea funcțiilor unidimensionale. 499
12.4.4 Regresia kernelului. 502
12.4.5 Tratarea datelor euclidiene de înaltă dimensiune. 502
12.4.6 Tratarea distanțelor neeuclidiene. 504
12.4.7 Exerciții pentru secțiunea 12.4. 504
12.5 Arbori de decizie. 505
12.5.1 Utilizarea unui arbore de decizie. 506
12.5.2 Măsuri de impuritate. 507
12.5.3 Proiectarea unui nod arbore de decizie. 508
12.5.4 Selectarea unui test folosind o caracteristică numerică. 509
12.5.5 Selectarea unui test folosind o caracteristică categorică. 511
12.5.6 Proiectarea în paralel a arborilor de decizie. 513
12.5.7 Tunderea nodului. 514
12.5.8 Păduri de decizie. 515
12.5.9 Exerciții pentru secțiunea 12.5. 516
12.6 Compararea metodelor de învățare. 517
12.7 Rezumatul capitolului 12. 518
12.8 Referințe pentru capitolul 12. 520
13 Rețele neuronale și învățare profundă
523
13.1 Introducere în plasele neuronale. 524
13.1.1 Rețelele neuronale, în general. 525
13.1.2 Interconectări între noduri. 527
13.1.3 Rețele neuronale convoluționale. 527
13.1.4 Probleme de proiectare pentru plasele neuronale. 528
13.1.5 Exerciții pentru secțiunea 13.1. 528
13.2 Rețele Feedforward dense. 529
13.2.1 Notare algebră liniară. 529
13.2.2 Funcții de activare. 531
13.2.3 Sigmoidul. 532
13.2.4 Tangenta hiperbolică. 532
13.2.5 Softmax. 533
13.2.6 Unitate liniară certificată. 534
13.2.7 Funcții de pierdere. 535
13.2.8 Pierderea de regresie. 536
13.2.9 Pierderea clasificării. 537
13.2.10 Exerciții pentru secțiunea 13.2. 538
13.3 Propagarea înapoi și coborârea în gradient. 539
13.3.1 Calculați graficele. 540
13.3.2 Gradienți, iacobieni și regula lanțului. 541

13.3.3 Algoritmul Backpropagation. 542
13.3.4 Coborârea gradientului iterant. 545
13.3.5 Tensori. 546
13.3.6 Exerciții pentru secțiunea 13.3. 548
13.4 Rețele neuronale convoluționale. 548
13.4.1 Straturi convoluționale. 549
13.4.2 Convoluție și corelație încrucișată. 552
13.4.3 Straturile de punere în comun. 553
13.4.4 Arhitectura CNN. 553
13.4.5 Implementare și instruire. 555
13.4.6 Exerciții pentru secțiunea 13.4. 556
13.5 Rețele neuronale recurente. 557
13.5.1 Instruirea RNN-urilor. 559
13.5.2 Gradiente de dispariție și explozie. 561
13.5.3 Memorie pe termen scurt (LSTM). 562
13.5.4 Exerciții pentru secțiunea 13.5. 564
13.6 Regularizare. 565
13.6.1 Penalizări ale normelor. 565
13.6.2 abandon. 566
13.6.3 Oprirea timpurie. 566
13.6.4 Augmentarea setului de date. 567

13.7 Rezumatul capitolului 13. 567
13.8 Referințe pentru capitolul 13. 569

Pagina 20

Pagina 21

Capitolul 1
Exploatarea datelor

În acest capitol introductiv începem cu esența mineritului de date și o discuție despre modul în care mineritul de date este tratat de diferitele discipline care tribut acestui domeniu. Acoperim „Principiul lui Bonferroni”, care este într-adevăr un avertisment despre utilizarea excesivă a capacității de extragere a datelor. Acest capitol este, de asemenea, locul unde rezumăm câteva idei utile care nu sunt minerite de date în sine, dar sunt folositoare înțelegerea unor concepte importante de data mining. Acestea includ TF.IDF măsură a importanței cuvintelor, comportamentul funcțiilor hash și a indexurilor, și identități care implică e, baza logaritmilor naturali. În cele din urmă, oferim un schița subiectelor tratate în bilanțul cărții.

1.1 Ce este Data Mining?

În anii 1990, „exploatarea datelor” era un nou concept interesant și popular. În jurul În 2010, oamenii au început să vorbească despre „big data”. Astăzi, termenul popular este ,,Știința datelor”. Cu toate acestea, în tot acest timp, conceptul a rămas același: utilizați cel mai puternic hardware, cele mai puternice sisteme de programare și cei mai eficienți algoritmi pentru rezolvarea problemelor din știință, comerț, asistență medicală, guvern, umaniste și multe alte domenii ale efortului uman.

1.1.1 Modelare

Pentru mulți, extragerea datelor este procesul de creare a unui model din date, adesea de către procesul de învățare automată, pe care îl menționăm în secțiunea 1.1.3 și îl discutăm mai complet în capitolul 12. Cu toate acestea, mai general, obiectivul de extragere a datelor este un algoritm. De exemplu, discutăm hashingul sensibil la localitate în capitolul 3 și o serie de algoritmi de extragere a fluxurilor în capitolul 4, dintre care niciunul nu implică un model. Cu toate acestea, în multe aplicații importante, partea dificilă este crearea de model și, odată ce modelul este disponibil, algoritmul de utilizare a modelului este direct.

Pagina 22

Exemplul 1.1: luați în considerare problema detectării e-mailurilor care sunt phishing atacuri. Cea mai comună abordare este de a construi un model de e-mailuri de phishing, poate examinând e-mailurile pe care oamenii le-au raportat recent ca fiind phishing atacuri și căutarea cuvintelor sau frazelor care apar neobișnuit de des în acele e-mailuri, cum ar fi „prințul nigerian” sau „verificați contul”. Modelul ar putea fii greutăți asupra cuvintelor, cu greutăți pozitive pentru cuvintele care apar frecvent în e-mailuri de phishing și greutăți negative pentru cuvintele care nu. Apoi algoritmul detectarea e-mailurilor de phishing este simplă. Aplicați modelul la fiecare e-mail, adică sumă greutățile cuvintelor din acel e-mail și spun că e-mailul este phishing dacă și numai dacă suma este pozitivă. Găsirea celor mai bune greutăți este o problemă dificilă, noi va prelua în secțiunea 12.2. ✷

1.1.2 Modelare statistică

Statisticienii au fost primii care au folosit termenul „minerit de date”. Inițial, „date minerit ”sau„ dragare de date ”a fost un termen derogatoriu referitor la încercările de a extrageți informații care nu au fost acceptate de date. Secțiunea 1.2 ilustrează genul de erori pe care le poți face încercând să extragi ceea ce nu este într-adevăr în date. Astăzi, „exploatarea datelor” a luat un sens pozitiv. Acum, statisticienii văd extragerea datelor ca construcție a unui model statistic, adică un suport distribuție din care sunt extrase datele vizibile.

Exemplul 1.2: Să presupunem că datele noastre sunt un set de numere. Aceste date sunt mult

mai simplu decât datele care ar fi extrase de date, dar va servi ca exemplu. A statisticianul ar putea decide că datele provin dintr-o distribuție gaussiană și utilizați o formulă pentru a calcula parametrii cei mai probabili ai acestui Gauss. Media și deviația standard a acestei distribuții gaussiene caracterizează complet distribuție și ar deveni modelul datelor. ✷

1.1.3 Învățarea automată

Există unii care consideră că mineritul de date este sinonim cu învățarea automată. Nu există nicio îndoială că unele date mining folosesc în mod adecvat algoritmi de la învățare automată. Practicanții de învățare automată folosesc datele ca un set de instruire, pentru a instrui un algoritm de unul dintre numeroasele tipuri utilizate pentru învățarea automată, cum ar fi ca rețele Bayes, mașini vector-suport, arbori de decizie, modele ascunse Markov, și o mare varietate de alții.

Există situații în care utilizarea datelor în acest mod are sens. Tipic cazul în care învățarea automată este o abordare bună este atunci când nu avem prea puțină idee ce spun datele despre problema pe care încercăm să o rezolvăm. De exemplu, ea este destul de neclar despre ce este vorba despre filme care îi face pe anumiți pasionați de film sau nu-mi place. Astfel, răspunzând la „provocarea Netflix” de a concepe un algoritm care prezice evaluările filmelor de către utilizatori, pe baza unui eșantion de răspunsuri ale acestora, algoritmii de învățare automată s-au dovedit destul de reuși ți. Vom discuta despre formă simplă a acestui tip de algoritm în secțiunea 9.4.

Cu toate acestea, învățarea automată poate fi necompetitivă în situațiile în care putem descrieți obiectivele mineritului mai direct. Un caz interesant este încercarea lui WhizBang! Laboratoarele [1] pentru a utiliza învățarea automată pentru a localiza oamenii CV-uri pe web. Nu a reușit să facă mai bine decât algoritmii proiectați de mână pentru a căuta câteva dintre cuvintele și frazele evidente care apar în CV tipic. Din moment ce toți cei care au privit sau au scris un CV au un o idee destul de bună despre ce conțin CV-uri, nu a existat niciun mister despre ce face ca o pagină Web să fie un CV. Astfel, nu a existat niciun avantaj învățarea asupra proiectării directe a unui algoritm pentru a descoperi CV-uri.

O altă problemă cu unele metode de învățare automată este că acestea sunt deseori produce un model care, deși poate fi destul de precis, nu este explicabil. În unele cazuri, explicabilitatea nu este importantă. De exemplu, dacă întrebați Google de ce a clasificat un Gmail ca spam, de obicei spune ceva de genul „arată ca altul mesaje pe care oamenii le-au identificat ca spam. ” Adică, e-mailul se potrivește orice model de spam Google a dezvoltat în acea zi, fără îndoială, folosind un tehnică din arsenalul algoritmilor de învățare automată. Această explicație este probabil satisfăcător. Chiar nu ne pasă ce face Google, atâta timp cât este ia decizia corectă de spam / non-spam.

Pe de altă parte, luați în considerare o companie de asigurări auto care creează un model al riscului asociat fiecărui șofer și alocă prime diferite fiecare, după model. Dacă prima dvs. crește, s-ar putea să doriți o explicație a ceea ce face noul model și de ce a schimbat estimarea riscul tău. Din păcate, în multe metode de învățare automată, în special „adânc învățare ”, în care modelul implică strat peste strat de elemente mici, fiecare dintre ele care ia o decizie pe baza intrărilor din stratul anterior, s-ar putea să nu fie este posibil să se dea o explicație coerentă a ceea ce face modelul.

1.1.4 Abordări computaționale ale modelării

Spre deosebire de abordarea statistică, informaticienii tind să privească datele mineritul ca o problemă algoritmică. În acest caz, un model al datelor este pur și simplu răspunsul la o interogare complexă despre aceste date. De exemplu, având în vedere setul din numerele din Exemplul 1.2, am putea calcula media și standardul acestora deviere. Rețineți că este posibil ca aceste valori să nu fie parametrii Gaussianului care se potrivește cel mai bine datelor, deși aproape sigur vor fi foarte apropiate dacă dimensiunea datelor este mare, iar sursa datelor este cu adevărat gaussiană.

Există multe abordări diferite pentru modelarea datelor. Avem deja a menționat posibilitatea construirii unui proces aleator prin care datele s-ar fi putut genera. Cele mai multe alte abordări ale modelării pot fi descrise ca oricare

1. Rezumând datele succint și aproximativ sau
2. Extragerea celor mai proeminente caracteristici ale datelor și ignorarea celorlalte.

[1] Această pornire a încercat să folosească învățarea automată pentru a extrage date pe scară largă și a angajat pe mulți dintre cei mai buni oameni care învață mașina să facă acest lucru. Din păcate, nu a reușit să supraviețuiască.

Vom explora aceste două abordări în secțiunile următoare.

1.1.5 Rezumare

Una dintre cele mai interesante forme de rezumare este ideea PageRank, care
Google a reușit Google și pe care îl vom aborda în capitolul 5. În acest formular
de minerit Web, întreaga structură complexă a Web-ului este rezumată printr-un
număr unic pentru fiecare pagină. Acest număr, „PageRank” al paginii, este
(simplificând oarecum) probabilitatea ca un mers aleatoriu pe grafic
ar fi la acea pagină la un moment dat. În mod remarcabil, acest clasament reflectă foarte mult
Ei bine, „importanța” paginii - gradul în care ar face căutătorii tipici
ca pagina respectivă a revenit ca răspuns la interogarea lor de căutare.
O altă formă importantă de rezumat - gruparea - va fi acoperită în capitolul
ter 7. Aici, datele sunt privite ca puncte într-un spațiu multidimensional. Puncte
care sunt „aproape” în acest spațiu sunt atribuite aceluiași cluster. Clusterele
ele însele sunt rezumate, poate prin oferirea centrului clusterului și
distanța medie față de centroidul de puncte din cluster. Acest grup
rezumatele devin apoi rezumatul întregului set de date.
Exemplul 1.3: A avut loc o faimoasă instanță de grupare pentru a rezolva o problemă
cu mult timp în urmă la Londra și s-a făcut în întregime fără calculatoare. [2] Medicul
John Snow, care se ocupa de un focar de holeră, a trasat cazurile pe o hartă a
oraș. O mică ilustrație care sugerează procesul este prezentată în Fig. 1.1.
Figura 1.1: Plotarea cazurilor de holeră pe o hartă a Londrei
Cazurile s-au grupat în jurul unor intersecții de drumuri. Aceste inter-
secțiunile erau locațiile puțurilor care deveniseră contaminate; oamenii care
[2] A se vedea http://en.wikipedia.org/wiki/1854 Broad Street focar de holera.

---

au trăit în apropierea acestor fântâni, s-au îmbolnăvit, în timp ce oamenii care trăiau mai aproape de fântâni care
nu a fost contaminat nu s-a îmbolnăvit. Fără capacitatea de a grupa
date, cauza holerei nu ar fi fost descoperită. ✷

1.1.6 Extragerea caracteristicilor

Modelul tipic bazat pe caracteristici caută cele mai extreme exemple de fenomene
nume și reprezintă datele prin aceste exemple. Dacă sunteți familiarizați cu
Rețelele Bayes, o ramură a învățării automate și un subiect pe care nu îl abordăm în acest sens
carte, știi cum este reprezentată o relație complexă între obiecte
găsirea celor mai puternice dependențe statistice dintre aceste obiecte și utilizarea
numai cei care reprezintă toate conexiunile statistice. Unele dintre cele importante
tipurile de extragere a caracteristicilor din date pe scară largă pe care le vom studia sunt:
1. Elemente frecvente. Acest model are sens pentru datele care constau din „
kets ”de seturi mici de articole, ca în problema coșului de piață pe care o vom face
discutați în capitolul 6. Căutăm seturi mici de elemente care apar împreună
în multe coșuri, iar aceste „obiecte frecvente” sunt caracterizarea
datele pe care le căutăm. Aplicarea inițială a acestui tip de minerit a fost
coșuri de piață adevărate: seturi de articole, cum ar fi hamburger și ketchup,
că oamenii tind să cumpere împreună atunci când fac check-out la casa de marcat
a unui magazin sau super market.
2. Elemente similare. Adesea, datele dvs. arată ca o colecție de seturi și
obiectivul este de a găsi perechi de seturi care au o fracțiune relativ mare de
elementele lor în comun. Un exemplu este tratarea clienților la o
magazin de linii precum Amazon ca set de articole pe care le-au cumpărat. În ordine
pentru ca Amazon să le recomande altceva care le-ar putea plăcea, Amazon poate
căutați clienți „similari” și recomandați multe dintre acestea
clienții au cumpărat. Acest proces se numește „filtrare colaborativă”.
Dacă clienții erau singuri, adică nu cumpărau decât un fel de
lucru, atunci ar putea funcționa gruparea clienților. Cu toate acestea, din moment ce clienții
tind să aibă interese în multe lucruri diferite, este mai util să găsești,
pentru fiecare client, un număr mic de alți clienți care sunt similari
în gusturile lor și reprezintă datele prin aceste conexiuni. Discutam
similaritate în capitolul 3.

1.2 Limite statistice privind exploatarea datelor

Un tip obișnuit de problemă de extragere a datelor implică descoperirea evenimentelor neobișnuite
ascunse în cantități masive de date. Această secțiune este o discuție despre
problemă, inclusiv „Principiul lui Bonferroni”, un avertisment împotriva utilizării excesive de zel

de minerit de date.

1.2.1 Conștientizarea informațiilor totale

În urma atacului terorist din 11 septembrie 2001, s-a observat că există
patru persoane înscrise în diferite școli de zbor, învățând cum să piloteze reclame
aeronave, deși nu erau afiliate cu nicio companie aeriană. A fost presupus
că informațiile necesare pentru prezicerea și stingerea atacului erau disponibile în
date, dar că nu exista atunci nicio modalitate de a examina datele și de a detecta suspiciunile
evenimente cious. Răspunsul a fost un program numit TIA sau Total Information
Conștientizarea, care a fost menită să extragă toate datele pe care le-a putut găsi, inclusiv
chitanțe de card de credit, înregistrări hoteliere, date de călătorie și multe alte tipuri de informații
pentru a urmări activitatea teroristă. Acum integrarea informațiilor -
ideea de a relaționa și combina diferite surse de date pentru a obține informații care sunt
nu este disponibil din nicio sursă - este adesea un pas cheie pe calea spre rezolvare
o problemă importantă.
TIA a provocat în mod natural o mare îngrijorare în rândul susținătorilor confidențialității și
ject a fost în cele din urmă ucis de Congres. Scopul acestei cărți nu este acela de a
discutați problema dificilă a compromisului de confidențialitate-securitate. Cu toate acestea, perspectiva
TIA sau un sistem ca acesta ridică multe întrebări tehnice cu privire la
sibilitate. În această secțiune, dorim să ne concentrăm pe o anumită problemă tehnică:
dacă vă uitați în date pentru prea multe lucruri în același timp, veți vedea
lucruri care par interesante, dar care sunt de fapt pur și simplu artefacte statistice și au
nici o semnificație. Adică, dacă vă căutați datele pentru activități care arată
comportament terorist, nu veți găsi multe activități inocente - sau chiar
activități ilicite care nu sunt teroriste - care vor duce la vizite ale poliției
și poate mai rău decât doar o vizită? Răspunsul este că totul depinde de cum
definiți în mod restrâns activitățile pe care le căutați. Statisticienii au văzut
această problemă în multe moduri și au o teorie, pe care o introducem în următoarea
secțiunea, pentru a evita acest tip de eroare.

1.2.2 Principiul lui Bonferroni

Să presupunem că aveți o anumită cantitate de date și căutați evenimente
Tain tip în datele respective. Vă puteți aștepta să apară evenimente de acest tip, chiar dacă
datele sunt complet aleatorii și numărul de apariții al acestor evenimente
va crește pe măsură ce crește dimensiunea datelor. Aceste întâmplări sunt „false” în
simt că nu au altă cauză decât datele întotdeauna aleatorii
un număr de caracteristici neobișnuite care arată semnificativ, dar nu sunt. O teoremă
de statistici, cunoscută sub numele de corecția Bonferroni oferă un mod statistic sănătos
pentru a evita majoritatea acestor răspunsuri pozitive false la o căutare prin date.
Fără a intra în detaliile statistice, oferim o versiune informală, Bon-
principiul ferroni, care ne ajută să evităm tratarea evenimentelor aleatorii ca și cum ar fi ele
au fost reale. Calculați numărul așteptat de apariții ale evenimentelor care sunteți
căutând, presupunând că datele sunt aleatorii. Dacă acest număr este semnificativ
cu mult mai mare decât numărul de instanțe reale pe care speraţi să le găsiți, atunci trebuie
aștepta-vă că aproape orice veți găsi ca fiind fals, adică mai degrabă un artefact statistic

---

decât dovezi despre ceea ce căutați. Această observație este informală
afirmație a principiului lui Bonferroni.
Într-o situație precum căutarea teroriștilor, în care ne așteptăm să existe
puțini teroriști care acționează în același timp, principiul lui Bonferroni spune că noi
poate detecta teroriști doar căutând evenimente atât de rare încât sunt
este puțin probabil să apară în date aleatorii. Dăm mai jos un exemplu extins.

1.2.3 Un exemplu al principiului lui Bonferroni

Să presupunem că se crede că există niște „răufăcători” acolo și că vrem
pentru a le detecta. Să presupunem în plus că avem motive să credem că periodi-
cally, răufăcătorii se adună la un hotel pentru a-și complota răul. Să facem următoarele
ipoteze despre dimensiunea problemei:
1. Există un miliard de oameni care ar putea fi răufăcători.
2. Toată lumea merge la un hotel într-o zi din 100.
3. Un hotel găzduiește 100 de persoane. Prin urmare, există 100.000 de hoteluri - suficient pentru
deține 1% dintr-un miliard de oameni care vizitează un hotel într-o zi dată.

4. Vom examina înregistrările hotelului timp de 1000 de zile.
Pentru a găsi răufăcători în aceste date, vom căuta oameni care, pe două diferite zile, erau amândoi la același hotel. Să presupunem, totuși, că nu există cu adevărat răufăcători. Adică toată lumea se comportă la întâmplare, decidând cu probabilitate 0,01 să vizitați un hotel într-o anumită zi și, dacă da, alegeți unul dintre cele 10^5 hoteluri de la Aleatoriu. Am găsi vreo pereche de oameni care să pară răufăcători?
Putem face un calcul aproximativ simplu după cum urmează. Probabilitatea de oricare două persoane care decid ambele să viziteze un hotel într-o zi dată este de .0001. The șansa ca aceștia să viziteze același hotel este această probabilitate împărțită la 10^5, numărul de hoteluri. Astfel, șansa ca aceștia să viziteze același hotel pe o zi dată este 10^{-9}. Șansa ca aceștia să viziteze același hotel pe doi zile diferite diferite este pătratul acestui număr, 10^{-18}. Rețineți că hotelurile poate fi diferit în cele două zile.
Acum, trebuie să luăm în considerare câte evenimente vor indica faptele rele. Un eveniment" în acest sens este o pereche de oameni și o pereche de zile, astfel încât cei doi oameni au fost la același hotel în fiecare dintre cele două zile. Pentru a simplifica aritmetica, notați că pentru n mare, $\binom{n}{2}$ este de aproximativ $n^2/2$. Vom folosi această aproximare în ce urmează. Astfel, numărul de perechi de persoane este $\binom{10^9}{2} = 5 \times 10^{17}$. Numarul de perechi de zile este $\binom{1000}{2} = 5 \times 10^5$. Numărul așteptat de evenimente care arată la fel ca răul de a face este produsul numărului de perechi de oameni, al numărului de perechi de zile și probabilitatea ca oricare pereche de oameni și pereche de zile este o instanță a comportamentului pe care îl căutăm. Acest număr este
$5 \times 10^{17} \times 5 \times 10^5 \times 10^{-18} = 250,000$

Adică vor exista un sfert de milion de perechi de oameni care arătau răi lucrători, deși nu sunt.
Acum, să presupunem că există cu adevărat 10 perechi de răufăcători acolo. Politia va trebui să investigheze un sfert de milion de alte perechi pentru a găsi realul răufăcători. Pe lângă pătrunderea în viața a jumătate de milion de nevinovați oameni, munca implicată este suficient de mare încât această abordare a găsirii răufăcătorii probabil nu este fezabil.

1.2.4 Exerciții pentru secțiunea 1.2

Exercițiul 1.2.1: Folosind informațiile din secțiunea 1.2.3, care ar fi numărul de perechi suspectate dacă au fost aduse următoarele modificări datelor (și toate celelalte numere au rămas așa cum erau în acea secțiune)?
(a) Numărul de zile de observație a fost ridicat la 2000.
(b) Numărul de persoane observate a fost crescut la 2 miliarde (și au existat deci 200.000 de hoteluri).
(c) Am raportat o pereche ca suspectă doar dacă se aflau în același hotel la hotel aceeași oră în trei zile diferite.
! Exercițiul 1.2.2: Să presupunem că avem informații despre scopul supermarketului urmăriri de 100 de milioane de oameni. Fiecare persoană merge la supermarket de 100 de ori într-un an și cumpără 10 din cele 1000 de articole pe care le vinde supermarketul. Noi credem că o pereche de teroriști vor cumpăra exact același set de 10 articole (poate ingrediente pentru o bombă?) la un moment dat în cursul anului. Dacă căutăm perechi de oameni care au cumpărat același set de articole, ne-am aștepta ca vreunul dintre acestea oamenii găsiți erau cu adevărat teroriști? [3]

1.3 Lucruri utile de Știut

În această secțiune, oferim scurte introduceri la subiecte pe care le puteți sau le puteți nu ați văzut în studiul dvs. de alte cursuri. Fiecare va fi util în studiu de minerit de date. Ei includ:
1. Măsura TF.IDF a importanței cuvântului.
2. Funcțiile Hash și utilizarea acestora.
3. Stocarea secundară (disc) și efectul acesteia asupra timpului de rulare al algoritmilor.
4. Baza e a logaritmilor și identităților naturale care implică această constantă.
5. Legile puterii.

[3] Adică, presupunem ipoteza noastră că teroriștii vor cumpăra cu siguranță un set de 10 obiecte în comun la un moment dat pe parcursul anului. Nu vrem să abordăm problema teroriștilor sau nu neapărat ar face acest lucru.

1.3.1 Importanţa cuvintelor în documente

În mai multe aplicaţii de extragere a datelor, ne vom confrunta cu problema clasificarea documentelor (secvenţe de cuvinte) după subiectul lor. De obicei, subiecte sunt identificate prin găsirea cuvintelor speciale care caracterizează documentele despre acel subiect. De exemplu, articolele despre baseball ar tinde să aibă multe apariţii de cuvinte precum „minge”, „liliac”, „pitch”, „alergare” şi aşa mai departe. Odată ce noi au documente clasificate pentru a determina că sunt despre baseball, nu este greu să observăm că astfel de cuvinte apar neobişnuit de frecvent. Cu toate acestea, până când am făcut clasificarea, nu este posibil să identificăm aceste cuvinte ca caracteristică.

Astfel, clasificarea începe adesea prin analizarea documentelor şi găsirea cuvinte semnificative din acele documente. Prima noastră presupunere ar putea fi că cuvintele care apar cel mai frecvent într-un document sunt cele mai semnificative. In orice caz, acea intuiţie este exact opusă adevărului. Cele mai frecvente cuvinte vor cel mai sigur sunt cuvintele obişnuite precum „cel” sau „şi”, care ajută la construirea idei, dar nu au nici o însemnătate. De fapt, cele câteva sute cele mai frecvente cuvinte în engleză (numite stop words) sunt adesea eliminate documente înainte de orice încercare de clasificare a acestora.

De fapt, indicatorii subiectului sunt cuvinte relativ rare. Cu toate acestea, nu toate cuvintele rare sunt la fel de utile ca indicatori. Există anumite cuvinte, pentru exemplu „fără a aduce atingere” sau „deşi”, care apar rar într-o colecţie de documente, dar nu ne spune nimic util. Pe de altă parte, un cuvânt de genul ,,Chukker” este probabil la fel de rar, dar ne sugerează că este vorba despre document sportul poloului. Diferenţa dintre cuvintele rare care ne spun ceva şi cei care nu au legătură cu concentrarea cuvintelor utile în doar a puţine documente. Adică, prezenţa unui cuvânt ca „deşi” într-un document o face nu face să fie teribil mai probabil să apară de mai multe ori. In orice caz, dacă un articol menţionează „chukker” o dată, este probabil să ne spună ce s-a întâmplat în ,,primul chukker”, apoi „al doilea chukker” şi aşa mai departe. Adică cuvântul este probabil să se repete dacă apare deloc.

Măsura formală a cât de concentrat este în relativ puţine documente apariţiile unui cuvânt dat se numeşte TF.IDF (Term Frequency times Invers. Frecvenţa documentului). În mod normal, se calculează după cum urmează. Să presupunem că noi au o colecţie de N documente. Definiţi f_{ij} pentru a fi frecvenţa (numărul de apariţii) ale termenului (cuvântului) i din documentul j. Apoi, definiţi termenul de frecvenţă TF_{ij} să fie:

$$TF_{ij} = \frac{f_{ij}}{\max_k f_{kj}}$$

Adică, termenul frecvenţă al termenului i din documentul j este f_{ij} normalizat prin împărţire prin numărul maxim de apariţii al oricărui termen (poate excluzând stop cuvinte) în acelaşi document. Astfel, termenul cel mai frecvent din documentul j primeşte un TF de 1, iar alţi termeni obţin fracţii ca frecvenţă a termenului pentru aceasta document.

IDF pentru un termen este definit după cum urmează. Să presupunem că termenul i apare în n_i din

cele N documente din colecţie. Apoi $IDF_i = \log_2 (N / n_i)$. TF.IDF

Scorul pentru termenul i din documentul j este apoi definit ca fiind $TF_{ij} \times IDF_i$. Termenii cu cel mai mare scor TF.IDF sunt adesea termenii care caracterizează cel mai bine subiectul documentului.

Exemplul 1.4: Să presupunem că depozitul nostru este format din 2^{20} = 1.048.576 documente. Să presupunem că cuvântul w apare în 2^{10} = 1024 din aceste documente. Atunci $IDF_w = \log_2 (2 \text{ de } {}^{20} / 2^{10}) = \log 2 (2^{10}) = 10$. Să considerăm un document j în care w apare 20 ori, şi acesta este numărul maxim de ori în care apare orice cuvânt (poate după eliminarea cuvintelor stop). Apoi $TF_{wj} = 1$ şi scorul TF.IDF pentru w din documentul j este 10.

Să presupunem că în documentul k, cuvântul w apare o dată, în timp ce maximul numărul de apariţii al oricărui cuvânt din acest document este 20. Atunci $TF_{wk} = 1/20$, iar scorul TF.IDF pentru w din documentul k este 1/2. ✷

1.3.2 Funcţii Hash

Cititorul a auzit probabil de tabele hash și poate le-a folosit în Java
clase sau pachete similare. Funcțiile hash care fac fezabile tabelele hash
sunt, de asemenea, componente esențiale într-o serie de algoritmi de extragere a datelor, unde
tabelul hash ia o formă necunoscută. Vom revizui elementele de bază aici.
În primul rând, o funcție hash h ia o valoare de cheie hash ca argument și produce
un număr de găleată ca rezultat. Numărul cupei este un număr întreg, în mod normal în
intervalul 0 la B - 1, unde B este numărul de cupe. Hash-tastele pot fi de orice
tip. Există o proprietate intuitivă a funcțiilor hash pe care le „randomizează”
chei hash. Pentru a fi precis, dacă hash-cheile sunt trase aleatoriu dintr-un rezonabil
populație de posibile taste hash, atunci h va trimite numere aproximativ egale
de chei hash pentru fiecare dintre gălețile B. Ar fi imposibil să se facă acest lucru dacă, pentru
de exemplu, populația de posibile chei hash a fost mai mică decât B. O astfel de
populația nu ar fi „rezonabilă”. Cu toate acestea, pot exista reacții mai subtile
fii de ce o funcție hash nu reușește să realizeze o distribuție aproximativ uniformă
în găleți.
Exemplul 1.5: Să presupunem că tastele hash sunt numere întregi pozitive. Un lucru comun și simplu
funcția hash este de a alege h (x) = x mod B, adică restul când x este
împărțit la B. Această alegere funcționează bine dacă populația noastră de chei hash este tot
numere întregi pozitive. 1 / Bth din numărul întreg va fi atribuit fiecărei găleți.
Totuși, să presupunem că populația noastră este numărul întreg și B = 10. Numai atunci
gălețile 0, 2, 4, 6 și 8 pot fi valoarea lui h (x), iar funcția hash este
distinct non-aleatoriu în comportamentul său. Pe de altă parte, dacă am ales B = 11,
atunci am descoperi că 1/11 din numărul întreg este trimis la fiecare dintre cele 11
găleți, deci funcția hash ar funcționa bine în acest caz. ✷
Generalizarea Exemplului 1.5 este că atunci când tastele hash sunt numere întregi, alege
B, deci are orice factor comun cu toate (sau chiar cele mai multe dintre) posibile hash-
tastele vor avea ca rezultat distribuirea non-aleatorie în cupe. Astfel, este în mod normal

---

am preferat să alegem B pentru a fi un prim. Această alegere reduce șansa de a
comportament non-aleatoriu, deși trebuie totuși să luăm în considerare posibilitatea ca toate
tastele hash au B ca factor. Desigur, există multe alte tipuri de hash
funcții care nu se bazează pe aritmetica modulară. Nu vom încerca să rezumăm
opțiunile de aici, dar unele surse de informații vor fi menționate în
note bibliografice.
Ce se întâmplă dacă cheile hash nu sunt întregi? Într-un sens, toate tipurile de date au valori
care sunt compuse din biți, iar secvențele de biți pot fi întotdeauna interpretate ca
tegers. Cu toate acestea, există câteva reguli simple care ne permit să convertim comun
tipuri la numere întregi. De exemplu, dacă tastele hash sunt șiruri, convertiți fiecare caracter
la echivalentul său ASCII sau Unicode, care poate fi interpretat ca o mică inte-
ger. Sumați numerele întregi înainte de a împărți la B. Atâta timp cât B este mai mic decât
suma tipică a codurilor de caractere pentru populația de șiruri, distribuția
în găleți va fi relativ uniform. Dacă B este mai mare, atunci putem partiționa
caracterele unui șir în grupuri de mai multe caractere fiecare. Tratează concantul-
națiunea codurilor pentru caracterele unui grup ca un singur întreg. Sumați
numere întregi asociate cu toate grupurile unui șir și se împart la B ca înainte.
De exemplu, dacă B este în jur de un miliard, sau 2^{30}, atunci gruparea caracterelor patru la
un timp ne va da întregi pe 32 de biți. Suma mai multor dintre acestea va fi distribuită
destul de uniform într-un miliard de găleți.
Pentru tipuri de date mai complexe, putem extinde ideea utilizată pentru conversie
șiruri la numere întregi, recursiv.

- Pentru un tip care este o înregistrare, fiecare dintre ale cărui componente are propriul tip,
converti recursiv valoarea fiecărei componente într-un număr întreg, folosind
algoritm adecvat pentru tipul componentei respective. Sumați numerele întregi
pentru componente și convertiți suma întreagă în cupe împărțind
de B.
- Pentru un tip care este o matrice, un set sau o pungă de elemente de un singur tip,
convertiți valorile tipului elementelor în numere întregi, însumați numărul întreg și
împarte la B.

1.3.3 Indici

Un index este o structură de date care face eficientă recuperarea obiectelor date de
valoarea unuia sau mai multor elemente ale acelor obiecte. Cea mai frecventă situație
este una în care obiectele sunt înregistrări, iar indexul se află pe unul dintre câmpurile din
acel record. Având o valoare v pentru acel câmp, indexul ne permite să preluăm toate
înregistrări cu valoarea v în acel câmp, fără a fi nevoie să preluați toate înregistrările din

fișierul. De exemplu, am putea avea un fișier de (nume, adresă, telefon) triple,
și un index pe câmpul telefonic. Având în vedere un număr de telefon, indexul ne permite
pentru a găsi rapid înregistrarea sau înregistrările cu acel număr de telefon.
Există multe modalități de a implementa indexuri și nu vom încerca
cercetează problema aici. Notele bibliografice oferă sugestii în continuare
citind. Cu toate acestea, un tabel hash este un mod simplu de a construi un index. Campul

---

Pagina 32

sau câmpurile pe care se bazează indexul formează cheia hash pentru o funcție hash.
Aplicăm funcția hash aplicată valorii cheii hash pentru fiecare înregistrare,
iar înregistrarea în sine este plasată în cupa al cărei număr este determinat de
funcție hash. Bucket-ul ar putea fi o listă de înregistrări din memoria principală sau un disc
bloc, de exemplu.
Apoi, având o valoare hash-key, o putem hash, găsi găleata și trebuie
căutați doar acea bucket pentru a găsi înregistrările cu acea valoare pentru cheia hash. Dacă
alegem numărul de găleți B pentru a fi comparabil cu numărul de înregistrări
în fișier, atunci vor fi relativ puține înregistrări în orice bucket și o vom face
găsiți puține, dacă există, înregistrări în găleată cu o cheie hash care nu este cea care suntem noi
căuta. Astfel, căutarea înregistrărilor dorite este destul de eficientă, comparată
cu căutarea întregului fișier pentru înregistrări cu cheia hash dorită.

-1
B
17
0
h (800−555−1212)
Înregistrează cu h (telefon) = 17
Matrice de
găleată
anteturi
.
.
.
.
.
.
Sally Jones Maple St 800−555−1212

Figura 1.2: Un tabel hash folosit ca index; numerele de telefon sunt tăiate în cupe,
iar întreaga înregistrare este plasată în cupa al cărei număr este valoarea hash a
telefonul

Exemplul 1.6: Figura 1.2 sugerează cu ce are un index de memorie principală a înregistrărilor
numele, adresa și câmpurile de telefon ar putea arăta astfel. Aici, indexul este la telefon
câmpul și gălețile sunt liste legate. Arătăm telefonul 800-555-1212 hashed la
cupa cu numărul 17. Există o serie de anteturi pentru cupă, al căror al i-lea element este
capul unei liste legate pentru cupa numerotată i. Arătăm, în extins
, unul dintre elementele listei legate. Conține o înregistrare cu nume,
adresa și câmpurile de telefon. Această înregistrare este de fapt una cu numărul de telefon
800-555-1212. Alte înregistrări din acel compartiment pot avea sau nu acest telefon
număr. Știm doar că orice număr de telefon pe care îl au este un telefon care
hashes la 17. ✷

Pagina 33

1.3.4 Depozitare secundară

Este important, atunci când avem de-a face cu date la scară largă, să avem un bun
înțelegerea diferenței de timp luate pentru efectuarea calculelor atunci când
datele sunt inițial pe disc, spre deosebire de timpul necesar dacă datele sunt inițial în
memoria principala. Caracteristicile fizice ale discurilor sunt un alt subiect pe care
am putea spune multe, dar vom spune doar puțin și vom lăsa cititorul interesat
urmați notele bibliografice.
Discurile sunt organizate în blocuri, care sunt unitățile minime pe care le operează
sistemul de operare folosește pentru a muta datele între memoria principală și disc. De exemplu,
sistemul de operare Windows utilizează blocuri de 64K octeți (adică 2^{16} = 65.536 octeți
mai exact). Este nevoie de aproximativ zece milisecunde pentru a accesa (mutați discul
mergeți la pista blocului și așteptați ca blocul să se rotească sub cap)
și citiți un bloc de disc. Această întârziere este de cel puțin cinci ordine de mărime (un factor

de 10^5) mai lent decât timpul necesar citirii unui cuvânt din memoria principală, deci dacă totul vrem să facem este să accesăm câțiva octeți, avem un avantaj copleșitor date din memoria principală. De fapt, dacă vrem să facem ceva simplu fiecărui octet a unui bloc de disc, de exemplu, tratați blocul ca o găleată a unui tabel hash și căutați o valoare particulară a cheii hash dintre toate înregistrările din acel compartiment, atunci timpul necesar pentru mutarea blocului de pe disc în memoria principală va fi mult mai mare decât timpul necesar pentru a face calculul.

Prin organizarea datelor noastre astfel încât datele aferente să fie pe un singur cilindru (colecție de blocuri accesibile la o rază fixă de la centrul discului și prin urmare accesibil fără a muta capul discului), putem citi toate blocurile pe cilindru în memoria principală în considerabil mai puțin de 10 milisecunde pe bloc. Puteți presupune că un disc nu poate transfera date în memoria principală la mai mult de o sută de milioane de octeți pe secundă, indiferent de cât sunt aceste date organizat. Aceasta nu este o problemă atunci când setul de date este un megabyte. Dar a setul de date de o sută de gigaocteți sau un terabyte prezintă probleme doar la accesare să nu mai vorbim de nimic util cu el.

1.3.5 Baza logaritmilor naturali

Constanta e = 2.7182818 ··· are o serie de proprietăți speciale utile.În în special, e este limita de $(1 + \frac{1}{x})^x$ ca x merge la infinit. Valorile acestui lucru expresia pentru x = 1, 2, 3, 4 sunt aproximativ 2, 2,25, 2,37, 2,44, deci ar trebui consideră că este ușor să crezi că limita acestei serii este în jur de 2,72.

Unele algebre ne permit să obținem aproximări la multe aparent complexe expresii. Luați în considerare $(1 + a)^b$, unde a este mic. Putem rescrie expresia ca $(1 + a)^{(1/a)(ab)}$. Apoi înlocuiți a = 1 / x și 1 / a = x, deci avem $(1+ \frac{1}{x})^{x(ab)}$,

care este

$((1 + \frac{1}{X})^x)^{ab}$

Deoarece a este presupus mic, x este mare, deci subexpresia $(1 + \frac{1}{x})^x$ va fi aproape la valoarea limitativă a e. Putem astfel să aproximăm $(1 + a)^b$, pentru a mic, ca e^{ab} .

Identități similare sunt valabile atunci când a este negativ. Adică, limita la care se îndreaptă x infinitate de $(1 - \frac{1}{x})^x$ este 1 / e. Rezultă că aproximarea $(1 + a)^b = e^{ab}$ se menține chiar și atunci când a este un număr negativ mic. Altfel spus, $(1 - a)^b$ este aproximativ e^{-ab} când a este mic.

Unele alte aproximări utile decurg din expansiunea Taylor a lui e^x .

Adică $e^x = \sum_{i=0}^{\infty} x^i / i!$, sau $e^x = 1 + x^2/2 + x^3/6 + x^4/$ deschisă 24 + °. Cand x este mare, seria de mai sus converge încet, deși converge deoarece n! crește mai repede decât x^n pentru orice constantă x. Cu toate acestea, atunci când x este mic, fie pozitiv sau negativ, seria converge rapid și doar câțiva termeni sunt necesar pentru a obține o bună aproximare.

Exemplul 1.7: Fie x = 1/2. Apoi

$e^{1/2} = 1 + \frac{1}{2} + \frac{1}{8} + \frac{1}{48} + \frac{1}{}$

384+ · · ·
sau aproximativ $e^{1/2} = 1,64844$.
Fie $x = -1$. Apoi
$e^{-1} = 1 - 1 +$
1
2 -
1
6
+
1
24 -
1
120
+
1
720 -
1
5040+ · · ·
sau aproximativ $e^{-1} = 0,367786$. ✷

1.3.6 Legile puterii

Există multe fenomene care raportează două variabile printr-o lege a puterii, adică a relație liniară între logaritmii variabilelor. Figura 1.3 sugerează o astfel de relație. Dacă x este axa orizontală și y este axa verticală, atunci relația este $\log_{10} y = 6 - 2 \log_{10} x$.

Exemplul 1.8: S-ar putea să examinăm vânzările de cărți de pe Amazon.com și să lăsăm x să resentați rangul cărților după vânzări. Atunci y este numărul de vânzări al x-lea cea mai bine vândută carte într-o anumită perioadă. Implicația graficului din Fig. 1.3 ar fi faptul că cea mai bine vândută carte a vândut 1.000.000 de exemplare, cea de-a 10-a cea mai bine vândută cartea a vândut 10.000 de exemplare, cea de-a 100-a cea mai bine vândută carte a vândut 100 de exemplare și așa mai departe pentru toate rangurile dintre aceste numere și dincolo. Implicația de mai sus rangul 1000, vânzările sunt o fracțiune dintr-o carte, este prea extremă și, de fapt, am face-o așteptați ca linia să se aplatizeze pentru ranguri mult mai mari decât 1000. Mai mult, panta liniei din Fig. 1.3 este probabil mult prea abruptă pentru a descrie vânzările de cărți, deși o linie care scade mai puțin precipitat ar fi aproape de ceea ce se întâmplă în practică. ✷

Forma generală a unei legi a puterii referitoare la x și y este $\log y = b + a \log x$. Dacă ridicăm baza logaritmului (baza nu contează de fapt în ecuație), să spunem e, la valorile de pe ambele părți ale acestei ecuații, obținem $y = e^b e^{a \log x} = e^b x^a$. Deoarece e^b este doar „o constantă”, să o înlocuim cu constantă c. Astfel, o lege a puterii poate fi scrisă ca $y = cx^a$ pentru unele constante a și c.

15
1
10 100 1000 10.000
1
10
100
1000
10.000
100.000
1.000.000
10.000.000
Figura 1.3: O lege a puterii cu o pantă de -2

Exemplul 1.9: În Fig. 1.3 vedem că când $x = 1$, $y = 10^6$ și când $x = 1000$, $y = 1$. Efectuând prima înlocuire, vedem $10^6 = c$. Al doilea substituția ne dă $1 = c(1000)^a$. Deoarece acum știm $c = 10^6$, al doilea ecuația ne dă $1 = 10^6 (1000)^a$, din care vedem $a = -2$. Adică legea exprimat prin Fig. 1.3 este $y = 10^6 x^{-2}$ sau $y = 10^6 / x^2$. ✷

Vom întâlni în această carte multe moduri în care legile puterii guvernează fenomenele. Aici sunt cateva exemple:

1. Gradele nodurilor din graficul web: ordonați toate paginile după numărul de linkuri către acea pagină. Fie x poziția unei pagini în această ordonare și fii y numărul de linkuri către pagina a x-a. Apoi y în funcție de x seamănă foarte mult cu Fig. 1.3. Exponentul a este puțin mai mare decât −2 afișat acolo; a fost găsit mai aproape de 2.1.
2. Vânzările de produse: Comandați produse, să spunem cărți de pe Amazon.com, de la acestea

vânzări în ultimul an. Fie y numărul de vânzări din a x-a cea mai populară
ular carte. Din nou, funcția y (x) va arăta ceva asemănător cu figura 1.3. noi
va discuta consecințele acestei distribuții a vânzărilor în secțiunea 9.1.2,
unde preluăm problema „cozii lungi”.
3. Dimensiunile site-urilor web: numărați numărul de pagini de pe site-urile web și comandați
site-urilor după numărul paginilor lor. Fie y numărul de pagini de la
al X-lea site. Din nou, funcția y (x) urmează o lege a puterii.
4. Legea lui Zipf: Această lege a puterii se referea inițial la frecvența cuvintelor
într-o colecție de documente. Dacă comandați cuvinte după frecvență și lăsați-o pe y

---

Efectul Matei

Adesea, existența legilor puterii cu valori ale exponentului mai mari decât
1 sunt explicate prin efectul Matei. În Cartea biblică a lui Matei,
există un verset despre „bogații devin mai bogați”. Multe fenomene prezintă acest lucru
comportament, în cazul în care obținerea unei valori ridicate a unor proprietăți cauzează acest lucru
proprietate de mărit. De exemplu, dacă o pagină Web are mai multe linkuri, atunci
este mai probabil ca oamenii să găsească pagina și pot alege să se conecteze la aceasta
una dintre paginile lor, de asemenea. Ca un alt exemplu, dacă o carte se vinde bine
pe Amazon, atunci este probabil să fie promovat atunci când clienții merg la
Site-ul Amazon. Unii dintre acești oameni vor alege să cumpere și cartea,
crescând astfel vânzările acestei cărți.
fie de câte ori apare al x-lea cuvânt din ordine, atunci veți obține
o lege a puterii, deși cu o pantă mult mai mică decât cea din Fig. 1.3.
Observația lui Zipf a fost că $y = cx^{-1/2}$. Interesant este că o serie de altele
tipuri de date respectă această lege specială a puterii. De exemplu, dacă comandăm
state din SUA după populație și să fie populația a x-a
cel mai populat stat, atunci x și y respectă legea Zipf aproximativ.

1.3.7 Exerciții pentru secțiunea 1.3

Exercițiul 1.3.1: Să presupunem că există un depozit de zece milioane de documente. Ce
(până la cel mai apropiat număr întreg) este IDF pentru un cuvânt care apare în (a) 40 de documente
(b) 10.000 de documente?
Exercițiul 1.3.2: Să presupunem că există un depozit de zece milioane de documente și
cuvântul w apare în 320 dintre ele. Într-un anumit document d, maximul
numărul de apariții al unui cuvânt este 15. Aproximativ ce este TF.IDF
scor pentru w dacă acel cuvânt apare (a) o dată (b) de cinci ori?
! Exercițiul 1.3.3: Să presupunem că tastele hash sunt extrase din populația tuturor
numere întregi negative care sunt multipli ai unei constante c și funcția hash h (x)
este x mod 15. Pentru ce valori ale lui c va fi o funcție hash adecvată, adică a
o gamă largă aleatorie de taste hash va fi împărțită aproximativ în mod egal în găleți?
Exercițiul 1.3.4: În termeni de e, dați aproximări la
(a) $(1{,}01)^{500}$ (b) $(1{,}05)^{1000}$ (c) $(0{,}9)^{40}$
Exercițiul 1.3.5: Utilizați expansiunea Taylor a lui e^x pentru a calcula, la trei zecimale
locuri: (a) $e^{1/10}$ (b) $e^{-1/10}$ (c) e^2 .

1.4 Schița cărții

Această secțiune oferă rezumate scurte ale capitolelor rămase ale cărții.
Capitolul 2 nu se referă la extragerea datelor în sine. Mai degrabă ne prezintă
sisteme de programare care facilitează prelucrarea în paralel a unor cantități masive de
date. Discutăm arhitectura cloud-computing, care utilizează un număr mare
de procesoare conectate. Discutăm în detaliu sistemele de programare bazate pe
MapReduce și oferă algoritmi bazați pe MapReduce pentru o serie de obișnuite
operații folosind în procesarea seturilor de date masive.
Capitolul 3 este despre găsirea unor articole similare. Punctul nostru de plecare este că articolele
pot fi reprezentate prin seturi de elemente, iar seturi similare sunt cele care au un
mare parte din elementele lor în comun. Tehnicile cheie ale minhashing-ului
și se explică hashingul sensibil la localitate. Aceste tehnici au numeroase
aplicații și oferă adesea soluții surprinzător de eficiente la problemele care se
peră imposibilă pentru seturi masive de date.
În capitolul 4, luăm în considerare datele sub forma unui flux. Diferența

între un flux și o bază de date înseamnă că datele dintr-un flux se pierd dacă o faceți
nu faceți ceva despre asta imediat. Exemple importante de fluxuri sunt
fluxuri de interogări de căutare la un motor de căutare sau clicuri pe un site Web popular. În
în acest capitol, vedem câteva dintre aplicațiile surprinzătoare ale hashing-ului pe care le fac
gestionarea datelor fluxului este fezabilă.
Capitolul 5 este dedicat unei singure aplicații: calculul PageRank.
Acest calcul este ideea care a făcut ca Google să se distingă de alte căutări
și este încă o parte esențială a modului în care motoarele de căutare știu ce pagini
este probabil ca utilizatorul să vrea să vadă. Extensiile PageRank sunt, de asemenea, esențiale în
lupta împotriva spamului (numită eufemistic „optimizarea motorului de căutare”) și
vom examina cele mai noi extensii ale ideii în scopul combaterii
spam.
Apoi, Capitolul 6 introduce modelul de coș de piață al datelor și
probleme de reguli de asociere și de găsire a unor obiecte frecvente. În piață-
model de coș, datele constau dintr-o mare colecție de coșuri, fiecare dintre ele
are un set mic de articole. Oferim o secvență de algoritmi capabili de găsire
toate perechile frecvente de obiecte, adică perechi de obiecte care apar împreună în multe
coșuri. O altă secvență de algoritmi este utilă pentru găsirea majorității
obiecte frecvente mai mari decât perechile, cu eficiență ridicată.
Capitolul 7 examinează problema grupării. Presupunem un set de articole
cu o măsură a distanței care definește cât de aproape sau de departe este un articol de altul.
Scopul este de a examina o cantitate mare de date și de a le împărți în subseturi
(clustere), fiecare cluster format din elemente care sunt toate aproape una de alta, încă
departe de elementele din celelalte clustere.
Capitolul 8 este dedicat publicității on-line și problemelor de calcul
aceasta generează. Introducem noțiunea de algoritm on-line - unul în care a
un răspuns bun trebuie dat imediat, mai degrabă decât să așteptăm până când avem
a văzut întregul set de date. Ideea raportului competitiv este un alt lucru important
concept acoperit în acest capitol; este raportul dintre performanța garantată a

---

un algoritm online comparat cu performanța algoritmului optim
care este permis să vadă toate datele înainte de a lua orice decizie. Aceste idei sunt
folosit pentru a oferi algoritmi buni care corespund ofertelor de către agenții de publicitate pentru dreptul la
afișați anunțul ca răspuns la o interogare împotriva interogărilor de căutare care ajung la un
motor de căutare.
Capitolul 9 este dedicat sistemelor de recomandare. Multe aplicații web
implică sfătuirea utilizatorilor cu privire la ceea ce le-ar putea plăcea. Provocarea Netflix este una
exemplu, în cazul în care se dorește prezicerea ce filme ar dori un utilizator sau Ama-
problema zonului de a prezenta un produs unui client pe baza informațiilor despre
ce ar putea fi interesați să cumpere. Există două abordări de bază pentru
recomandare. Putem caracteriza articolele după caracteristici, de exemplu, stelele unui
film și recomandă articole cu aceleași caracteristici ca cele pe care le cunoaște utilizatorul
a place. Sau ne putem uita la alți utilizatori cu preferințe similare cu cele ale
utilizator în cauză și vedeți ce le-a plăcut (o tehnică cunoscută sub numele de colaborare
filtrare).
În capitolul 10, studiem rețelele sociale și algoritmii pentru analiza lor.
Exemplul canonic al unei rețele sociale este graficul prietenilor de pe Facebook,
unde nodurile sunt oameni și marginile conectează doi oameni dacă sunt prieteni.
Graficele direcționate, cum ar fi urmăritorii de pe Twitter, pot fi, de asemenea, privite ca sociale
rețele. Un exemplu obișnuit de problemă care trebuie abordată este identificarea
,,Comunități”, adică mici seturi de noduri cu un număr neobișnuit de mare de
marginile dintre ele. Alte întrebări despre rețelele sociale sunt întrebări generale
despre grafice, cum ar fi calcularea închiderii tranzitive sau a diametrului unui grafic,
dar sunt îngreunate de dimensiunea rețelelor tipice.
Capitolul 11 analizează reducerea dimensionalității. Ni se oferă un foarte mare
matrice, de obicei rar. Gândiți-vă la matrice ca reprezentând o relație
între două tipuri de entități, de exemplu, evaluările filmelor de către spectatori. Intuitiv,
există un număr mic de concepte, mult mai puține concepte decât există
filme sau spectatori, care explică de ce anumitor spectatori le plac anumite filme. Noi
oferă mai mulți algoritmi care simplifică matricile prin descompunerea lor într-un
produs de matrici care sunt mult mai mici într-una din cele două dimensiuni. unu
matricea corelează entități de un fel cu numărul mic de concepte și altul
relaționează conceptele cu celălalt tip de entitate. Dacă este făcut corect, produsul
dintre matricile mai mici vor fi foarte apropiate de matricea originală.
Capitolul 12 discută algoritmii pentru învățarea automată din seturi de date foarte mari.

Tehnicile acoperite includ perceptroni, mașini vector-suport, găsirea modurilor
altele prin coborârea în gradient, modelele cele mai apropiate și arborii de decizie.
În cele din urmă, Capitolul 13 introduce rețelele neuronale și învățarea profundă, în special
lar. Pe lângă ideea generală a unei rețele neuronale, acest capitol acoperă
cazurile importante importante ale rețelelor neuronale convoluționale, neuronale recurente
rețele și rețele de memorie pe termen scurt.

1.5 Rezumatul capitolului 1

♦ Data Mining: Acest termen se referă la aplicarea instrumentelor puternice ale computerului
știință pentru a rezolva probleme din știință, industrie și multe alte aplicații
zone. În mod frecvent, cheia unei aplicații de succes este construirea unui model
a datelor, adică un rezumat sau o reprezentare relativ succintă a
cele mai relevante caracteristici ale datelor.

♦ Principiul lui Bonferroni: Dacă suntem dispuși să vedem ca o temă interesantă
de date despre care se poate aștepta să existe multe cazuri
în datele aleatorii, atunci nu ne putem baza pe faptul că astfel de caracteristici sunt semnificative.
Această observație ne limitează capacitatea de a extrage date pentru caracteristici care nu sunt
suficient de rare în practică.

♦ TF.IDF: Măsura numită TF.IDF ne permite să identificăm cuvintele dintr-o colecție
a documentelor care sunt utile pentru determinarea temei fiecărui document.
Un cuvânt are un scor TF.IDF ridicat într-un document dacă apare în relativ puține
documente, dar apare în acesta și când apare într-un document
tinde să apară de multe ori.

♦ Funcții Hash: O funcție hash mapează tastele hash de un anumit tip de date
numere de cupe întregi. O funcție hash bună distribuie posibilul
valorile cheie hash aproximativ uniform între găleți. Orice tip de date poate
fie domeniul unei funcții hash.

♦ Indexuri: un index este o structură de date care ne permite să stocăm și să recuperăm
înregistrări de date în mod eficient, având în vedere valoarea într-unul sau mai multe dintre câmpurile din
record. Hashing este o modalitate de a construi un index.

♦ Stocare pe disc: Când datele trebuie stocate pe disc (memorie secundară),
este nevoie de mult mai mult timp pentru a accesa un element de date dorit decât dacă este același
datele au fost stocate în memoria principală. Când datele sunt mari, este important
că algoritmii se străduiesc să păstreze datele necesare în memoria principală.

♦ Legile puterii: Multe fenomene respectă o lege care poate fi exprimată ca
$y = cx^{a}$ pentru o anumită putere a, adesea în jurul valorii de -2. Astfel de fenomene includ
vânzările celei de-a X-a cărți cele mai populare sau numărul de link-uri către a X-a
cea mai populară pagină.

1.6 Referințe pentru capitolul 1

[8] este o introducere clară la elementele de bază ale exploatării datelor. [3] acoperă exploatarea datelor
în principal din punctul de vedere al învățării automate și al statisticii. The
diferența dintre abordarea statistică și abordarea de calcul a
extragerea datelor este exprimată în [1].
Pentru construcția funcțiilor hash și a tabelelor hash, consultați [5]. Detalii despre
Măsura TF.IDF și alte aspecte legate de procesarea documentelor pot fi

---

găsit în [6]. Consultați [4] pentru mai multe despre gestionarea indexurilor, tabelelor hash și a datelor
pe disc.
Legile puterii referitoare la Web au fost explorate de [2]. Efectul Matei
a fost observată prima dată în [7].

1. L. Breiman, „Modelarea statistică: cele două culturi”, Știința statistică
16: 3, pp. 199-215, 2001.
2. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R.
Stata, A. Tomkins și J. Weiner, „Structura graficului pe web”, Com-
puter Networks 33: 1-6, pp. 309-320, 2000.
3. MM Gaber, Exploatarea datelor științifice și descoperirea cunoștințelor - Prin-
ciples and Foundations, Springer, New York, 2010.

4. H. Garcia-Molina, JD Ullman și J. Widom, Sisteme de baze de date: The
Ediția a doua carte completă, Prentice-Hall, Upper Saddle River, NJ,
2009.
5. DE Knuth, The Art of Computer Programming Vol. 3 (Sortare și
Căutare), Ediția a doua, Addison-Wesley, Upper Saddle River, NJ,
1998.
6. CP Manning, P. Raghavan și H. Schütze, Introducere în informații
Recuperare, Cambridge Univ. Presă, 2008.
7. RK Merton, „Efectul Matei în știință”, Știința 159: 3810, pp. 56–
63, 5 ianuarie 1968.
8. P.-N. Tan, M. Steinbach și V. Kumar, Introducere în exploatarea datelor,
Addison-Wesley, Upper Saddle River, NJ, 2005.

capitolul 2
MapReduce și Noul Stivă de software

Aplicațiile moderne de extragere a datelor, deseori numite analize „big-data”, ne necesită
pentru a gestiona rapid cantități imense de date. În multe dintre aceste aplicații,
datele sunt extrem de regulate și există o mare oportunitate de a exploata paralelismul.
Exemple importante sunt:
1. Clasarea paginilor Web după importanță, care implică o iterație
multiplicare matrice-vector unde dimensiunea este de multe miliarde. Acest
aplicația, numită „PageRank”, face obiectul capitolului 5.
2. Căutări în rețelele „prieteni” de pe site-urile de socializare, care implică
grafice cu sute de milioane de noduri și multe miliarde de muchii.
Operațiunile pe grafice de acest tip sunt prezentate în capitolul 10.
Pentru a face față unor aplicații precum acestea, a evoluat o nouă stivă de software.
Aceste sisteme de programare sunt concepute pentru a obține paralelismul lor nu dintr-un
,,Supercomputer”, dar din „clustere de calcul” - colecții mari de produse
hardware, inclusiv procesoare convenționale („noduri de calcul”) conectate
prin cabluri Ethernet sau comutatoare ieftine. Stiva software începe cu
o nouă formă de sistem de fișiere, numită „sistem de fișiere distribuit”, care are caracteristici
unități mult mai mari decât blocurile de disc dintr-un sistem de operare convențional. Dis-
sistemele de fișiere tributate oferă, de asemenea, replicarea datelor sau redundanța de protejat
împotriva frecventelor defecțiuni media care apar la distribuirea datelor
mii de noduri de calcul ieftine.
Pe lângă aceste sisteme de fișiere, multe sisteme diferite de programare la nivel superior
au fost dezvoltate teme. Punctul central al noii stive software este programarea
sistemul numit MapReduce. Implementările MapReduce permit multe dintre
cele mai frecvente calcule pe date la scară largă care trebuie efectuate pe calcul
clustere eficient și într-un mod care este tolerant la defecțiunile hardware din timpul
calcul.

Sistemele MapReduce evoluează și se extind rapid. Astăzi, este
lun pentru programele MapReduce care urmează să fie create din programare la nivel superior
sisteme, adesea o implementare a SQL. Mai mult, MapReduce se dovedește a fi
un caz util, dar simplu, de idei mai generale și mai puternice. Includem
în acest capitol o discuție despre generalizările MapReduce, mai întâi la sisteme
care suportă fluxuri de lucru aciclice și apoi la sisteme care implementează recursive
algoritmi.
Ultimul nostru subiect pentru acest capitol este proiectarea unor algoritmi buni MapReduce,
un subiect care adesea diferă semnificativ de problema proiectării bunului
algoritmi paraleli care se execută pe un supercomputer. La proiectarea MapReduce
algoritmi, de multe ori descoperim că cel mai mare cost este comunicarea. Noi
investigați astfel costul comunicării și ceea ce ne spune despre cele mai eficiente
Algoritmi MapReduce. Pentru mai multe aplicații comune ale MapReduce suntem

capabil să ofere familiilor de algoritmi care tranzacționează în mod optim costul comunicării
împotriva gradului de paralelism.

2.1 Sisteme de fișiere distribuite

Majoritatea calculelor se fac pe un singur procesor, cu memoria sa principală, cache și
disc local (un nod de calcul). În trecut, aplicațiile care solicitau paralelism
prelucrarea, cum ar fi calculele științifice de mari dimensiuni, a fost făcută în scopuri speciale
calculatoare paralele cu multe procesoare și hardware specializat. In orice caz,
prevalența serviciilor web pe scară largă a cauzat din ce în ce mai multă informatică
de făcut pe instalații cu mii de noduri de calcul care funcționează mai mult
sau mai puțin independent. În aceste instalații, nodurile de calcul sunt mărfuri
hardware, ceea ce reduce foarte mult costul în comparație cu paralelele cu destinație specială
mașini.
Aceste noi facilități de calcul au dat naștere unei noi generații de programe
sisteme de gramming. Aceste sisteme profită de puterea paralelismului
și, în același timp, evita problemele de fiabilitate care apar atunci când
hardware-ul constă din mii de componente independente, dintre care oricare
ar putea eșua în orice moment. În această secțiune, discutăm atât caracteristicile
aceste instalații de calcul și sistemele de fișiere specializate care au fost
dezvoltat pentru a profita de ele.

2.1.1 Organizarea fizică a nodurilor de calcul

Noua arhitectură de calcul paralel, uneori numită cluster computing,
este organizat după cum urmează. Nodurile de calcul sunt stocate pe rafturi, poate 8–64
pe un raft. De obicei, nodurile de pe un singur rack sunt conectate printr-o rețea
gigabit Ethernet. Pot exista multe rafturi de noduri de calcul, iar rafturile sunt
conectat de un alt nivel de rețea sau de un comutator. Lățimea de bandă a inter-rack-ului
comunicarea este oarecum mai mare decât Ethernet intrarack, dar având în vedere
numărul de perechi de noduri care ar putea avea nevoie să comunice între rafturi, acesta

---

Pagina 43

lățimea de bandă poate fi esențială. Figura 2.1 sugerează arhitectura unui
sistem de calcul la scară. Cu toate acestea, pot exista mult mai multe rafturi și multe altele
mai multe noduri de calcul pe rack.
Intrerupator
Rafturi de noduri de calcul
Figura 2.1: Nodurile de calcul sunt organizate în rafturi, iar rafturile sunt interconectate
conectat printr-un comutator
Este un fapt al vieții că componentele eșuează și cu cât mai multe componente, cum ar fi
calculează nodurile și legăturile de comunicare, un sistem are, cu atât mai frecvent
ceva din sistem nu va funcționa la un moment dat. Pentru sisteme
cum ar fi Fig. 2.1, principalele moduri de eșec sunt pierderea unui singur nod (de exemplu,
discul de la acel nod se blochează) și pierderea unui întreg rack (de exemplu, rețeaua
conectarea nodurilor sale între ele și la lumea exterioară eșuează).
Unele calcule importante durează minute sau chiar ore pe mii de
calculează noduri. Dacă ar trebui să avortăm și să repornim calculul de fiecare dată
o componentă a eșuat, atunci calculul s-ar putea să nu se finalizeze niciodată cu succes.
Soluția la această problemă ia două forme:
1. Fișierele trebuie stocate redundant. Dacă nu am duplicat fișierul la mai multe
calculați nodurile, apoi dacă un nod eșua, toate fișierele sale ar fi indisponibile
până când nodul este înlocuit. Dacă nu am făcut backup deloc a fișierelor și a fișierului
discul se blochează, fișierele ar fi pierdute pentru totdeauna. Discutăm despre gestionarea fișierelor
în secțiunea 2.1.2.
2. Calculele trebuie împărțite în sarcini, astfel încât, dacă o sarcină eșuează
pentru a fi executat până la finalizare, acesta poate fi repornit fără a afecta alte sarcini.
Această strategie este urmată de sistemul de programare MapReduce pe care noi
introduceți în secțiunea 2.2.

Pagina 44

Implementări DFS

Există mai multe sisteme de fișiere distribuite de tipul pe care l-am descris
care sunt folosite în practică. Printre acestea:
1. Sistemul de fișiere Google (GFS), originalul clasei.

2. Sistemul de fișiere distribuite Hadoop (HDFS), un DFS open-source utilizat
cu Hadoop, o implementare a MapReduce (vezi secțiunea 2.2)
și distribuite de Apache Software Foundation.
3. Colossus este o versiune îmbunătățită a GFS, despre care a fost puțin
publicat. Cu toate acestea, un obiectiv al Colossus este de a furniza fișiere în timp real
serviciu.

2.1.2 Organizarea sistemului de fișiere pe scară largă

Pentru a exploata calculul cluster, fișierele trebuie să arate și să se comporte oarecum diferit
din sistemele de fișiere convenționale găsite pe computere individuale. Acest fișier nou
sistem, denumit adesea sistem de fișiere distribuit sau DFS (deși acest termen are
avea alte semnificații în trecut), este de obicei folosit după cum urmează.

- Fișierele pot fi enorme, eventual cu o dimensiune terabyte. Dacă ai doar mici
fișiere, nu are rost să folosiți un DFS pentru ele.
- Fișierele sunt rar actualizate. Mai degrabă, acestea sunt citite ca date pentru unele calcule
și, eventual, date suplimentare sunt adăugate fișierelor din când în când.

De exemplu, un sistem de rezervare a unei companii aeriene nu ar fi potrivit pentru un
DFS, chiar dacă datele au fost foarte mari, deoarece datele sunt modificate astfel
frecvent.
Fișierele sunt împărțite în bucăți, care au de obicei 64 de megaocteți.
Bucățile sunt reproduse, poate de trei ori, la trei noduri de calcul diferite.
Mai mult, nodurile care dețin copii ale unei bucăți ar trebui să fie amplasate pe diferite
rafturi, deci nu pierdem toate copiile din cauza unei defecțiuni a raftului. De obicei, un rack „eșuează”
deoarece interconectarea dintre nodurile de calcul de pe rack eșuează și
rack nu mai poate comunica cu nimic din afara sa. În mod normal, ambele
mărimea bucății și gradul de replicare poate fi decis de către utilizator.
Pentru a găsi bucățile unui fișier, există un alt fișier mic numit nod master
sau nodul de nume pentru acel fișier. Nodul master este el însuși replicat și un director
deoarece sistemul de fișiere în ansamblu știe unde să găsească copiile sale. Directorul în sine
poate fi replicat și toți participanții care folosesc DFS știu unde se află directorul
copii sunt.

2.2 MapReduce

MapReduce este un stil de calcul care a fost implementat în mai multe sisteme
, inclusiv implementarea internă Google (numită pur și simplu MapReduce)
și populara implementare open source Hadoop care poate fi obținută,
împreună cu sistemul de fișiere HDFS de la Apache Foundation. Poți să folosești
o implementare a MapReduce pentru a gestiona multe com-
expuneri într-un mod care tolerează defecțiunile hardware. Tot ce trebuie să scrieți
sunt două funcții, numite Map și Reduce. Sistemul administrează paralela
executarea și coordonarea sarcinilor care execută Map sau Reduce. Sistemul
tratează, de asemenea, posibilitatea ca una dintre aceste sarcini să nu poată fi executată. În
pe scurt, un calcul MapReduce se execută după cum urmează:
1. Unora dintre anumite sarcini Hărți li se oferă câte unul sau mai multe bucăți dintr-un
sistem de fișiere distribuit. Aceste sarcini de hartă transformă bucata într-o secvență
a perechilor cheie-valoare. Modul în care sunt produse perechile cheie-valoare din intrare
datele sunt determinate de codul scris de utilizator pentru funcția Hartă.
2. Perechile cheie-valoare din fiecare sarcină Hărți sunt colectate de către o maestră
troller și sortate după cheie. Cheile sunt împărțite între toate Reduce
sarcini, deci toate perechile cheie-valoare cu aceeași cheie se încheie la același Re-
duce sarcina.
3. Sarcinile de reducere funcționează pe o tastă la un moment dat și combină toate valorile
asociate cu acea cheie într-un fel. Modul de combinare
de valori este determinat de codul scris de utilizator pentru Reduce
funcție.
Figura 2.2 sugerează acest calcul.

2.2.1 Sarcinile hărții

Vedem fișierele de intrare pentru o sarcină de hartă ca fiind formate din elemente, care pot fi
orice tip: un tuplu sau un document, de exemplu. O bucată este o colecție de
elemente și niciun element nu este stocat pe două bucăți. Din punct de vedere tehnic, toate intrările
pentru a harta sarcinile și ieșirile din Reducerea sarcinilor sunt în formă de pereche cheie-valoare,
dar în mod normal, tastele elementelor de intrare nu sunt relevante și vom avea tendința să
ignoră-i. Insistarea acestui formular pentru intrări și ieșiri este motivată de
dorința de a permite compunerea mai multor procese MapReduce.

Funcția Hartă ia ca argument un element de intrare și îl produce
zero sau mai multe perechi cheie-valoare. Tipurile de chei și valori sunt fiecare arbitrare.
Mai mult, cheile nu sunt „chei” în sensul obișnuit; nu trebuie să fie unice.
Mai degrabă o sarcină de hartă poate produce mai multe perechi cheie-valoare cu aceeași cheie, chiar
din același element.
Exemplul 2.1: Vom ilustra un calcul MapReduce cu ceea ce are
devine exemplul standard de aplicație: numărarea numărului de apariții

---

Pagina 46

Intrare
bucăți
grup
prin chei
Cheie-valoare
(k, v)
perechi
valorile lor
Cheile cu toate
ieșire
Combinat
Hartă
sarcini
Reduce
sarcini
(k, [v, w, ...])
Figura 2.2: Schema unui calcul MapReduce
pentru fiecare cuvânt dintr-o colecție de documente. În acest exemplu, fișierul de intrare este un
depozit de documente, iar fiecare document este un element. Funcția Hartă
pentru acest exemplu folosește chei de tip String (cuvintele) și valori care
sunt numere întregi. Sarcina Hartă citește un document și îl împarte în ordinea sa
de cuvinte w_1, w_2, ..., w_n. Apoi emite o secvență de perechi cheie-valoare în cazul în care
valoarea este întotdeauna 1. Adică rezultatul sarcinii Hărți pentru acest document este
secvența de perechi cheie-valoare:
(w_1, 1), (w_2, 1), ..., (w_n, 1)
Rețineți că o singură sarcină de hartă va procesa de obicei multe documente - toate
documentele într-unul sau mai multe bucăți. Astfel, rezultatul său va fi mai mult decât
secvența pentru documentul sugerat mai sus. Rețineți, de asemenea, că, dacă un cuvânt w
apare de câte ori printre toate documentele atribuite acelei sarcini, apoi acolo
vor fi m perechi cheie-valoare (w, 1) printre ieșirile sale. O opțiune, pe care o discutăm
în secțiunea 2.2.4, este pentru această sarcină hartă să combine aceste perechi m într-o singură
pair (w, m), dar putem face asta doar pentru că, așa cum vom vedea, sarcinile Reduce
aplicați o operație asociativă și comutativă, adăugată, valorilor. ✷

2.2.2 Gruparea după cheie

De îndată ce toate sarcinile Hărții s-au finalizat cu succes, perechile cheie-valoare sunt
grupate după cheie, iar valorile asociate fiecărei chei sunt formate într-o singură
lista valorilor pentru acea cheie. Gruparea este realizată de sistem, indiferent

---

Pagina 47

despre ceea ce fac sarcinile Hărți și Reducere. Procesul controlorului principal știe
câte sarcini de reducere vor exista, să spunem r astfel de sarcini. Utilizatorul spune de obicei
sistemul MapReduce ce ar trebui să fie r. Apoi controlerul principal alege un
funcție hash care ia o cheie ca argument și produce un număr bucket de la
De la 0 la r - 1. Fiecare cheie care este lansată de o sarcină de hartă este hashată și valoarea-cheie a acesteia
perechea este pusă într-unul dintre r fișiere locale. Fiecare fișier este destinat pentru unul dintre r Reduce
sarcini. [1]
Pentru a efectua gruparea după cheie și distribuție la sarcinile Reduce,
controlerul principal fuzionează fișierele din fiecare sarcină de hartă pentru care sunt destinate
o anumită sarcină Reduce și alimentează fișierul fuzionat în acel proces ca un set
frecvența perechilor cheie / listă de valori. Aceasta este, pentru fiecare tastă k, intrarea în
Reducerea sarcinii care tratează cheia k este o pereche de formă (k, [v_1, v_2, ..., v_n]), unde
(k, v_1), (k, v_2), ..., (k, v_n) sunt toate perechile cheie-valoare cu cheia k provenind de la
toate sarcinile Hărții.

2.2.3 Sarcinile de reducere

Argumentul funcției Reduce este o pereche constând dintr-o cheie și lista sa de valorile asociate. Ieșirea funcției Reduce este o secvență de zero sau mai multe perechi cheie-valoare. Aceste perechi cheie-valoare pot fi de un tip diferit de cele trimise de la sarcini de hartă pentru a reduce sarcinile, dar adesea sunt de același tip. Ne vom referi la aplicarea funcției Reduce la o singură tastă și a acesteia lista de valori asociată ca reductor.

O sarcină Reduce primește una sau mai multe chei și listele lor de valori asociate. Adică, o sarcină Reduce execută unul sau mai multe reductoare. Ieșirile de la toate sarcinile de reducere sunt îmbinate într-un singur fișier.

Exemplul 2.2: Să continuăm cu exemplul de numărare a cuvintelor din Exemplul 2.1. Funcția Reduce adaugă pur și simplu toate valorile. Ieșirea unui reductor constă din cuvânt și sumă. Astfel, rezultatul tuturor sarcinilor Reduce este un secvență de (w, m) perechi, unde w este un cuvânt care apare cel puțin o dată printre toate documentele de intrare și m este numărul total de apariții ale w dintre acele documente. ✷

2.2.4 Combinatoare

Uneori, o funcție Reduce este asociativă și comutativă. Adică valorile care trebuie combinate pot fi combinate în orice ordine, cu același rezultat. Adăugarea efectuată în exemplul 2.2 este un exemplu asociativ și operație comutativă. Nu contează cum comandăm sau grupăm o listă de numerele $v_1, v_2, ..., v_n$; suma va fi aceeași.

Când funcția Reduce este asociativă și comutativă, putem împinge o parte din ceea ce reducerii fac cu sarcinile Hărții. De exemplu, în loc de fiecare

[1] Opțional, utilizatorii își pot specifica propria funcție hash sau altă metodă de atribuire a cheilor pentru a reduce sarcinile. Cu toate acestea, indiferent de algoritmul utilizat, fiecare cheie este atribuită unei singure una Reduceți sarcina.

Reductoare, sarcini reduse, calculează noduri și înclinare

Dacă dorim un paralelism maxim, atunci am putea folosi o sarcină de reducere pentru a executa fiecare reductor, adică o singură cheie și lista de valori asociate. În plus, am putea executa fiecare activitate de reducere la un nod de calcul diferit, așa că toți ar executa în paralel. Acest plan nu este de obicei cel mai bun. unu problema este că există o cheltuială asociată cu fiecare sarcină pe care o creăm, deci s-ar putea să dorim să menținem numărul de sarcini Reduce mai mic decât numărul a diferitelor chei. Mai mult, de multe ori există mult mai multe chei decât există calculează nodurile disponibile, astfel încât să nu obținem niciun beneficiu dintr-un număr mare de Reduce sarcinile.

În al doilea rând, există adesea variații semnificative în lungimile valorii liste pentru diferite chei, astfel încât reductorii diferiți necesită diferite perioade de timp. Dacă facem din fiecare reductor o sarcină separată de reducere, atunci sarcinile în sine va prezenta o înclinație - o diferență semnificativă în intervalul de timp fiecare ia. Putem reduce impactul înclinării folosind mai puține sarcini de reducere decât există reductori. Dacă tastele sunt trimise aleatoriu pentru a reduce sarcinile, noi se poate aștepta ca va exista o medie a timpului total necesar de diferitele sarcini de reducere. Putem reduce și mai mult distorsiunea folosind mai mult Reduceți sarcinile decât există noduri de calcul. În acest fel, lung Reduce sarcinile ar putea ocupa un nod de calcul pe deplin, în timp ce mai multe Reduce mai scurte sarcinile pot rula secvențial la un singur nod de calcul.

Sarcină de mapare în Exemplul 2.1 producând multe perechi (w, 1), (w, 1), ..., am putea aplicați funcția Reduce în cadrul fiecărei sarcini Map, înainte de ieșirile din Sarcinile de hartă sunt supuse grupării și agregării. Aceste perechi cheie-valoare ar fi astfel înlocuit cu o pereche cu cheia w și valoare egală cu suma lui toate cele 1 din toate acele perechi. Adică perechile cu cheia w generate de un singur Sarcina hărții ar fi înlocuită de o pereche (w, m), unde m este de câte ori că w apare printre documentele gestionate de această sarcină Hartă. Rețineți că este încă necesar să faceți grupare și agregare și să transmiteți rezultatul sarcinile Reduce, deoarece de obicei va exista o pereche cheie-valoare cu cheia w provenind din fiecare dintre sarcinile Hărții.

2.2.5 Detalii despre executarea MapReduce

Să analizăm acum mai detaliat cum este executat un program care folosește MapReduce. Figura 2.3 oferă o schiță a modului în care procesele, sarcinile și fișierele interacționează. Luând avantajul unei biblioteci furnizate de un sistem MapReduce precum Hadoop, programul utilizatorului bifurează un proces de control master și un anumit număr de lucrători procesează la diferite noduri de calcul. În mod normal, un lucrător se ocupă de oricare hartă sarcini (un lucrător de pe hartă) sau sarcini de reducere (un lucrător de reducere), dar nu ambele.

Maestrul are multe responsabilități. Una este crearea unui număr de

---

Pagina 49

Program
Utilizator
Maestru
Muncitor
Muncitor
Muncitor
Muncitor
Muncitor
Date
Intrare
Fișier
Ieșire
furculiță
furculiță
furculiță
Hartă
atribui
atribui
Reduce
Intermediar
Dosare
Figura 2.3: Prezentare generală a execuției unui program MapReduce
Hărți sarcini și un număr de sarcini Reduce, aceste numere fiind selectate
de către programul utilizator. Aceste sarcini vor fi atribuite proceselor lucrătorilor de către
Maestru. Este rezonabil să creați o sarcină de hartă pentru fiecare bucată din intrare
fișier (e), dar este posibil să dorim să creăm mai puține sarcini de reducere. Motivul limitării
numărul de sarcini Reduce este acela că este necesar să se creeze fiecare sarcină Hartă
un fișier intermediar pentru fiecare activitate de reducere și, dacă există prea multe reduceri
sarcini explodează numărul de fișiere intermediare.
Maestrul ține evidența stării fiecărei Hărți și Reduce sarcina (inactiv,
executând la un anumit lucrător sau finalizat). Un proces muncitor raportează către
Maestrul când termină o sarcină, iar o nouă sarcină este programată de Maestru
pentru acel proces muncitor.
Fiecărei sarcini de Hărți i se atribuie unul sau mai multe bucăți din fișierele de intrare și
execută pe acesta codul scris de utilizator. Sarcina Hartă creează un fișier pentru
fiecare Reduceți sarcina de pe discul local al Lucrătorului care execută sarcina Hartă.
Maestrul este informat despre locația și dimensiunile fiecăruia dintre aceste fișiere și despre
Reduceți sarcina pentru care fiecare este destinat. Când o sarcină Reduce este atribuită de
Maestru pentru un proces Worker, acelei sarcini i se oferă toate fișierele care formează intrarea sa.
Activitatea Reduce execută codul scris de utilizator și își scrie ieșirea într-un
fișier care face parte din sistemul de fișiere distribuit din jur.

Pagina 50

2.2.6 Tratarea defecțiunilor nodului

Cel mai rău lucru care se poate întâmpla este că nodul de calcul la care Maestrul
execută eșuează. În acest caz, întregul job MapReduce trebuie repornit.
Dar numai acest nod poate aduce întregul proces în jos; alte eșecuri vor fi
administrat de Master, iar jobul MapReduce se va finaliza în cele din urmă.
Să presupunem că nodul de calcul la care se află un lucrător Map nu reușește. Acest eșec
Ureul va fi detectat de către Maestru, deoarece îl pică periodic pe Lucrător
procese. Toate sarcinile Hărții care au fost atribuite acestui Lucrător vor trebui
să fie refăcute, chiar dacă ar fi terminat. Motivul refacerii Hărții completate
task-uri este că ieșirea lor destinată sarcinilor Reduce se află la acel calcul
nod, iar acum nu este disponibil pentru sarcinile Reduce. Maestrul stabilește starea
a fiecăreia dintre aceste sarcini de Hărți să rămână inactiv și le va programa pe un Lucrător când
unul devine disponibil. Maestrul trebuie, de asemenea, să informeze fiecare sarcină de reducere că
locația intrării sale din sarcina hartă respectivă s-a modificat.
Gestionarea unei eșecuri la nodul unui lucrător Reduce este mai simplă. Maestrul
pur și simplu stabilește starea în care se execută în prezent Reduce sarcinile la inactiv. Aceste
va fi reprogramat pe un alt muncitor Reduce ulterior.

2.2.7 Exerciții pentru secțiunea 2.2

Exercițiul 2.2.1: Să presupunem că executăm programul MapReduce de numărare a cuvintelor
descris în această secțiune pe un depozit mare, cum ar fi o copie a Web-ului. Vom
folosiți 100 de sarcini de hartă și un număr de sarcini de reducere.

(a) Să presupunem că nu folosim un combinator la sarcinile Hartă. Te aștepți acolo
să fie o distorsiune semnificativă în timpii preluați de diferiții reductori
lista lor de valori? De ce sau de ce nu?
(b) Dacă combinăm reductoarele într-un număr mic de sarcini de reducere, să zicem 10
sarcini, la întâmplare, vă așteptați ca distorsiunea să fie semnificativă? Ce se întâmplă dacă noi
în schimb combinați reductoarele în 10.000 Reduceți sarcinile?
! (c) Să presupunem că folosim un combinator la sarcinile 100 Map. Te aștepți la o înclinare
a fi semnificativ? De ce sau de ce nu?

2.3 Algoritmi utilizând MapReduce

MapReduce nu este o soluție pentru fiecare problemă, nici măcar pentru fiecare problemă care
profitabil poate folosi multe noduri de calcul care funcționează în paralel. După cum am menționat
în Secțiunea 2.1.2, întregul mediu distribuit de sistem de fișiere are sens numai atunci când
fișierele sunt foarte mari și sunt rar actualizate la locul lor. De exemplu, nu am face-o
așteptați să utilizați fie un DFS, fie o implementare a MapReduce pentru gestionare
vânzări cu amănuntul on-line, chiar dacă utilizează un mare comerciant cu amănuntul on-line, precum Amazon.com
mii de noduri de calcul la procesarea cererilor pe Web. Motivul
este că operațiunile principale pe datele Amazon implică răspunsul la căutări

---

pentru produse, înregistrarea vânzărilor și așa mai departe, procese care implică relativ puțin
calcul și care schimbă baza de date. [2] Pe de altă parte, Amazon ar putea
utilizați MapReduce pentru a efectua anumite interogări analitice pe cantități mari de date,
precum găsirea pentru fiecare utilizator a acelor utilizatori ale căror modele de cumpărare erau cele mai mari
similar.
Scopul inițial pentru care implementarea de către Google a MapReduce
a fost creat pentru a executa multiplicări matrice-vector foarte mari ca și cum sunt
necesare în calculul PageRank (vezi Capitolul 5). Vom vedea asta
calculele matrice-vector și matrice-matrice se încadrează frumos în MapReduce
stilul de calcul. O altă clasă importantă de operații care poate utiliza MapRe-
efectiv sunt operațiile relaționale-algebră. Vom examina
MapReduce și execuția acestor operațiuni.

2.3.1 Multiplicarea Matricii-Vector prin MapReduce

Să presupunem că avem o matrice n × n M, al cărei element din rândul i și coloana j este
notat m_{ij} . Să presupunem că avem și un vector v de lungime n, al cărui jth element
este v_j . Atunci produsul matrice-vector este vectorul x de lungime n, al cărui ith
elementul x_i este dat de

$$x_i = \sum_{j=1}^{n} m_{ij} v_j$$

Dacă n = 100, nu dorim să folosim un DFS sau MapReduce pentru acest calcul.
Dar acest tip de calcul se află în centrul clasamentului paginilor web care
continuă la motoarele de căutare și acolo, n este de ordinul trilioanelor. [3]
Permiteți-ne
presupuneți mai întâi că n este mare, dar nu atât de mare încât vectorul v nu se potrivește în main
memorie și astfel să fie disponibil pentru fiecare sarcină de pe hartă.
Fiecare matrice M și vectorul v vor fi stocate într-un fișier al DFS. Noi
presupunem că coordonatele rând-coloană ale fiecărui element de matrice vor fi descoperite
erabil, fie din poziția sa în fișier, fie pentru că este stocat cu explicit
coordonate, ca triplă (i, j, m_{ij}). De asemenea, asumăm poziția elementului v_j în
vectorul v va putea fi descoperit în mod analog.
Funcția Map: Funcția Map este scrisă pentru a se aplica unui element din
M. Totuși, dacă v nu este deja citit în memoria principală la nodul de calcul
executarea unei sarcini de hartă, apoi v este citit mai întâi, în întregime, și ulterior va fi
să fie disponibil pentru toate aplicațiile funcției Map efectuate la această sarcină Map.
Fiecare sarcină Hartă va funcționa pe o bucată din matricea M. Din fiecare matrice
elementul m_{ij} produce perechea cheie-valoare (i, m_{ij} v_j). Astfel, toți termenii
suma care alcătuiește componenta x_i a produsului matrice-vector va obține
aceeași cheie, i.

[2] Amintiți-vă că, chiar dacă vă uitați la un produs pe care nu îl cumpărați, Amazon își amintește acest lucru te-ai uitat la el.

[3] Matricea este rară, având în medie 10 până la 15 elemente diferite de zero pe rând, din moment ce matrix reprezintă legăturile din Web, cu m_{ij} diferit de zero dacă și numai dacă există un link de la pagina j la pagina i. Rețineți că nu există nicio modalitate de a putea stoca o matrice densă a cărei latură era 10^{12} , deoarece ar avea 10^{24} de elemente.

Funcția Reduce: Funcția Reduce sumează pur și simplu toate valorile ca-
asociat cu o cheie dată i. Rezultatul va fi o pereche (i, x_i).

2.3.2 Dacă Vectorul v nu se poate potrivi în memoria principală

Cu toate acestea, este posibil ca vectorul v să fie atât de mare încât să nu se încadreze în
integral în memoria principală. Nu este necesar ca v să se încadreze în memoria principală a
calculează nodul, dar dacă nu, atunci va exista un număr foarte mare de
accesează pe disc pe măsură ce mutăm bucăți din vector în memoria principală pentru a le înmulți
componente prin elemente ale matricei. Astfel, ca alternativă, ne putem împărți
matricea în dungi verticale de lățime egală și împarte vectorul într-un egal
numărul de dungi orizontale, de aceeași înălțime. Scopul nostru este să folosim suficient
dungi, astfel încât porțiunea vectorului dintr-o bandă să poată încadra convenabil în
memorie principală la un nod de calcul. Figura 2.4 sugerează aspectul partiției
ca și cum matricea și vectorul sunt împărțite fiecare în cinci dungi.

M
Matrice
Vectorul **v**

Figura 2.4: Împărțirea unei matrice și a unui vector în cinci dungi

Banda i a matricei înmulțește numai componentele din banda i
a vectorului. Astfel, putem împărți matricea într-un fișier pentru fiecare bandă și
faceți același lucru pentru vector. Fiecărei sarcini de Hărți i se atribuie o bucată dintr-unul din
dungi ale matricei și obține întreaga bandă corespunzătoare a vectorului. The
Activitățile de hartă și reducere pot acționa exact așa cum a fost descris în secțiunea 2.3.1
pentru cazul în care sarcinile Hărții obțin întregul vector.

Vom lua din nou multiplicarea matrice-vector folosind MapReduce din nou în
Secțiunea 5.2. Acolo, din cauza aplicației particulare (calculul PageRank
), avem o constrângere suplimentară conform căreia vectorul rezultat ar trebui să fie
menționat în același mod ca vectorul de intrare, astfel încât ieșirea poate deveni
intrare pentru o altă iterație a multiplicării matrice-vector. Vom vedea
acolo că cea mai bună strategie implică partiționarea matricei M în pătrat
blocuri, mai degrabă decât dungi.

2.3.3 Operații relaționale-algebră

Există o serie de operații pe date la scară largă care sunt utilizate în baza de date
întrebări. Multe aplicații tradiționale de baze de date implică recuperarea de mici dimensiuni
o mulțime de date, chiar dacă baza de date în sine poate fi mare. De exemplu, a
interogarea poate solicita soldul bancar al unui anumit cont. Astfel de întrebări sunt
aplicații nefolositoare ale MapReduce.

Cu toate acestea, există multe operații pe date care pot fi descrise cu ușurință în
termenii primitivelor comune de interogare a bazei de date, chiar dacă interogările în sine
nu sunt executate în cadrul unui sistem de gestionare a bazelor de date. Astfel, un bun început
punctul pentru explorarea aplicațiilor MapReduce este luarea în considerare a standardului
operațiuni asupra relațiilor. Presupunem că sunteți familiarizați cu sistemele de baze de date,
limbajul de interogare SQL și modelul relațional, dar pentru a revizui, o relație este
un tabel cu anteturi de coloană numite atribute. Se numesc rânduri ale relației
tupluri. Setul de atribute ale unei relații se numește schema sa. De multe ori scriem
o expresie ca R (A_1 , A_2 , ..., A_n) pentru a spune că numele relației este R și al său
atributele sunt A_1 , A_2 , ..., A_n .

De la catre
url1
url2
url1
url3
url2
url3
url2
url4
. . .
. . .

Figura 2.5: Legăturile de relație constă din setul de perechi de URL-uri astfel încât
primul are unul sau mai multe linkuri către al doilea

Exemplul 2.3: În Fig. 2.5 vedem o parte din relația Link-uri care descrie

structura Web-ului. Există două atribute, De la și Până. Un rând sau
tuplu, din relație este o pereche de URL-uri astfel încât să existe cel puțin un link de la
primul URL către al doilea. De exemplu, primul rând din Fig. 2.5 este perechea
(url1, url2). Acest tuplu spune că pagina web url1 are un link către pagina url2. In timp ce
am arătat doar patru tupluri, relația reală a Webului sau porțiunea de
care ar fi stocat de un motor de căutare tipic, are trilioane de tupluri. ✷
O relație, oricât de mare, poate fi stocată ca fișier într-un sistem de fișiere distribuit.
Elementele acestui fișier sunt tuplurile relației.
Există mai multe operațiuni standard privind relațiile, adesea denumite re-
algebră națională, care sunt utilizate pentru a implementa interogări. Întrebările în sine
de obicei sunt scrise în SQL. Operațiunile relaționale-algebră le vom discuta
sunt:
1. Selectare: Aplicați o condiție C fiecărui tuplu din relație și produceți
ca rezultat doar acele tupluri care satisfac C. Rezultatul acestei selecții este
notat σ_C (R).

Pagina 54

2. Proiecție: Pentru unele subseturi S ale atributelor relației, produceți
din fiecare tuplu doar componentele pentru atributele din S. Rezultatul
din această proiecție se notează π_S (R).
3. Unire, intersecție și diferență: Aceste operațiuni bine-cunoscute stabilite
se aplică seturilor de tupluri în două relații care au aceeași schemă.
Există, de asemenea, versiuni bag (multiset) ale operațiilor în SQL, cu
definiții oarecum neintuitive, dar nu vom intra în versiunile de sac
a acestor operațiuni aici.
4. Unire naturală: având în vedere două relații, comparați fiecare pereche de tupluri, una din
fiecare relație. Dacă tuplurile sunt de acord cu toate atributele comune
la cele două scheme, apoi produceți un tuplu care are componente pentru fiecare
a atributelor din fiecare schemă și este de acord cu cele două tupluri de pe fiecare
atribut. Dacă tuplurile nu sunt de acord cu unul sau mai multe atribute partajate, atunci
nu produce nimic din această pereche de tupluri. Îmbinarea naturală a relațiilor R
și S este notat R ⋈ S. În timp ce vom discuta despre executarea numai a naturii
ural se alătură cu MapReduce, toate echijoins (se alătură în cazul în care acordul tuplu
condiția implică egalitatea atributelor din cele două relații care nu
neapărat au același nume) pot fi executate în același mod. Noi
va da o ilustrare în exemplul 2.4.
5. Gruparea și agregarea: [4] Având în vedere o relație R, partiționați tuplurile sale
conform valorilor lor într-un set de atribute G, numit grupare
atribute. Apoi, pentru fiecare grup, agregați valorile în anumite alte
omagii. Agregările permise în mod normal sunt SUM, COUNT, AVG,
MIN și MAX, cu semnificațiile evidente. Rețineți că MIN și MAX
necesită ca atributele agregate să aibă un tip care poate fi comparat,
de exemplu, numere sau șiruri, în timp ce SUM și AVG necesită ca tipul să permită
operații aritmetice, de obicei numai numere. COUNT poate fi efectuat
pe date de orice tip. Desemnăm o operațiune de grupare și agregare pe
o relație R cu γ_X (R), unde X este o listă de elemente care sunt fiecare
(a) Un atribut de grupare sau
(b) O expresie θ(A), unde θeste una dintre cele cinci operații de agregare
ca SUM, și A este un atribut care nu se află în grup
atribute.
Rezultatul acestei operații este un tuplu pentru fiecare grup. Acest tuplu are
o componentă pentru fiecare dintre atributele de grupare, cu valoarea comună
la tupluri din acel grup. De asemenea, are o componentă pentru fiecare agregare,
cu valoarea agregată pentru grupul respectiv. Vom vedea o ilustrare în
Exemplul 2.5.

[4] Unele descrieri ale algebrei relaționale nu includ aceste operații și chiar ele
nu au făcut parte din definiția originală a acestei algebre. Cu toate acestea, aceste operațiuni sunt așa
important în SQL, că tratamentele moderne ale algebrei relaționale le includ.

Pagina 55

Exemplul 2.4: Să încercăm să găsim căile de lungime două în Web, folosind
relația Legături din Fig. 2.5. Adică vrem să găsim triplurile URL-urilor
(u, v, w) astfel încât există o legătură de la u la v și o legătură de la v la w. Noi

în esență, dorim să luăm unirea naturală a Link-urilor cu sine, dar mai întâi avem nevoie
să ne imaginăm că este vorba de două relații, cu scheme diferite, așa că putem descrie
conexiunea dorită ca unire naturală. Imaginați-vă astfel că există două exemplare
de legături, și anume L1 (U1, U2) și L2 (U2, U3). Acum, dacă calculăm L1 ⋈ L2,
vom avea exact ceea ce ne dorim. Adică pentru fiecare tuplu t1 din L1 (adică
fiecare tupl de legături) și fiecare tuplu t2 de L2 (un alt tuplu de legături, eventual
chiar același tuplu), vedeți dacă componentele lor U2 sunt aceleași. Rețineți că
aceste componente sunt a doua componentă a t1 și prima componentă a
t2. Dacă aceste două componente sunt de acord, atunci produceți un tuplu pentru rezultat, cu
schemă (U1, U2, U3). Acest tuplu este format din prima componentă a t1,
a doua componentă a t1 (care trebuie să fie egală cu prima componentă a t2) și
a doua componentă a t2.
Este posibil să nu dorim întreaga cale de lungimea a doua, ci doar perechile
(u, w) de URL-uri astfel încât să existe cel puțin o cale de la u la w cu lungimea doi. Dacă
deci, putem proiecta componentele medii calculând $\pi_{U1, U3}$ (L1 ⋈ L2).
✷
Exemplul 2.5: imaginați-vă că un site de socializare are o relație
Prieteni (utilizator, prieten)
Această relație are tupluri care sunt perechi (a, b) astfel încât b este prietenul lui a. Site-ul
ar putea dori să dezvolte statistici despre numărul de prieteni pe care membrii îi au.
Primul lor pas ar fi să calculeze un număr al numărului de prieteni din fiecare
utilizator. Această operație se poate face prin grupare și agregare, în mod specific
$\gamma_{\text{Utilizator, COUNT (Prieten)}}$ (Prieteni)
Această operație grupează toate tuplurile după valoarea din prima lor componentă, deci
există un grup pentru fiecare utilizator. Apoi, pentru fiecare grup, numărul numărului
de prieteni ai acelui utilizator este făcut. Rezultatul va fi un tuplu pentru fiecare grup și
un tuplu tipic ar arăta ca (Sally, 300), dacă utilizatorul „Sally” are 300 de prieteni. ✷

2.3.4 Selecții de calcul prin MapReduce

Selecțiile nu au nevoie de întreaga putere a MapReduce. Se pot face
cel mai convenabil doar în porțiunea de hartă, deși s-ar putea face și ele
numai în porția de reducere. Iată o implementare MapReduce a selecției
σ_C (R).
Funcția Hartă: Pentru fiecare tuplu t din R, testați dacă îndeplinește C. Dacă da, produceți
perechea cheie-valoare (t, t). Adică, atât cheia, cât și valoarea sunt t. Dacă t nu
satisfac C, atunci mapatorul pentru t nu produce nimic.

Funcția Reduce: Funcția Reduce este identitatea. Pur și simplu trece
fiecare pereche cheie-valoare la ieșire.
Rețineți că rezultatul nu este exact o relație, deoarece are perechi cheie-valoare.
Cu toate acestea, o relație poate fi obținută utilizând numai componentele valorice (sau
numai componentele cheie) ale ieșirii.

2.3.5 Calcularea proiecțiilor prin MapReduce

Proiecția se realizează similar cu selecția. Totuși, pentru că proiecția
poate face ca același tupl să apară de mai multe ori, funcția Reduce trebuie
elimina duplicatele. Putem calcula π_S (R) după cum urmează.
Funcția Hărți: Pentru fiecare tuplu t din R, construiește un tuplu t' eliminând
din t acele componente ale căror atribute nu se află în S. Ieșiți valoarea-cheie
pereche (t', t').
Funcția de reducere: pentru fiecare tastă t' produsă de oricare dintre sarcinile Hărții,
vor exista una sau mai multe perechi cheie-valoare (t', t'). După ce sistemul grupează tasta-
perechi de valori după tastă, funcția Reduce transformă (t', [t', t', ..., t']) în (t', t'), deci
produce exact o pereche (t', t') pentru această cheie t'.
Observați că operația Reduce este eliminarea duplicat. Această operațiune
este asociativ și comutativ, deci un combinator asociat cu fiecare sarcină de hartă
poate elimina orice duplicat este produs local. Cu toate acestea, Reduce
sunt încă necesare sarcini pentru a elimina două tupluri identice provenite din diferite
Sarcini de hartă.

2.3.6 Unire, intersecție și diferență prin MapReduce

În primul rând, luați în considerare unirea a două relații. Să presupunem că relațiile R și S au
aceeași schemă. Activităților de hartă li se vor atribui bucăți din R sau S; nu
care contează. Sarcinile Hărții nu fac nimic, cu excepția transmiterii lor
tupluri ca perechi cheie-valoare la sarcinile Reduce. Acesta din urmă trebuie doar să elimine
duplicate ca pentru proiecție.
Funcția Hartă: Transformați fiecare tuple de intrare t într-o pereche cheie-valoare (t, t).

Funcția de reducere: asociată cu fiecare tastă t va exista fie una, fie
două valori. Produceți ieșire (t, t) în ambele cazuri.
Pentru a calcula intersecția, putem folosi aceeași funcție Map. In orice caz,
funcția Reduce trebuie să producă un tuplu numai dacă ambele relații au tuplul.
Dacă tasta t are o listă de două valori [t, t] asociate, atunci Reduceți
sarcina pentru t ar trebui să producă (t, t). Cu toate acestea, dacă lista de valori asociată cu cheia
t este doar [t], apoi unul dintre R și S lipsește t, deci nu vrem să producem un
tuplu pentru intersecție.
Funcția Hartă: transformați fiecare tuplu t într-o pereche cheie-valoare (t, t).
Funcția Reduce: Dacă tasta t are lista de valori [t, t], atunci produce (t, t). Altul
altfel, nu produce nimic.

Pagina 57

Diferența R - S necesită ceva mai multă gândire. Singurul mod în care un tuplu
t poate apărea în ieșire, dacă este în R, dar nu în S. Funcția Hartă
poate trece tupluri de la R și S, dar trebuie să informeze funcția Reduce
dacă tuplul a venit de la R sau S. Vom folosi astfel relația ca
valoare asociată tastei t. Iată o specificație pentru cele două funcții.
Funcția Hartă: Pentru un tuplu t în R, produceți perechea cheie-valoare (t, R) și
pentru un tuplu t în S, produceți perechea cheie-valoare (t, S). Rețineți că intenția este aceea
valoarea este numele lui R sau S (sau mai bine, un singur bit care indică dacă
relația este R sau S), nu întreaga relație.
Funcția de reducere: pentru fiecare tastă t, dacă lista valorilor asociate este [R], atunci
produce (t, t). Altfel, nu produce nimic.

2.3.7 Calcularea Natural Join prin MapReduce

Ideea din spatele implementării unirii naturale prin MapReduce poate fi văzută dacă noi
uită-te la cazul specific al îmbinării R (A, B) cu S (B, C). [5] Trebuie să găsim tupluri
care sunt de acord asupra componentelor lor B, adică a doua componentă din tupluri
de R și prima componentă a tuplurilor de S. Vom folosi valoarea B a tuplurilor
din oricare dintre relații ca cheie. Valoarea va fi cealaltă componentă și
numele relației, astfel încât funcția Reduce poate ști de unde a venit fiecare tuplu
din.
Funcția de hartă: pentru fiecare tuplu (a, b) din R, produceți perechea cheie-valoare
(sutien)). Pentru fiecare tuplu (b, c) din S, produceți perechea cheie-valoare (b, (S, c)).
Funcția Reduce: Fiecare valoare a tastei b va fi asociată cu o listă de perechi
care sunt fie de formă (R, a), fie (S, c). Construiți toate perechile formate dintr-una
cu prima componentă R și cealaltă cu prima componentă S, să zicem (R, a) și
(S, c). Ieșirea din această listă de chei și valori este o secvență de perechi cheie-valoare.
Cheia este irelevantă. Fiecare valoare este una dintre triplele (a, b, c) astfel încât (R, a)
și (S, c) sunt pe lista de introducere a valorilor pentru tasta b.
Același algoritm funcționează dacă relațiile au mai mult de două atribute.
Vă puteți gândi la A ca reprezentând toate acele atribute în schema lui R, dar
nu S. B reprezintă atributele din ambele scheme și C reprezintă atributele
numai în schema lui S. Cheia pentru un tuplu de R sau S este lista valorilor în toate
atributele care se află în schemele R și S. Valoarea pentru un tuplu
lui R este numele R împreună cu valorile tuturor atributelor aparținând
R dar nu la S, iar valoarea pentru un tuplu de S este numele S împreună cu
valorile atributelor aparținând lui S, dar nu și lui R.
Funcția Reduce analizează toate perechile cheie-valoare cu o tastă dată și
combină acele valori din R cu acele valori ale lui S în toate modurile posibile. Din
fiecare împerechere, tuplul produs are valorile din R, valorile cheie și
valorile din S.

[5] Dacă sunteți familiarizat cu implementarea bazei de date, veți recunoaște MapReduce
implementarea join-ului ca clasică join hash paralelă.

Pagina 58

2.3.8 Gruparea și agregarea după MapReduce

Așa cum am făcut pentru aderare, vom discuta aici doar exemplul minim de grupare
agregare, unde există un atribut de grupare (A), unul agregat
atribut (B) și un atribut (C) care nu este nici grupat, nici agregat.
Fie R (A, B, C) o relație la care aplicăm operatorul $\gamma_{A,\ \theta(B)}$ (R). Hartă
va efectua gruparea, în timp ce Reduce face agregarea.

Funcția Hartă: pentru fiecare tuplu (a, b, c) produce perechea cheie-valoare (a, b).
Funcția Reduce: Fiecare tastă a reprezintă un grup. Aplicați agregarea operator θla lista $[b_1, b_2, ..., b_n]$ a valorilor B asociate cu cheia a. The ieșirea este perechea (a, x), unde x este rezultatul aplicării θlistei. Pentru de exemplu, dacă θeste SUM, atunci $x = b_1 + b_2 + \cdots + b_n$, iar dacă θeste MAX, atunci x este cel mai mare dintre $b_1, b_2, ..., b_n$.
Dacă există mai multe atribute de grupare, atunci cheia este lista valorilor a unui tuplu pentru toate aceste atribute. Dacă există mai multe agregări, atunci funcția Reduce aplică fiecare dintre ele la lista de valori asociate o cheie dată și produce un tuplu format din cheie, inclusiv componente pentru toate atributele de grupare dacă există mai multe, urmate de rezultatele fiecare dintre agregări.

2.3.9 Multiplicarea matricei

Dacă M este o matrice cu elementul m_{ij} în rândul i și coloana j, iar N este o matrice cu elementul n_{jk} în rândul j și coloana k, atunci produsul P = MN este matricea P cu elementul p_{ik} în rândul i și coloana k, unde

$$p_{ik} = \Sigma_j m_{ij} n_{jk}$$

Este necesar ca numărul de coloane ale lui M să fie egal cu numărul de rânduri de N, deci suma peste j are sens.
Ne putem gândi la o matrice ca la o relație cu trei atribute: numărul rândului, numărul coloanei și valoarea din acel rând și coloană. Astfel, am putea vedea matricea M ca relație M (I, J, V), cu tupluri (i, j, m_{ij}) și am putea vedea matricea N ca relație N (J, K, W), cu tupluri (j, k, n_{jk}). Pe măsură ce sunt matricile mari de multe ori rare (mai ales 0) și, din moment ce putem omite tuplele pentru elementele matricei care sunt 0, această reprezentare relațională este adesea una foarte bună pentru un mare matrice. Cu toate acestea, este posibil ca i, j și k să fie implicați în poziția a element matrice din fișierul care îl reprezintă, mai degrabă decât scris explicit cu elementul în sine. În acest caz, funcția Hartă va trebui să fie concepută pentru a construiți componentele I, J și K ale tuplurilor din poziția datelor.
Produsul MN este aproape o îmbinare naturală urmată de grupare și adunare. Adică, îmbinarea naturală a lui M (I, J, V) și N (J, K, W), având numai atributul J în comun, ar produce tupluri (i, j, k, v, w) din fiecare tuplu (i, j, v) în M și tuplu (j, k, w) în N. Acest tuplu cu cinci componente reprezintă

---

pereche de elemente matrice (m_{ij}, n_{jk}). Ceea ce ne dorim în schimb este produsul aceste elemente, adică tuplul cu patru componente (i, j, k, v × w), pentru că reprezintă produsul $m_{ij} n_{jk}$. Odată ce avem această relație ca rezultat al uneia Operațiunea MapReduce, putem efectua gruparea și agregarea, cu I și K ca atribute de grupare și suma lui V × W ca agregare. Acea adică, putem implementa multiplicarea matricii ca cascadă a două MapReduce operațiuni, după cum urmează. Primul:
Funcția Hartă: Pentru fiecare element matrice m_{ij}, produceți perechea de valori cheie (j, (M, i, m_{ij})). La fel, pentru fiecare element matricial n_{jk}, produceți valoarea cheie pereche (j, (N, k, n_{jk})). Rețineți că M și N din valori nu sunt matrici înșiși. Mai degrabă sunt nume ale matricilor sau, mai exact, un singur bit care indică dacă elementul provine de la M sau N (așa cum am menționat în ceea ce privește funcția de hartă similară pe care am folosit-o pentru îmbinarea naturală) :.
Funcția de reducere: Pentru fiecare tastă j, examinați lista valorilor asociate.
Pentru fiecare valoare care vine de la M, să spunem (M, i, m_{ij}) și fiecare valoare care vine din N, să zicem (N, k, n_{jk}), producem o pereche cheie-valoare cu cheie egală cu (i, k) și valoare egală cu produsul acestor elemente, $m_{ij} n_{jk}$.
Acum, efectuăm o grupare și agregare printr-o altă operațiune MapReduce aplicată la ieșirea primei operații MapReduce.
Funcția Hartă: Această funcție este doar identitatea. Adică pentru fiecare intrare element cu cheie (i, k) și valoare v, produce exact această pereche cheie-valoare.
Funcția de reducere: Pentru fiecare tastă (i, k), produceți suma listei de valorile asociate cu această cheie. Rezultatul este o pereche ((i, k), v), unde v este valoarea elementului din rândul i și coloana k a matricei P = MN.

2.3.10 Multiplicarea matricei cu un singur pas MapReduce

Există adesea mai multe modalități de a utiliza MapReduce pentru a rezolva o problemă. Tu poate doriți să utilizați doar o singură trecere MapReduce pentru a efectua multiplicarea matricei P = MN. [6] Este posibil să facem acest lucru dacă depunem mai multă muncă în cele două funcții. Începeți prin utilizarea funcției Map pentru a crea seturile de elemente matrice care sunt necesar pentru a calcula fiecare element al răspunsului P. Observați că un element al

M sau N contribuie la multe elemente ale rezultatului, deci un element de intrare va
să fie transformat în mai multe perechi cheie-valoare. Cheile vor fi perechi (i, k), unde i este a
rândul lui M și k este o coloană a lui N. Iată un sinopsis al Hărții și Reducerii
funcții.
Funcția Hartă: Pentru fiecare element m_{ij} al lui M, produceți toată valoarea-cheie
perechi ((i, k), (M, j, m_{ij})) pentru k = 1, 2, ... până la numărul de coloane de
N. În mod similar, pentru fiecare element n_{jk} din N, produceți toate perechile cheie-valoare
((i, k), (N, j, n_{jk})) pentru i = 1, 2, ... până la numărul de rânduri de M. După cum este-
în primul rând, M și N sunt într-adevăr biți pentru a spune care dintre cele două matrice vine o valoare
din.
[6] Cu toate acestea, arătăm în secțiunea 2.6.7 că două pase ale MapReduce sunt de obicei mai bune decât
una pentru multiplicarea matricei.

Pagina 60

Funcția Reduce: Fiecare tastă (i, k) va avea o listă asociată cu toate
valorile (M, j, m_{ij}) și (N, j, n_{jk}), pentru toate valorile posibile ale lui j. Reduce
funcția trebuie să conecteze cele două valori din listă care au aceeași valoare de
j, pentru fiecare j. O modalitate ușoară de a face acest pas este să sortați după j valorile care încep
cu M și sortați după j valorile care încep cu N, în liste separate. Al julea
valorile din fiecare listă trebuie să aibă a treia componentă, m_{ij} și n_{jk} extrase
și multiplicat. Apoi, aceste produse sunt însumate și rezultatul este asociat cu
(i, k) în ieșirea funcției Reduce.
Puteți observa că dacă un rând al matricei M sau o coloană a matricei N
este atât de mare încât nu se va potrivi în memoria principală, atunci sarcinile de reducere vor fi
forțat să folosească un sortare externă pentru a ordona valorile asociate cu o cheie dată
(i, k). Cu toate acestea, în acest caz, matricele în sine sunt atât de mari, poate 10^{20}
elemente, că este puțin probabil să încercăm acest calcul dacă matricile
erau dense. Dacă sunt rare, atunci ne-am aștepta să fie mult mai puține valori
asociate cu orice cheie și ar fi fezabil să se facă suma produselor
în memoria principală.

2.3.11 Exerciții pentru secțiunea 2.3

Exercițiul 2.3.1: Proiectați algoritmi MapReduce pentru a lua un fișier foarte mare de
numere întregi și produc ca ieșire:
(a) Cel mai mare număr întreg.
(b) Media tuturor numerelor întregi.
(c) Același set de numere întregi, dar cu fiecare număr întreg care apare o singură dată.
! (d) Numărul numărului de numere întregi distincte din intrare.
În fiecare parte, puteți presupune că cheia fiecărei perechi de ieșire va fi ignorată
sau scăpat.
Exercițiul 2.3.2: Formularea noastră de multiplicare matrice-vector a presupus că
matricea M era pătrată. Generalizați algoritmul în cazul în care M este un
matricea r-c pentru un număr de rânduri r și coloane c.
! Exercițiul 2.3.3: Sub forma algebrei relaționale implementate în SQL,
acțiunile nu sunt seturi, ci pungi; adică tuplurile au voie să apară mai mult decât
o singura data. Există definiții extinse ale uniunii, intersecției și diferenței pentru
pungi, pe care le vom defini mai jos. Scrieți algoritmi MapReduce pentru calcul
următoarele operațiuni pe sacii R și S:
(a) Uniunea sacilor, definită ca fiind punga de tupluri în care apare tuplul t
suma numărului de ori în care apare în R și S.
(b) Intersecția sacului, definită ca fiind punga de tupluri în care apare tuplul t
minimul numărului de ori în care apare în R și S.

Pagina 61

(c) Diferența pungii, definită ca punga de tupluri în care numărul de
de câte ori apare un tuplu t este egal cu numărul de apariții în R
minus de câte ori apare în S. Un tuplu care apare mai mult
ori în S decât în R nu apare în diferență.
! Exercițiul 2.3.4: Selecția poate fi efectuată și pe pungi. Oferiți un MapReduce
implementare care produce numărul adecvat de copii ale fiecărui tuplu care
trece condiția de selecție. Adică, produce perechi cheie-valoare din care
rezultatul corect al selecției poate fi obținut cu ușurință din valori.
Exercițiul 2.3.5: Operația relațională-algebră R (A, B) $\lhd\rhd_{B<C}$ S (C, D)
produce toate tuplurile (a, b, c, d) astfel încât tuplul (a, b) este în relație R, tuplul (c, d) este

în S și b <c. Dați o implementare MapReduce a acestei operațiuni, presupunând
R și S sunt mulțimi.
! Exercițiul 2.3.6: În secțiunea 2.3.5 am susținut că eliminarea duplicat este un
operație asociativă și comutativă. Dovediți acest fapt.

2.4 Extensii la MapReduce

MapReduce s-a dovedit atât de influent încât a generat o serie de extensii și
modificări. Aceste sisteme împărtășesc de obicei o serie de caracteristici cu
Sisteme MapReduce:
1. Sunt construite pe un sistem de fișiere distribuit.
2. Ei gestionează un număr foarte mare de sarcini care sunt instantanee ale unui
număr mic de funcții scrise de utilizator.
3. Acestea încorporează o metodă pentru a face față majorității eșecurilor pe care
apar în timpul executării unui job mare, fără a fi nevoie să reporniți acest lucru
job de la început.
Începem această secțiune cu o discuție despre sistemele de „flux de lucru”, care
tind MapReduce prin sprijinirea rețelelor aciclice de funcții, fiecare funcție
implementat printr-o colecție de sarcini. În timp ce multe astfel de sisteme au fost
implementat (a se vedea notele bibliografice pentru acest capitol), un
alegerea ular este Spark-ul lui UC Berkeley. De asemenea, câștigă importanță și Google
TensorFlow. Acesta din urmă, deși nu este recunoscut în general ca sistem de flux de lucru
de fapt, din cauza direcționării sale specifice aplicațiilor de învățare automată
are la bază o arhitectură de flux de lucru.
O altă familie de sisteme utilizează un model grafic de date. Se face calculul
la nodurile graficului, iar mesajele sunt trimise din orice nod către oricare adiacent
nodul. Sistemul original de acest tip a fost Google Pregel, care are propriul său sistem
propriul mod unic de a face față eșecurilor. Dar acum a devenit obișnuit
implementați o facilitate de model grafic deasupra unui sistem de flux de lucru și utilizați
sistemul de fișiere din urmă și facilitatea de gestionare a defecțiunilor.

2.4.1 Sistemele de flux de lucru

Sistemele de flux de lucru extind MapReduce din fluxul de lucru simplu în doi pași (
Funcția Hartă alimentează funcția Reduce) la orice colecție de funcții, cu un
grafic aciclic reprezentând fluxul de lucru printre funcții. Adică există un
grafic de flux aciclic ale cărui arce a → b reprezintă faptul că funcționează ieșirea a
este o intrare pentru funcția b.
Datele transmise de la o funcție la alta sunt un fișier de elemente ale uneia
tip. Dacă o funcție are o singură intrare, atunci această funcție se aplică fiecăruia
intrare independentă, la fel cum funcțiile Hărți și Reducere sunt aplicate lor
elemente de intrare individual. Ieșirea funcției este un fișier colectat din
rezultatul aplicării funcției la fiecare intrare. Dacă o funcție are intrări de la
mai mult de un fișier, elementele din fiecare dintre fișiere pot fi combinate în diverse
căi. Dar funcția în sine este aplicată combinațiilor de elemente de intrare, la
cele mai multe din fiecare fișier de intrare. Vom vedea exemple de astfel de combinații atunci când
discutăm despre implementarea uniunii și aderarea relațională în secțiunea 2.4.2.
h
f
g
eu
j
Figura 2.6: Un exemplu de flux de lucru care este mai complex decât alimentarea cu hărți
Reduce
Exemplul 2.6: O sugestie despre cum ar putea arăta un flux de lucru este în Fig. 2.6.
Acolo, cinci funcții, de la f la j, transmit datele de la stânga la dreapta în moduri specifice,
deci fluxul de date este aciclic și nicio sarcină nu trebuie să furnizeze date înainte
întreaga sa intrare este disponibilă. De exemplu, funcția h își ia intrarea de la a
fișier preexistent al sistemului de fișiere distribuit. Fiecare dintre elementele de ieșire ale h este
a trecut la funcțiile i și j, în timp ce i ia ieșirile atât ale lui f cât și ale h
intrări. Ieșirea lui j este fie stocată în sistemul de fișiere distribuit, fie este
a trecut la o aplicație care a invocat acest flux de date. ✷
În analogie cu funcțiile Map și Reduce, fiecare funcție a unui flux de lucru poate
fi executat de mai multe sarcini, cărora li se atribuie o parte din intrare
functia. Un controler principal este responsabil pentru împărțirea lucrării între
sarcinile care implementează o funcție, eventual prin hashul elementelor de intrare la
decide asupra sarcinii adecvate pentru a primi un element. Astfel, la fel ca sarcinile Hărți, fiecare sarcină
implementarea unei funcții f are un fișier de ieșire cu date destinate fiecăruia dintre

sarcini care implementează funcția (funcțiile) succesorale a lui f. Aceste fișiere sunt livrate

de către controlorul principal la momentul potrivit - după finalizarea sarcinii
munca sa.
Funcțiile unui flux de lucru și, prin urmare, sarcinile, sunt partajate cu MapReduce
atribuie o proprietate importantă: proprietatea de blocare, prin faptul că livrează numai
ieșire după finalizare. Ca rezultat, dacă o sarcină eșuează, aceasta nu a fost livrată
ieșire către oricare dintre succesorii săi din graficul de flux. [7]
Un controler master poate
prin urmare, reporniți sarcina eșuată la un alt nod de calcul, fără să vă faceți griji
că ieșirea sarcinii repornite va duplica ieșirea care a fost anterior
trecut la o altă sarcină.
Unele aplicații ale sistemelor de flux de lucru sunt în mod eficient cascade de MapReduce locuri de munca. Un exemplu ar fi îmbinarea a trei relații, unde o MapReduce job se alătură primelor două relații, iar un al doilea job MapReduce se alătură
a treia relație cu rezultatul alăturării primelor două relații. Ambele slujbe ar face-o
utilizați un algoritm precum cel din secțiunea 2.3.7.
Există un avantaj în implementarea unor astfel de cascade ca un singur flux de lucru.
De exemplu, fluxul de date între sarcini și replicarea acestuia pot fi gestionate
de către controlerul master, fără a fi nevoie să stocați fișierul temporar care este în afara
a pus o lucrare MapReduce în sistemul de fișiere distribuit. Prin localizarea sarcinilor la
calculează noduri care au o copie a intrării lor, putem evita o mare parte din
comunicație care ar fi necesară dacă stocăm rezultatul unui MapReduce
job și apoi a inițiat un al doilea job MapReduce (deși Hadoop și altele
Sistemele MapReduce încearcă, de asemenea, să localizeze sarcinile de hartă unde se află o copie a intrării lor
deja prezent).

2.4.2 Scânteie

Spark este, în centrul său, un sistem de flux de lucru. Cu toate acestea, este un avans peste
sisteme timpurii de flux de lucru în mai multe moduri, inclusiv:
1. Un mod mai eficient de a face față eșecurilor.
2. Un mod mai eficient de grupare a sarcinilor între noduri de calcul și planificare
executarea funcțiilor.
3. Integrarea caracteristicilor limbajului de programare, cum ar fi looping (care tehnologie
îl scoate din clasa fluxului de lucru aciclic al sistemelor) și funcționează
biblioteci.
Abstracția centrală de date a Spark se numește Rezilient Distribuit
Set de date sau RDD. Un RDD este un fișier de obiecte de un singur tip. Exemplul principal
a unui RDD pe care l-am văzut până acum este fișierele perechilor cheie-valoare care sunt utilizate
în sistemele MapReduce. Acestea sunt, de asemenea, fișierele care sunt trecute printre funcții
despre care am vorbit în legătură cu Fig. 2.6. RDD-urile sunt „distribuite” în
sensul că un RDD este în mod normal divizat în bucăți care pot fi ținute la

[7] După cum vom discuta în secțiunea 2.4.5, proprietatea de blocare este valabilă doar pentru fluxurile de lucru aciclice,
iar sistemele care acceptă recursivitatea nu o pot folosi pentru a gestiona eșecurile.

diferite noduri de calcul. Sunt „rezistenți” în sensul în care ne așteptăm să fie
capabil să se recupereze după pierderea oricăror sau a tuturor bucăților unui RDD. Cu toate acestea, spre deosebire de
abstracția cheie-valoare-pereche a MapReduce, nu există nicio restricție asupra tipului
a elementelor care cuprind un RDD.
Un program Spark constă dintr-o succesiune de pași, fiecare dintre aceștia fiind de obicei
aplică o anumită funcție unui RDD pentru a produce un alt RDD. Astfel de operațiuni
se numesc transformări. De asemenea, este posibil să luați date din împrejurimi
sistem de fișiere, cum ar fi HDFS, și transformați-l într-un RDD, și pentru a lua un RDD și
returnează-l la sistemul de fișiere din jur sau pentru a produce un rezultat care este transmis înapoi
la o aplicație care a numit un program Spark. Ultimele tipuri de operațiuni
se numesc acțiuni.
Nu vom încerca să enumerăm toate transformările și acțiunile disponibile care
Sunt disponibile. Nici nu ne vom fixa asupra dictărilor unei anumite programe
limba, deoarece operațiunile Spark sunt concepute pentru a fi exprimabile într-un număr
a diferitelor limbaje de programare. Cu toate acestea, iată câteva dintre cele mai frecvente
operații utilizate.
Hartă, hartă plană și filtru

Transformarea Hărții ia un parametru care este o funcție și îl aplică
funcționează la fiecare element al unui RDD, producând un alt RDD. Această operațiune
ar trebui să ne amintească de Harta MapReduce, dar nu este exact același lucru.
În primul rând, în MapReduce, o funcție Map se poate aplica numai unei perechi cheie-valoare.
În al doilea rând, în MapReduce, o funcție Map produce un set de perechi cheie-valoare și
fiecare pereche cheie-valoare este considerată un element independent al rezultatului
Funcția de hartă. În Spark, o funcție Map se poate aplica oricărui tip de obiect, dar ea
produce exact un obiect ca rezultat. Tipul obiectului rezultat poate
fi un set, dar asta nu este același lucru cu a produce multe obiecte dintr-o singură intrare
obiect. Dacă doriți să produceți un set de obiecte dintr-un singur obiect, Spark
vă oferă o altă transformare numită Flatmap, care este similară cu
Harta MapReduce, dar fără cerința ca toate tipurile să fie valoare-cheie
perechi.
Exemplul 2.7: Să presupunem că intrarea noastră RDD este un fișier de documente, ca în „cuvântul-
număr ”din Exemplul 2.1. Am putea scrie o funcție Spark Map care necesită una
documentează și produce un set de perechi, cu fiecare pereche a formei (w, 1), unde
w este unul dintre cuvintele din document. Cu toate acestea, dacă facem acest lucru, atunci rezultatul
RDD este o listă de seturi, fiecare set constând din toate cuvintele unui document,
fiecare cuvânt asociat cu numărul întreg 1. Dacă vrem să duplicăm funcția Hartă
descris în Exemplul 2.1, atunci trebuie să folosim transformarea Flatmap a lui Spark.
Această operație aplicată RDD a documentelor va produce un alt RDD,
fiecare dintre ale cărui elemente este o singură pereche (w, 1). ✷
Spark oferă, de asemenea, o operațiune similară cu o formă limitată de hartă, numită
Filtru. În loc de o funcție ca parametru, transformarea filtrului ia un
predicat care se aplică tipului de obiecte din RDD de intrare. Predicatul

---

returnează adevărat sau fals pentru fiecare obiect, iar RDD-ul de ieșire al unui filtru trans-
formarea constă numai din acele obiecte din RDD de intrare pentru care filtrul
funcția returnează adevărat.
Exemplul 2.8: Continuarea exemplului 2.7, să presupunem că vrem să evităm numărarea
cuvinte de oprire: cele mai frecvente cuvinte precum „cel” sau „și”. Am putea scrie un
funcția de filtrare care a încorporat în ea lista de cuvinte pe care vrem să le eliminăm.
Când este aplicată unei perechi (w, 1), această funcție returnează adevărat dacă și numai dacă w nu este
pe listă. Apoi putem scrie un program Spark care se aplică mai întâi Flatmap
RDD-ul documentelor, producând un RDD R_1 format dintr-o pereche (w, 1) pentru
fiecare apariție a cuvântului w în oricare dintre documente. Programul atunci
aplică filtrul de eliminare a cuvintelor de oprire la R_1, producând un alt RDD, R_2.
Ultimul RDD constă dintr-o pereche (w, 1) pentru fiecare apariție a cuvântului w în oricare
din documente, dar numai dacă w nu este un cuvânt de oprire. ✷
Reduce
În Spark, operația Reduce este o acțiune, nu o transformare. Acesta este,
operația Reduce se aplică unui RDD, dar returnează o valoare și nu alta
RDD. Reduce ia un parametru care este o funcție care ia două elemente ale
un anumit tip T și returnează un alt element de același tip T. Când
aplicat unui RDD ale cărui elemente sunt de tip T, Reduce se aplică în mod repetat
la fiecare pereche de elemente consecutive, reducându-le la un singur element. Cand
rămâne un singur element, care devine rezultatul operației Reduce.
De exemplu, dacă parametrul este funcția de adăugare și această instanță
din Reduce se aplică unui RDD ale cărui elemente sunt numere întregi, apoi rezultatul
va fi un singur întreg care este suma tuturor numerelor întregi din RDD. Atâta timp
deoarece parametrul funcției este o funcție asociativă și comutativă, cum ar fi
în plus, nu contează în ce ordine sunt elementele intrării RDD
combinate. Cu toate acestea, este posibilă și utilizarea unei funcții arbitrare, atâta timp cât
suntem mulțumiți de combinația de elemente în orice ordine.
Operațiuni de baze de date relaționale
Există o serie de operații Spark încorporate care se comportă ca relațional-
operatori de algebră pe relații care sunt reprezentate de RDD-uri. Adică gândiți-vă
a elementelor RDD-urilor ca tupluri ale unei relații. Transformarea Alătură-te
ia două RDD-uri, fiecare reprezentând una dintre relații. Tipul fiecărei RDD
trebuie să fie o pereche cheie-valoare, iar tipurile de cheie ale ambelor relații trebuie să fie aceleași.
Transformarea Unirii caută apoi două obiecte, câte unul din fiecare intrare
RDD-uri, astfel încât valorile cheie să fie aceleași, să zicem (k, x) și (k, y). Pentru fiecare
o astfel de pereche găsită, Join produce perechea cheie-valoare (k, (x, y)) și ieșirea
RDD constă din toate aceste obiecte.
Operațiunea de grupare a SQL este, de asemenea, implementată în Spark de către

formare GroupByKey. Această transformare ia ca intrare un RDD al cărui
tipul este perechi cheie-valoare. RDD de ieșire este, de asemenea, un set de perechi cheie-valoare cu

---

Pagina 66

același tip de cheie. Tipul valorii pentru ieșire este o listă de valori ale intrării
tip. GroupByKey își ordonează intrarea RDD după cheie și pentru fiecare tastă k produce
perechea (k, [v_1, v_2, ..., v_n]) astfel încât v_i sunt toate valorile asociate cu
tasta k din intrarea RDD. Observați că GroupByKey este exact operația care
este realizat în culise de MapReduce pentru a grupa ieșirea din
funcția Harta prin tastă.

2.4.3 Implementarea scânteii

Există o serie de moduri în care implementarea Spark diferă de Hadoop sau
alte implementări MapReduce. Vom discuta despre două îmbunătățiri importante
mente: evaluarea leneșă a RDD-urilor și a genealogiei pentru RDD-uri. Înainte să o facem, noi
ar trebui să menționăm un mod în care Spark este similar cu MapReduce: modul
sunt gestionate RDD-uri mari.
Amintiți-vă că atunci când aplicați Harta unui fișier mare, MapReduce împarte acel fișier
în bucăți și creează o sarcină de hartă pentru fiecare bucată sau grup de bucăți. The
bucăți și sarcinile lor sunt de obicei distribuite între mai multe calcule diferite
noduri. La fel, multe activități Reduce pot rula în paralel pe calcule diferite
noduri și fiecare dintre aceste sarcini ia o porțiune din întregul set de perechi cheie-valoare
care sunt transmise de la hartă la reducere. Spark permite, de asemenea, divizarea oricărui RDD
în bucăți, pe care le numește despărțiri. Fiecare împărțire poate fi dată unui calcul diferit
nod, iar transformarea pe acel RDD poate fi efectuată în paralel pe
fiecare dintre despicături.
Evaluare leneșă
După cum sa menționat în secțiunea 2.4.1, este obișnuit ca sistemele de flux de lucru să fie exploatate
proprietatea de blocare pentru tratarea erorilor. Pentru a face acest lucru, o funcție este aplicată unui
un singur fișier intermediar (analog unui RDD) și ieșirea acelei funcții
este pus la dispoziția consumatorilor acelei ieșiri numai după finalizarea funcției.
Cu toate acestea, Spark nu aplică de fapt transformări RDD-urilor până când nu este
trebuie să facă acest lucru, de obicei pentru că trebuie să aplice o acțiune, de exemplu, stocarea
un RDD calculat în sistemul de fișiere din jur sau returnarea unui rezultat la un fișier
cerere.
Avantajul acestei strategii de evaluare leneșă este că multe CDD nu sunt
construite dintr-o dată. Când se creează o divizare dintr-un RDD la un nod, se poate
fi utilizat imediat la același nod de calcul pentru a aplica o altă transformare
ziune. Avantajul acestui startegy este că acest RDD nu este stocat niciodată pe disc și
nu s-a transmis niciodată unui alt nod de calcul, salvând astfel ordinele de mărime
în timpul de funcționare în unele cazuri.
Exemplul 2.9: Luați în considerare situația sugerată în exemplul 2.8, unde Flat-
harta este aplicată unui singur RDD, la care vom denumi R_0. Rețineți că RDD R_0
este creat prin convertirea fișierului extern de documente într-un RDD. Ca R_0 este
un fișier mare, vom dori să-l împărțim în divizări și să operăm asupra divizărilor în
paralel.

Pagina 67

Prima transformare pe R_0 aplică Flatmap pentru a crea un set de perechi
(w, 1) pentru fiecare cuvânt. Pentru fiecare împărțire a lui R_0, o împărțire a RDD rezultat, care
am numit R_1 în Exemplul 2.8, este creat la același nod de calcul. Această despărțire
R_1 este apoi trecut la filtrul de transformare, care elimină perechile ale căror
prima componentă este un cuvânt de oprire. Când acest filtru este aplicat divizării,
rezultatul este o împărțire a RDD R_2, situat la același nod de calcul.
Cu toate acestea, nici transformările Flatmap, nici Filtrul nu apar, cu excepția cazului în care
acțiunea se aplică la R_2. De exemplu, programul Spark poate stoca R_2 în
sistemul de fișiere înconjurător sau efectuați o operație de reducere care contorizează aparițiile
a cuvintelor. Numai când programul atinge această acțiune, Spark aplică
Transformări Flatmap și Filter la R_0, executând aceste transformări la
fiecare dintre nodurile de calcul care deține o divizare de R_0, în paralel. Astfel, despărțirile
din R_1 și R_2 există doar local la nodul de calcul care le-a creat și
cu excepția cazului în care programatorul solicită în mod explicit ca acestea să fie întreținute, aceste împărțiri
sunt abandonate imediat ce sunt utilizate local. ✷

Rezistența RDD-urilor
S-ar putea întreba în mod natural ce se întâmplă în Exemplul 2.9 dacă un nod de calcul nu reușește
după crearea unei diviziuni de R_1 și înainte de a transforma acea divizare într-o divizare de
R_2. Deoarece R_1 nu este copiat în sistemul de fișiere, nu este pierdut pentru totdeauna? Spark's
înlocuitorul stocării redundante a valorilor intermediare este înregistrarea descendenței
din fiecare RDD pe care îl creează. Linia îi spune sistemului Spark cum să se recreeze
RDD sau o divizare a RDD, dacă este necesar.
Exemplul 2.10: Luând în considerare din nou situația descrisă în exemplul 2.9,
descendența pentru R_2 ar spune că este creată prin aplicarea la R_1 a particularității
Operație de filtrare care elimină cuvintele stop. La rândul său, R_1 este creat din R_0 de către
operația Flatmap care transformă cuvintele unui document în (w, 1) perechi. Și
R_0 a fost creat dintr-un anumit fișier al sistemului de fișiere din jur.
De exemplu, dacă pierdem o fracțiune de R_2, știm că o putem reconstrui
împărțirea corespunzătoare a lui R_1. Dar, din moment ce această împărțire există la același calcul
nod, probabil că am pierdut și divizarea. Dacă da, am putea să o reconstruim din
împărțirea corespunzătoare a lui R_0, care este, de asemenea, probabil pierdută dacă acest nod de calcul
a gresit. Dar știm că putem reconstitui împărțirea lui R_0 de la
sistemul de fișiere din jur, care este probabil redundant și nu se va pierde.
Astfel, Spark va găsi un alt nod de calcul, va reconstrui divizarea pierdută a lui R_0
din sistemul de fișiere de acolo și apoi aplicați transformările cunoscute necesare
pentru a reconstitui împărțirile corespunzătoare ale lui R_1 și R_2. ✷
După cum putem vedea din Exemplul 2.10, recuperarea dintr-un eșec al nodului poate fi mai mare
complex în Spark decât în MapReduce sau în sistemele de flux de lucru care stochează inter-
mediază valorile redundant. Cu toate acestea, compromiterea recuperării mai complexe
când lucrurile merg prost împotriva unei viteze mai mari atunci când lucrurile merg bine este, în general, un
buna asta. Cu cât rulează mai repede un program Spark, cu atât mai puține șanse există un nod
eșec în timpul rulării.

Pagina 68

Ar trebui să contrastăm nevoia lui Spark de a putea executa un program în față
de eșecuri cu necesitatea stocării redundante a fișierelor care se așteaptă să existe
pentru o perioadă îndelungată. Pe o perioadă lungă de timp, eșecurile sunt aproape sigure, așa că suntem foarte
probabil să pierdem bucăți dintr-un fișier dacă nu îl stocăm redundant. Dar peste scurt
perioadă - minute sau chiar ore - există șanse mari de a evita eșecurile.
Astfel, este rezonabil să fiți dispuși să plătiți mai mult atunci când există un eșec în acest sens
caz.

2.4.4 TensorFlow

TensorFlow este un sistem open-source dezvoltat inițial la Google pentru a sprijini
aplicații de învățare automată. La fel ca Spark, TensorFlow oferă un program-
interfață ming în care se scrie o secvență de pași. Programele sunt de obicei
aciclice, deși la fel ca Spark este posibil să iterați blocuri de cod.
O diferență majoră între Spark și TensorFlow este tipul de date
care este transmis între pașii programului. În locul RDD, TensorFlow
folosește tensori; un tensor este pur și simplu o matrice multidimensională.
Exemplul 2.11: O constantă, de exemplu 3.14159, este considerată ca o dimensiune 0 cu zece dimensiuni
sor. Un vector este un tensor unidimensional. De exemplu, vectorul (1, 2, 3) poate
să fie scris în tensorFlow ca [1., 2., 3.]. O matrice este un tensor bidimensional. Pentru
de exemplu, matricea
1
2
3
4
5
6
7
8
9 10 11 12
este exprimat ca [[1., 2., 3., 4.], [5., 6., 7., 8.], [9., 10., 11., 12.]].
Sunt posibile și matrici cu dimensiuni superioare. De exemplu, un 2-pe-2-pe-2
cubul lui 0 este reprezentat ca [[[0., 0.], [0., 0.]], [[0., 0.], [0., 0.]]]. ✷
Deși tensorii sunt de fapt o formă limitată de RDD, puterea tensorului
Flow provine din selecția sa de operațiuni încorporate. Operații de algebră liniară
sunt disponibile ca funcții. De exemplu, dacă doriți ca matricea C să fie produsul
din matricile A și B, puteți scrie
C = tensorflow.matmul (A, B)
Chiar mai puternice sunt abordările comune ale învățării automate care

sunt încorporate ca operațiuni. o singură afirmație în limba TensorFlow poate
face ca un model care este un tensor să fie construit din datele de antrenament, care este
reprezentat, de asemenea, ca un tensor, folosind o metodă cum ar fi coborârea în gradient. (Discutam
coborârea gradientului în secțiunile 9.4.5 și 12.3.4).

Pagina 69

2.4.5 Extensii recursive la MapReduce

Multe calcule la scară largă sunt într-adevăr recursive. Un exemplu important este
PageRank, care face obiectul capitolului 5. Acest calcul este, în
termeni multipli, calculul punctului fix al unei multiplicări matrice-vector.
Este calculat sub sistemele MapReduce prin aplicația iterată a
algoritm de multiplicare matricială-vector descris în secțiunea 2.3.1 sau cu un alt
strategie complexă pe care o vom introduce în secțiunea 5.2. Tipurile de iterație
continuă calic pentru un număr necunoscut de pași, fiecare pas fiind un MapReduce
până când rezultatele a două iterații consecutive sunt suficient de apropiate încât
credem că s-a produs convergența. Un al doilea exemplu important de recurență
algoritmul sive pe date masive este descendența în gradient, pe care tocmai am menționat-o în
conexiune cu TensorFlow.
Recursiunile prezintă o problemă pentru recuperarea eșecului. Sarcini recursive inerent
lipsesc proprietatea de blocare necesară pentru repornirea independentă a sarcinilor eșuate. Aceasta
este imposibil pentru o colecție de sarcini recursive reciproc, fiecare dintre acestea având un
ieșire care este introdusă la cel puțin unele dintre celelalte sarcini, pentru a produce numai ieșire
la sfârșitul sarcinii. Dacă toți ar urma această politică, nicio sarcină nu ar primi vreodată
orice intrare și nimic nu a putut fi realizat. Ca urmare, un anumit mecanism
altele decât repornirea simplă a sarcinilor eșuate trebuie implementate într-un sistem care
gestionează fluxurile de lucru recursive (grafice de flux care nu sunt aciclice). Vom începe
prin studierea unui exemplu de recursivitate implementat ca flux de lucru și apoi
discutați abordări pentru rezolvarea eșecurilor sarcinilor.
Exemplul 2.12: Să presupunem că avem un grafic direcționat ale cărui arce sunt reprezentate
prin relația E (X, Y), ceea ce înseamnă că există un arc de la nodul X la nodul Y.
Dorim să calculăm relația de căi P (X, Y), ceea ce înseamnă că există o cale
de lungime 1 sau mai mult de la nodul X la nodul Y. Adică, P este închiderea tranzitivă
din E. Un algoritm recursiv simplu pentru a face acest lucru este:
1. Începeți cu P (X, Y) = E (X, Y).
2. În timp ce apar modificări ale relației P, adăugați la P toate tuplurile din
$\pi_{X,Y}$ (P (X, Z) ◁▷ P (Z, Y))
Adică, găsiți perechi de noduri X și Y astfel încât pentru un anumit nod Z să existe
cunoscut a fi o cale de la X la Z și, de asemenea, o cale cunoscută de la Z la Y.
Figura 2.7 sugerează modul în care am putea organiza sarcini recursive pentru a realiza acest lucru
calcul. Există două tipuri de sarcini: Asociere sarcini și Dup-elimin sarcini.
Există n Asociați sarcini, pentru unele n, și fiecare corespunde unei bucăți de hash
funcția h. Un tuplu de cale P (a, b), atunci când este descoperit, devine intrare în două
Alăturați-vă sarcinilor: cele numerotate h (a) și h (b). Sarcina celei de-a doua sarcini Join, când
primește tuplul de intrare P (a, b), este de a găsi anumite alte tupluri văzute anterior
(și stocate local prin acea sarcină).
1. Păstrați local P (a, b).

Pagina 70

A te alatura
sarcină
0
A te alatura
sarcină
1
A te alatura
sarcină
eu
sarcină
0
Dup − elimin
sarcină
1
Dup − elimin
sarcină
j
Dup − elimin
.
.
.

.
.
.
.
.
.
.
.
.
P (a, b) dacă
h (a) = i sau
h (b) = i
P (c, d) dacă
g (c, d) = j
P (c, d) dacă nu este niciodată
văzut înainte
Pentru a vă alătura sarcinii h (d)
Pentru a vă alătura sarcinii h (c)

Figura 2.7: Implementarea închiderii tranzitive printr-o colecție de recursive sarcini

2. Dacă h (a) = i atunci căutați tuplele P (x, a) și produceți tuplul de ieșire P (x, b).
3. Dacă h (b) = i, atunci căutați tuplele P (b, y) și produceți tuplul de ieșire P (a, y).

Rețineți că, în cazuri rare, avem h (a) = h (b), deci atât (2) cât și (3) sunt executate. Dar, în general, doar unul dintre acestea trebuie executat pentru un anumit tupl. Există, de asemenea, m sarcini Dup-elimin și fiecare corespunde unei bucăți de a funcția hash g care ia două argumente. Dacă P (c, d) este o ieșire a unor Join task, apoi este trimis la Dup-elimin task j = g (c, d). La primirea acestui tuplu, Jth Dup-elimin task verifică dacă nu a mai primit acest tupl înainte, de la job este eliminarea duplicat. Dacă a fost primit anterior, tuplul este ignorat. Dar dacă asta tuplul este nou, este stocat local și trimis la două sarcini Join, cele numerotate h (c) și h (d).

Fiecare activitate Join are m fișiere de ieșire - unul pentru fiecare activitate Dup-elimin - și fiecare Sarcina Dup-elimin are n fișiere de ieșire - câte una pentru fiecare sarcină Join. Aceste fișiere pot fi distribuite conform oricăreia dintre mai multe strategii. Inițial, tuplurile E (a, b) reprezentarea arcurilor grafului sunt distribuite sarcinilor Dup-elimin, cu E (a, b) fiind trimis ca P (a, b) la sarcina Dup-elimin g (a, b). Controlerul master

---

așteaptă până când fiecare sarcină de Alăturare și-a procesat întreaga intrare pentru o rundă. Apoi, toate fișierele de ieșire sunt distribuite sarcinilor Dup-elimin, care își creează propriile ieșire. Respectiva ieșire este distribuită sarcinilor Unire și devine intrarea lor pentru runda următoare. ✷

În Exemplul 2.12 nu este esențial să aveți două tipuri de sarcini. Mai degrabă, Alătură-te sarcinile ar putea elimina duplicatele pe măsură ce sunt primite, deoarece acestea trebuie stocate intrările lor primite anterior oricum. Cu toate acestea, acest aranjament are un avantaj atunci când trebuie să ne recuperăm după un eșec al sarcinii. Dacă fiecare sarcină stochează toate fișierele de ieșire pe care le-a creat vreodată și plasăm sarcinile Join pe diferite rafturi din sarcinile Dup-elimin, atunci putem face față oricărui singur nod de calcul sau eșecul unui singur rack. Adică, o sarcină Join care trebuie repornită poate obține toate intrările generate anterior de care are nevoie din sarcinile Dup-elimin și vice versa.

În cazul particular al calculării închiderii tranzitive, nu este necesar împiedica o sarcină repornită să genereze rezultate generate de sarcina inițială ated anterior. În calculul închiderii tranzitive, redescoperirea o cale nu influențează eventualul răspuns. Cu toate acestea, multe calcule nu poate tolera o situație în care atât versiunea originală, cât și cea repornită a unui task trece aceeași ieșire la o altă sarcină. De exemplu, dacă pasul final al fișierului calculul a fost o agregare, să spunem un număr al numărului de noduri atinse de fiecare nod din grafic, atunci am primi un răspuns greșit dacă am număra o cale de două ori.

Există cel puțin trei abordări diferite care au fost utilizate pentru a face față cu eșecuri în timpul executării unui program recursiv.

1. Iterated MapReduce: Scrieți recursiunea ca executare repetată a unei Map-Reduceți jobul sau o secvență de joburi MapReduce. Ne putem baza apoi pe mecanismul de eșec al implementării MapReduce pentru a gestiona eșecurile la orice pas. Primul exemplu al unui astfel de sistem a fost HaLoop (vezi note bibliografice pentru acest capitol).
2. Abordarea Spark: Limbajul Spark include de fapt iterativ declarații, cum ar fi buclele for care permit implementarea recursivelor. Aici, managementul eșecurilor este implementat folosind evaluarea leneșă și mecanisme genealogice ale Spark. În plus, programatorul Spark are opțiuni pentru stocarea stărilor intermediare ale recursiunii.

3. Sisteme sincrone în vrac: Aceste sisteme utilizează un model bazat pe graf
calcul pe care îl vom descrie în continuare. De obicei folosesc altul
abordarea rezilienței: puncte de control periodice.

2.4.6 Sisteme sincrone în vrac

O altă abordare a implementării algoritmilor recursivi pe un cluster de calcul
este reprezentat de sistemul Google Pregel, care a fost primul exemplu de

---

Pagina 72

sistem bazat pe grafice, sincronizat în bloc, pentru procesarea unor cantități masive de date.
Un astfel de sistem își vede datele ca pe un grafic. Fiecare nod al graficului corespunde
aproximativ la o sarcină (deși, în practică, multe noduri ale unui grafic mare ar fi
grupate într-o singură sarcină, la fel ca în Unirea sarcinilor din Exemplul 2.12). Fiecare grafic
nodul generează mesaje de ieșire care sunt destinate altor noduri ale graficului,
iar fiecare nod grafic procesează intrările pe care le primește de la alte noduri.
Exemplul 2.13: Să presupunem că datele noastre sunt o colecție de arce ponderate ale unui grafic,
și vrem să găsim, pentru fiecare nod al graficului, lungimea celei mai scurte căi
către fiecare dintre celelalte noduri. Pe măsură ce algoritmul se execută, fiecare nod a va stoca
un set de perechi (b, w), unde w este lungimea celei mai scurte căi de la nodul a la
nodul b care este cunoscut în prezent.
Inițial, fiecare nod grafic a stochează setul de perechi (b, w) astfel încât acolo
este un arc de la a la b de greutate w. Aceste fapte sunt trimise către toate celelalte noduri, ca
tripluri (a, b, w), cu semnificația intenționată că nodul a știe despre o cale de
lungimea w până la nodul b. [8] Când nodul a primește un triplu (c, d, w), acesta trebuie să decidă
dacă acest fapt implică o cale mai scurtă decât o știe deja despre ea însăși
a nodul d. Nodul a caută distanța actuală până la c; adică găsește perechea
(c, v) stocate local, dacă există. De asemenea, găsește perechea (d, u) dacă există una.
Dacă w + v <u, atunci perechea (d, u) se înlocuiește cu (d, w + v) și dacă a existat
fără pereche (d, u), atunci perechea (d, w + v) este stocată la nodul a. De asemenea, celălalt
nodurilor li se trimite mesajul (a, d, w + v) în oricare dintre aceste două cazuri. ✷
Calculele în Pregel sunt organizate în superstepte. Într-un singur pas, toate
mesajele care au fost primite de oricare dintre noduri la superpasul anterior
(sau inițial, dacă este primul superstep) sunt procesate, apoi toate mesajele
generate de acele noduri sunt trimise la destinație. Este acest ambalaj de
multe mesaje într-unul care dă acestei abordări numele de „bloc-sincron”.
Există un avantaj foarte important în gruparea mesajelor în acest fel.
Comunicarea pe o rețea necesită, în general, o cantitate mare de cheltuieli generale
pentru a trimite orice mesaj, oricât de scurt. Să presupunem că în Exemplul 2.13 am trimis
un singur nou fapt pe distanța cea mai scurtă către nodul relevant de fiecare dată când era
descoperit. Numărul de mesaje trimise ar fi enorm dacă graficul ar fi
mare și nu ar fi realist să implementăm un astfel de algoritm. In orice caz,
într-un sistem sincronizat în bloc, o sarcină care are responsabilitatea gestionării
multe noduri ale graficului pot grupa împreună toate mesajele trimise de
nodurile sale către oricare dintre noduri fiind gestionate de o altă sarcină. Această alegere
salvează de obicei ordine de mărime în timpul necesar pentru a trimite toate cele necesare
mesaje.
Gestionarea eșecurilor în Pregel
În caz de eșec al nodului de calcul, nu există nicio încercare de a reporni sarcinile eșuate
la acel nod de calcul. Mai degrabă, Pregel își verifică întregul calcul după aceea

[8] Acest algoritm utilizează mult prea multă comunicare, dar va servi drept exemplu simplu
a modelului de calcul Pregel.

Pagina 73

unele dintre supersteps. Un punct de control constă în realizarea unei copii a întregului
starea fiecărei sarcini, deci poate fi repornită din acel punct, dacă este necesar. Dacă există
nodul de calcul nu reușește, întreaga lucrare este repornită de la cel mai recent punct de control.
Deși această strategie de recuperare provoacă multe sarcini care nu au eșuat
reface munca lor, este satisfăcătoare în multe situații. Reamintim că motivul
Sistemele MapReduce acceptă repornirea numai a sarcinilor eșuate pe care le dorim
asigurarea că timpul așteptat pentru finalizarea întregului loc de muncă în fața eșecului
Ures nu este mult mai mare decât timpul pentru executarea lucrării fără eșecuri.
Orice sistem de gestionare a defecțiunilor va avea acea proprietate atâta timp cât
recuperarea după un eșec este mult mai mică decât timpul mediu dintre eșecuri.
Astfel, este necesar doar ca punctele de verificare Pregel să își calculeze calculele după o

ber de supersteps astfel încât probabilitatea unei eșecuri în timpul acelui număr de supersteps este scăzut.

2.4.7 Exerciții pentru secțiunea 2.4

! Exercițiul 2.4.1: Să presupunem că o lucrare constă din n sarcini, fiecare dintre ele necesită timp t secunde. Astfel, dacă nu există eșecuri, suma peste toate nodurile de calcul ale timpul necesar pentru executarea sarcinilor la acel nod este nt. Să presupunem, de asemenea, că probabilitatea a unei sarcini care eșuează este p pe lucrare pe secundă, iar atunci când o sarcină eșuează, cheltuielile generale pentru gestionarea repornirii este de așa natură încât adaugă 10t secunde la execuția totală timpul slujbei. Care este timpul total de execuție așteptat al lucrării?

! Exercițiul 2.4.2: Să presupunem că un loc de muncă Pregel are o probabilitate p de eșec în timpul orice superstep. Să presupunem, de asemenea, că timpul de execuție (însumat peste tot calculează noduri) de a lua un punct de control este de c ori timpul necesar pentru executarea unui superstep. Pentru a minimiza timpul de execuție așteptat al lucrării, câte superstep-uri ar trebui să treacă între punctele de control?

2.5 Modelul de comunicare-cost

În această secțiune vom introduce un model pentru măsurarea calității algoritmilor implementat pe un cluster de calcul de tipul discutat până acum în acest capitol. Presupunem că calculul este descris de un flux de lucru aciclic, după cum sa discutat în secțiunea 2.4.1. Pentru multe aplicații, blocajul mută datele printre sarcini, cum ar fi transportul rezultatelor sarcinilor de hartă la reducerea corespunzătoare sarcini. De exemplu, explorăm calculul îmbinărilor multidirecționale ca simple MapReduce joburile și vedem că, în unele situații, această abordare este mai mult eficient decât cascada simplă a îmbinărilor bidirecționale.

2.5.1 Costul comunicării pentru rețelele de activități

Imaginați-vă că un algoritm este implementat de o rețea aciclică de sarcini. Aceste sarcinile ar putea fi Hărți sarcini hrănire Reduceți sarcini, ca într-un MapReduce standard algoritm sau ar putea fi mai multe joburi MapReduce în cascadă sau mai general

---

structura fluxului de lucru, cum ar fi o colecție de sarcini care implementează fiecare fluxul de lucru din Fig. 2.6. 9 Costul comunicării unei sarcini este dimensiunea intrării la sarcină. Această dimensiune poate fi măsurată în octeți. Cu toate acestea, din moment ce vom fi folosind ca exemple operațiile bazei de date relaționale, vom folosi adesea numărul de tupluri ca măsură de mărime.

Costul comunicării unui algoritm este suma comunicării costul tuturor sarcinilor de implementare a algoritmului respectiv. Ne vom concentra pe comunicare costul de nicare ca modalitate de măsurare a eficienței unui algoritm. În special, nu luăm în considerare timpul necesar fiecărei sarcini pentru a fi executat atunci când este temporizarea timpului de rulare al unui algoritm. Deși există excepții, unde timpul de execuție al sarcinilor domină, aceste excepții sunt rare în practică. Noi poate explica și justifica importanța costului comunicării după cum urmează.

- Algoritmul executat de fiecare sarcină tinde să fie foarte simplu, adesea liniar în mărimea intrării sale.
- Viteza tipică de interconectare pentru un cluster de calcul este de 1 gigabit per al doilea. Acest lucru poate părea foarte mult, dar este lent în comparație cu viteza la care un procesor execută instrucțiuni. Mai mult, în multe grupuri arhitecturi, există concurență pentru interconectare atunci când mai multe nodurile pute trebuie să comunice în același timp. Ca urmare, nodul de calcul poate face multă muncă pe un element de intrare primit în timpul necesar livrării acelui element.
- Chiar dacă o sarcină se execută la un nod de calcul care are o copie a bucății pe care operează sarcina, acea bucată va fi stocată în mod normal pe disc, iar timpul necesar pentru a muta datele în memoria principală poate depăși timpul necesar pentru a opera datele odată ce acestea sunt disponibile în memorie.

Presupunând că costul comunicării este costul dominant, s-ar putea să ne întrebăm de ce numărăm doar dimensiunea de intrare, și nu dimensiunea de ieșire. Răspunsul la această întrebare implică două puncte:

1. Dacă ieșirea unei sarcini τ este introdusă într-o altă sarcină, atunci dimensiunea lui τ ieșirea va fi luată în considerare la măsurarea dimensiunii de intrare pentru sarcină. Astfel, nu există niciun motiv pentru a număra dimensiunea oricărei ieșiri, cu excepția pentru acele sarcini a căror ieșire formează rezultatul întregului algoritm.
2. Dar, în practică, ieșirea algoritmului este rareori mare în comparație cu de intrare sau datele intermediare produse de algoritm. Motivul este că rezultatele masive nu pot fi utilizate decât dacă sunt rezumate sau

agregate într-un fel. De exemplu, deși am vorbit în Exemplul 2.12
de a calcula întreaga închidere tranzitivă a unui grafic, în practică am face-o
doresc ceva mult mai simplu, cum ar fi numărul numărului de noduri
9 Amintiți-vă că această figură reprezenta funcții, nu sarcini. Ca rețea de sarcini, acolo
ar fi, de exemplu, multe sarcini de implementare a funcției f, fiecare dintre care alimentează date către fiecare
a sarcinilor pentru funcția g și fiecare dintre sarcinile pentru funcția i.

---

accesibil din fiecare nod sau setul de noduri accesibil dintr-un singur
nodul.

Exemplul 2.14: Să evaluăm costul comunicării pentru algoritmul de asociere
din secțiunea 2.3.7. Să presupunem că alăturăm R (A, B) ⋈ S (B, C) și dimensiunile
ale relațiilor R și S sunt respectiv r și s. Fiecare bucată din fișierele care conțin
R și S sunt introduse într-o sarcină de hartă, deci suma costurilor de comunicare pentru toți
sarcinile Hartă sunt r + s. Rețineți că într-o execuție tipică, sarcinile Harta vor fi
fiecare va fi executat la un nod de calcul care deține o copie a bucății la care se află
se aplică. Astfel, nu este necesară nici o comunicare internode pentru sarcinile Hartă, dar
totuși trebuie să își citească datele de pe disc. Deoarece toate sarcinile Hărții sunt de a crea un
transformarea simplă a fiecărui tupl de intrare într-o pereche cheie-valoare, ne așteptăm la asta
costul de calcul va fi mic în comparație cu costul comunicării,
indiferent dacă intrarea este locală pentru sarcină sau trebuie transportată către
nodul său de calcul.

Suma rezultatelor sarcinilor Hărții este aproximativ la fel de mare ca și
pune. Fiecare pereche cheie-valoare de ieșire este trimisă exact la o sarcină de reducere și este
este puțin probabil ca această sarcină de reducere să se execute la același nod de calcul. Acolo-
prin urmare, comunicarea de la sarcini de hartă pentru a reduce sarcinile va fi probabil peste tot
interconectarea clusterului, mai degrabă decât memoria-pe-disc. Această comunicare
este O (r + s), deci costul de comunicare al algoritmului de îmbinare este O (r + s).

Activitățile Reduce execută reductorul (aplicarea funcției Reduce
la o cheie și lista de valori asociate) pentru una sau mai multe valori ale atributului B.
Fiecare reductor ia intrările pe care le primește și le împarte între tupluri care
au venit din R și cele care au venit din S. Fiecare tuplu din R se împerechează cu fiecare
tuplu de la S pentru a produce o singură ieșire. Dimensiunea de ieșire pentru îmbinare poate fi fie
mai mare sau mai mic decât r + s, în funcție de cât de probabil este un anumit R-tuplu
se uneste cu un S-tupl dat. De exemplu, dacă există multe valori B diferite,
ne-am aștepta ca ieșirea să fie mică, în timp ce dacă există puține valori B, o mare
rezultatul este probabil.

Dacă ieșirea este mare, atunci costul de calcul al generării tuturor ieșirilor
de la un reductor ar putea fi mult mai mare decât O (r + s). Cu toate acestea, ne vom baza pe a noastră
presupunem că dacă ieșirea îmbinării este mare, atunci probabil că există unele
agregarea făcându-se pentru a reduce dimensiunea ieșirii. Va fi necesar să
comunicați rezultatul asocierii cu o altă colecție de sarcini pe care le îndepliniți
această agregare și, prin urmare, costul comunicării va fi cel puțin proporțional
la calculul necesar pentru a produce ieșirea îmbinării. ✷

2.5.2 Ora ceasului de perete

În timp ce costul comunicării influențează adesea alegerea algoritmului nostru de utilizat
un mediu de calcul cluster, trebuie să fim conștienți de importanța
timpul ceasului de perete, timpul necesar unui algoritm paralel pentru a finaliza. Folosind neglijent
raționament, s-ar putea minimiza costul total al comunicării prin atribuirea tuturor
lucrați la o singură sarcină și, prin urmare, reduceți comunicarea totală. Însă

---

timpul ceasului de perete al unui astfel de algoritm ar fi destul de mare. Algoritmii noi
sugerați sau ați sugerat până acum să aveți proprietatea că opera este împărțită
corect între sarcini. Prin urmare, timpul ceasului de perete ar fi aproximativ
cât de mic ar putea fi, având în vedere numărul de noduri de calcul disponibile.

2.5.3 Îmbinări multiple

Pentru a vedea cum analiza costului comunicării ne poate ajuta să alegem un algoritm
în mediul de calcul cluster, vom examina cu atenție cazul unui
multiway join. Există o teorie generală în care noi:

1. Selectați anumite atribute ale relațiilor implicate în îmbinarea naturală a
trei sau mai multe relații pentru a avea valorile lor hash, fiecare la un anumit număr
de găleți.

2. Selectați numărul de găleți pentru fiecare dintre aceste atribute, în funcție de
constrângerea că produsul numărului de găleți pentru fiecare atribut
este k, numărul reductoarelor care vor fi utilizate.
3. Identificați fiecare dintre k reductoare cu un vector de numere de cupe. Aceste
vectorii au o componentă pentru fiecare dintre atributele selectate la pasul (1).
4. Trimiteți tupluri ale fiecărei relații la toate acele reductoare unde ar putea găsi tupluri
a se alătura cu. Adică, tuplul dat t va avea valori pentru unele dintre
atributele selectate la pasul (1), astfel încât să putem aplica funcția (funcțiile) hash
acele valori pentru a determina anumite componente ale vectorului care identifică
reductoarele. Alte componente ale vectorului sunt necunoscute, deci nu trebuie
să fie trimise reductorilor pentru toți vectorii care au orice valoare în aceste necunoscute
componente.
Câteva exemple ale acestei tehnici generale apar în exerciții. Aici,
ne vom uita doar la îmbinarea R (A, B) ◁▷ S (B, C) ◁▷ T (C, D) ca exemplu.
Să presupunem că relațiile R, S și T au dimensiunile r, s, respectiv t, și
pentru simplitate, să presupunem că p este probabilitatea ca
1. Un R-tuplu și și un S-tuplu sunt de acord cu B și, de asemenea, cu privire la probabilitatea ca
2. Un tuplu S și un tuplu T sunt de acord asupra lui C.
Dacă ne alăturăm mai întâi lui R și S, folosind algoritmul MapReduce din secțiunea 2.3.7,
atunci costul comunicării este O (r + s) și dimensiunea îmbinării intermediare
R ◁▷ S este prs. Când alăturăm acest rezultat cu T, comunicarea acestuia
a doua lucrare MapReduce este O (t + prs). Astfel, întregul cost de comunicare al
algoritmul format din două îmbinări bidirecționale este O (r + s + t + prs). Dacă noi în schimb
unim mai întâi S și T, apoi unim R cu rezultatul, obținem un alt algoritm
al cărui cost de comunicare este O (r + s + t + pst).
O a treia modalitate de a lua acest join este de a utiliza o singură lucrare MapReduce care se alătură
cele trei relații deodată. Să presupunem că intenționăm să folosim reductori k pentru aceasta

---

loc de munca. Alegeți numerele b și c reprezentând numărul de găleți în care noi
trebuie să aibă valori hash B, respectiv C. Fie h o funcție hash care trimite
Valorile B în b bucket și să fie g o altă funcție hash care trimite valori C
în găleți c. Avem nevoie de bc = k; adică fiecărui reductor îi corespunde
o pereche de găleți, una pentru valoarea B și una pentru valoarea C. Reductorul
corespunzător perechii de găleată (i, j) este responsabil pentru unirea tuplelor R (u, v),
S (v, w) și T (w, x) ori de câte ori h (v) = i și g (w) = j.
Ca rezultat, sarcinile de hartă care trimit tupluri de R, S și T către reductoare
care au nevoie de ele trebuie să trimită tupluri R și T la mai multe reductoare. Pentru un
S-tuplu S (v, w), cunoaștem valorile B și C, deci putem trimite acest tuplu numai către
reductorul pentru (h (v), g (w)). Cu toate acestea, luați în considerare un R-tuplu R (u, v). Noi stim
trebuie doar să meargă la reductoare care corespund la (h (v), y), pentru unele y. Dar
nu stim noi; valoarea lui C ar putea fi orice din câte știm. Prin urmare,
trebuie să trimitem R (u, v) către reductoare c, deoarece y ar putea fi oricare dintre c gălețile pentru
C-valori. În mod similar, trebuie să trimitem T-tuplul T (w, x) la fiecare dintre reductoare
(z, g (w)) pentru orice z. Există b astfel de reductoare.
0
1
2
3
2
1
0
g (TC) = 1
g (C) =
h (SB) = 2 și g (SC) = 1
h (B) =
h (RB) = 2
3
Figura 2.8: Șaisprezece reductoare realizează împreună o îmbinare cu 3 căi
Exemplul 2.15: Să presupunem că b = c = 4, deci k = 16. Cele șaisprezece reductoare
poate fi considerat așezat într-un dreptunghi, așa cum sugerează Fig. 2.8. Acolo,
vedem un S-tuplu ipotetic S (v, w) pentru care h (v) = 2 și g (w) = 1. Aceasta
tuplul este trimis prin sarcina sa de hartă numai reductorului pentru cheie (2, 1). Vedem și noi
un R-tuplu R (u, v). Deoarece h (v) = 2, acest tupl este trimis la toate reductoarele (2, y), pt
y = 1, 2, 3, 4. În cele din urmă, vedem un T-tuplu T (w, x). Deoarece g (w) = 1, acest tuplu este
trimis la toate reductoarele (z, 1) pentru z = 1, 2, 3, 4. Observați că aceste trei tupluri se unesc,
și se întâlnesc exact la un reductor, reductor pentru cheie (2, 1). ✷

Acum, să presupunem că dimensiunile lui R, S și T sunt diferite; amintim că folosim r, s, respectiv t, pentru acele dimensiuni. Dacă hash-B valori pentru b găleți și

---

Pagina 78

Costul de calcul al îmbinării cu 3 căi

Fiecare dintre reductoare trebuie să calculeze îmbinarea părților din cele trei relații, și este rezonabil să ne întrebăm dacă această înscriere poate fi luată la timp cât mai mic posibil: liniar în suma dimensiunilor de intrare și ieșire pentru acea sarcină Reduce. În timp ce îmbinările mai complexe s-ar putea să nu fie calculabile în timp liniar, îmbinarea exemplului nostru de rulare poate fi executată la fiecare Reduceți procesul eficient. Mai întâi, creați un index pe RB, pentru a organiza R-tuplurile primite. La fel, creați un index pe TC pentru T-tupluri.

Apoi, luați în considerare fiecare S-tuplu primit, S (v, w). Utilizați indexul de pe RB pentru găsiți toate R-tupluri cu RB = v și utilizați indexul pe TC pentru a găsi toate T-tupluri cu TC = w.

Valori C până la c cupe, unde bc = k, atunci costul total de comunicare pentru mutarea tuplelor la reductoarele corespunzătoare este suma:

1. s pentru a trimite fiecare tuplu S (v, w) la reductor (h (v), g (w)).
2. cr pentru a trimite fiecare tuplu R (u, v) la reductorii c (h (v), y) pentru fiecare dintre c valorile posibile ale lui y.
3. bt pentru a trimite fiecare tuplu T (w, x) la reductorii b (z, g (w)) pentru fiecare dintre b valorile posibile ale z.

Există, de asemenea, un cost r + s + t pentru a face ca fiecare tuplu din fiecare relație să fie introdus una dintre sarcinile Hărții. Acest cost este fix, independent de b, c și k.

Trebuie să selectăm b și c, sub rezerva constrângerii bc = k, pentru a minimiza s + cr + bt. Vom folosi tehnica multiplicatorilor Lagrangean pentru a găsi locul unde funcția s + cr + bt - λ(bc - k) are derivatele sale cu respect la b și c egal cu 0. Adică trebuie să rezolvăm ecuațiile r - λb= 0 și t - λc= 0. Deoarece r = λb și t = λc, putem înmulți laturile corespunzătoare ale lui aceste ecuații pentru a obține rt = λ^2 bc. Deoarece bc = k, obținem rt = λ^2 k sau $\lambda = \sqrt{rt} / k$.

Astfel, costul minim de comunicare este obținut când c = t / $\lambda = \sqrt{kt} / r$, și b = r / $\lambda = \sqrt{kr} / t$.

Dacă substituim aceste valori în formula s + cr + bt, obținem s + $2\sqrt{krt}$.

Acesta este costul comunicării pentru sarcinile de reducere, la care trebuie să adăugăm costul s + r + t pentru costul comunicării sarcinilor Hărții. Totalul costul comunicării este astfel r + 2s + t + $2\sqrt{krt}$. În majoritatea circumstanțelor, putem neglijează r + t, deoarece va fi mai mic de $2\sqrt{krt}$, de obicei cu un factor de O ($\sqrt{k}$).

Exemplul 2.16: Să vedem în ce circumstanțe îmbinarea cu 3 căi are o valoare mai mică costul comunicării decât cascada a două îmbinări bidirecționale. Pentru a rezolva problemele simplu, să presupunem că R, S și T sunt toate aceeași relație R, care reprezintă relația „prieteni” într-o rețea socială precum Facebook. Sunt

---

Pagina 79

aproximativ un miliard de abonați pe Facebook, cu o medie de 300 de prieteni fiecare, deci relația R are r = 3×10^{11} tupluri. Să presupunem că vrem să calculăm R ◁▷ R ◁▷ R, poate ca parte a unui calcul pentru a găsi numărul de prieteni ai prietenilor prietenii pe care îi are fiecare abonat sau poate doar persoana cu cel mai mare număr de prieteni de prieteni de prieteni. [10] Costul comunicării îmbinării cu 3 căi a lui R cu sine este 4r + $2r\sqrt{k}$; 3r reprezintă costul sarcinilor Hărții și r + $2\sqrt{kr}$ [2] este costul sarcinilor de reducere. Deoarece presupunem r = 3×10^{11}, acest cost este $1{,}2 \times 10^{12} + 6 \times 10^{11} \sqrt{k}$.

Acum ia în considerare costul de comunicare al alăturării lui R cu el însuși și apoi alăturând din nou rezultatul cu R. Sarcinile Hărți și Reducere pentru prima alăturare fiecare au un cost de 2r, deci primul join are doar un cost de comunicare 4r = $1{,}2 \times 10^{12}$.

Dar dimensiunea lui R ◁▷ R este mare. Nu putem spune exact cât de mari sunt, de la prieteni tind să cadă în clici și, prin urmare, o persoană cu 300 de prieteni va avea mulți mai puțin decât numărul maxim posibil de prieteni de prieteni, care este de 90.000.

Să estimăm conservator că dimensiunea lui R ◁▷ R nu este de 300r, ci doar 30r, sau 9×10^{12}. Costul comunicării pentru a doua îmbinare a (R ◁▷ R) ◁▷ R este astfel $1{,}8 \times 10^{13} + 6 \times 10^{11}$. Prin urmare, costul total al celor două îmbinări este $1.2 \times 10^{12} + 1.8 \times 10^{13} + 6 \times 10^{11} = 1.98 \times 10^{13}$.

Trebuie să ne întrebăm dacă costul îmbinării cu 3 căi, care este

$1,2 \times 10^{12} + 6 \times 10^{11} \sqrt{k}$
este mai mic de $1,98 \times 10^{13}$. Așa este, cu condiția ca $6 \times 10^{11} \sqrt{k} < 1,86 \times 10^{13}$ sau
$\sqrt{k} < 31$. Adică, îmbinarea cu 3 căi va fi preferabilă, cu condiția să nu mai folosim
decât $31^2 = 961$ reductoare. ✷

2.5.4 Exerciții pentru secțiunea 2.5

Exercițiul 2.5.1: Care este costul comunicării pentru fiecare dintre următoarele
algoritmi, în funcție de mărimea relațiilor, matricilor sau vectorilor la
care sunt aplicate?
(a) Algoritmul de multiplicare matrice-vector din secțiunea 2.3.2.
(b) Algoritmul uniunii din secțiunea 2.3.6.
(c) Algoritmul de agregare din secțiunea 2.3.8.
(d) Algoritmul de multiplicare a matricii din secțiunea 2.3.10.
! Exercițiul 2.5.2: Să presupunem că relațiile R, S și T au dimensiunile r, s și t, respectiv
și vrem să luăm îmbinarea cu 3 căi R (A, B) ⋈ S (B, C) ⋈ T (A, C),
folosind k reductoare. Vom hash valori ale atributelor A, B și C la a, b și c
găleți, respectiv, unde abc = k. Fiecare reductor este asociat cu un vector

[10] Această persoană, sau mai general, persoanele cu cercuri extinse de prieteni, sunt bune
oamenii să folosească pentru a începe o campanie de marketing oferindu-le mostre gratuite.

Pagina 80

Star Se alătură

O structură comună pentru extragerea datelor de date comerciale este unirea în stea.
De exemplu, un lanț de magazine precum Walmart păstrează un tabel de informații a cărui
ples fiecare reprezintă o singură vânzare. Această relație arată ca F (A_1 , A_2 , ...),
unde fiecare atribut A_i este o cheie care reprezintă una dintre companiile importante
elemente ale vânzării, cum ar fi cumpărătorul, articolul achiziționat, magazinul
sucursală sau data. Pentru fiecare atribut cheie există un tabel de dimensiuni
oferind informații despre participant. De exemplu, dimensiunea ta-
D (A_1 , B_{11} , B_{12} , ...) ar putea reprezenta cumpărătorii. A_1 este cumpărătorul
ID, cheia acestei relații. B_{1i} s“ar putea da numele cumpărătorului,
adresa, telefonul și așa mai departe. De obicei, tabelul de fapt este mult mai mare decât
tabelele de dimensiuni. De exemplu, ar putea exista un tabel de fapte de un miliard
tupluri și zece tabele de dimensiuni a câte un milion de tupluri fiecare.
Analiștii extrag aceste date întrebând întrebări analitice care se alătură de obicei
tabelul cu mai multe tabele de dimensiuni (o „asociere stea”) și apoi
agregați rezultatul într-o formă utilă. De exemplu, ar putea întreba un analist

,,Dă-mi un tabel de vânzări de pantaloni, defalcat pe regiuni și culori, pentru
în fiecare lună a anului 2016. ” În cadrul modelului de comunicare-cost al acestei secțiuni,
aderarea la tabelul de date și tabelele de dimensiuni printr-o îmbinare multipluă este aproape
sigur că sunt mai eficienți decât unirea relațiilor în perechi. De fapt, este
poate avea sens să stocați tabelul de date peste oricâte noduri de calcul
sunt disponibile și reproduceți tabelele de dimensiuni permanent în exact
în același mod în care le-am reproduce dacă ar trebui să luăm unirea
tabelul de fapte și toate tabelele de dimensiuni. În acest caz special, numai
atributele cheie (A-urile de mai sus) sunt hash la găleți și numărul de
gălețile pentru fiecare atribut cheie sunt proporționale cu dimensiunea dimensiunii sale
masa.
de găleți, câte una pentru fiecare dintre cele trei funcții hash. Găsiți, în funcție de r, s,
t și k, valorile lui a, b și c care minimizează costul de comunicare al
algoritm.
! Exercițiul 2.5.3: Să presupunem că luăm o îmbinare stelară a unui tabel de fapte F (A_1 , A_2 , ..., A_m)
cu tabele de dimensiuni D_i (A_i , B_i) pentru i = 1, 2, ..., m. Să existe k reductoare,
fiecare asociat cu un vector de găleți, câte unul pentru fiecare dintre atributele cheie
A_1 , A_2 , ..., A_m . Să presupunem că numărul de găleți în care avem hash A_i este un $_i$.
Bineînțeles, $a_1 a_2 \cdots a_m = k$. În cele din urmă, să presupunem că fiecare tabel de dimensiuni D_i are dimensiune
d_i , iar dimensiunea tabelului de date este mult mai mare decât oricare dintre aceste dimensiuni. Găsi
valorile lui a_i care minimizează costul luării stelei se alătură ca una
Operațiune MapReduce.

Pagina 81

2.6 Teoria complexității pentru MapReduce

Acum vom explora mai detaliat proiectarea algoritmilor MapReduce.
Secțiunea 2.5 a introdus ideea că comunicarea între hartă și Re-
sarcinile de producție sunt adesea blocajul atunci când se efectuează un calcul MapReduce.
Aici, vom analiza modul în care costul comunicării este legat de alte deziderate
pentru algoritmii MapReduce, în special dorința noastră de a micșora timpul ceasului de perete
și să execute fiecare reductor din memoria principală. Amintiți-vă că un „reductor" este
executarea funcției Reduce pe o singură tastă și lista de valori asociate acesteia.
Scopul explorării din această secțiune este că pentru multe probleme există un
spectru de algoritmi MapReduce care necesită cantități diferite de comunicare-
ziune. Mai mult, cu cât un algoritm utilizează mai puțină comunicare, cu atât poate fi mai rău
în alte privințe, inclusiv timpul ceasului de perete și cantitatea de memorie principală
necesită.

2.6.1 Dimensiunea reductorului și rata de replicare

Să introducem acum cei doi parametri care caracterizează familiile MapRe-
duce algoritmi. Primul este dimensiunea reductorului, pe care o notăm cu q. Acest
parametrul este limita superioară a numărului de valori care sunt permise să se
pere în lista asociată cu o singură cheie. Dimensiunea reductorului poate fi selectată cu
cel puțin două obiective în minte.

1. Prin reducerea dimensiunii reductorului, putem forța să existe mai multe reductoare,
adică, multe taste diferite în funcție de care este împărțită intrarea problemei
prin sarcinile Hartă. Dacă creăm și multe sarcini de reducere - chiar și una pentru
fiecare reductor - atunci va exista un grad ridicat de paralelism și putem
așteptați un timp redus al ceasului de perete.
2. Putem alege un reductor suficient de mic încât să fim siguri că
calculul asociat cu un singur reductor poate fi executat în întregime în
memoria principală a nodului de calcul în care se află sarcina Reduce.
Indiferent de calculul făcut de reductoare, timpul de funcționare
va fi mult redus dacă putem evita să mutăm date în mod repetat
între memoria principală și disc.

Al doilea parametru este rata de replicare, notată r. Definim r to
fie numărul de perechi cheie-valoare produse de toate sarcinile Hărți pe toate
intrări, împărțit la numărul de intrări. Adică, rata de replicare este
comunicare medie de la sarcini de hartă pentru a reduce sarcinile (măsurată prin numărare
perechi cheie-valoare) pe intrare.

Exemplul 2.17: Să luăm în considerare algoritmul de multiplicare a matricei cu o singură trecere
din secțiunea 2.3.10. Să presupunem că toate matricile implicate sunt $n \times n$ matrici.
Atunci rata de replicare r este egală cu n. Acest fapt este ușor de văzut, deoarece pentru
fiecare element m_{ij} , există n perechi cheie-valoare produse; acestea au toate cheile de

---

Pagina 82

forma (i, k), pentru $1 \leq k \leq n$. La fel, pentru fiecare element al celeilalte matrice,
să zicem n_{jk} , producem n perechi cheie-valoare, fiecare având una dintre chei (i, k), pentru
$1 \leq i \leq n$. În acest caz, nu numai că n este numărul mediu de perechi cheie-valoare
produs pentru un element de intrare, dar fiecare intrare produce exact acest număr de
perechi.

Vedem, de asemenea, că q, dimensiunea reductorului necesar, este de 2n. Adică pentru fiecare cheie
(i, k), există n perechi cheie-valoare reprezentând elementele m_{ij} ale primei matrice
și alte n perechi cheie-valoare derivate din elementele n_{jk} ale celei de-a doua
matrice. În timp ce această pereche de valori reprezintă doar un algoritm special pentru
multiplicarea matricii cu o singură trecere, vom vedea că face parte dintr-un spectru de
algoritmi și, de fapt, reprezintă un punct extrem, unde q este cât se poate de mic
fi, iar r este la maxim. Mai general, există un compromis între r și
q, care poate fi exprimat ca $qr \geq 2n^2$. ✷

2.6.2 Un exemplu: Asocieri similare

Pentru a vedea compromisul dintre r și q într-o situație realistă, vom examina a
problemă cunoscută sub denumirea de asemănare. În această problemă, ni se oferă un set mare de
elementele X și o măsură de similitudine s (x, y) care arată cât de asemănătoare sunt două elemente
x și y din mulțimea X sunt. În capitolul 3 vom afla despre cele mai importante
noțiuni de similitudine și, de asemenea, să învățăm câteva trucuri care ne permit să găsim perechi similare
repede. Dar aici, vom lua în considerare doar forma brută a problemei, unde
trebuie să ne uităm la fiecare pereche de elemente ale lui X și să stabilim asemănarea lor cu
aplicând funcția s. Presupunem că s este simetric, deci s (x, y) = s (y, x),
dar nu presupunem nimic altceva despre s. Ieșirea algoritmului este acele perechi
a cărui similitudine depășește un prag dat t.

De exemplu, să presupunem că avem o colecție de câte un milion de imagini, fiecare de dimensiunea unu megabyte. Astfel, setul de date are dimensiunea de un terabyte. Nu vom face încercați să descrieți funcția de similitudine, dar s-ar putea, să zicem, să dea mai mare valori atunci când imaginile au aproximativ aceeași distribuție a culorilor sau când imaginile au regiuni corespunzătoare cu aceeași distribuție a culorilor. Scopul ar fi fie să descoperiți perechi de imagini care arată același tip de obiect sau scenă. Acest problema este extrem de grea, dar clasificarea după distribuția culorilor este în general de unii ajută la atingerea acestui obiectiv.

Să ne uităm la modul în care am putea face calculul folosind MapReduce pentru a exploata paralelismul natural găsit în această problemă. Intrarea este perechi cheie-valoare (i, P_i), unde i este un ID pentru imagine și P_i este imaginea în sine. Noi vrem pentru a compara fiecare pereche de imagini, așa că haideți să folosim o tastă pentru fiecare set de două ID-uri {i, j}. Există aproximativ 5×10^{11} perechi de două ID-uri. Ne dorim fiecare tasta {i, j} pentru a fi asociată cu cele două valori P_i și P_j, deci intrarea la reducătorul corespunzător va fi ({i, j}, [P_i, P_j]). Apoi, funcția Reduce poate pur și simplu aplicați funcția de similaritate s la cele două imagini din lista sa de valori, adică este, calculați s (P_i, P_j) și decideți dacă asemănarea celor două imagini este peste prag. Perechea ar fi rezultată dacă da.

Din păcate, acest algoritm va eșua complet. Dimensiunea reductorului este mică, deoarece nu

---

lista are mai mult de două valori, sau un total de 2 MB de intrare. Deși noi nu știm exact cum funcționează funcția de similitudine, ne putem aștepta în mod rezonabil că nu va necesita mai mult decât memoria principală disponibilă. Însă rata de replicare este de 999.999, deoarece pentru fiecare imagine generăm acel număr de chei-perechi de valori, una pentru fiecare dintre celelalte imagini din setul de date. Numărul total de octeți comunicați de la sarcini de hartă pentru a reduce sarcinile este de 1.000.000 (pentru imagini) ori 999.999 (pentru replicare), ori 1.000.000 (pentru dimensiunea fiecăruia imagine). Adică aproximativ 10^{18} octeți sau un exabyte. A comunica această cantitate de date pe Ethernet gigabit ar dura 10^{10} secunde, sau aproximativ 300 de ani. [11]

Din fericire, acest algoritm este doar punctul extrem într-un spectru de algoritmi ble. Putem caracteriza acești algoritmi prin gruparea imaginilor în grupuri g, fiecare din 10^6 / g imagini.

Funcția Hartă: Luați un element de intrare (i, P_i) și generați g - 1 tastă-perechi de valori. Pentru fiecare, cheia este unul dintre seturile {u, v}, unde u este grupul care imagine i aparține, și v este unul dintre celelalte grupuri. Valoarea asociată este perechea (i, P_i).

Funcția de reducere: luați în considerare cheia {u, v}. Lista valorilor asociate va avea elementele 2×10^6 / g (j, P_j), unde j aparține fie grupului u, fie grupul v. Funcția Reduce ia fiecare (i, P_i) și (j, P_j) pe această listă, unde i și j aparțin grupurilor diferite și aplică funcția de similaritate s (P_i, P_j).

În plus, trebuie să comparăm imaginile care aparțin aceluiași grup, dar nu vrem să facem aceeași comparație la fiecare dintre reductoarele g - 1 a cărei cheie conține un număr de grup dat. Există multe modalități de a rezolva acest lucru problemă, dar o modalitate este următoarea. Comparați membrii grupului u la reductor {u, u + 1}, unde „+1” este luat în sensul final. Acesta este, dacă u = g (adică, u este ultimul grup), atunci u + 1 este grupul 1. În caz contrar, u + 1 este grup al cărui număr este unul mai mare decât u.

Putem calcula rata de replicare și dimensiunea reductorului în funcție de număr de grupuri g. Fiecare element de intrare este transformat în g - 1 perechi cheie-valoare. Adică, rata de replicare este g - 1, sau aproximativ r = g, deoarece presupunem că numărul grupurilor este încă destul de mare. Dimensiunea reductorului este de 2×10^6 / g, deoarece acesta este numărul de valori din listă pentru fiecare reductor. Fiecare valoare este de aproximativ a megaocteți, deci numărul de octeți necesari pentru stocarea intrării este de 2×10^{12} / g.

Exemplul 2.18: Dacă g este 1000, atunci intrarea consumă aproximativ 2 GB. Asta e suficient pentru a ține totul într-o memorie principală tipică. Mai mult, totalul numărul de octeți comunicat este acum $10^6 \times 999 \times 10^6$, sau aproximativ 10^{15} octeți. Deși este încă o cantitate imensă de date de comunicat, este de 1000 de ori mai mică decât cea a algoritmului forței brute discutate mai întâi. Mai mult, există

[11] Într-un cluster tipic, există multe comutatoare care conectează subseturile nodurilor de calcul, deci toate datele nu trebuie să treacă printr-un singur comutator gigabit. Cu toate acestea, totalul disponibil comunicarea este încă suficient de mică încât să nu fie fezabilă implementarea acestui algoritm pentru scara datelor am emis ipoteza.

încă aproximativ jumătate de milion de reductoare. Deoarece este puțin probabil să avem la dispoziție asta multe noduri de calcul, putem împărți toate reductoarele într-un număr mai mic de
Reduceți sarcinile și păstrați în continuare toate nodurile de calcul ocupate; adică putem obține cât mai mult paralelismul pe care ni-l oferă clusterul nostru de calcul. ✷
Costul de calcul pentru algoritmi din această familie este independent de numărul de grupuri g, atâta timp cât intrarea în fiecare reductor se potrivește în memoria principală. Motivul este că cea mai mare parte a calculului este aplicarea funcției s la perechile de poze. Indiferent ce valoare are g, s se aplică fiecărei perechi o singură dată. Astfel, deși activitatea algoritmilor din familie poate să fie împărțiți între reductori în moduri diferite, toți membrii familiei faceți același calcul.

2.6.3 Un model de grafic pentru problemele MapReduce

În această secțiune, începem studiul unei tehnici care ne va permite să dovedim limite inferioare ale ratei de replicare, în funcție de dimensiunea reductorului, pentru un număr de probleme. Primul nostru pas este să introducem un model grafic al problemelor. Acest graficul descrie modul în care rezultatele problemei depind de intrări. The ideea cheie care trebuie exploatată este că, deoarece reductoarele funcționează independent, pentru fiecare ieșire trebuie să existe un reductor care obține toate intrările necesare pentru a calcula acea ieșire. Pentru fiecare problemă rezolvabilă de un algoritm MapReduce există:

1. Un set de intrări.
2. Un set de ieșiri.
3. O relație multe-multe între intrări și ieșiri, care describi care intrări sunt necesare pentru a produce ce ieșiri.

Exemplul 2.19: Figura 2.9 prezintă graficul pentru problema asemănării discutat în secțiunea 2.6.2, dacă există patru imagini mai degrabă decât un milion. The intrările sunt imaginile, iar ieșirile sunt cele șase perechi posibile de imagini. Fiecare ieșire este legată de cele două intrări care sunt membre ale perechii sale. Acest forma problemei, unde ieșirile sunt toate perechile de intrări, este comună și ne vom referi la aceasta ca problema tuturor perechilor. ✷
Exemplul 2.20: Înmulțirea matricei prezintă un grafic mai complex. Dacă noi înmulțiți n × n matrici M și N pentru a obține matricea P, atunci există $2n^2$ intrări, m_{ij} și n_{jk}, și există n^2 ieșiri p_{ik}. Fiecare ieșire p_{ik} este legată de 2n intrări: $m_{i1}, m_{i2}, ..., m_{in}$ și $n_{1k}, n_{2k}, ..., n_{nk}$. Mai mult, fiecare intrare este legată la n ieșiri. De exemplu, m_{ij} este legat de $p_{i1}, p_{i2}, ..., p_{in}$. Figura 2.10 prezintă relația intrare-ieșire pentru multiplicarea matricei pentru cazul simplu al 2 × 2 matrice, în mod specific
[ab
cd] [
ef
gh] = [
ij
kl]
✷

P_1
P_2
P_3
P_4
P_1
P_1
P_1
P_2
P_3
P_4
P_2
P_2
P_3
P_4
$P_3 P_4$
{,}
{,}
{,}
{,}

{,}
{,}
Figura 2.9: Relația intrare-ieșire pentru o îmbinare de similaritate
În problemele din exemplele 2.19 și 2.20, intrările și ieșirile au fost
în mod clar toți cei prezenți. Cu toate acestea, există alte probleme în care intrările și / sau
s-ar putea ca toate rezultatele să nu fie prezente în niciun caz al problemei. Un exemplu
a unei astfel de probleme este îmbinarea naturală a lui R (A, B) și S (B, C) discutate în
Secțiunea 2.3.7. Presupunem că atributele A, B și C au fiecare un domeniu finit,
deci există doar un număr finit de intrări și ieșiri posibile. Intrările sunt
toate posibilele R-tupluri, cele care constau dintr-o valoare din domeniul A asociat
cu o valoare din domeniul B și toate posibilele S-tupluri - perechi din
domeniile B și C. Ieșirile sunt toate triplurile posibile, cu componente de la
domeniile A, B și C în această ordine. Ieșirea (a, b, c) este conectată la
două intrări, și anume R (a, b) și S (b, c).
Dar într-o instanță a calculului de îmbinare, doar unele dintre intrările posibile
vor fi prezente și, prin urmare, vor fi produse doar unele dintre rezultatele posibile.
Acest fapt nu influențează graficul problemei. Mai trebuie să știm
modul în care fiecare ieșire posibilă se referă la intrări, indiferent dacă este sau nu ieșirea respectivă
produs într-o instanță dată.

2.6.4 Scheme de mapare

Acum că vedem cum să reprezentăm problemele abordabile de MapReduce ca
grafice, putem defini cerințele pentru rezolvarea unui algoritm MapReduce
o problemă dată. Fiecare astfel de algoritm trebuie să aibă o schemă de mapare, care
exprimă modul în care ieșirile sunt produse de diferiții reductori folosiți de algoritm
ritm. Adică o schemă de mapare pentru o problemă dată cu un reductor dat
dimensiunea q este o atribuire de intrări la unul sau mai multe reductoare, astfel încât:
1. Niciun reductor nu este alocat mai mult de q intrări.

A
b
c
d
e
f
g
h
eu
j
k
l
Figura 2.10: Relația intrare-ieșire pentru multiplicarea matricei
2. Pentru fiecare ieșire a problemei, există cel puțin un reductor
a atribuit toate intrările care sunt legate de acea ieșire. Spunem acest reductor
acoperă ieșirea.
Punctul (1) este pur și simplu definiția „dimensiunii reductorului”. Punctul (2) este justificat prin
faptul că reductoarele pot vedea doar intrările care le sunt date. Dacă nu există reductor
vede toate intrările de care depinde o ieșire, apoi niciun reductor nu poate corect
produce acea ieșire și, prin urmare, presupusul algoritm nu va funcționa. Aceasta
se poate argumenta că existența unei scheme de cartografiere pentru orice dimensiune a reductorului este
ce distinge problemele care pot fi rezolvate printr-o singură lucrare MapReduce
cei care nu pot.
Exemplul 2.21: Să reconsiderăm strategia de „grupare” pe care am discutat-o în
legătura cu problema tuturor perechilor din secțiunea 2.6.2. Pentru a generaliza problema,
să presupunem că intrarea este p imagini, pe care le plasăm în g grupuri de dimensiuni egale de p / g
intrări fiecare. Numărul de ieșiri este (p
2), sau aproximativ $p^2/2$ ieșiri. A
reductorul va primi intrările de la două grupuri - adică 2p / g intrări - deci reductorul
dimensiunea de care avem nevoie este q = 2p / g. Fiecare imagine este trimisă reductoarelor corespunzătoare
perechile formate din grupul său și oricare dintre celelalte grupe g - 1. Astfel,
rata de replicare este g - 1, sau aproximativ g. Dacă înlocuim g cu replicarea
rata r în q = 2p / g, concluzionăm că r = 2p / q. Adică rata de replicare
este invers proporțional cu dimensiunea reductorului. Această relație este comună; the
cu cât dimensiunea reductorului este mai mică, cu atât rata de replicare este mai mare și, prin urmare, cu atât este mai mare
comunicarea.
Această familie de algoritmi este descrisă de o familie de scheme de cartografiere, una
pentru fiecare q posibil. În schema de mapare pentru q = 2p / g, există (g

$_2$), sau
aproximativ g $_2$ /2 reductoare. Fiecare reductor corespunde unei perechi de grupuri,
iar o intrare P este atribuită tuturor reductoarelor a căror pereche include grupul de

---

P. Astfel, niciunui reductor nu i se alocă mai mult de 2p / g intrări; de fapt fiecare reductor
i se atribuie exact acel număr. Mai mult, fiecare ieșire este acoperită de unele
reductor. Mai exact, dacă ieșirea este o pereche din două grupuri diferite u și v,
atunci această ieșire este acoperită de reductor pentru perechea de grupuri {u, v}. Dacă
ieșirea corespunde intrărilor dintr-un singur grup u, apoi ieșirea este acoperită
de mai multe reductoare - cele corespunzătoare setului de grupuri {u, v} pentru oricare
v = u. Rețineți că algoritmul pe care l-am descris are doar unul dintre acești reductori
calculând ieșirea, dar oricare dintre ele ar putea să o calculeze. ✷
Faptul că o ieșire depinde de o anumită intrare înseamnă că atunci când
intrarea este procesată la sarcina Hartă, va exista cel puțin o pereche cheie-valoare
generat pentru a fi utilizat la calcularea ieșirii. Este posibil ca valoarea să nu fie
exact intrarea (așa cum a fost cazul în Exemplul 2.21), dar este derivată din aceasta
intrare. Ceea ce este important este că pentru fiecare intrare și ieșire conexă există
o pereche cheie-valoare care trebuie comunicată. Rețineți că există tehnic
niciodată nu este nevoie de mai multe perechi cheie-valoare pentru o anumită intrare și ieșire,
deoarece intrarea ar putea fi transmisă reductorului ca el însuși și oricum
transformările pe intrare au fost aplicate de funcția Harta ar putea în schimb
să fie aplicat de funcția Reduce la reductor pentru acea ieșire.

2.6.5 Când nu sunt prezente toate intrările

Exemplul 2.21 descrie o problemă în care știm că fiecare intrare posibilă este pre-
trimise, deoarece putem defini setul de intrare pentru a fi acele imagini care de fapt
există în setul de date. Totuși, așa cum sa discutat la sfârșitul secțiunii 2.6.3, acolo
sunt probleme precum calcularea îmbinării, unde graficul de intrări și ieșiri
descrie intrările care ar putea exista și ieșirile care se fac doar când cel puțin
una dintre intrări există în setul de date. De fapt, pentru unire, ambele intrări au fost legate
la o ieșire trebuie să existe dacă vrem să realizăm ieșirea respectivă.
Un algoritm pentru o problemă în care rezultatele pot lipsi are încă nevoie de
schema de cartografiere. Justificarea este că toate intrările sau orice subset al acestora,
ar putea fi prezent, deci un algoritm fără o schemă de mapare nu ar fi
capabil să producă fiecare ieșire posibilă dacă toate intrările legate de acea ieșire
s-a întâmplat să fie prezent și totuși niciun reductor nu acoperea ieșirea respectivă.
Singurul mod în care absența unor intrări face diferența este că noi
poate doriți să regândim valoarea dorită a dimensiunii reductorului q atunci când selectăm un
algoritm din familia posibililor algoritmi. Mai ales, dacă valoarea lui q
selectăm este acel număr astfel încât să putem fi siguri că intrarea se va potrivi doar în principal
memorie, atunci s-ar putea să dorim să creștem q pentru a lua în considerare acea fracțiune
dintre intrări nu sunt chiar acolo.
Exemplul 2.22: Să presupunem că știm că putem executa funcția Reduce
în memoria principală a unei taste și lista sa asociată de valori q. Cu toate acestea, și noi
să știți că doar 5% din intrările posibile sunt într-adevăr prezente în setul de date.
Apoi, o schemă de mapare pentru dimensiunea reductorului q va trimite cu adevărat aproximativ q / 20 din
intrări care există la fiecare reductor. Altfel spus, am putea folosi algoritmul

---

pentru dimensiunea reductorului 20q și așteptați-vă că va apărea efectiv o medie de q intrări
pe lista pentru fiecare reductor. Astfel, putem alege 20q ca dimensiune a reductorului sau de atunci
va exista o oarecare aleatorie în numărul de intrări care apar efectiv la
pentru fiecare reductor, am putea dori să alegem o valoare puțin mai mică a dimensiunii reductorului, de exemplu
ca 18q. ✷

2.6.6 Limite mai mici la rata de replicare

Familia algoritmilor de asociere similară descrisă în Exemplul 2.21 ne permite să tranzacționăm
opriți comunicarea împotriva dimensiunii reductorului și, prin intermediul dimensiunii reductorului, faceți comerțul
comunicare împotriva paralelismului sau împotriva capacității de a executa Reducerea
funcție în memoria principală. De unde știm că obținem cât mai bine posibil
compromis? Putem ști că avem o comunicare minimă posibilă numai dacă
putem dovedi o limită inferioară potrivită. Folosind existența unei scheme de cartografiere ca
punctul de plecare, putem dovedi adesea o astfel de limită inferioară. Iată o schiță

a tehnicii.
1. Dovediți o limită superioară la câte ieșiri poate face un reductor cu q intrări acoperi. Apelați această legătură g (q). Acest pas poate fi dificil, dar pentru exemple la fel ca problema tuturor perechilor, este de fapt destul de simplă.
2. Determinați numărul total de ieșiri produse de problemă.
3. Să presupunem că există k reductoare, iar reductorul ith are $q_i < q$ intrări. Observați că $\sum_{i=1}^{k} g(q_i)$ nu trebuie să fie mai mic decât numărul de ieșiri calculat în etapa (2).
4. Manipulați inegalitatea din (3) pentru a obține o limită inferioară pe $\sum_{i=1}^{k} q_i$.
Adesea, trucul folosit la acest pas este de a înlocui unii factori de q_i cu lor limita superioară q, dar lăsați un singur factor de q_i în termenul pentru i.
5. Din moment ce $\sum_{i=1}^{k} q_i$ este comunicarea totală de la sarcini de hartă pentru a reduce sarcinile, împărțiți limita inferioară de la (4) pe această cantitate la numărul de intrări. Rezultatul este o limită inferioară a ratei de replicare.
Exemplul 2.23: Această secvență de pași poate părea misterioasă, dar să ne considerăm problema tuturor perechilor ca exemplu care sperăm să clarifice lucrurile. Reamintim că în Exemplul 2.21 am dat o limită superioară asupra ratei de replicare r de $r \leq 2p / q$, unde p a fost numărul de intrări și q a fost dimensiunea reductorului. Vom arăta o limită inferioară pe r care este jumătate din suma respectivă, ceea ce implică că, deși sunt posibile îmbunătățiri ale algoritmului, [12] orice reducere a comunicarea pentru o dimensiune redusă dată va fi de cel mult 2.
Pentru pasul (1), observați că, dacă un reductor primește q intrări, acesta nu poate acoperi mai mult decât $\binom{q}{2}$, sau aproximativ $q^2/2$ ieșiri. Pentru pasul (2), știm că există un

[12] De fapt, există un algoritm cu r foarte apropiat de p / q, pentru cel puțin unele valori ale lui p.

---

total de $\binom{p}{2}$, sau aproximativ $p^2/2$ ieșiri care fiecare trebuie să fie acoperite. The inegalitatea construită la etapa (3) este astfel

$$\sum_{i=1}^{k} q_i^2 / 2 \geq p^2/2$$

sau, înmulțind ambele părți cu 2,

$$\sum_{i=1}^{k} q_i^2 \geq p^2 \tag{2.1}$$

Acum, trebuie să facem manipularea pasului (4). În urma indiciului, observăm că există doi factori de q_i în fiecare termen din stânga ecuației (2.1), deci înlocuim un factor cu q și îl lăsăm pe celălalt ca q_i. Deoarece $q \geq q_i$, putem doar măriți partea stângă făcând acest lucru și astfel inegalitatea continuă să se mențină:

$$q \sum_{i=1}^{k} q_i \geq p^2$$

sau, împărțind la q:

$$\sum_{i=1}^{k} q_i \geq p^2 / q \tag{2.2}$$

Pasul final, care este pasul (5), este împărțirea ambelor părți ale ecuației 2.2 la p, numărul de intrări. Ca urmare, partea stângă, care este $(\sum_{i=1}^{k} q_i) / p$ este egal la rata de replicare, iar partea dreaptă devine p / q. Adică am dovedit limita inferioară pe r:

$r \geq p / q$
După cum sa susținut, acest lucru arată că familia algoritmilor din Exemplul 2.21 au toate o rată de replicare care este cel mult de două ori cea mai mică rată de replicare posibilă. ✷

2.6.7 Studiu de caz: Multiplicarea matricei

În această secțiune vom aplica tehnica de limită inferioară matricei cu o singură trecere-algoritmi de multiplicare. Am văzut un astfel de algoritm în secțiunea 2.3.10, dar acesta este doar un caz extrem al unei familii de algoritmi posibili. În special, pentru acel algoritm, un reductor corespunde unui singur element al matricei de ieșire.
La fel cum am grupat intrările în problema tuturor perechilor pentru a reduce comunicarea în detrimentul unei dimensiuni mai mari a reductorului, putem grupa rânduri și coloane ale două matrice de intrare în benzi. Fiecare pereche constând dintr-o bandă de rânduri de prima matrice și o bandă de coloane a celei de-a doua matrice este utilizată de un reductor pentru a produce un pătrat de elemente ale matricei de ieșire. Se sugerează un exemplu după Fig. 2.11.

Pagina 90

=
Figura 2.11: Împărțirea matricilor în benzi pentru a reduce comunicarea
Mai detaliat, să presupunem că vrem să calculăm MN = P și toate cele trei matrice sunt $n \times n$. Grupați rândurile de M în g benzi de n / g rânduri fiecare și grupați coloane de N în g benzi de n / g coloane fiecare. Această grupare este cea sugerată după Fig. 2.11. Tastele corespund a două grupuri (benzi), una din M și una din N.
Funcția Map: Pentru fiecare element al lui M, funcția Map generează g perechi cheie-valoare. Valoarea în fiecare caz este elementul în sine, împreună cu elementul său rândul și numărul coloanei, astfel încât să poată fi identificat prin funcția Reduce. The cheia este grupul căruia îi aparține elementul, asociat cu oricare dintre grupuri a matricei N. În mod similar, pentru fiecare element al lui N, funcția Harta generează g perechi cheie-valoare. Cheia este grupul acelui element asociat cu oricare dintre grupuri de M, iar valoarea este elementul în sine plus rândul și coloana sa.
Funcția de reducere: reductorul corespunzător tastei (i, j), unde i este un grup de M și j este un grup de N, primește o listă de valori constând din toate elementele din banda a lui M și banda a X-a a N. Are astfel toate valorile de care are nevoie pentru a calcula elementele lui P al căror rând este unul dintre acele rânduri care cuprinde banda a lui M și a cărei coloană este una dintre cele care cuprind a doua bandă a lui N. De exemplu, Fig. 2.11 sugerează al treilea grup de M și al patrulea grup de N, combinându-se pentru a calcula un pătrat de P la reductor (3, 4).
Fiecare reductor obține n (n / g) elemente din fiecare dintre cele două matrice, deci $q = 2n^2 / g$. Rata de replicare este g, deoarece fiecare element din fiecare matrice este trimis către g reductoare. Adică r = g. Combinând r = g cu $q = 2n^2 / g$ putem concluziona că $r = 2n^2 / q$. Adică, la fel ca și pentru asocierea similară, rata de replicare variază invers cu dimensiunea reductorului.
Se pare că această limită superioară la rata de replicare este, de asemenea, o limită inferioară. Adică, nu putem face mai bine decât familia de algoritmi pe care am descris-o mai sus într-o singură rundă de MapReduce. Interesant, vom vedea că putem obține un comunicare totală mai mică pentru aceeași dimensiune a reductorului, dacă folosim două treceri de MapReduce așa cum am discutat în secțiunea 2.3.9. Nu vom da completul dovada limitei inferioare, dar va sugera elementele importante.

Pagina 91

Pentru pasul (1) trebuie să obținem o limită superioară la câte ieșiri are un reductor de dimensiunea q poate acoperi. Mai întâi, observați că, dacă un reductor introduce unele dintre elemente un rând de M, dar nu toate, atunci elementele acelui rând sunt inutile; the reductor nu poate produce nicio ieșire în acel rând de P. În mod similar, dacă un reductor primește o parte, dar nu toate coloanele de N, aceste intrări sunt, de asemenea, inutile. Prin urmare, putem presupune că cea mai bună schemă de mapare va trimite fiecărui reductor numărul de rânduri complete de M și un număr de coloane complete de N. Acest reductor este apoi capabil să producă elementul de ieșire p_{ik} dacă și numai dacă a primit întregul al doilea rând al lui M și întreaga a cincea coloană a lui N.
argumentul pentru etapa (1) este de a demonstra că este acoperit cel mai mare număr de ieșiri când reductorul primește același număr de rânduri ca și coloanele. Lăsăm asta parte ca exercițiu.

Totuși, presupunând că un reductor primește k rânduri de M și k coloane de N, atunci $q = 2nk$ și ieșirile k^2 sunt acoperite. Adică g (q), numărul maxim de ieșiri acoperite de un reductor care primește q intrări, este $q^2 / 4n^2$. Pentru pasul (2), știm că numărul de ieșiri este n^2. La pasul (3) observăm că dacă există k reductoare, cu reductorul ith primind $q_i \leq q$ intrări, atunci

$$\sum_{i=1}^{k} q_i^2 / 4n^2 \geq n^2$$

sau

$$\sum_{i=1}^{k} q_i^2 \geq 4n^4$$

Din această inegalitate, puteți obține

$$r \geq 2n^2 / q$$

Lăsăm manipularea algebrică, care este similară cu cea din Exemplul 2.23, ca exercițiu.

Acum, să luăm în considerare generalizarea multiplicării matricei în două treceri algoritm pe care l-am descris în secțiunea 2.3.9. În primul rând, observați că am putea avea a proiectat prima trecere pentru a utiliza un reductor pentru fiecare triplu (i, j, k). Acest reductor ar obține doar cele două elemente m_{ij} și n_{jk}. Putem generaliza această idee la utilizați reductoare care obțin seturi mai mari de elemente din fiecare matrice; aceste seturi de elementele formează pătrate în matricile lor respective. Ideea este sugerată după Fig. 2.12. Putem împărți rândurile și coloanele ambelor matrice de intrare M și N în g grupuri de n / g rânduri sau coloane fiecare. Intersecțiile dintre grupurile împart fiecare matrice în g^2 pătrate de n^2 / g^2 elemente fiecare. Pătratul M corespunzător setului de rânduri I și setului de coloane J linii cu pătratul lui N corespunzător setului de rânduri J și setului de coloane K. Aceste două pătrate calculează câțiva dintre termenii necesari pentru a produce pătratul matricei de ieșire P care are set de rânduri I și set de coloane K. Cu toate acestea, aceste două pătrate nu calculează valoarea completă a acestor elemente ale P; mai degrabă produc o contribuție la sumă. Alte perechi de pătrate, una

---

Pagina 92

Figura 2.12: Partiționarea matricilor în pătrate pentru un MapReduce al-doi goritm

din M și unul din N, contribuie la același pătrat al lui P. Aceste contribuții sunt sugerate în Fig. 2.12. Acolo, vedem cum toate pătratele lui M cu o valoare fixă pentru setul de rânduri pe care le împerechez cu toate pătratele lui N care au un fix valoare pentru setul de coloane K, lăsând setul J să varieze.

Deci, în prima trecere, calculăm produsele pătratului (I, J) al lui M cu pătratul (J, K) al lui N, pentru tot I, J și K. Apoi, în a doua trecere, pentru fiecare I și K însumăm produsele pe toate seturile posibile J. Mai detaliat, primul Jobul MapReduce face următoarele.

Funcția Hartă: Tastele sunt tripluri de seturi de rânduri și / sau numere de coloane bers (I, J, K). Să presupunem că elementul m_{ij} aparține grupului de rânduri I și grup de coloane J. Apoi din m_{ij} generăm g perechi cheie-valoare cu valoare egală cu m_{ij}, împreună cu numerele sale de rânduri și coloane, i și j, pentru a identifica matricea element. Există o pereche cheie-valoare pentru fiecare cheie (I, J, K), unde K poate fi oricare din grupele g de coloane ale lui N. În mod similar, din elementul n_{jk} al lui N, dacă j aparține pentru a grupa J și k pentru a grupa K, funcția Hartă generează g perechi cheie-valoare cu valoare constând din n_{jk}, j și k și cu taste (I, J, K) pentru orice grup I.

Funcția de reducere: reductorul corespunzător (I, J, K) primește ca introduceți toate elementele m_{ij} unde i este în I și j este în J, și primește tot elementele n_{jk}, unde j este în J și k este în K. Calculează

$$x_{iJk} = \sum_{j \text{ în } J} m_{ij} n_{jk}$$

Pagina 93

pentru tot i în I și k în K.
Observați că rata de replicare pentru primul job MapReduce este g și că to-
comunicarea tal este deci $2gn^2$. De asemenea, observați că fiecare reductor primește $2n^2 / g^2$
intrări, deci $q = 2n^2 / g^2$. În mod echivalent, $g = n\sqrt{2 / q}$. Astfel, comunicarea totală
$2gn^2$ poate fi scris în termeni de q ca $2\sqrt{2n^3 / \sqrt{q}}$.
A doua lucrare MapReduce este simplă; rezumă x_{ijk} pe toate seturile J.
Funcția Hărți: Presupunem că sarcinile Hărți se execută la orice
nodurile de calcul au executat sarcinile de reducere ale jobului anterior. Astfel, nici o com-
este necesară comunicarea între locuri de muncă. Funcția Hartă ia ca intrare una
elementul x_{ijk}, care presupunem că reductoarele anterioare au rămas etichetate cu i
și k deci știm la ce element al matricei P contribuie acest termen. unu
se generează perechea cheie-valoare. Cheia este (i, k), iar valoarea este x_{ijk}.
Funcția Reduce: Funcția Reduce sumează pur și simplu valorile asociate
cu tasta (i, k) pentru a calcula elementul de ieșire P_{ik}.
Comunicarea dintre sarcinile Harta și Reducerea celui de-al doilea job este
gn^2, deoarece există n valori posibile ale lui i, n valori posibile ale lui k și g posibile
valorile setului J și fiecare x_{ijk} este comunicată o singură dată. Dacă ne amintim de la
analiza noastră despre primul job MapReduce că $g = n\sqrt{2 / q}$, putem scrie
comunicare pentru al doilea job ca $n^2 g = \sqrt{2n^3 / \sqrt{q}}$. Această sumă este exact
jumătate din comunicarea pentru primul loc de muncă, deci comunicarea totală pentru
algoritmul cu două treceri este $3\sqrt{2n^3 / \sqrt{q}}$. Deși nu vom examina acest punct
aici, se dovedește că putem face ceva mai bine dacă împărțim matricile M
și N nu în pătrate, ci în dreptunghiuri care sunt de două ori mai lungi pe o parte
ca pe de alta. În acest caz, obținem constanta 4 puțin mai mică în loc
de $3\sqrt{2} = 4.24$ și obținem un algoritm în două treceri cu o comunicare egală cu
$4n^3 / \sqrt{q}$.
Acum, amintiți-vă că costul de comunicare pe care l-am calculat pentru o singură trecere
algoritmul este $4n^4 / q$. La fel de bine putem presupune că q este mai mic decât n^2, sau altfel putem
pur și simplu utilizați un algoritm serial la un nod de calcul și nu utilizați deloc MapReduce.
Astfel, $n^3 / \sqrt{q}$ este mai mic decât n^4 / q, iar dacă q este aproape de minimul său posibil
valoarea de 2n, [13] atunci algoritmul cu două treceri bate algoritmul cu o singură trecere cu a
factorul $O(\sqrt{n})$ în comunicare. Mai mult, ne putem aștepta la diferență
în comunicare să fie diferența semnificativă de cost. Ambii algoritmi o fac
aceleași operații aritmetice $O(n^3)$. Metoda în două treceri are în mod natural
mai multe sarcini de gestionare a cheltuielilor generale decât metoda cu un singur job. Pe de altă parte
mână, a doua trecere a algoritmului cu două treceri aplică o funcție Reduce
adică asociativ și comutativ. Astfel, ar putea fi posibil să salvați unele
costul comunicării prin utilizarea unui combinator pe acel permis.

2.6.8 Exerciții pentru secțiunea 2.6

Exercițiul 2.6.1: Descrieți graficele care modelează următoarele probleme.

[13] Dacă q este mai mic de 2n, atunci un reductor nu poate obține nici măcar un rând și o coloană și, prin urmare nu poate calcula deloc ieșiri.

Pagina 94

(a) Înmulțirea unei matrice $n \times n$ cu un vector de lungime n.
(b) Îmbinarea naturală a lui R (A, B) și S (B, C), unde A, B și C au
rețele de dimensiuni a, b și respectiv c.
(c) Gruparea și agregarea pe relația R (A, B), unde A este
atributul de grupare și B este agregat de operația MAX. Să presupunem
A și B au domenii de mărime a și respectiv b.
! Exercițiul 2.6.2: Furnizați detaliile dovezii că o matrice cu o singură trecere
algoritmul de multiplicare necesită o rată de replicare cel puțin $r \geq 2n^2 / q$, inclusiv:
(a) Dovada că, pentru o dimensiune redusă fixă, numărul maxim de ieșiri
sunt acoperite de un reductor atunci când acel reductor primește un număr egal de
rânduri de M și coloane de N.
(b) Manipularea algebrică necesară, începând cu $\sum_{i=1}^{k} q_i^2 \geq 4n^4$.
!! Exercițiul 2.6.3: Să presupunem că intrările noastre sunt șiruri de biți de lungime b și ieșirile
corespund perechilor de corzi la distanța de Hamming 1. [14]
(a) Dovediți că un reductor de dimensiunea q poate acoperi cel mult $(q / 2) \log_2 q$ ieșiri.
(b) Utilizați partea (a) pentru a afișa limita inferioară a ratei de replicare: $r \geq b / \log_2 q$.
(c) Arătați că există algoritmi cu rata de replicare, așa cum este dat de partea (b)
pentru cazurile $q = 2$, $q = 2^b$ și $q = 2^{b/2}$.
!! Exercițiul 2.6.4: Pentru p care este pătratul unui prim, arată că există un

schema de mapare pentru problema tuturor perechilor care are $r \leq 1 + p / q$.

2.7 Rezumatul capitolului 2

♦ Cluster Computing: O arhitectură comună pentru aplicații la scară foarte mare este un grup de noduri de calcul (cip procesor, memorie principală și disc). Nodurile de calcul sunt montate în rack-uri, iar nodurile de pe rack sunt conectat, de obicei prin gigabit Ethernet. Rack-urile sunt, de asemenea, conectate printr-un rețea sau comutator de mare viteză.

♦ Sisteme de fișiere distribuite: o arhitectură pentru sisteme de fișiere la scară foarte mare-tems s-a dezvoltat recent. Fișierele sunt compuse din bucăți de aproximativ 64 megaocteți, iar fiecare bucată este reprodusă de mai multe ori, pe com-noduri sau rafturi pute.

Șirurile de 14 biți au distanța Hamming 1 dacă diferă exact într-o poziție de biți. Poți priviți înainte la secțiunea 3.5.6 pentru definiția generală.

♦ MapReduce: Acest sistem de programare permite exploatarea paralelismului inerent în calculul cluster și gestionează eșecurile hardware care poate apărea în timpul unui calcul lung pe multe noduri. Multe sarcini de hartă și multe sarcini Reduce sunt gestionate de un proces Master. Sarcini pe un nodul de calcul eșuat este reluat de către Master.

♦ Funcția Hartă: Această funcție este scrisă de utilizator. Este nevoie de un colecție de obiecte de intrare și transformă fiecare în zero sau mai multe valori-cheie perechi. Cheile nu sunt neapărat unice.

♦ Funcția Reduce: un sistem de programare MapReduce sortează toate perechi cheie-valoare produse de toate sarcinile Hărții, formează toate valorile asociat cu o cheie dată într-o listă și distribuie perechi de liste de chei pentru a Reduce sarcini. Fiecare activitate Reduce combină elementele de pe fiecare listă, aplicând funcția scrisă de utilizator. Rezultatele produse de toate Reduce sarcinile formează rezultatul procesului MapReduce.

♦ Reductoare: este adesea convenabil să faceți referire la aplicarea Reducerii funcționează la o singură tastă și la lista de valori asociate acesteia ca „reductor”.

♦ Hadoop: Acest sistem de programare este o implementare open-source a unui sistem de fișiere distribuite (HDFS, sistemul de fișiere distribuite Hadoop) și MapReduce (Hadoop în sine). Este disponibil prin baza Apache ziune.

♦ Gestionarea eșecurilor de calcul-nod: sistemele MapReduce acceptă repornirea de sarcini care eșuează din cauza nodului lor de calcul sau a rack-ului care îl conține nod, eșuează. Deoarece sarcinile Hărți și Reducere își produc rezultatele numai după se termină (proprietatea de blocare), este posibil să reporniți o sarcină eșuată fără grija pentru o posibilă repetare a efectelor acelei sarcini. Este necesar pentru a reporni întreaga lucrare numai dacă nodul la care Maestrul execută eșuează.

♦ Aplicații MapReduce: Deși nu toți algoritmii paraleli sunt adecvați pentru implementarea în cadrul MapReduce, există im-plementări de multiplicare matrice-vector și matrice-matrice. De asemenea, principalii operatori ai algebrei relaționale sunt ușor de implementat în MapReduce.

♦ Sisteme de flux de lucru: MapReduce a fost generalizat la sisteme care suportă port orice colecție aciclică de funcții, fiecare dintre acestea putând fi instanțiată prin orice număr de sarcini, fiecare responsabil pentru executarea acelei funcții o parte din date.

♦ Spark: Acest popular sistem de flux de lucru introduce Rezilient, Distribuit Seturi de date (RDD-uri) și un limbaj în care se operează multe operații comune RDD-urile pot fi scrise. Spark are o serie de eficiențe, inclusiv leneșe evaluarea RDD-urilor pentru a evita stocarea secundară a rezultatelor intermediare

---

și înregistrarea liniei pentru RDD-uri, astfel încât acestea să poată fi reconstruite ca Necesar.

♦ TensorFlow: Acest sistem de flux de lucru este conceput special pentru a suporta
învățare automată. Datele sunt reprezentate ca tablouri multidimensionale sau zece
sors și operațiunile încorporate efectuează multe operații puternice, cum ar fi
algebră liniară și antrenament model.

♦ Fluxuri de lucru recursive: la implementarea unei colecții recursive de funcții
nu este întotdeauna posibil să se păstreze capacitatea de a reporni orice eșuat
sarcini recursive este posibil să fi produs rezultate care erau
însumată de o altă sarcină înainte de eșec. O serie de scheme de verificare
arătând părți ale calculului pentru a permite repornirea sarcinilor individuale sau repornirea
toate sarcinile dintr-un punct recent au fost propuse.

♦ Comunicare-Cost: Multe aplicații MapReduce sau sisteme similare
temurile fac lucruri foarte simple pentru fiecare sarcină. Apoi, costul dominant este
de obicei costul transportului datelor de unde sunt create până unde
este folosit. În aceste cazuri, eficiența unui algoritm MapReduce poate fi
estimată prin calcularea sumei dimensiunilor intrărilor la toate
sarcini.

♦ Îmbinări multiple: uneori este mai eficient să replici tupluri ale
relațiile implicate într-o asociere și au asocierea a trei sau mai multe relații
calculat ca un singur job MapReduce. Tehnica mul- lagrangeană
tipliers pot fi utilizate pentru a optimiza gradul de replicare pentru fiecare dintre
relații participante.

♦ Star Joins: interogările analitice implică adesea un tabel de fapte foarte mare alăturat
cu tabele de dimensiuni mai mici. Aceste îmbinări se pot face întotdeauna eficient
prin tehnica multi-join. O alternativă este distribuirea faptului
tabela și reproduceți tabelele de dimensiuni permanent, folosind același lucru
strategie așa cum ar fi folosită dacă luăm unirea multiplă a faptului
tabel și fiecare tabel de dimensiuni.

♦ Rata de replicare și dimensiunea reductorului: este adesea convenabil să se măsoare
comunicare prin rata de replicare, care este comunicarea per
intrare. De asemenea, dimensiunea reductorului este numărul maxim de intrări asociate
cu orice reductor. Pentru multe probleme, este posibil să se obțină o valoare mai mică
legat de rata de replicare în funcție de dimensiunea reductorului.

♦ Reprezentarea problemelor ca grafice: Este posibil să se reprezinte multe probleme
Lemuri care sunt supuse calculului MapReduce printr-un grafic în care
nodurile reprezintă intrări și ieșiri. O ieșire este conectată la toate fișierele
intrări care sunt necesare pentru a calcula ieșirea respectivă.

♦ Scheme de mapare: Având în vedere graficul unei probleme și având în vedere dimensiunea reductorului,
o schemă de mapare este o atribuire a intrărilor la unul sau mai multe reductoare

---

Pagina 97

astfel încât niciunui reductor nu i se alocă mai multe intrări decât permite dimensiunea reductorului,
și totuși pentru fiecare ieșire există un reductor care obține toate intrările
necesare pentru a calcula acea ieșire. Cerința ca să existe o cartografiere
schema pentru orice algoritm MapReduce este o bună expresie a ceea ce face
Algoritmi MapReduce diferiți de calculele paralele generale.

♦ Multiplicarea matricei prin MapReduce: Există o familie de hartă cu o singură trecere
Reduceți algoritmii care efectuează multiplicarea matricilor n × n cu
rata minimă de replicare posibilă $r = 2n^2 / q$, unde q este reductorul
mărimea. Pe de altă parte, un algoritm MapReduce în două treceri pentru același lucru
problemă cu aceeași dimensiune a reductorului poate utiliza până la un factor de n mai puțin com-
municare.

2.8 Referințe pentru capitolul 2

GFS, sistemul de fișiere Google, a fost descris în [13]. Hârtia de pe Google
MapReduce este [10]. Informații despre Hadoop și HDFS pot fi găsite la [15].
Mai multe detalii despre relații și algebră relațională pot fi găsite în [25].
Câteva dintre cele mai vechi sisteme de flux de lucru au fost Clustera [11] la Univ. de
Wisconsin, și Hyracks (denumit anterior Hyrax) [6], de la UC Irvine, și
Dryad Microsoft [17] și mai târziu DryadLINQ [26].
Flink este un sistem open-source de flux de lucru conceput pentru a gestiona fluxul de date
[12]. A fost inițial dezvoltat în cadrul proiectului Stratosphere [5] de la TU Berlin.
Multe dintre inovațiile din Spark sunt descrise în [27]. Imprimarea open-source
plementarea Spark este la [21]. Pagina de informații despre TensorFlow este
[24].

Abordarea iterată-MapReduce a implementării recursivității este de la Habucla [7]. Pentru o discuție despre implementarea clusterului de recursivitate, consultați [1]. Pregel este din [19]. Există o versiune open-source a lui Pregel numită Giraph [14]. GraphLab [18] este un alt sistem de implementare paralel notabil pentru grafic algoritmi. GraphX [22] este un front-end grafic pentru Spark.
Există o serie de alte sisteme construite pe un sistem de fișiere distribuit și / sau MapReduce, care nu au fost acoperite aici, dar ar putea fi demn de știut despre. [8] descrie BigTable, o implementare Google a unui depozit de obiecte de dimensiune foarte mare. O direcție oarecum diferită a fost luată la Yahoo! cu nuci [9]. Acesta din urmă acceptă o formă limitată de procesare a tranzacțiilor, de exemplu. PIG [20] este o implementare a algebrei relaționale deasupra Hadoop. Similarly, Hive [16] implementează o formă restricționată de SQL deasupra Hadoop. Scânteie oferă, de asemenea, un front-end de tip SQL [23].
Modelul de comunicare-cost pentru algoritmii MapReduce și optimul implementările îmbinărilor multiple sunt de la [3]. Materialul privind rata de replicare, dimensiunea reductorului, iar relația lor este de la [2]. Soluții la exercițiile 2.6.2 și 2.6.3 poate fi găsit acolo. Soluția la exercițiul 2.6.4 este în [4].

Pagina 98

1. FN Afrati, V. Borkar, M. Carey, A. Polyzotis și JD Ullman, „Cluster computer, recursivity and Datalog ”, să apară în Proc. Datalog 2.0 Atelier, Elsevier, 2011.
2. FN Afrati, A. Das Sarma, S. Salihoglu și JD Ullman, „Upper și limite inferioare ale costului unui calcul MapReduce. ” să apară în Proc. Intl. Conf. pe baze de date foarte mari, 2013. Disponibil și ca CoRR, abs / 1206.4377.
3. FN Afrati și JD Ullman, „Optimizarea îmbinărilor într-un mediu MapReduce ,,Proc. A treisprezecea Intl. Conf. despre extinderea tehnologiei bazelor de date, 2010.
4. FN Afrati și JD Ullman, „Limite potrivite pentru toate perechile MapReduce problema, ”IDEAS 2013, pp. 3-4.
5. A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinlander, MJ Sax, S. Schelter, M. Hoger, K. Tzoumas și D. Warneke, „Platforma Stratosphere pentru analiza datelor mari”, VLDB J. 23: 6, pp. 939–964, 2014.
6. VR Borkar, MJ Carey, R, Grover, N. Onose și R. Vernica, „Hyracks: O bază flexibilă și extensibilă pentru calculul intensiv în date, ”Intl. Conf. on Data Engineering, pp. 1151–1162, 2011.
7. Y. Bu, B. Howe, M. Balazinska și M. Ernst, „HaLoop: eficient iterprelucrarea activă a datelor pe clustere mari, ”Proc. Intl. Conf. pe Foarte mare Baze de date, 2010.
8. F. Chang, J. Dean, S. Ghemawat, WC Hsieh, DA Wallach, M. Burrows, T. Chandra, A. Fikes și RE Gruber, „Bigtable: un spațiu de stocare distribuit sistem pentru date structurate ”, ACM Transactions on Computer Systems 26: 2, pp. 1–26, 2008.
9. BF Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver și R. Yerneni, „Pnuts: Yaplatforma de servire a datelor găzduită de hoo! ”, PVLDB 1: 2, pp. 1277–1288, 2008.
10. J. Dean și S. Ghemawat, „Mapreduce: prelucrare simplificată a datelor pe clustere mari ", com. ACM 51: 1, pp. 107-113, 2008.
11. DJ DeWitt, E. Paulson, E. Robinson, JF Naughton, J. Royalty, S. Shankar și A. Krioukov, „Clustera: un calcul integrat și sistem de gestionare a datelor, ”PVLDB 1: 1, pp. 28-41, 2008.
12. flink.apache.org, Apache Foundation.
13. S. Ghemawat, H. Gobioff și S.-T. Leung, „Sistemul de fișiere Google” Al 19-lea Simpozion ACM privind principiile sistemelor de operare, 2003.

Pagina 99

14. giraph.apache.org, Fundația Apache.
15. hadoop.apache.org, Fundația Apache.
16. hadoop.apache.org/hive, Apache Foundation.

17. M. Isard, M. Budiu, Y. Yu, A. Birrell și D. Fetterly. „Dryad: dis-
programe paralele de date tributate din blocuri de construcție secvențiale, ”Procede-
a doua Conferință europeană ACM SIGOPS / EuroSys
puter Systems, pp. 59-72, ACM, 2007.
18. Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola și JM Heller-
stein, „Distribuit GraphLab: un cadru pentru învățarea automată și date
minerit în nor ”, - Proc. VLDB Endowment 5: 8, pp. 716-727,
2012.
19. G. Malewicz, MN Austern, AJC Sik, JC Denhert, H. Horn, N. Leiser,
și G. Czajkowski, „Pregel: un sistem pentru prelucrarea graficelor pe scară largă”
Proc. Conferința ACM SIGMOD, 2010.
20. C. Olston, B. Reed, U. Srivastava, R. Kumar și A. Tomkins, „Pig latin:
o limbă nu atât de străină pentru prelucrarea datelor, ”Proc. ACM SIGMOD Con-
ference, pp. 1099–1110, 2008.
21. spark.apache.org, Fundația Apache.
22. spark.apache.org/graphx, Apache Foundation.
23. spark.apache.org/sql, Apache Foundation.
24. www.tensorflow.org.
25. JD Ullman și J. Widom, Un prim curs de sisteme de baze de date, al treilea
Ediție, Prentice-Hall, Upper Saddle River, NJ, 2008.
26. Y. Yu, M. Isard, D. Fetterly, M. Budiu, I. Erlingsson, PK Gunda și
J. Currey, „DryadLINQ: un sistem pentru date distribuite de uz general-
calcul paralel folosind un limbaj la nivel înalt ", OSDI, pp. 1-14, USENIX
Asociația, 2008.
27. M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, MJ
Franklin, S. Shenker și I. Stoica, „Seturi de date distribuite rezistente: A
abstracție tolerantă la erori pentru calculul clusterului în memorie ”, Proc. 9
Conferința USENIX despre proiectarea și implementarea sistemelor în rețea,
Asociația USENIX, 2012.

Pagina 100

Pagina 101

capitolul 3
Găsirea unor articole similare

O problemă fundamentală de extragere a datelor este examinarea datelor pentru elemente „similare”. Noi va prelua cererile în secțiunea 3.1, dar un exemplu ar fi analiza unui colectarea de pagini Web și găsirea de pagini aproape duplicate. Aceste pagini ar putea fi plagiatele, de exemplu, sau ar putea fi oglinzi care au aproape la fel conținut, dar diferă în ceea ce privește informațiile despre gazdă și despre alte oglinzi.
Abordarea naivă de a găsi perechi de articole similare ne impune să analizăm o pereche de articole. Când avem de-a face cu un set de date mare, ne uităm la toate perechile de articole pot fi prohibitive, chiar și având în vedere o abundență de resurse hardware.
De exemplu, chiar și un milion de articole ne oferă o jumătate de trilion de perechi de examinat și un milion de articole este considerat un set de date „mic” conform standardelor actuale.
Prin urmare, este o surpriză plăcută să aflăm despre o familie de tehnici numite hash sensibil la localitate sau LSH, care ne permite să ne concentrăm pe perechi care sunt probabil a fi asemănător, fără a fi nevoie să te uiți la toate perechile. Astfel, este posibil ca noi poate evita creșterea pătratică a timpului de calcul care este cerută de algoritm naiv. Există, de obicei, un dezavantaj al hashingului sensibil la localitate, datorat la prezența negativelor false, adică a perechilor de elemente care sunt similare încă nu sunt incluse în setul de perechi pe care le examinăm, ci printr-o acordare atentă poate reduce fracția de negative negative prin creșterea numărului de perechi noi considera.
Ideea generală din spatele LSH este că vom hash items folosind multe diferite funcții hash. Aceste funcții hash nu sunt tipul convențional de hash funcții. Mai degrabă, acestea sunt atent concepute pentru a avea proprietatea pe care o împerechează sunt mult mai susceptibile de a ajunge în aceeași găleată a unei funcții hash dacă articolele sunt similare decât dacă nu sunt similare. Atunci putem examina doar perechi candidate, care sunt perechi de articole pentru care se termină în aceeași găleată cel puțin una dintre funcțiile hash.

Începem discuția noastră despre LSH cu o examinare a problemei găsirii
documente similare - cele care împărtășesc o mulțime de text comun. Mai întâi arătăm
cum să convertim documentele în seturi (secțiunea 3.2) într-un mod care ne permite să vizualizăm
similaritatea textuală a documentelor ca seturi cu o suprapunere mare. Mai precis,

---

măsurăm similaritatea seturilor prin similaritatea lor Jaccard, raportul dintre
dimensiunile intersecției și uniunii lor. Un al doilea truc cheie de care avem nevoie este minhashing
(Secțiunea 3.3), care este o modalitate de a converti seturi mari în reprezentanți mult mai mici
tări, numite semnături, care încă ne permit să estimăm îndeaproape Jaccard
similaritatea seturilor reprezentate. În cele din urmă, în secțiunea 3.4 vedem cum să aplicăm
ideea de colectare inerentă în LSH a semnăturilor.
În secțiunea 3.5 începem studiul nostru despre cum să aplicăm LSH la alte elemente decât
seturi. Considerăm noțiunea generală a unei măsuri de distanță care spune la ce
elementele de grad sunt similare. Apoi, în secțiunea 3.6 luăm în considerare ideea generală a
hashing sensibil la localitate, iar în secțiunea 3.7 vedem cum să facem LSH pentru unele date
alte tipuri decât seturile. Apoi, Secțiunea 3.8 examinează în detaliu mai multe aplicații
a ideii LSH. În cele din urmă, luăm în considerare în secțiunea 3.9 câteva tehnici de găsire
seturi similare care pot fi mai eficiente decât LSH atunci când gradul de similaritate
vrem este foarte mare.

3.1 Aplicații ale asemănării seturilor

Ne vom concentra inițial pe o noțiune particulară de „similitudine”: asemănarea
setează uitându-se la dimensiunea relativă a intersecției lor. Această noțiune de similitudine
se numește similaritate Jaccard, care este introdusă în secțiunea 3.1.1. Atunci noi
examinați câteva dintre utilizările de a găsi seturi similare. Acestea includ găsirea textuală
documente similare și filtrare colaborativă prin găsirea de clienți similari și
produse similare. Pentru a transforma problema asemănării textuale a documentelor
într-una dintre intersecțiile stabilite, folosim tehnica numită șindrilă, care este
subiectul secțiunii 3.2.

3.1.1 Similitudinea seturilor Jaccard

Similitudinea Jaccard a mulțimilor S și T este $|S \cap T| / |S \cup T|$, adică raportul
de mărimea intersecției dintre S și T la dimensiunea uniunii lor. Vom
denotați asemănarea Jaccard a lui S și T de către SIM (S, T).
Exemplul 3.1: În Fig. 3.1 vedem două seturi S și T. Există trei elemente
în intersecția lor și un total de opt elemente care apar în S sau T sau ambele.
Astfel, SIM (S, T) = 3/8. ✷

3.1.2 Asemănarea documentelor

O clasă importantă de probleme pe care similaritatea Jaccard le abordează bine este aceea
de a găsi documente similare textual într-un corpus mare, cum ar fi Web sau a
colecție de articole de știri. Ar trebui să înțelegem că aspectul asemănării
ne uităm aici la similitudinea la nivel de caracter, nu la „semnificație similară”, care
ne cere să examinăm cuvintele din documente și utilizările acestora. Această problemă
este, de asemenea, interesant, dar este abordat prin alte tehnici, pe care le-am sugerat în

---

0
0
1
1
0
0
1
1
0
0
1
1
0

0
1
1
0
0
1
1
0
0
1
1
0
0
1
1
0
0
1
1
T
S
Figura 3.1: Două seturi cu similaritate Jaccard 3/8
Secțiunea 1.3.1. Cu toate acestea, similaritatea textuală are și utilizări importante. Multe acestea implică găsirea de duplicate sau aproape de duplicate. În primul rând, să observăm acest lucru testarea dacă două documente sunt duplicate exacte este ușoară; compară doar două documente caracter cu caracter și dacă diferă vreodată, atunci nu sunt aceeași. Cu toate acestea, în multe aplicații, documentele nu sunt încă identice împărtășesc porțiuni mari din textul lor. Aici sunt cateva exemple:
Plagiat
Găsirea documentelor plagiate testează capacitatea noastră de a găsi similitudine textuale. The plagiatorul poate extrage doar unele părți ale unui document pentru propria sa persoană. El poate modificați câteva cuvinte și poate modifica ordinea în care propozițiile originalului apărea. Cu toate acestea, documentul rezultat poate conține în continuare o mare parte din original. Nu proces simplu de comparare a documentelor caracter cu caracter va detecta un plagiat sofisticat.
Pagini oglindă
Este obișnuit ca site-urile web importante sau populare să fie duplicate la un număr de gazde, pentru a împărți sarcina. Paginile acestor site-uri oglindă vor fi destul de asemănătoare, dar rareori sunt identice. De exemplu, fiecare poate conține informațiile asociate cu gazda lor particulară și fiecare dintre acestea ar putea avea linkuri către celelalte site-uri oglindă, dar nu către ele însele. Un fenomen înrudit este reutilizarea paginilor Web de la o clasă academică la alta. Aceste pagini ar putea include note de curs, sarcini și diapozitive de curs. Se pot schimba pagini similare numele cursului, anul și efectuați mici modificări de la an la an. Aceasta este important să puteți detecta pagini similare de acest gen, deoarece căutați motoarele produc rezultate mai bune dacă evită afișarea a două pagini aproape identice în prima pagină de rezultate.

Articole din aceeași sursă
Este obișnuit ca un reporter să scrie un articol de știri care să fie distribuit, spun prin Associated Press, la multe ziare, care apoi publică articolul de pe site-urile lor web. Fiecare ziar schimbă oarecum articolul. Aceștia pot tăia paragrafe sau chiar adăuga materiale proprii. Ei cel mai mult probabil va înconjura articolul cu propriile sigle, reclame și linkuri către alte articole la site-ul lor. Cu toate acestea, nucleul paginii fiecărui ziar va fi originalul articol. Agregatorii de știri, cum ar fi Știrile Google, încearcă să găsească toate versiunile acestora un articol, pentru a afișa doar unul, iar această sarcină necesită găsirea când două Paginile web sunt similare din punct de vedere textual, deși nu sunt identice. 1

3.1.3 Filtrarea colaborativă ca o problemă similară

O altă clasă de aplicații în care similitudinea seturilor este foarte importantă se numește filtrare colaborativă, un proces prin care recomandăm utilizatorilor elemente care au fost apreciat de alți utilizatori care au prezentat gusturi similare. Vom investiga

filtrarea colaborativă în detaliu în secțiunea 9.3, dar pentru moment să vedem câteva exemple comune.

Achiziții on-line

Amazon.com are milioane de clienți și vinde milioane de articole. Baza de date înregistrează ce articole au fost cumpărate de către care clienți. Putem spune două tomers sunt similare dacă seturile lor de articole cumpărate au o similaritate mare Jaccard. La fel, două articole care au seturi de cumpărători cu similaritate mare Jaccard va fi considerat similar. Rețineți că, deși ne-am putea aștepta ca site-urile oglindă să aibă Similitudine Jaccard peste 90%, este puțin probabil ca vreunul dintre doi clienți să aibă similitudinea cardului este atât de mare (cu excepția cazului în care au achiziționat un singur articol). Chiar și un Asemănarea Jaccard, cum ar fi 20%, ar putea fi suficient de neobișnuită pentru a identifica clienții cu gusturi similare. Aceeași observație este valabilă și pentru obiecte; Asemănările Jaccard au nevoie nu fi foarte mare pentru a fi semnificativ.

Filtrarea colaborativă necesită mai multe instrumente, pe lângă găsirea de elemente similare clienți sau articole, așa cum discutăm în capitolul 9. De exemplu, două Amazon clienții cărora le place ficțiunea științifică ar putea cumpăra fiecare cărți de ficțiune, dar doar câteva dintre acestea vor fi în comun. Cu toate acestea, prin combinarea descoperind cu clustering (Capitolul 7), am putea fi în măsură să descoperim că cărțile de ficțiune se aseamănă reciproc și le plasează într-un singur grup. Atunci, putem obțineți o noțiune mai puternică de similitudine cu clienții întrebând dacă au făcut-o achiziții în cadrul multora dintre aceleași grupuri.

1 Agregarea știrilor implică, de asemenea, găsirea articolelor care au aproximativ același subiect, chiar dacă nu asemănătoare din punct de vedere textual. Și această problemă poate genera o căutare de similaritate, dar necesită alte tehnici decât similitudinea seturilor Jaccard.

Evaluări de filme

Netflix înregistrează ce filme au închiriat fiecare dintre clienții săi, precum și evaluările atribuite acestor filme de către clienți. Putem considera filmele ca fiind similare dacă au fost închiriate sau apreciate de mulți dintre aceiași clienți și vezi clienți la fel de asemănători dacă au închiriat sau au apreciat foarte multe dintre aceleași filme. Aceleași observații pe care le-am făcut pentru Amazon mai sus se aplică în această situație: similitudinile nu trebuie să fie mari pentru a fi semnificative și gruparea filmelor pe genuri va ușura lucrurile.

Când datele noastre constau mai degrabă în evaluări decât în decizii binare (cumpărat / făcut nu cumpără sau mi-a plăcut / nu mi-a plăcut), nu ne putem baza doar pe seturi ca reprezentări ale clienți sau articole. Unele opțiuni sunt:

1. Ignorați perechile de clienți / filme cu rating redus; adică tratează aceste evenimente ca și cum clientul nu a vizionat niciodată filmul.
2. Când comparați clienții, imaginați-vă două elemente stabilite pentru fiecare film, ,,Plăcut” și „urât”. Dacă un client a apreciat un film la un nivel înalt, puneți „apreciat” pentru acel film din platoul clientului. Dacă au acordat un rating scăzut unui film, puneți ,,Urât” pentru acel film din platoul lor. Apoi, putem căuta Jaccard înalt similaritate între aceste seturi. Putem folosi un truc similar atunci când comparăm filme.
3. Dacă evaluările sunt de la 1 la 5 stele, introduceți un film în setul clientului de n ori, dacă au evaluat filmul cu stele n. Apoi, utilizați similaritatea Jaccard pentru pungi atunci când măsurând similitudinea clienților. Similitudinea Jaccard pentru genți B și C sunt definite prin numărarea unui element de n ori în intersecție dacă n este minimul de câte ori apare elementul în B și C. În uniune, numărăm elementul suma numărului de ori apare în B și în C. 2

Exemplul 3.2: Similitudinea sacilor sacilor {a, a, a, b} și {a, a, b, b, c} este 1/3. Intersecția contează a de două ori și b o dată, astfel încât dimensiunea sa este de 3. Dimensiunea unirea a două pungi este întotdeauna suma mărimilor celor două pungi, sau 9 în aceasta caz. Deoarece cea mai mare similitudine posibilă pentru saci este 1/2, scorul din 1/3 indică faptul că cele două pungi sunt destul de asemănătoare, așa cum ar trebui să reiasă dintr-un examinarea conținutului acestora. ✷

2 Deși uniunea pentru saci este în mod normal (de exemplu, în standardul SQL) definită ca având suma numărului de exemplare din fiecare dintre cele două pungi, această definiție provoacă o anumită inconsecvență cu similaritatea Jaccard pentru seturi. Conform acestei definiții a uniunii de saci, Jaccard maxim similitudinea este 1/2, nu 1, deoarece unirea unui set cu sine are de două ori mai multe elemente decât intersecția aceluiași set cu sine. Dacă preferăm să avem similaritatea Jaccard a unui set cu el însuși să fie 1, putem redefini unirea pungilor pentru ca fiecare element să pară maxim de câte ori apare în oricare dintre cele două pungi. Această schimbare oferă, de asemenea, un motiv rezonabil măsura asemănării pungii.

3.1.4 Exerciţii pentru secţiunea 3.1

Exerciţiul 3.1.1: Calculaţi asemănările Jaccard ale fiecărei perechi din următoarele trei seturi: {1, 2, 3, 4}, {2, 3, 5, 7} şi {2, 4, 6}.

Exerciţiul 3.1.2: Calculaţi asemănarea sacului Jaccard cu fiecare pereche de folie coborârea a trei pungi: {1, 1, 1, 2}, {1, 1, 2, 2, 3} şi {1, 2, 3, 4}.

!! Exerciţiul 3.1.3: Să presupunem că avem un set universal U de n elemente, iar noi alegeţi două subseturi S şi T la întâmplare, fiecare cu m din n elemente. Ce este valoarea aşteptată a asemănării Jaccard dintre S şi T?

3.2 Shingling de documente

Cel mai eficient mod de a reprezenta documentele ca seturi, în scopul identificării certificarea unor documente similare din punct de vedere lexical înseamnă a construi din document setul de şiruri scurte care apar în interiorul său. Dacă facem acest lucru, atunci documentele care împărtăşesc piesele scurte ca propoziţiile sau chiar frazele vor avea multe elemente comune în seturile lor, chiar dacă aceste propoziţii apar în ordine diferite în cele două documente menti. În această secţiune, introducem cea mai simplă şi mai obişnuită abordare, şindrilă, precum şi o variaţie interesantă.

3.2.1 k-zona zoster

Un document este un şir de caractere. Definiţi un k-şindrilă pentru a fi un document orice şir de lungime k găsit în document. Apoi, ne putem asocia cu fiecare document, setul de k-zoster care apare de una sau mai multe ori în interior acel document.

Exemplul 3.3: Să presupunem că documentul nostru D este şirul abcdabd şi că alegem k = 2. Atunci setul de 2 şindrilă pentru D este {ab, bc, cd, da, bd}.

Reţineţi că şirul ab apare de două ori în D, dar apare o singură dată ca o şindrilă. O variaţie de şindrilă produce mai degrabă o pungă decât un set, deci fiecare zoster ar apărea în rezultat de câte ori apare în document.

Cu toate acestea, nu vom folosi aici saci de şindrilă. ✷

Există mai multe opţiuni cu privire la modul în care spaţiul alb (gol, filă, linie nouă, etc.) este tratat. Probabil că are sens să înlocuiţi orice secvenţă a uneia sau mai multor caractere în spaţiu alb cu un singur gol. În acest fel, distingem zona zoster acoperă două sau mai multe cuvinte din cele care nu fac asta.

Exemplul 3.4: Dacă folosim k = 9, dar eliminăm spaţiul alb cu totul, atunci noi ar vedea o oarecare similaritate lexicală în propoziţiile „Avionul era pregătit pentru aterizare". şi „fundaşul a marcat un touchdown”. Cu toate acestea, dacă noi reţineţi semifabricatele, apoi primul are şindrilă care atinge dow şi aruncă în jos, în timp ce al doilea are touchdown. Dacă am elimina golurile, atunci amândouă ar avea aterizare. ✷

3.2.2 Alegerea dimensiunii şindrilelor

Putem alege k pentru a fi orice constantă ne place. Cu toate acestea, dacă alegem k prea mic, atunci ne-am aştepta ca majoritatea secvenţelor de k caractere să apară în majoritatea documentelor. Dacă da, atunci am putea avea documente ale căror seturi de şindrilă aveau similitude Jaccard ridicate. laritate, totuşi documentele nu aveau niciuna din aceleaşi propoziţii sau chiar fraze. La fel de un exemplu extrem, dacă folosim k = 1, majoritatea paginilor web vor avea cea mai mare parte a caractere comune şi alte câteva caractere, deci aproape toate paginile web vor avea asemănare mare.

Cât de mare ar trebui să fie k depinde de cât de lungi sunt documentele tipice şi de cum mare setul de personaje tipice este. Important de reţinut este:

- k ar trebui să fie ales suficient de mare încât să existe probabilitatea unei anumite şindrilă care apare în orice document dat este redus.

Astfel, dacă corpusul nostru de documente este e-mailuri, alegerea k = 5 ar trebui să fie în regulă. Pentru a vedea de ce, să presupunem că numai literele şi un caracter general în spaţiul alb apar pere în e-mailuri (deşi în practică, majoritatea caracterelor ASCII tipărite se poate aştepta să apară ocazional). Dacă da, atunci ar exista 27^5 = 14.348.907 şindrilă posibilă. Deoarece e-mailul tipic este mult mai mic decât 14 lungime de milioane de caractere, ne-am aştepta ca k = 5 să funcţioneze bine, şi într-adevăr funcţionează. Cu toate acestea, calculul este ceva mai subtil. Cu siguranţă, mai mult de 27 de caracteristici tere apar în e-mailuri, cu toate acestea, toate caracterele nu apar cu probe egale bilitate. Literele obişnuite şi spaţiile libere domină, în timp ce „z” şi alte litere care au o valoare punctuală ridicată în Scrabble sunt rare. Astfel, chiar şi e-mailurile scurte vor avea

multe 5 șindrilă constând din litere comune și șanse de a nu avea legătură
e-mailurile care împărtășesc aceste sindrile comune sunt mai mari decât ar fi implicat de
calcul în paragraful de mai sus. O regulă bună este să ne imaginăm asta
există doar 20 de caractere și estimați numărul de k-zoster ca 20^k. Pentru
documentele mari, cum ar fi articolele de cercetare, alegerea k = 9 este considerată sigură.

3.2.3 Hinging Shinging

În loc să folosim șiruri sub formă de șindrilă, putem alege o funcție hash
care mapează șiruri de lungime k la un anumit număr de găleți și tratează rezultatul
numărul găleții ca șindrilă. Setul care reprezintă un document este apoi
set de numere întregi care sunt numere de cupă ale unuia sau mai multor k-zoster care apar
în document. De exemplu, am putea construi setul de 9 șindrilă pentru a
documentează și apoi mapează fiecare dintre cele 9 șindrilă la un număr de cupă din
intervalul 0 la 2^{32} - 1. Astfel, fiecare șindrilă este reprezentată în schimb de patru octeți
din nouă. Nu numai că datele au fost compactate, dar acum le putem manipula
(hash) șindrilă prin operații de mașină cu un singur cuvânt.
Observați că putem diferenția mai bine documentele dacă folosim 9 șindrilă și
Hash-le până la patru octeți decât pentru a utiliza 4-șindrilă, chiar dacă spațiul folosit
a reprezenta o șindrilă este la fel. Motivul a fost atins în secțiunea 3.2.2.
Dacă folosim 4 șindrilă, cele mai multe secvențe de patru octeți sunt puțin probabil sau imposibil

---

Pagina 108

găsiți în documente tipice. Astfel, numărul efectiv al diferitelor zoster este
mult mai puțin de $2^{32} - 1$. Dacă, la fel ca în secțiunea 3.2.2, presupunem că sunt doar 20 de caractere
frecvent în textul în limba engleză, apoi numărul diferitelor 4-șindrilă care este probabil
a se produce este doar $(20)^4$ = 160.000. Cu toate acestea, dacă folosim 9 șindrilă, există multe
peste 2^{32} de zona zoster probabilă. Când le hashem până la patru octeți, putem
așteptați aproape orice secvență de patru octeți să fie posibilă, așa cum sa discutat în
Secțiunea 1.3.2.

3.2.4 Zoster construit din cuvinte

O formă alternativă de șindrilă s-a dovedit eficientă pentru problema identificării
articole de știri similare, menționate în secțiunea 3.1.2. Distincția exploatabilă pentru
această problemă este că articolele de știri sunt scrise într-un stil destul de diferit decât
sunt alte elemente care apar de obicei pe pagina cu articolul. Știri
articolele și majoritatea prozelor au multe cuvinte de oprire (a se vedea secțiunea 1.3.1), cel mai mult
cuvinte uzuale precum „și”, „tu”, „către” și așa mai departe. În multe aplicații,
vrem să ignorăm cuvintele stop, deoarece acestea nu ne spun nimic util
articolul, precum subiectul acestuia.
Cu toate acestea, pentru problema găsirii unor articole de știri similare, s-a constatat că
definirea unei șindrilă ca fiind un cuvânt de oprire urmat de următoarele două cuvinte, indiferent
dacă au fost sau nu cuvinte de oprire, au format un set util de șindrilă. The
avantajul acestei abordări este că articolul de știri ar contribui mai mult
zoster la setul care reprezintă pagina web decât ar fi elementele din jur
menti. Amintiți-vă că scopul exercițiului este de a găsi pagini care au aceleași
articole, indiferent de elementele din jur. Prin părtinirea setului de șindrilă
în favoarea articolului, pagini cu același articol și înconjurătoare diferite
materialul are o asemănare Jaccard mai mare decât paginile cu același înconjurător
material dar cu un articol diferit.
Exemplul 3.5: un anunț poate avea textul simplu „Cumpărați Sudzo”. Cu toate acestea, a
articolul de știri cu aceeași idee ar putea citi ceva de genul „Un purtător de cuvânt
căci Sudzo Corporation a dezvăluit astăzi că studiile au arătat că este
bine pentru oameni să cumpere produse Sudzo. ” Aici, am pus cursiv toate
probabil să oprească cuvintele, deși nu există un număr stabilit din cele mai frecvente cuvinte
care ar trebui considerate cuvinte stop. Primele trei sindrile realizate dintr-un
stop word și următoarele două următoarele sunt:
Un purtător de cuvânt pentru
pentru Sudzo
Corporația Sudzo
Există nouă șindrilă din propoziție, dar niciuna din „anunț”. ✷

3.2.5 Exerciții pentru secțiunea 3.2

Exercițiul 3.2.1: Care sunt primele zece 3 șindrilă din prima teză din sec.
secțiunea 3.2?

Pagina 109

Exercițiul 3.2.2: Dacă folosim șindrilele bazate pe cuvinte stop din secțiunea 3.2.4 și luăm cuvintele stop pentru a fi toate cuvintele a trei sau mai puține litere, apoi ce sunt zona zoster din prima teză a secțiunii 3.2?

Exercițiul 3.2.3: Care este cel mai mare număr de k-zoster un document de n octeți pot avea? Puteți presupune că dimensiunea alfabetului este suficient de mare că numărul șirurilor posibile de lungime k este cel puțin n.

3.3 Rezumate ale seturilor care păstrează similitudinea

Seturile de șindrilă sunt mari. Chiar dacă le hashem la patru octeți fiecare, spațiul necesar pentru a stoca un set este încă de aproximativ patru ori mai mare decât spațiul ocupat de document. Dacă avem milioane de documente, este posibil să nu fie posibilă stocarea tuturor documentelor seturi de șindrilă în memoria principală. [3]

Scopul nostru în această secțiune este de a înlocui seturile mari cu reprezentanți mult mai mici. numite „semnături”. Proprietatea importantă de care avem nevoie pentru semnături este că putem compara semnăturile a două seturi și putem estima simularea Jaccard ilaritatea seturilor subiacente numai de la semnături. Nu este posibil ca. semnăturile oferă similaritatea exactă a seturilor pe care le reprezintă, dar estimarea colegii pe care îi oferă sunt apropiați, iar cu cât semnăturile sunt mai mari, cu atât sunt mai exacte estimările. De exemplu, dacă înlocuim seturile de șindrilă de 200.000 de octeți care derivă din documente de 50.000 de octeți prin semnături de 1000 de octeți, putem ajung de obicei la câteva procente.

3.3.1 Reprezentarea matricială a seturilor

Înainte de a explica cum este posibil să construiți semnături mici din mari seturi, este util să vizualizați o colecție de seturi ca matrice caracteristică a acestora. Coloanele matricei corespund seturilor, iar rândurile corespund elemente ale mulțimii universale din care sunt extrase elemente ale mulțimilor. Acolo este 1 în rândul r și coloana c dacă elementul pentru rândul r este un membru al setului pentru coloana c. În caz contrar, valoarea în poziția (r, c) este 0.

Element
S_1
S_2
S_3
S_4
A
1
0
0
1
b
0
0
1
0
c
0
1
0
1
d
1
0
1
1
e
0
0
1
0

Figura 3.2: O matrice reprezentând patru seturi

[3] Există o altă preocupare serioasă: chiar dacă seturile se încadrează în memoria principală, numărul de perechi poate fi prea mare pentru noi pentru a evalua similitudinea fiecărei perechi. Luăm soluția pentru această problemă în secțiunea 3.4.

Exemplul 3.6: În Fig. 3.2 este un exemplu de matrice reprezentând seturi alese din mulțimea universală {a, b, c, d, e}. Aici, S_1 = {a, d}, S_2 = {c}, S_3 = {b, d, e},

și S_4 = {a, c, d}. Rândul de sus și coloanele din stânga nu fac parte din matrice, dar sunt prezente doar pentru a ne reaminti ce reprezintă rândurile și coloanele. ✷

Este important să ne amintim că este puțin probabil ca matricea caracteristică să fie modul în care sunt stocate datele, dar este util ca o modalitate de a vizualiza datele. Pentru un motiv pentru a nu stoca date ca o matrice, aceste matrice sunt aproape întotdeauna rare (au mult mai multe 0 decât 1) în practică. Economisește spațiu pentru a reprezenta un matricea rară de 0 și 1 după pozițiile în care apar 1. Pentru altul motiv, datele sunt de obicei stocate într-un alt format în alte scopuri.

De exemplu, dacă rândurile sunt produse, iar coloanele sunt clienții, reprezentați după setul de produse pe care le-au cumpărat, atunci aceste date ar apărea într-adevăr într-un tabelul de achiziții din baza de date. Un tuplu din acest tabel ar lista elementul, cumpărător și, probabil, alte detalii despre achiziție, cum ar fi data și cardul de credit utilizat.

3.3.2 Minhashing

Semnăturile pe care dorim să le construim pentru seturi sunt compuse din rezultatele unui un număr mare de calcule, să zicem câteva sute, fiecare dintre ele fiind un „minhash” a matricei caracteristice. În această secțiune, vom învăța cum este un minhash calculat în principiu, iar în secțiunile ulterioare vom vedea cum un bun marea la minhash este calculată în practică.

Pentru a minimiza un set reprezentat de o coloană a matricei caracteristice, alegeți o permutare a rândurilor. Valoarea minhash a oricărei coloane este numărul de primul rând, în ordinea permutată, în care coloana are 1.

Exemplul 3.7: Să presupunem că alegem ordinea rândurilor beadc pentru matrice din Fig. 3.2. Această permutare definește o funcție minhash h care mapează seturile la rânduri. Să calculăm valoarea minhash a mulțimii S_1 conform h. Primul coloana, care este coloana pentru setul S_1, are 0 în rândul b, deci trecem la rândul e, al doilea în ordinea permutată. Există din nou un 0 în coloana pentru S_1, deci trecem la rândul a, unde găsim un 1. Astfel. h (S_1) = a.

Element
S_1
S_2
S_3
S_4
b
0
0
1
0
e
0
0
1
0
A
1
0
0
1
d
1
0
1
1
c
0
1
0
1

Figura 3.3: O permutare a rândurilor din Fig. 3.2

Deși nu este posibil din punct de vedere fizic permutarea unei caracteristici foarte mari matrice, funcția minhash h reordonează implicit rândurile matricei de

Fig. 3.2 deci devine matricea din Fig. 3.3. În această matrice, putem citi valorile lui h prin scanarea de sus până ajungem la un 1. Astfel, vedem că h (S_2) = c, h (S_3) = b și h (S_4) = a. ✷

3.3.3 Minhashing și similaritatea Jaccard

Există o legătură remarcabilă între minhashing și similaritatea Jaccard dintre seturile care sunt minhashed.

- Probabilitatea ca minhash să funcționeze pentru o permutare aleatorie a rândurile produc aceeași valoare pentru două seturi egal cu asemănarea Jaccard dintre acele seturi.

Pentru a vedea de ce, trebuie să ne imaginăm coloanele pentru aceste două seturi. Dacă restrângem noi înșine la coloanele pentru seturile S_1 și S_2, apoi rândurile pot fi împărțite în trei clase:

1. Tipul X rândurile au 1 în ambele coloane.
2. Tastați Y rândurile au 1 într-una dintre coloane și 0 în cealaltă.
3. Tipul Z rândurile au 0 în ambele coloane.

Deoarece matricea este rară, majoritatea rândurilor sunt de tip Z. Cu toate acestea, este raportul a numerelor de rândurile de tip X și de tip Y care determină atât $\text{SIM}(S_1, S_2)$ și probabilitatea ca $h(S_1) = h(S_2)$. Să existe x rânduri de tip X și y rânduri de tip Y. Apoi $\text{SIM}(S_1, S_2) = x / (x + y)$. Motivul este că x este dimensiunea din $S_1 \cap S_2$ și $x + y$ este mărimea lui $S_1 \cup S_2$.

Acum, considerați probabilitatea ca $h(S_1) = h(S_2)$. Dacă ne imaginăm rândurile permutat aleatoriu, și continuăm de sus, probabilitatea că vom întâlnim un rând de tip X înainte să întâlnim un rând de tip Y este $x / (x + y)$. Dar dacă primul rând din partea de sus, altul decât rândurile de tip Z, este un rând de tip X, apoi cu siguranță $h(S_1) = h(S_2)$. Pe de altă parte, dacă primul rând, altul decât un rând de tip Z pe care îl întâlnim este un rând de tip Y, apoi setul cu 1 primește acel rând ca minhash valoare. Cu toate acestea, setul cu 0 în acel rând devine cu siguranță un rând mai jos lista permutată. Astfel, știm $h(S_1) = h(S_2)$ dacă întâlnim mai întâi un rând de tip Y. Concluzionăm probabilitatea ca $h(S_1) = h(S_2)$ să fie $x / (x + y)$, care este și Similitudinea Jaccard a S_1 și S_2.

3.3.4 Semnături Minhash

Gândiți-vă din nou la o colecție de seturi reprezentate de matricea lor caracteristică M. Pentru a reprezenta seturi, alegem la întâmplare un număr n de permutări ale rânduri de M. Poate că 100 de permutări sau câteva sute de permutări vor face. Apelați funcțiile minhash determinate de aceste permutații $h_1, h_2, ..., h_n$. Din coloana care reprezintă setul S, construiți semnătura minhash pentru S, vectorul $[h_1(S), h_2(S), ..., h_n(S)]$. În mod normal, reprezentăm această listă de valori hash

---

ca o coloană. Astfel, putem forma din matricea M o matrice de semnătură, în care coloana a-a a lui M este înlocuită cu semnătura minhash pentru (setul) a doua coloană.

Rețineți că matricea semnăturii are același număr de coloane ca M, dar doar n rânduri. Chiar dacă M nu este reprezentat în mod explicit, ci în unele comprimate formă potrivită pentru o matrice rară (de exemplu, prin locațiile 1), este normal pentru ca matricea semnăturii să fie mult mai mică decât M.

Remarcabilul despre matricile de semnături este că le putem folosi coloane pentru a estima similaritatea Jaccard a seturilor care corespund coloane din matricea semnăturii. Prin teorema demonstrată în secțiunea 3.3.3, știm că probabilitatea ca două coloane să aibă aceeași valoare într-un rând dat de matricea semnăturii este egală cu asemănarea Jaccard a seturilor corespunzătoare acele coloane. Mai mult, din moment ce permutațiile pe care valorează minhash au fost alese independent, ne putem gândi la fiecare rând al semnăturii matricea ca experiment independent. Astfel, numărul așteptat de rânduri în care două coloane sunt de acord este egal cu asemănarea Jaccard cu cea corespunzătoare seturi. Mai mult, cu cât folosim mai multe minhashings, adică cu atât mai multe rânduri în matrice naturii, cu atât este mai mică eroarea așteptată în estimarea Jaccard asemănarea va fi.

3.3.5 Calcularea semnăturilor Minhash în practică

Nu este fezabil să permutăm în mod explicit o matrice caracteristică mare. Chiar și culegerea o permutare aleatorie de milioane sau miliarde de rânduri necesită mult timp și sortarea necesară a rândurilor ar dura chiar mai mult timp. Astfel, permutat matrici precum cea sugerată în Fig. 3.3, deși atractive din punct de vedere conceptual, nu sunt implementabil.

Din fericire, este posibil să simulați efectul unei permutări aleatorii prin o funcție hash aleatorie care mapează numerele de rânduri la câte găleți există sunt rânduri. O funcție hash care mapează numerele întregi 0, 1, ..., k - 1 la numerele bucket De la 0 la $k - 1$ va mapa de obicei unele perechi de numere întregi la aceeași bucket și

lăsați alte găleți neumplute. Cu toate acestea, diferența nu este importantă atâta timp cât
k este mare și nu există prea multe coliziuni. Putem menține ficțiunea
că funcția noastră hash h „permite” rândul r în poziția h (r) în permutat
Ordin.
Astfel, în loc să alegem n permutări aleatorii de rânduri, alegem n aleatoriu
funcțiile hash alese h_1 , h_2 , ..., h_n pe rânduri. Construim semnătura
matrice luând în considerare fiecare rând în ordinea dată. Fie SIG (i, c) elementul
a matricei de semnătură pentru funcția hash și coloana c. Inițial, setează
SIG (i, c) la ∞pentru toate i și c. Ne ocupăm de rândul r făcând următoarele:
1. Calculați h_1 (r), h_2 (r), ..., h_n (r).
2. Pentru fiecare coloană c faceți următoarele:
(a) Dacă c are 0 în rândul r, nu faceți nimic.

(b) Cu toate acestea, dacă c are 1 în rândul r, atunci pentru fiecare i = 1, 2, ..., n setați SIG (i, c)
la cea mai mică dintre valoarea curentă a SIG (i, c) și h_i (r).
Rând
S_1
S_2
S_3
S_4
x +1 mod5 3x +1 mod5
0
1
0
0
1
1
1
1
0
0
1
0
2
4
2
0
1
0
1
3
2
3
1
0
1
1
4
0
4
0
0
1
0
0
3
Figura 3.4: Funcții Hash calculate pentru matricea din Fig. 3.2
Exemplul 3.8: Să reconsiderăm matricea caracteristică din Fig. 3.2, care
reproducem cu câteva date suplimentare ca în Fig. 3.4. Am înlocuit
litere numind rândurile după numere întregi de la 0 la 4. De asemenea, am ales două hash
funcții: h_1 (x) = x + 1 mod 5 și h_2 (x) = 3x + 1 mod 5. Valorile acestor
două funcții aplicate numerelor rândurilor sunt date în ultimele două coloane ale
Fig. 3.4. Observați că aceste funcții hash simple sunt permutări adevărate ale
rânduri, dar o permutare adevărată este posibilă doar deoarece numărul de rânduri, 5, este
un prim. În general, vor exista coliziuni, unde două rânduri obțin același hash

valoare.
Acum, să simulăm algoritmul pentru calcularea matricei de semnături.
Inițial, această matrice este formată din toate ∞-urile:
S_1
S_2
S_3
S_4
h_1
∞
∞
∞
∞
h_2
∞
∞
∞
∞
În primul rând, considerăm rândul 0 din Fig. 3.4. Vedem că valorile $h_1(0)$ și
$h_2(0)$ sunt ambele 1. Rândul numerotat 0 are 1 în coloane pentru seturile S_1 și
S_4, deci numai aceste coloane din matricea semnăturii se pot modifica. Deoarece 1 este mai mic decât
∞, modificăm de fapt ambele valori din coloanele pentru S_1 și S_4. Curentul
estimarea matricei semnăturii este astfel:
S_1
S_2
S_3
S_4
h_1
1
∞
∞
1
h_2
1
∞
∞
1
Acum, trecem la rândul numerotat 1 din Fig. 3.4. Acest rând are 1 numai în
S_3, iar valorile sale hash sunt $h_1(1) = 2$ și $h_2(1) = 4$. Astfel, setăm SIG (1, 3) la 2
și SIG (2, 3) până la 4. Toate celelalte intrări de semnături rămân așa cum sunt deoarece lor
coloanele au 0 în rândul numerotat 1. Noua matrice de semnături:
S_1
S_2
S_3
S_4
h_1
1
∞
2
1
h_2
1
∞
4
1

Rândul din Fig. 3.4 numerotat 2 are 1 în coloane pentru S_2 și S_4 și
valorile sale hash sunt $h_1(2) = 3$ și $h_2(2) = 2$. Am putea modifica valorile din
semnătura pentru S_4, dar valorile din această coloană a matricei de semnături, [1, 1], sunt
fiecare mai mică decât valorile hash corespunzătoare [3, 2]. Cu toate acestea, încă din coloană
pentru că S_2 are încă ∞, îl înlocuim cu [3, 2], rezultând:

S_1
S_2
S_3
S_4
h_1
1
3
2
1
h_2
1
2
4
1
Urmează rândul numerotat 3 din Fig. 3.4. Aici, toate coloanele, cu excepția S_2, au 1, iar valorile hash sunt $h_1(3) = 4$ și $h_2(3) = 0$. Valoarea 4 pentru h_1 depășește ceea ce este deja în matricea semnăturii pentru toate coloanele, așa că nu vom face modificați orice valori din primul rând al matricei de semnături. Cu toate acestea, valoarea 0 pentru h_2 este mai mic decât ceea ce este deja prezent, așa că coborâm SIG (2, 1), SIG (2, 3) și SIG (2, 4) la 0. Rețineți că nu putem coborî SIG (2, 2) deoarece coloana pentru S_2 în Fig. 3.4 are 0 în rândul pe care îl luăm în considerare în prezent. Semnătura rezultată matrice:
S_1
S_2
S_3
S_4
h_1
1
3
2
1
h_2
0
2
0
0
În cele din urmă, considerați rândul din Fig. 3.4 numerotat 4. $h_1(4) = 0$ și $h_2(4) = 3$. Deoarece rândul 4 are 1 numai în coloana pentru S_3, comparăm doar curentul coloană de semnătură pentru setul respectiv, [2, 0] cu valorile hash [0, 3]. De la 0 <2, noi schimbați SIG (1, 3) la 0, dar din 3> 0 nu schimbăm SIG (2, 3). Finala matricea semnăturii este:
S_1
S_2
S_3
S_4
h_1
1
3
0
1
h_2
0
2
0
0
Putem estima asemănările Jaccard ale seturilor subiacente din aceasta matricea semnăturii. Observați că coloanele 1 și 4 sunt identice, deci presupunem că SIM (S_1, S_4) = 1,0. Dacă ne uităm la Fig. 3.4, vedem că adevărata similitudine Jaccard din S_1 și S_4 este 2/3. Amintiți-vă că fracția de rânduri care sunt de acord în matricea semnăturii este doar o estimare a adevăratei similitudini Jaccard și aceasta exemplu este mult prea mic pentru ca legea numărului mare să asigure că estimările sunt apropiate. Pentru exemple suplimentare, coloanele de semnătură pentru S_1 și S_3 sunt de acord în jumătate de rânduri (adevărată similaritate 1/4), în timp ce semnăturile lui S_1 și S_2 estimează 0 ca asemănare Jaccard (valoarea corectă). ✷

3.3.6 Accelerarea Minhashing-ului

Procesul de minhashing consumă mult timp, deoarece trebuie să examinăm întreaga matrice K-rând M pentru fiecare funcție minhash dorită. Să ne întoarcem mai întâi

la modelul Secțiunii 3.3.2, unde ne imaginăm că rândurile sunt permutate de fapt.
Dar pentru a calcula o funcție minhash pe toate coloanele, nu vom merge pe toate drumul până la sfârșitul permutării, dar uitați-vă doar la primul m din k rânduri. Dacă facem m mic în comparație cu k, reducem munca cu mare factor, k / m.

Cu toate acestea, există un dezavantaj în a face m mic. Atâta timp cât fiecare coloană are cel puțin un 1 în primele m rânduri în ordine permutată, rândurile după luna nu au niciun efect asupra valorii minhash și, la fel de bine, nu pot fi luate în considerare. Dar ce se întâmplă dacă unele coloane sunt toate-0 în primele m rânduri? Nu avem valoare minhash pentru acele coloane și va trebui să folosim un simbol special, pentru care noi va folosi ∞.

Când examinăm semnăturile minhash din două coloane pentru a estima împerechează asemănarea Jaccard a seturilor lor subiacente, așa cum avem în secțiunea 3.3.4 să se ia în considerare posibilitatea ca una sau ambele coloane să aibă ∞ca lor valoarea minhash pentru unele componente ale semnăturii. Există trei cazuri:

1. Dacă niciuna dintre coloane nu are ∞într-un rând dat, atunci nu este necesară nicio modificare. Numărați acest rând ca exemplu de valori egale dacă cele două valori sunt același și ca exemplu de valori inegale, dacă nu.
2. O coloană are ∞și cealaltă nu. În acest caz, dacă am fi folosit toate rândurile matricei permutate M originale, coloana care are ∞ i s-ar fi dat în cele din urmă un anumit număr de rând și acel număr cu siguranță nu va fi unul dintre primele m rânduri în ordinea permutată. Cu exceptia cealaltă coloană are o valoare care este unul dintre primele m rânduri. Astfel, noi cu siguranță avem un exemplu de valori inegale ale minhash-ului și numărăm acest rând a matricei de semnături ca atare exemplu.
3. Acum, să presupunem că ambele coloane au ∞în rând. Apoi, în originalul permutat matrice M, primele m rânduri ale ambelor coloane erau toate 0. Astfel nu avem informații despre similaritatea Jaccard a seturilor corespunzătoare; acea similitudinea este doar o funcție a ultimelor k - m rânduri, pe care le-am ales să nu te uiți. Prin urmare, socotim acest rând al matricei de semnături ca fiind nici un exemplu de valori egale, nici de valori inegale.

Atâta timp cât al treilea caz, în care ambele coloane au ∞, este rar, obținem aproape la fel de multe exemple de mediat pe cât există rânduri în matricea semnăturii. Acest efect va reduce precizia estimărilor noastre privind distanța Jaccard oarecum, dar nu mult. Și din moment ce acum putem calcula minhash valori pentru toate coloanele mult mai rapid decât dacă am examina toate rândurile lui M, ne putem permite timpul să mai aplicăm câteva funcții minhash. Devenim uniformi o precizie mai bună decât inițial și o facem mai repede decât înainte.

3.3.7 Accelerarea utilizând funcțiile Hash

Ca și până acum, există motive pentru a nu permuta fizic rândurile în acest mod presupus în secțiunea 3.3.6. Cu toate acestea, ideea permutărilor adevărate face mai multe

sens în contextul secțiunii 3.3.6 decât în secțiunea 3.3.2. Motivul este că nu trebuie să construim o permutare completă a k elemente, ci doar alegeți un număr mic m din rândurile k, apoi alegeți o permutare aleatorie din acele rânduri. În funcție de valoarea lui m și de modul în care este stocată matricea M, aceasta ar putea avea sens să urmăm literal algoritmul sugerat de secțiunea 3.3.6.

Cu toate acestea, este mai probabil să fie necesară o strategie asemănătoare secțiunii 3.3.5. Acum, rândurile lui M sunt fixe și nu permutate. Alegem o funcție hash care hashează numerele rândurilor și calculează valorile hash doar pentru primele m rânduri. Adică urmăm algoritmul Secțiunii 3.3.5, dar numai până când ajungem la rândul m, după care ne oprim și, și pentru fiecare coloană, luăm minimul valoarea hash văzută până acum ca valoarea minhash pentru acea coloană.

Deoarece unele coloane pot avea 0 în toate m rândurile, este posibil ca unele dintre valorile minhash vor fi ∞. Presupunând că m este suficient de mare încât ∞minhash valorile sunt rare, obținem încă o estimare bună a similarității Jaccard a seturilor de comparând coloanele matricei de semnături. Să presupunem că T este ansamblul elementelor a mulțimii universale care sunt reprezentate de primele m rânduri de matrice M. Fie S_1 și S_2 sunt mulțimile reprezentate de două coloane ale lui M. Apoi primele m rânduri lui M reprezintă mulțimile $S_1 \cap T$ și $S_2 \cap T$. Dacă ambele aceste mulțimi sunt goale (adică, ambele coloane sunt toate-0 în primele lor rânduri m), atunci această funcție minhash va fi

∞ în ambele coloane și va fi ignorat atunci când se estimează similaritatea Jaccard
a seturilor subiacente ale coloanelor.
Dacă cel puțin una dintre mulțimile $S_1 \cap T$ și $S_2 \cap T$ nu este goală, atunci proba-
bilitatea celor două coloane cu valori egale pentru această funcție minhash este
Asemănarea Jaccard a acestor două seturi, adică
$| S_1 \cap S_2 \cap T |$
$| (S_1 \cup S_2) \cap T |$
Atâta timp cât T este ales pentru a fi un subset aleatoriu al setului universal, cel așteptat
valoarea acestei fracții va fi aceeași cu asemănarea Jaccard a S_1 și S_2.
Cu toate acestea, vor exista unele variații aleatorii, deoarece, în funcție de T, am putea
găsiți mai mult sau mai puțin decât un număr mediu de rânduri de tip X (1 în ambele coloane)
și / sau tastați rânduri Y (1 într-o coloană și 0 în cealaltă) printre primii m
rânduri de matrice M.
Pentru a atenua această variație, nu folosim același set T pentru fiecare minhashing
pe care o facem. Mai degrabă, împărțim rândurile lui M în grupuri k / m. [4] Apoi pentru fiecare
funcția hash, calculăm o valoare minhash examinând doar primul m
rânduri de M, o valoare minhash diferită examinând doar al doilea m rând,
și așa mai departe. Obținem astfel valori k / m minhash dintr-o singură funcție hash și
o singură trecere peste toate rândurile lui M. De fapt, dacă k / m este suficient de mare, putem
obțineți toate rândurile matricei de semnături de care avem nevoie printr-o singură funcție hash
aplicat fiecărui subset de rânduri de M.
[4] În cele ce urmează, presupunem că m împarte k în mod egal, pentru comoditate. Nu este important, așa cum
atâta timp cât k / m este mare, dacă unele rânduri nu sunt incluse în niciun grup deoarece k nu este un număr întreg
multiplu de m.

Mai mult, folosind fiecare dintre rândurile lui M pentru a calcula unul dintre aceste minhash
valori, avem tendința de a echilibra erorile în estimarea similarității Jaccard
datorită oricărui subset specific al rândurilor. Adică asemănarea Jaccard
din S_1 și S_2 determină raportul rândurilor de tip X și de tip Y. Tot tipul X
rândurile sunt distribuite între seturile de rânduri k / m și, de asemenea, rândurile de tip Y.
Astfel, în timp ce un set de m rânduri poate avea mai multe tipuri de rând decât media,
trebuie să existe apoi un alt set de m rânduri cu mai puțin decât media dintre acestea
acelasi tip.
Exemplul 3.9: În Fig. 3.5 vedem o matrice reprezentând trei seturi S_1, S_2 și
S_3, cu un set universal de opt elemente; adică k = 8. Să alegem m = 4, deci unul
trecerea prin rânduri produce două valori minhash, una bazată pe primele patru
rânduri și celălalt pe al doilea patru rânduri.

S_1	S_2	S_3
0	0	1
0	1	0
0	1	1
0	1	1
0	1	1
0	1	1
0	1	1
0	0	0
0	1	1
1	0	0
0	0	0
1	0	0
0	0	0
0	0	0

Figura 3.5: O matrice booleană reprezentând trei seturi
În primul rând, rețineți că asemănările Jaccard ale celor trei seturi sunt SIM (S_1, S_2) =

1/2, SIM (S_1, S_3) = 1/5 și SIM (S_2, S_3) = 1/2. Uită-te acum la primele patru numai rânduri. Oricare ar fi funcția hash pe care o folosim, valoarea minhash pentru S_1 va fi ∞, valoarea minhash pentru S_2 va fi valoarea hash a celui de-al 4-lea rând și valoarea minhash pentru S_3 va fi cea mai mică dintre valorile hash pentru a treia și al patrulea rând. Astfel, valorile minhash pentru S_1 și S_2 nu vor fi niciodată de acord. Acea are sens, deoarece dacă T este ansamblul elementelor reprezentate de primele patru rânduri, atunci $S_1 \cap T = \varnothing$ și, prin urmare, SIM ($S_1 \cap T$, $S_2 \cap T$) = 0. Cu toate acestea, în al doilea patru rânduri, asemănarea Jaccard a S_1 și S_2 limitată la elemente reprezentat de ultimele patru rânduri este 2/3.

Concluzionăm că dacă generăm semnături constând din două valori minhash folosind această funcție hash, una bazată pe primele patru rânduri și a doua bazată pe ultimele patru rânduri, numărul așteptat de meciuri pe care le obținem între semnăturile pentru S_1 și S_2 sunt media de 0 și 2/3 sau 1/3. Întrucât actualul Similitudinea Jaccard a S_1 și S_2 este 1/2, există o eroare, dar nu prea mare eroare. În exemple mai mari, unde valorile minhash se bazează pe mult mai mult decât patru rânduri, eroarea așteptată se va apropia de zero.

În mod similar, putem vedea efectul împărțirii rândurilor pe celelalte două perechi de coloane. Între S_1 și S_3, jumătatea superioară reprezintă seturi cu un Jaccard

---

similaritatea de 0, în timp ce jumătatea de jos reprezintă seturi cu o similaritate Jaccard 1/3. Prin urmare, numărul așteptat de meciuri în semnăturile S_1 și S_3 este media acestora, sau 1/6. Aceasta se compară cu adevărata similitudine Jaccard SIM (S_1, S_3) = 1/5. În cele din urmă, atunci când comparăm S_2 și S_3, observăm că Similitudinea Jaccard a acestor coloane în primele patru rânduri este 1/2, la fel și a lor Similitudine Jaccard în cele patru rânduri de jos. Media, 1/2, este de asemenea de acord exact cu SIM (S_2, S_3) = 1/2. ✷

3.3.8 Exerciții pentru secțiunea 3.3

Exercițiul 3.3.1: Verificați teorema din secțiunea 3.3.3, care leagă Jac-similitudinea cardului cu probabilitatea de minhashing la valori egale, pentru ular caz din Fig. 3.2.

(a) Calculați asemănarea Jaccard a fiecărei perechi de coloane din Fig. 3.2.

! (b) Calculați, pentru fiecare pereche de coloane din acea cifră, fracția din 120 permutările rândurilor care fac ca cele două coloane să fie hash la fel valoare.

Exercițiul 3.3.2: Folosind datele din Fig. 3.4, adăugați la semnăturile fișierului coloanele valorilor următoarelor funcții hash:

(a) $h_3(x) = 2x + 4 \bmod 5$.

(b) $h_4(x) = 3x - 1 \bmod 5$.

Element
S_1
S_2
S_3
S_4
0
0
1
0
1
1
0
1
0
0
2
1
0
0
1
3
0
0
1
0
4
0

0
1
1
5
1
0
0
0
Figura 3.6: Matrice pentru exercițiul 3.3.3
Exercițiul 3.3.3: În Fig. 3.6 este o matrice cu șase rânduri.
(a) Calculați semnătura minhash pentru fiecare coloană dacă folosim următoarele
trei funcții hash: $h_1(x) = 2x + 1 \bmod 6$; $h_2(x) = 3x + 2 \bmod 6$;
$h_3(x) = 5x + 2 \bmod 6$.
(b) Care dintre aceste funcții hash sunt permutări adevărate?

Pagina 119

(c) Cât de apropiate sunt asemănările Jaccard estimate pentru cele șase perechi de coloane
la adevăratele asemănări Jaccard?
! Exercițiul 3.3.4: Acum că știm că similitudinea Jaccard este legată de probabilitatea
bilitatea că două setează minhash la aceeași valoare, reconsiderați exercițiul 3.1.3. Poate sa
utilizați această relație pentru a simplifica problema calculării așteptărilor
Similitudinea Jaccard a seturilor alese la întâmplare?
! Exercițiul 3.3.5: Dovediți că dacă similitudinea Jaccard a două coloane este 0, atunci
minhashing oferă întotdeauna o estimare corectă a asemănării Jaccard.
!! Exercițiul 3.3.6: Ne-am putea aștepta să putem estima simularea Jaccard
laritatea coloanelor fără a utiliza toate permutările posibile ale rândurilor. De exemplu,
am putea permite doar permutări ciclice; adică începeți de la un rând ales aleatoriu
r, care devine primul în ordine, urmat de rândurile r + 1, r + 2 și așa
pe, în jos până la ultimul rând, și apoi continuând cu primul rând, al doilea rând,
și așa mai departe, până la rândul r - 1. Există n astfel de permutări, dacă există
n rânduri. Cu toate acestea, aceste permutări nu sunt suficiente pentru a estima Jaccard
similaritate corect. Dați un exemplu de matrice cu două coloane în care se face media
peste toate permutările ciclice nu oferă asemănarea Jaccard.
! Exercițiul 3.3.7: Să presupunem că dorim să folosim un cadru MapReduce pentru a calcula
semnături minhash. Dacă matricea este stocată în bucăți care corespund unor
coloane, atunci este destul de ușor să exploatezi paralelismul. Fiecare sarcină Hartă primește câteva
a coloanelor și a tuturor funcțiilor hash și calculează semnăturile minhash
dintre coloanele sale date. Cu toate acestea, să presupunem că matricea a fost blocată de rânduri, deci
că unei sarcini Harta i se oferă funcțiile hash și un set de rânduri pe care să lucreze. Proiecta
Funcții Map și Reduce pentru a exploata MapReduce cu date în acest formular.
! Exercițiul 3.3.8: După cum am observat în secțiunea 3.3.6, avem probleme când a
coloana are doar 0. Dacă calculăm o funcție minhash folosind coloane întregi
(ca în secțiunea 3.3.2), atunci singura dată când obținem toate 0 într-o coloană este dacă
coloana reprezintă setul gol. Cum ar trebui să ne ocupăm de setul gol de realizat
sigur nu sunt introduse erori în estimarea similarității Jaccard?
!! Exercițiul 3.3.9: în exemplul 3.9, fiecare dintre cele trei estimări ale Jaccard sim-
ilaritatea obținută a fost fie mai mică, fie aceeași cu adevăratul Jaccard
similitudine. Este posibil ca pentru o altă pereche de coloane să fie media
Similitudinile Jaccard ale jumătăților superioare și inferioare vor depăși Jaccard-ul real
similaritatea coloanelor?

3.4 Hashing sensibil la localitate pentru documente

Chiar dacă putem folosi minhashing pentru a comprima documente mari în mici
semnăturile și păstrează asemănarea așteptată a oricărei perechi de documente
totuși poate fi imposibil să găsești în mod eficient perechile cu cea mai mare similaritate. The

---

Pagina 120

motivul este că numărul de perechi de documente poate fi prea mare, chiar dacă există
nu sunt prea multe documente.
Exemplul 3.10: Să presupunem că avem un milion de documente și că folosim semnături
de lungime 250. Apoi folosim 1000 de octeți per document pentru semnături și
toate datele se încadrează într-un gigabyte - mai puțin decât o memorie principală tipică a unui laptop.
Cu toate acestea, există (1.000.000

^{2}
) sau jumătate de trilion de perechi de documente. Dacă este nevoie de un
microsecundă pentru a calcula similaritatea a două semnături, apoi este nevoie de aproape
șase zile pentru a calcula toate asemănările de pe acel laptop. ✷
Dacă scopul nostru este să calculăm similaritatea fiecărei perechi, noi nu suntem nimic
poate face pentru a reduce munca, deși paralelismul poate reduce timpul scurs.
Cu toate acestea, de multe ori ne dorim doar cele mai similare perechi sau toate perechile care sunt deasupra
unele limită inferioară în asemănare. Dacă da, atunci trebuie să ne concentrăm doar atenția
pe perechi care ar putea fi similare, fără a investiga fiecare pereche. Există
o teorie generală a modului de a oferi o astfel de concentrare, numită hashing sensibil la localitate
(LSH) sau căutare în vecinătate. În această secțiune vom lua în considerare o formă specifică
LSH, conceput pentru problema specială pe care am studiat-o: documente,
reprezentat de seturi de șindrilă, apoi minhashed la semnături scurte. În secțiunea 3.6
prezentăm teoria generală a hashului sensibil la localitate și o serie de
aplicații și tehnici conexe.

3.4.1 LSH pentru semnături Minhash

O abordare generală a LSH este de a „hash” elemente de mai multe ori, în așa fel încât
articole similare sunt mai susceptibile de a fi hash la aceeași găleată decât diferite
elementele sunt. Apoi, luăm în considerare orice pereche care a ajuns la aceeași găleată pentru oricare
dintre hashings pentru a fi o pereche candidată. Verificăm numai perechile de candidați pentru
similitudine. Speranța este că majoritatea perechilor diferite nu vor avea niciodată hash la
aceeași găleată și, prin urmare, nu va fi verificată niciodată. Acele perechi diferite care
face hash la aceeași găleată sunt fals pozitive; sperăm că acestea vor fi doar a
mică parte din toate perechile. Sperăm, de asemenea, că majoritatea perechilor cu adevărat similare
va hash la aceeași bucket sub cel puțin una dintre funcțiile hash. Acestea
care nu sunt negative negative; sperăm că acestea vor fi doar o mică parte din
perechile cu adevărat similare.
Dacă avem semnături minhash pentru articole, o modalitate eficientă de a alege
hashings este de a împărți matricea semnăturii în benzi b formate din r rânduri
fiecare. Pentru fiecare bandă, există o funcție hash care ia vectori de r numere întregi
(porțiunea unei coloane din acea bandă) și le hashează la unele mari
numărul de găleți. Putem folosi aceeași funcție hash pentru toate benzile, dar
folosim o matrice separată pentru fiecare bandă, deci coloane cu același vector
în diferite benzi, nu va avea hash la aceeași găleată.
Exemplul 3.11: Figura 3.7 prezintă o parte dintr-o matrice de semnături de 12 rânduri împărțite
în patru benzi de câte trei rânduri. Al doilea și al patrulea dintre cele explicite
coloanele afișate au fiecare vectorul de coloană [0, 2, 1] în prima bandă, deci ele

---

1 0 0 0 2
3 2 1 2 2
0 1 3 1 1
. . .
. . .
banda 1
banda 2
banda 3
banda 4

Figura 3.7: Împărțirea unei matrice de semnături în patru benzi de câte trei rânduri pe bandă
cu siguranță va hash la aceeași găleată în hashing pentru prima trupă. Prin urmare,
indiferent de cum arată acele coloane în celelalte trei benzi, această pereche
de coloane va fi o pereche de candidați. Este posibil ca alte coloane, cum ar fi
primele două afișate în mod explicit, vor, de asemenea, hash la aceeași găleată conform
hashing-ul primei trupe. Cu toate acestea, deoarece vectorii lor de coloană sunt diferiți,
[1, 3, 0] și [0, 2, 1], și există multe găleți pentru fiecare hashing, ne așteptăm ca
șansele ca o coliziune accidentală să fie foarte mici. În mod normal, vom presupune
că doi vectori hash la aceeași bucket dacă și numai dacă sunt identici.
Două coloane care nu sunt de acord în banda 1 au alte trei șanse să devină
o pereche de candidați; ar putea fi identici în oricare dintre aceste alte trupe.
Cu toate acestea, observați că cu cât sunt mai asemănătoare cele două coloane, cu atât este mai probabilă
că vor fi identici în unele trupe. Astfel, intuitiv strategia de bandare
face ca coloanele similare să fie mult mai susceptibile de a fi perechi candidate decât diferite
perechi. ✷

3.4.2 Analiza tehnicii de bandare

Să presupunem că folosim b benzi de r rânduri fiecare și să presupunem că o anumită pereche de
documentele au similitudini Jaccard. Reamintim din secțiunea 3.3.3 că probele
capacitatea semnăturilor minhash pentru aceste documente sunt de acord în orice anume

rândul matricei semnăturii este s. Putem calcula probabilitatea ca acestea
documentele (sau mai bine zis semnăturile lor) devin o pereche de candidați după cum urmează:
1. Probabilitatea ca semnăturile să fie de acord în toate rândurile unui anumit
banda este s^r .
2. Probabilitatea ca semnăturile să nu fie de acord în cel puțin un rând al unei părți
banda ticulară este $1 - s^r$.
3. Probabilitatea ca semnăturile să nu fie de acord în cel puțin un rând din fiecare
dintre benzi este $(1 - s^r)^b$.

Pagina 122

4. Probabilitatea ca semnăturile să fie de acord în toate rândurile a cel puțin una
bandă și, prin urmare, devin o pereche candidată, este $1 - (1 - s^r)^b$.
0
1
de documente
Similitudine Jaccard
Probabilitate
de a deveni
un candidat
Figura 3.8: Curba S.
Poate că nu este evident, dar indiferent de constantele alese b și r, aceasta
funcția are forma unei curbe S, așa cum este sugerat în Fig. 3.8. Pragul,
adică valoarea asemănării s la care probabilitatea de a deveni candidat
data este 1/2, este o funcție a lui b și r. Pragul este aproximativ acolo unde este creșterea
cel mai abrupt, iar pentru b și r mari găsim că perechile cu similitudine peste
pragul este foarte probabil să devină candidați, în timp ce cei sub prag
este puțin probabil să devină candidați - exact situația pe care o dorim. O aprox.
imitația la prag este $(1 / b)^{1/r}$. De exemplu, dacă b = 16 și r = 4, atunci
pragul este aproximativ la s = 1/2, deoarece a 4-a rădăcină a 1/16 este 1/2.
Exemplul 3.12: Să luăm în considerare cazul b = 20 și r = 5. Adică presupunem
avem semnături de lungime 100, împărțite în douăzeci de benzi de câte cinci rânduri fiecare.
Figura 3.9 tabelează unele dintre valorile funcției $1 - (1 - s^5)^{20}$. Înștiințare
că pragul, valoarea lui s la care curba a crescut la jumătatea drumului, este doar
ceva mai mult de 0,5. De asemenea, observați că curba nu este exact pasul ideal
funcție care sare de la 0 la 1 la prag, dar panta curbei
la mijloc este semnificativ. De exemplu, crește cu mai mult de 0,6 de la
s = 0,4 până la s = 0,6, deci panta din mijloc este mai mare de 3.
De exemplu, la s = 0,8, $1 - (0,8)^5$ este de aproximativ 0,672. Dacă crești acest număr
la puterea 20, veți obține aproximativ 0,00035. Scăderea acestei fracții din 1
produce 0,99965. Adică, dacă luăm în considerare două documente cu 80% similaritate, atunci
într-o singură bandă, au doar aproximativ 33% șanse de a fi de acord în toate cele cinci rânduri
și astfel devenind o pereche de candidați. Cu toate acestea, există 20 de trupe și deci 20

Pagina 123

s
$1 - (1 - s^r)^b$
.2 .006
.3 .047
.4 .186
.5 .470
.6 .802
.7 .975
.8 .9996
Figura 3.9: Valorile curbei S pentru b = 20 și r = 5
șanse de a deveni candidat. Doar aproximativ una din 3000 de perechi care sunt la fel de mari
întrucât 80% similar nu va reuși să devină o pereche de candidați și astfel să fie un fals negativ.
✷

3.4.3 Combinarea tehnicilor

Acum putem oferi o abordare pentru a găsi setul de perechi de candidați pentru similare
documente și apoi descoperirea documentelor cu adevărat similare dintre ele. Aceasta
trebuie subliniat faptul că această abordare poate produce negative negative - perechi de
documente similare care nu sunt identificate ca atare pentru că nu devin niciodată

o pereche de candidați. De asemenea, vor exista falsuri pozitive - perechi de candidați care sunt evaluate, dar se constată că nu sunt suficient de similare.
1. Alegeți o valoare k și construiți din fiecare document setul de șindrilă k. Opțional, hash-k-zoster pentru a numere mai mici cupe.
2. Sortați perechile document-șindrilă pentru a le ordona prin șindrilă.
3. Alegeți o lungime n pentru semnăturile minhash. Introduceți lista sortată în algoritmul Secțiunii 3.3.5 pentru a calcula semnăturile minhash pentru toate documente.
4. Alegeți un prag t care definește modul în care trebuie să se afle documente similare pentru ca aceștia să fie considerați ca o „pereche similară” dorită. Alegeți un număr a benzilor b și a unui număr de rânduri r astfel încât br = n și pragul t este aproximativ $(1 / b)^{1/r}$. Dacă este importantă evitarea negativelor false, poate doriți să selectați b și r pentru a produce un prag mai mic decât t; dacă viteza este importantă și doriți să limitați falsurile pozitive, selectați b și r la produc un prag mai mare.
5. Construiți perechi de candidați aplicând tehnica LSH din secțiunea 3.4.1.
6. Examinați semnăturile fiecărei perechi de candidați și stabiliți dacă fracțiunea componentele în care sunt de acord este cel puțin t.

---

7. Opțional, dacă semnăturile sunt suficient de similare, accesați documentele și verifică dacă sunt cu adevărat similare, mai degrabă decât documente care, din fericire, aveau semnături similare.

3.4.4 Exerciții pentru secțiunea 3.4

Exercițiul 3.4.1: Evaluează curba $S 1 - (1 - s^r)^b$ pentru s = 0,1, 0,2, ..., 0,9, pentru următoarele valori ale lui r și b:

- r = 3 și b = 10.
- r = 6 și b = 20.
- r = 5 și b = 50.

! Exercițiul 3.4.2: Pentru fiecare dintre perechile (r, b) din exercițiul 3.4.1, calculați prag, adică valoarea lui s pentru care valoarea $1- (1 - s^r)^b$ este exact 1/2. Cum se compară această valoare cu estimarea $(1 / b)^{1/r}$ care a fost sugerată în secțiunea 3.4.2?
! Exercițiul 3.4.3: Utilizați tehnicile explicate în secțiunea 1.3.5 pentru a aproxima curba $S 1 - (1 - s^r)^b$ când s^r este foarte mică.
! Exercițiul 3.4.4: Să presupunem că dorim să implementăm LSH prin MapReduce. Specifica-presupunem că bucăți din matricea semnăturii constau din coloane și elemente sunt perechi cheie-valoare unde cheia este numărul coloanei și valoarea este semnătura în sine (adică un vector de valori).
(a) Arătați cum să produceți gălețile pentru toate benzile ca ieșire a unui singur Procesul MapReduce. Sugestie: Rețineți că o funcție de hartă poate produce mai multe perechi cheie-valoare dintr-un singur element.
(b) Arătați cum un alt proces MapReduce poate converti ieșirea lui (a) în o listă de perechi care trebuie comparate. Mai exact, pentru fiecare coloană i, ar trebui să existe o listă a acelor coloane j> i cu care trebuie să fiu comparat.

3.5 Măsuri la distanță

Acum facem un scurt ocol pentru a studia noțiunea generală de măsuri la distanță. Similitudinea Jaccard este o măsură a cât de apropiate sunt seturile, deși nu este într-adevăr o măsură de distanță. Adică, cu cât seturile sunt mai apropiate, cu atât este mai mare Jaccard similitudine. Mai degrabă, 1 minus similaritatea Jaccard este o măsură a distanței, așa cum noi va vedea; se numește distanța Jaccard.
Cu toate acestea, distanța Jaccard nu este singura măsură de apropiere care o face sens. Vom examina în această secțiune alte măsuri de distanță care au aplicații. Apoi, în secțiunea 3.6 vedem cum măsoară unele dintre aceste distanțe

---

au, de asemenea, o tehnică LSH care ne permite să ne concentrăm pe punctele din apropiere fără comparând toate punctele. Alte aplicații ale măsurilor de distanță vor apărea când studiem gruparea în capitolul 7.

3.5.1 Definiția unei măsuri la distanță

Să presupunem că avem un set de puncte, numit spațiu. O măsură de distanță în acest sens spațiul este o funcție d (x, y) care ia două puncte în spațiu ca argumente și produce un număr real și îndeplinește următoarele axiome:

1. d (x, y) ≥ 0 (fără distanțe negative).
2. d (x, y) = 0 dacă și numai dacă x = y (distanțele sunt pozitive, cu excepția distanța de la un punct la sine).
3. d (x, y) = d (y, x) (distanța este simetrică).
4. d (x, y) ≤ d (x, z) + d (z, y) (inegalitatea triunghiului).

Inegalitatea triunghiului este cea mai complexă condiție. Spune, intuitiv, că pentru a călători de la x la y, nu putem obține niciun beneficiu dacă suntem obligați să călătorim prin un anumit al treilea punct z. Axioma triunghi-inegalitate este ceea ce face totul măsurătorile distanței se comportă ca și când distanța descrie lungimea unei căi mai scurte de la un punct la altul.

3.5.2 Distanțe euclidiene

Cea mai familiară măsură a distanței este cea pe care o considerăm în mod normal tance. ” Un spațiu euclidian n-dimensional este unul în care punctele sunt vectori de n numere reale. Măsura convențională a distanței în acest spațiu, pe care o vom face se referă la cum L_2 -normă, este definit:

$d([x_1, x_2, ..., x_n], [y_1, y_2, ..., y_n]) = \sqrt{\sqrt{\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}}}$

Adică, pătrăm distanța în fiecare dimensiune, însumăm pătratele și luăm rădăcina pătrată pozitivă.

Este ușor de verificat că primele trei cerințe pentru măsurarea distanței sunt mulțumit. Distanța euclidiană între două puncte nu poate fi negativă, pentru că se intenționează rădăcina pătrată pozitivă. Deoarece toate pătratele numerelor reale sunt nenegativ, orice i astfel încât $x_i = y_i$ forțează distanța să fie strict pozitivă. Pe de altă parte, dacă $x_i = y_i$ pentru tot i, atunci distanța este clar 0. Simetrie urmează deoarece $(x_i - y_i)^2 = (y_i - x_i)^2$. Inegalitatea triunghiului necesită un bine afacere de algebră de verificat. Cu toate acestea, este bine înțeles că este o proprietate a Spațiul euclidian: suma lungimilor oricărei două laturi ale unui triunghi nu este mai mică decât lungimea celei de-a treia laturi.

Există alte măsuri de distanță care au fost utilizate pentru spațiile euclidiene. Pentru orice constantă r, putem defini L_r -norma ca fiind măsura distanței d definit de:

$d([x_1, x_2, ..., x_n], [y_1, y_2, ..., y_n]) = (\sum_{i=1}^{n} |x_i - y_i|^r)^{1/r}$

Cazul r = 2 este de obicei L_2 -normă tocmai am menționat. O altă distanță comună măsura este distanța L_1 -normă sau Manhattan. Acolo, distanța dintre două puncte reprezintă suma mărimilor diferențelor din fiecare dimensiune. Se numește „distanța Manhattan”, deoarece este distanța pe care ar trebui să o faci călătoriți între puncte dacă unul ar fi constrâns să călătorească de-a lungul liniilor grilei, ca pe străzile unui oraș precum Manhattan.

O altă măsură de distanță interesantă este L_∞-norma, care este limita pe măsură ce r se apropie de infinitul L_r -norm. Pe măsură ce r devine mai mare, doar dimensiunea cu cea mai mare diferență contează, deci formal, norma L_∞ este definită ca maxim de $|x_i - y_i|$ peste toate dimensiunile i.

Exemplul 3.13: Luați în considerare spațiul euclidian bidimensional (plan ari) și punctele (2, 7) și (6, 4). Norma L_2 dă o distanță de $\sqrt{(2 - 6)^2 + (7-4)^2} = \sqrt{4^2 + 3^2} = 5$. L_1 -normă oferă o distanță de $|2 - 6| + |7 - 4| = 4 + 3 = 7$. L_∞-normul dă o distanță de $\max(|2 - 6|, |7 - 4|) = \max(4, 3) = 4$

✷

3.5.3 Distanța Jaccard

După cum sa menționat la începutul secțiunii, definim distanța Jaccard
de seturi de d (x, y) = 1 - SIM (x, y). Adică distanța Jaccard este 1 minus
raportul dintre dimensiunile intersecției și uniunii mulțimilor x și y. Trebuie să verificăm
că această funcție este o măsură a distanței.
1. d (x, y) nu este negativ deoarece dimensiunea intersecției nu poate depăși
mărimea uniunii.
2. d (x, y) = 0 dacă x = y, deoarece x ∪ x = x ∩ x = x. Cu toate acestea, dacă x = y, atunci
dimensiunea lui x ∩ y este strict mai mică decât dimensiunea lui x ∪ y, deci d (x, y) este strict
pozitiv.
3. d (x, y) = d (y, x) deoarece atât uniunea cât și intersecția sunt simetrice; adică
x ∪ y = y ∪ x și x ∩ y = y ∩ x.
4. Pentru inegalitatea triunghiului, reamintim din secțiunea 3.3.3 că SIM (x, y) este
probabilitatea unei funcții minhash aleatorii mapează x și y la aceeași valoare.
Astfel, distanța Jaccard d (x, y) este probabilitatea ca o min
funcția hash nu trimite x și y la aceeași valoare. Prin urmare, putem

---

Pagina 127

traduceți condiția d (x, y) ≤ d (x, z) + d (z, y) la afirmația că dacă
h este o funcție minhash aleatorie, atunci probabilitatea ca h (x) = h (y)
nu este mai mare decât suma probabilității ca h (x) = h (z) și
probabilitatea ca h (z) = h (y). Cu toate acestea, această afirmație este adevărată deoarece
ori de câte ori h (x) = h (y), cel puțin unul dintre h (x) și h (y) trebuie să fie diferit
de la h (z). Nu pot fi amândoi h (z), deoarece atunci h (x) și h (y)
ar fi la fel.
.

3.5.4 Distanța cosinusului

Distanța cosinusului are sens în spațiile care au dimensiuni, inclusiv Eu-
spații clidiene și versiuni discrete ale spațiilor euclidiene, cum ar fi spațiile unde
punctele sunt vectori cu componente întregi sau componente booleene (0 sau 1). În
într-un astfel de spațiu, punctele pot fi gândite ca direcții. Nu distingem
între un vector și un multiplu al acelui vector. Apoi distanța cosinusului dintre
două puncte este unghiul pe care îl fac vectorii față de acele puncte. Acest unghi va
să fie în intervalul 0 - 180 de grade, indiferent de câte dimensiuni are spațiul
are.
Putem calcula distanța cosinusului calculând mai întâi cosinusul lui
și apoi aplicarea funcției arc-cosinus pentru a se traduce la un unghi în
Interval 0-180 grade. Având în vedere doi vectori x și y, cosinusul unghiului dintre
ele este produsul punct xy împărțit la L_2 -normele lui x și y (adică ale lor
Distanțele euclidiene de la origine). Amintiți-vă că produsul punct al vectorilor
$[x_1, x_2, ..., x_n]$. $[y_1, y_2, ..., y_n]$ este $\sum_{i=1}^{n} x_i y_i$.
Exemplul 3.14: Fie cei doi vectori ai noștri x = [1, 2, −1] și = [2, 1, 1]. Punctul
produsul xy este 1 × 2 + 2 × 1 + (- 1) × 1 = 3.Norma L_2 a ambilor vectori este
$\sqrt{6}$. De exemplu, x are L_2 -norm $\sqrt{1^2 + 2^2 + (-1)^2} = \sqrt{6}$. Astfel, cosinusul
unghiul dintre x și y este $3 / (\sqrt{6}\sqrt{6})$ sau 1/2. Unghiul al cărui cosinus este 1/2
este de 60 de grade, deci aceasta este distanța cosinusului între x și y. ✷
Trebuie să arătăm că distanța cosinusului este într-adevăr o măsură a distanței. Noi
l-am definit astfel încât valorile să fie cuprinse între 0 și 180, deci nu există distanțe negative
sunt posibile. Doi vectori au unghiul 0 dacă și numai dacă sunt în aceeași direcție. [5]
Simetria este evidentă: unghiul dintre x și y este același cu unghiul
între y și x. Inegalitatea triunghiului este argumentată cel mai bine prin raționamentul fizic.
O modalitate de a roti de la x la y este de a roti la z și de acolo la y. Suma
aceste două rotații nu pot fi mai mici decât rotația directă de la x la y.
[5] Observați că pentru a satisface a doua axiomă, trebuie să tratăm vectori care sunt multipli ai lui
unii pe alții, de exemplu [1, 2] și [3, 6], ca aceeași direcție, care sunt. Dacă le-am privi pe acestea
ca vectori diferiți, le-am da distanța 0 și astfel vom încălca condiția că numai
d (x, x) este 0.

Pagina 128

3.5.5 Editare distanță

Această distanță are sens atunci când punctele sunt șiruri. Distanța dintre doi
șiruri $x = x_1 x_2 \cdots x_n$ și $y = y_1 y_2 \cdots y_m$ este cel mai mic număr de inserții

și ștergeri de caractere unice care vor converti x în y.
Exemplul 3.15: Distanța de editare dintre șirurile x = abcde și y =
acfdeg este 3. Pentru a converti x în y:
1. Ștergeți b.
2. Introduceți f după c.
3. Introduceți g după e.
Nicio secvență de mai puțin de trei inserții și / sau ștergeri nu va converti x în y.
Astfel, d (x, y) = 3. ✷
Un alt mod de a defini și calcula distanța de editare d (x, y) este să calculați
cea mai lungă subsecvență comună (LCS) a lui x și y. Un LCS de x și y este a
șir care este construit prin ștergerea pozițiilor din x și y, și care este ca
atâta timp cât orice șir care poate fi construit în acest fel. Distanța de editare d (x, y)
poate fi calculat ca lungimea lui x plus lungimea lui minus de două ori lungimea
din LCS-ul lor.
Exemplul 3.16: Șirurile x = abcde și y = acfdeg din Exemplul 3.15
au un LCS unic, care este acde. Putem fi siguri că este cel mai lung posibil,
deoarece conține fiecare simbol care apare atât în x, cât și în y. Din fericire, acestea
simbolurile comune apar în aceeași ordine în ambele șiruri, așa că suntem capabili
folosiți-le pe toate într-un LCS. Rețineți că lungimea lui x este 5, lungimea lui y este 6 și
lungimea LCS a acestora este 4. Distanța de editare este astfel 5 + 6 - 2 × 4 = 3, care
este de acord cu calculul direct din Exemplul 3.15.
Pentru un alt exemplu, considerați x = aba și y = bab. Distanța lor de editare este
2. De exemplu, putem converti x în y ștergând prima a și apoi inserând
b la final. Există două LCS-uri: ab și ba. Fiecare poate fi obținut de
ștergerea unui simbol din fiecare șir. Așa cum trebuie să fie cazul mai multor LCS
din aceeași pereche de șiruri, ambele LCS au aceeași lungime. Prin urmare, noi
poate calcula distanța de editare ca 3 + 3 - 2 × 2 = 2. ✷
Editarea distanței este o măsură a distanței. Cu siguranță, nicio distanță de editare nu poate fi negativă,
și doar două șiruri identice au o distanță de editare de 0. Pentru a vedea această editare
distanța este simetrică, rețineți că o secvență de inserții și ștergeri poate fi
inversat, fiecare inserție devenind o ștergere și invers. Triunghiul
inegalitatea este de asemenea simplă. O modalitate de a transforma un șir într-un șir t
este de a transforma s într-un șir u și apoi de a transforma u în t. Astfel, numărul de
modificările făcute mergând de la s la u, plus numărul de editări făcute mergând de la u la t
nu poate fi mai mic decât cel mai mic număr de editări care vor transforma s în t.

Spații neeuclidiene

Observați că mai multe dintre măsurile de distanță introduse în această secțiune sunt
nu spații euclidiene. O proprietate a spațiilor euclidiene pe care o vom găsi
important atunci când abordăm gruparea în capitolul 7 este că media
de puncte într-un spațiu euclidian există întotdeauna și este un punct din spațiu.
Cu toate acestea, luați în considerare spațiul seturilor pentru care am definit disecția Jaccard
tance. Noțiunea de „medie” a două seturi nu are sens. De asemenea,
spațiul șirurilor, unde putem folosi distanța de editare, nu ne lasă
ia „media” corzilor.
Spațiile vectoriale, pentru care am sugerat distanța cosinusului, pot sau pot
nu fii euclidian. Dacă componentele vectorilor pot fi orice număr real
bers, atunci spațiul este euclidian. Cu toate acestea, dacă restrângem componentele la
fiți întregi, atunci spațiul nu este euclidian. Observați că, de exemplu, noi
nu pot găsi o medie a vectorilor [1, 2] și [3, 1] în spațiul vectorilor
cu două componente întregi, deși dacă le-am trata ca pe membri ai
spațiul euclidian bidimensional, atunci am putea spune că media lor
a fost [2,0, 1,5].

3.5.6 Distanța de lovire

Având în vedere un spațiu de vectori, definim distanța de Hamming între doi vectori
să fie numărul de componente în care diferă. Ar trebui să fie evident
că distanța Hamming este o măsură a distanței. În mod clar distanța Hamming
nu poate fi negativ, iar dacă este zero, atunci vectorii sunt identici. Dis-
tanta nu depinde de care dintre doi vectori considerăm mai întâi. Triunghiul
inegalitatea ar trebui, de asemenea, să fie evidentă. Dacă x și z diferă în m componente, și z
și y diferă în n componente, atunci x și y nu pot diferi în mai mult de m + n
componente. Cel mai frecvent, distanța Hamming este utilizată atunci când vectorii sunt
Boolean; constau doar din 0 și 1. Cu toate acestea, în principiu, vectorii pot
au componente din orice set.
Exemplul 3.17: Distanța Hamming între vectorii 10101 și 11110

este 3. Adică acești vectori diferă în a doua, a patra și a cincea componentă,
în timp ce sunt de acord în prima și a treia componentă. ✷

3.5.7 Exerciții pentru secțiunea 3.5

! Exercițiul 3.5.1: Pe spațiul numerelor întregi nenegative, care dintre următoarele funcțiile sunt măsurători la distanță? Dacă da, dovedește-l; dacă nu, demonstrați că nu reușește satisfac una sau mai multe dintre axiome.
(a) max (x, y) = cel mai mare dintre x și y.

(b) dif (x, y) = | x - y | (magnitudinea absolută a diferenței dintre x și y).
(c) sumă (x, y) = x + y.
Exercițiul 3.5.2: Găsiți distanțele L_1 și L_2 dintre punctele (5, 6, 7) și (8, 2, 4).
!! Exercițiul 3.5.3: Dovediți că dacă i și j sunt numere întregi pozitive și i <j, atunci norma L_i dintre oricare două puncte este mai mare decât norma L_j dintre aceleași două puncte.
Exercițiul 3.5.4: Găsiți distanțele Jaccard între următoarele perechi de seturi:
(a) {1, 2, 3, 4} și {2, 3, 4, 5}.
(b) {1, 2, 3} și {4, 5, 6}.
Exercițiul 3.5.5: Calculați cosinusurile unghiurilor dintre fiecare dintre următoarele perechi mici de vectori. [6]
(a) (3, −1, 2) și (−2, 3, 1).
(b) (1, 2, 3) și (2, 4, 6).
(c) (5, 0, −4) și (−1, −6, 2).
(d) (0, 1, 1, 0, 1, 1) și (0, 0, 1, 0, 0, 0).
! Exercițiul 3.5.6: Dovediți că distanța cosinusului între oricare doi vectori ai lui 0 și 1, de aceeași lungime, are cel mult 90 de grade.
Exercițiul 3.5.7: Găsiți distanțele de editare (folosind doar inserții și ștergeri) între următoarele perechi de corzi.
(a) abcdef și bdaefc.
(b) abccdabc și acbdcab.
(c) abcdef și baedfc.
! Exercițiul 3.5.8: Există o serie de alte noțiuni de distanță de editare disponibile. De exemplu, putem permite, pe lângă inserții și ștergeri, următoarele operațiuni:

[6] Rețineți că ceea ce solicităm nu este tocmai distanța cosinusului, ci de la cosinus dintr-un unghi, puteți calcula unghiul în sine, poate cu ajutorul unui tabel sau bibliotecă funcție.

---

eu. Mutație, unde un simbol este înlocuit cu un alt simbol. Rețineți că a mutația poate fi întotdeauna efectuată printr-o inserție urmată de o ștergere, dar dacă permitem mutații, atunci această modificare contează doar 1, nu 2, când calculând distanța de editare.
ii. Transpunere, unde două simboluri adiacente au pozițiile schimbate. La fel ca o mutație, putem simula o transpunere printr-o inserție urmată printr-o singură ștergere, dar aici numărăm doar 1 pentru acești doi pași.
Repetați exercițiul 3.5.7 dacă distanța de editare este definită ca fiind numărul de inserții, ștergeri, mutații și transpuneri necesare pentru a transforma un șir în un alt.
! Exercițiul 3.5.9: Dovediți că distanța de editare discutată în Exercițiul 3.5.8 este într-adevăr o măsură de distanță.
Exercițiul 3.5.10: Găsiți distanțele Hamming între fiecare pereche de următori vectori lowing: 000000, 110011, 010101 și 011100.

3.6 Teoria funcțiilor sensibile la localitate

Tehnica LSH dezvoltată în secțiunea 3.4 este un exemplu de familie de funcții (funcțiile minhash) care pot fi combinate (prin tehnica de bandaj) pentru a distinge puternic între perechi la o distanță mică de perechi la o distanță mare tance. Abruptitatea curbei S din Fig. 3.8 reflectă cât de eficient putem evita fals pozitive și negative negative în rândul perechilor de candidați.

Acum, vom explora alte familii de funcții, pe lângă funcțiile minhash
, care pot servi la producerea eficientă a perechilor de candidați. Aceste funcții pot
se aplică spațiului seturilor și distanței Jaccard, sau altui spațiu și / sau
o altă măsură de distanță. Există trei condiții de care avem nevoie pentru o familie
de funcții:
1. Trebuie să fie mai probabil să facă perechi apropiate să fie perechi candidate decât
perechi îndepărtate. Facem această noțiune precisă în secțiunea 3.6.1.
2. Ele trebuie să fie independente din punct de vedere statistic, în sensul că este posibil
estimează probabilitatea ca două sau mai multe funcții să dea o anumită
răspuns prin regula produsului pentru evenimente independente.
3. Ele trebuie să fie eficiente, în două moduri:
(a) Trebuie să poată identifica cu mult mai puțin perechile de candidați
decât timpul necesar pentru a privi toate perechile. De exemplu, minhash
funcțiile au această capacitate, deoarece putem hash seturi la minhash
valori în timp proporționale cu dimensiunea datelor, mai degrabă decât
pătratul numărului de seturi din date. Din moment ce seturi cu comune
valorile sunt plasate într-o găleată, am produs implicit

Pagina 132

perechi candidate pentru o singură funcție minhash în timp mult mai mică decât
numărul de perechi de seturi.
(b) Acestea trebuie să fie combinabile pentru a construi funcții care să fie mai bine evitate-
fals pozitive și negative, iar funcțiile combinate trebuie
ia, de asemenea, un timp mult mai mic decât numărul de perechi. Pentru ex-
amplu, tehnica de bandare din secțiunea 3.4.1 necesită un minhash unic
funcții, care îndeplinesc condiția 3a, dar nu o au, de la sine
comportamentul curbei S pe care îl dorim și produce dintr-un număr de
funcții hash o funcție combinată care are forma curbei S.
Primul nostru pas este să definim „funcțiile sensibile la localitate” în general. Atunci noi
vezi cum se poate aplica ideea în mai multe aplicații. În cele din urmă, discutăm
cum se aplică teoria datelor arbitrare fie cu o distanță de cosinus, fie cu
Măsura distanței euclidiene.

3.6.1 Funcții sensibile la localitate

În sensul acestei secțiuni, vom lua în considerare funcțiile care includ două elemente
și să ia o decizie cu privire la dacă aceste articole ar trebui să fie o pereche de candidați.
În multe cazuri, funcția f va „hash” elemente, iar decizia se va baza
dacă rezultatul este sau nu egal. Deoarece este convenabil să utilizați
notația f (x) = f (y) pentru a însemna că f (x, y) este „da; faceți candidatul x și ya
pereche ”, vom folosi f (x) = f (y) ca prescurtare cu acest sens. De asemenea, folosim
f (x) = f (y) pentru a însemna „nu faceți pereche candidat x și ya decât dacă este altul
funcția concluzionează că ar trebui să facem acest lucru. ”
O colecție de funcții ale acestei forme va fi numită o familie de funcții.
De exemplu, familia de funcții minhash, fiecare bazată pe unul dintre posibile
permutările de rânduri ale unei matrice caracteristice, formează o familie.
Fie $d_1 < d_2$ două distanțe conform unei anumite măsuri de distanță d. A
se spune că familia F de funcții este (d_1, d_2, p_1, p_2) -sensibilă dacă pentru fiecare f din F:
1. Dacă $d(x, y) \leq d_1$, atunci probabilitatea ca f (x) = f (y) să fie cel puțin p_1.
2. Dacă $d(x, y) \geq d_2$, atunci probabilitatea ca f (x) = f (y) să fie cel mult p_2.
Figura 3.10 ilustrează ce ne așteptăm cu privire la probabilitatea ca un anumit
funcție într-o (d_1, d_2, p_1, p_2) -familia sensibilă va declara că două elemente pot fi-
pereche didat. Observați că nu spunem nimic despre ceea ce se întâmplă atunci când distanța
între elemente este strict între d_1 și d_2, dar putem face d_1 și d_2 ca
aproape cum ne dorim. Pedeapsa este că de obicei p_1 și p_2 sunt apoi aproape.
După cum vom vedea, este posibil să separați p_1 și p_2, păstrând în același timp d_1 și d_2
fix.

3.6.2 Familii sensibile la localitate pentru distanța Jaccard

Pentru moment, avem o singură modalitate de a găsi o familie de localități sensibile
funcții: utilizați familia de funcții minhash și presupuneți că distanța

---

Pagina 133

Probabilitate
de a fi

a declarat o
candidat
d
p
d
p
1
2
1
2
Distanță
Figura 3.10: Comportamentul unei funcții sensibile (d_1, d_2, p_1, p_2) măsura este distanța Jaccard. Ca și înainte, interpretăm o funcție minhash h pentru a face perechea de candidați x și ya dacă și numai dacă h (x) = h (y).

- Familia funcțiilor minhash este o familie (d_1, d_2, $1 - d_1$, $1 - d_2$) -sensibilă pentru orice d_1 și d_2, unde $0 \le d_1 < d_2 \le 1$.

Motivul este că dacă $d(x, y) \le d_1$, unde d este distanța Jaccard, atunci $\text{SIM}(x, y) = 1 - d(x, y) \ge 1 - d_1$. Dar știm că similitudinea Jaccard lui x și y este egal cu probabilitatea ca o funcție minhash să hash x și y la aceeași valoare. Un argument similar se aplică d_2 sau oricărei distanțe.

Exemplul 3.18: Am putea lăsa $d_1 = 0{,}3$ și $d_2 = 0{,}6$. Atunci putem afirma că familia funcțiilor minhash este o familie sensibilă (0,3, 0,6, 0,7, 0,4). Acesta este, dacă distanța Jaccard între x și y este de cel mult 0,3 (adică $\text{SIM}(x, y) \ge 0{,}7$) atunci există cel puțin 0,7 șanse ca o funcție minhash să trimită x și y către aceeași valoare și dacă distanța Jaccard între x și y este de cel puțin 0,6 (adică $\text{SIM}(x, y) \le 0{,}4$), atunci există cel mult 0,4 șanse ca x și y să fie trimise la aceeași valoare. Rețineți că am putea face aceeași afirmație cu alta alegerea d_1 și d_2; este necesar doar $d_1 < d_2$. ✷

3.6.3 Amplificarea unei familii sensibile la localitate

Să presupunem că ni se dă o familie sensibilă (d_1, d_2, p_1, p_2) F. Putem construi o noua familie F′ prin construcția AND pe F, care este definită după cum urmează. Fiecare membru al lui F′ este format din r membri ai lui F pentru un anumit r. Dacă f este în F′, și f este construit din mulțimea $\{f_1, f_2, ..., f_r\}$ a membrilor lui F, spunem f (x) = f (y) dacă și numai dacă $f_i(x) = f_i(y)$ pentru toate i = 1, 2, ..., r. Observați că această construcție reflectă efectul rândurilor r într-o singură bandă: banda face x și ya

---

pereche candidat dacă fiecare dintre rândurile r din bandă spune că x și y sunt egale (și, prin urmare, o pereche de candidați conform acelui rând).

Deoarece membrii F sunt aleși în mod independent pentru a deveni membru F′, putem afirma că F′ este o (d_1, d_2, $(p_1)^r$, $(p_2)^r$) -sensibilă familie. Adică pentru orice p, dacă p este probabilitatea ca un membru al F să declare (x, y) candidat pereche, atunci probabilitatea ca un membru al lui F′ să declare astfel este p^r.

Există o altă construcție, pe care o numim construcția OR, care se transformă a (d_1, d_2, p_1, p_2) -sensibilă familia F în a (d_1, d_2, $1 - (1 - p_1)^b$, $1 - (1 - p_2)^b$) -familie sensibilă F′. Fiecare membru f al lui F′ este construit din b membri ai lui F, spune f_1, f_2, ..., f_b. Definim f (x) = f (y) dacă și numai dacă $f_i(x) = f_i(y)$ pentru una sau mai multe valori ale i. Construcția OR reflectă efectul combinării mai multor benzi: x și y devin o pereche de candidați dacă orice bandă îi face candidat pereche.

Dacă p este probabilitatea ca un membru al F să declare (x, y) candidat pereche, atunci $1 - p$ este probabilitatea să nu declare așa. $(1 - p)^b$ este probabilitatea că niciunul dintre f_1, f_2, ..., $f_{b\,nu}$ va declara (x, y) o pereche candidată și $1 - (1 - p)^b$ este probabilitatea ca cel puțin un f_i să declare (x, y) o pereche candidată și prin urmare, f va declara (x, y) a fi o pereche candidată.

Observați că construcția AND scade toate probabilitățile, dar dacă alegem F și r în mod judicios, putem face ca probabilitatea mică p_2 să se apropie de 0, în timp ce probabilitatea mai mare p_1 rămâne semnificativ departe de 0. În mod similar, OR-construcția face ca toate probabilitățile să crească, dar alegând F și b judicios, putem face abordarea de probabilitate mai mare 1 în timp ce probabilitatea mai mică rămâne delimitat de 1. Putem construi în cascadă AND- și OR-construcții orice ordine pentru a face probabilitatea scăzută aproape de 0 și probabilitatea ridicată aproape la 1. Desigur, cu cât folosim mai multe construcții și cu atât sunt mai mari valorile lui r și b pe care le alegem, cu atât este mai mare numărul de funcții din familia originală pe care suntem obligați să le folosim. Astfel, cu cât familia finală de funcții este mai bună, cu atât este mai bună este nevoie de mai mult timp pentru a aplica funcțiile din această familie.

Exemplul 3.19: Să presupunem că începem cu o familie F. Folosim AND-construc-

cu r = 4 pentru a produce o familie F_1 . Apoi aplicăm construcția OR
la F_1 cu b = 4 pentru a produce o a treia familie F_2 . Rețineți că membrii F_2
fiecare este construit din 16 membri ai F și situația este similară cu cea de pornire
cu 16 funcții minhash și tratându-le ca patru benzi de câte patru rânduri fiecare.
Funcția AND cu 4 căi convertește orice probabilitate p în p^4 . Cand noi
urmați-l prin construcția OR cu 4 căi, probabilitatea fiind convertită în continuare
în $1-(1-p^4)^4$. Unele valori ale acestei transformări sunt indicate în Fig. 3.11.
Această funcție este o curbă S, rămânând scăzută o vreme, apoi crescând abrupt (deși
nu prea abrupt; panta nu devine niciodată mult mai mare decât 2) și apoi nivelarea
oprit la valori mari. Ca orice curbă S, are un punct fix, adică valoarea lui p
lăsată neschimbată atunci când aplicăm funcția curbei S. În acest caz,
punct fix este valoarea lui p pentru care $p = 1 - (1 - p^4)^4$. Putem vedea că
punctul fix este undeva între 0,7 și 0,8. Sub această valoare, probabilitățile sunt
au scăzut, iar deasupra ei sunt crescute. Astfel, dacă alegem o probabilitate mare

---

p	$1 - (1 - p^4)^4$
0,2	0,0064
0,3	0,0320
0,4	0,0985
0,5	0,2275
0,6	0,4260
0,7	0,66666
0,8	0,8585
0,9	0,9860

Figura 3.11: Efectul construcției AND cu 4 căi urmat de OR- cu 4 căi
constructie

deasupra punctului fix și o probabilitate scăzută sub acesta, vom avea ceea ce dorim
efectul că probabilitatea scăzută este scăzută și probabilitatea ridicată este crescută.
Să presupunem că F este funcțiile minhash, considerate ca (0,2, 0,6, 0,8, 0,4) -sens-
familie itivă. Apoi F_2 , familia construită printr-un AND cu 4 căi urmat de un
OR cu 4 căi, este o familie sensibilă (0,2, 0,6, 0,8855, 0,0985), așa cum putem citi din
rânduri pentru 0.8 și 0.4 în Fig. 3.11. Înlocuind F cu F_2 , am redus ambele
ratele fals-negative și fals-pozitive, cu prețul aplicării
funcțiile durează de 16 ori mai mult. ✷

p	$(1 - (1 - p)^4)^4$
0,1	0,0140
0,2	0,1215
0,3	0,3334
0,4	0,5740
0,5	0,7725
0,6	0,9015
0,7	0,9680
0,8	0,9936

Figura 3.12: Efectul construcției OR în 4 direcții urmat de AND- în 4 direcții
constructie

Exemplul 3.20: Pentru același cost, putem aplica o construcție OR cu 4 căi

urmat de o construcție ȘI cu 4 căi. Figura 3.12 oferă transformarea
asupra probabilităților implicate de această construcție. De exemplu, să presupunem că F este a
(0,2, 0,6, 0,8, 0,4) familie sensibilă. Atunci familia construită este o
(0,2, 0,6, 0,9936, 0,5740) -sensibil

---

familie. Această alegere nu este neapărat cea mai bună. Deși probabilitatea mai mare
s-a apropiat mult mai mult de 1, a crescut și probabilitatea mai mică, crescând
numărul falsurilor pozitive. ✷
Exemplul 3.21: Putem construi în cascadă cât de mult ne place. Pentru examen-
am putea folosi construcția Exemplului 3.19 pe familia minhash
funcții și apoi utilizați construcția Exemplului 3.20 pe familia rezultată.
Familia construită va avea apoi funcții construite fiecare din 256 minhash
funcții. Ar transforma, de exemplu, o familie sensibilă (0,2, 0,8, 0,8, 0,2)
într-o familie sensibilă (0,2, 0,8, 0,9991285, 0,0000004). ✷

3.6.4 Exerciții pentru secțiunea 3.6

Exercițiul 3.6.1: Care este efectul asupra probabilității de a începe cu familia
a funcțiilor minhash și aplicarea:
(a) O construcție AND cu 2 căi urmată de o construcție SAU cu 3 căi.
(b) O construcție SA cu 3 căi urmată de o construcție ȘI cu 2 căi.
(c) A urmat o construcție AND cu 2 căi, urmată de o construcție SAU cu 2 căi
printr-o construcție ȘI cu 2 căi.
(d) A urmat o construcție SA cu 2 căi, urmată de o construcție ȘI cu 2 căi
printr-o construcție SAU cu 2 căi urmată de o construcție ȘI cu 2 căi.
Exercițiul 3.6.2: Găsiți punctele fixe pentru fiecare dintre funcțiile construite în
Exercițiul 3.6.1.
! Exercițiul 3.6.3: Orice funcție de probabilitate p, cum ar fi cea din Fig. 3.11, are
o pantă dată de derivata funcției. Panta maximă este unde
acea derivată este un maxim. Găsiți valoarea lui p care dă o pantă maximă
pentru curbele S date de Fig. 3.11 și Fig. 3.12. Care sunt valorile acestora
pante maxime?
!! Exercițiul 3.6.4: Generalizați exercițiul 3.6.3 pentru a da, în funcție de r și b, valoarea
punctul pantei maxime și valoarea acelei pante, pentru familiile de funcții
definit din funcțiile minhash prin:
(a) O construcție R-way ȘI urmată de o construcție OR b-way.
(b) O construcție b-way SAU urmată de o construcție r-way ȘI.

3.7 Familii LSH pentru alte măsuri la distanță

Nu există nicio garanție că o măsură de distanță are o familie sensibilă la localitate
funcții hash. Până acum, am văzut astfel de familii doar pentru distanța Jaccard.
În această secțiune, vom arăta cum să construim familii sensibile la localitate
Distanța Hamming, distanța cosinusului și pentru distanța euclidiană normală.

3.7.1 Familii LSH pentru distanță de lovire

Este destul de simplu să construiești o familie de funcții sensibile la localitate pentru Ham-
ming distanță. Să presupunem că avem un spațiu de vectori d-dimensional și h (x, y)
denotă distanța Hamming între vectorii x și y. Dacă luăm pe cineva
poziția vectorilor, să spunem poziția ith, putem defini funcția f_i (x)
să fie bitul ith al vectorului x. Atunci f_i (x) = f_i (y) dacă și numai dacă vectorii x și
sunteți de acord în a doua poziție. Atunci probabilitatea ca f_i (x) = f_i (y) pentru o
domly ales i este exact 1 - h (x, y) / d; adică este fracția de poziții în
care x și y sunt de acord.
Această situație este aproape exact ca cea pe care am întâlnit-o pentru minhashing.
Astfel, familia F constând din funcțiile {f_1 , f_2 , ..., f_d } este a
(d_1 , d_2 , 1 - d_1 / d, 1 - d_2 / d) -sensibil
familie de funcții hash, pentru orice d_1 <d_2 . Există doar două diferențe
între această familie și familia funcțiilor minhash.
1. În timp ce distanța Jaccard variază de la 0 la 1, distanța Hamming pe a
spațiul vectorial al dimensiunii d rulează de la 0 la d. Prin urmare, este necesar să
scalați distanțele împărțind la d, pentru a le transforma în probabilități.
2. Deși există în esență o ofertă nelimitată de funcții minhash,

dimensiunea familiei F pentru distanța Hamming este doar d.
Primul punct nu are nicio consecință; necesită doar să împărțim la d at
vremuri potrivite. Al doilea punct este mai serios. Dacă d este relativ mic,
atunci suntem limitați în numărul de funcții care pot fi compuse folosind
construcțiile AND și OR, limitând astfel cât de abrupte putem face
S-curba fi.

3.7.2 Hiperavioane aleatorii și distanța cosinusului

Reamintim din secțiunea 3.5.4 că distanța cosinusului dintre doi vectori este
unghiul dintre vectori. De exemplu, vedem în Fig. 3.13 doi vectori x
și y care fac un unghi θ între ele. Rețineți că acești vectori pot fi
într-un spațiu de multe dimensiuni, dar ele definesc întotdeauna un plan și unghiul
între ele se măsoară în acest plan. Figura 3.13 este o „vedere de sus” a
plan care conține x și y.
Să presupunem că alegem un hiperplan prin origine. Acest hiperplan se intersectează
planul lui x și y într-o linie. Figura 3.13 sugerează două hiperplane posibile,
una a cărei intersecție este linia întreruptă, iar intersecția celuilalt este
linie punctata. Pentru a alege un hiperplan aleatoriu, alegem de fapt vectorul normal
la hiperplan, să spunem v. Hiperplanul este apoi setul de puncte al căror punct
produsul cu v este 0.
În primul rând, considerați un vector v care este normal pentru hiperplanul a cărui proiecție
este reprezentată de linia punctată în Fig. 3.13; adică x și y sunt diferite

---

Pagina 138

θ
X
y

Figura 3.13: Doi vectori fac un unghi θ

laturile hiperplanului. Atunci produsele dot vx și vy vor avea diferite
semne. Dacă presupunem, de exemplu, că v este un vector a cărui proiecție pe
planul lui x și y este deasupra liniei punctate din Fig. 3.13, atunci vx este pozitiv,
în timp ce vy este negativ. Vectorul normal v s-ar putea extinde în schimb
direcție, sub linia punctată. În acest caz, vx este negativ și vy este pozitiv,
dar semnele sunt încă diferite.
Pe de altă parte, vectorul ales aleatoriu v ar putea fi normal pentru a
hiperplan ca linia punctată din Fig. 3.13. În acest caz, atât vx cât și vy
au același semn. Dacă proiecția lui v se extinde spre dreapta, atunci ambele punctează
produsele sunt pozitive, în timp ce dacă v se extinde spre stânga, atunci ambele sunt negative.
Care este probabilitatea ca vectorul ales aleatoriu să fie normal pentru a
hiperplan care arată mai degrabă ca linia punctată decât linia punctată? Toate
unghiuri pentru linia care este intersecția hiperplanului aleatoriu și
planul lui x și y este la fel de probabil. Astfel, hiperplanul va arăta ca
linie punctată cu probabilitate θ/ 180 și va arăta ca linia punctată altfel.
Astfel, fiecare funcție hash f din familia noastră sensibilă la localitate F este construită din
un vector ales aleatoriu v_f . Având în vedere doi vectori x și y, spuneți f (x) = f (y) dacă
și numai dacă produsele punct v_f .x și v_f .y au același semn. Atunci F este a
familie sensibilă la localitate pentru distanța cosinusului. Parametrii sunt în esență
la fel ca pentru familia Jaccard-distance descrisă în secțiunea 3.6.2, cu excepția
scala distanțelor este mai degrabă 0-180 decât 0-1. Adică F este un
$(d_1, d_2, (180 - d_1) / 180, (180 - d_2) / 180)$ -sensibil
familie de funcții hash. Din această bază, putem amplifica familia după cum dorim,
la fel ca și pentru familia bazată pe minhash.

Pagina 139

3.7.3 Schițe

În loc să alegeți un vector aleatoriu din toți vectorii posibili, se dovedește a fi
suficient de aleatorii dacă ne restrângem alegerea la vectori ale căror componente sunt
+1 și −1. Produsul punct al oricărui vector x cu un vector v de +1 și −1
se formează prin adăugarea componentelor lui x unde v este +1 și apoi scăderea
celelalte componente ale lui x - cele unde v este -1.
Dacă alegem o colecție de vectori aleatori, să spunem $v_1, v_2, ..., v_n$, atunci putem
aplicați-le unui vector arbitrar x calculând v_1.x, v_2.x,. . . , v_n.x și apoi
înlocuind orice valoare pozitivă cu +1 și orice valoare negativă cu −1. Rezultatul este

numită schița lui x. Puteți gestiona 0 în mod arbitrar, de exemplu, alegând un rezultat +1 sau -1 la întâmplare. Deoarece există doar o mică probabilitate a unui produs cu punct zero, alegerea nu are în esență niciun efect.

Exemplul 3.22: Să presupunem că spațiul nostru este format din vectori 4-dimensionali, iar noi alegeți trei vectori aleatori: $v_1 = [+1, -1, +1, +1]$, $v_2 = [-1, +1, -1, +1]$ și $v_3 = [+1, +1, -1, -1]$. Pentru vectorul x = [3, 4, 5, 6], schița este [+1, +1, −1].

Adică $v_1 .x = 3-4 + 5 + 6 = 10$. Deoarece rezultatul este pozitiv, prima componentă din schiță este +1. În mod similar, $v_2 .x = 2$ și $v_3 .x = -4$, deci a doua componentă din schiță este +1 și a treia componentă este -1.

Se consideră vectorul y = [4, 3, 2, 1]. În mod similar putem calcula schița sa la fie [+1, −1, +1]. Deoarece schițele pentru x și y sunt de acord în 1/3 din poziții, estimăm că unghiul dintre ele este de 120 de grade. Adică o întâmplare hiperplanul ales este de două ori mai probabil să arate ca linia punctată din Fig. 3.13 decât ca linia punctată.

Concluzia de mai sus se dovedește a fi destul de greșită. Putem calcula cosinusul unghiului dintre x și y să fie xy, care este

$6 \times 1 + 5 \times 2 + 4 \times 3 + 3 \times 4 = 40$

împărțit la mărimile celor doi vectori. Aceste mărimi sunt

$\sqrt{6^2 + 5^2 + 4^2 + 3^2} = 9.274$

și $\sqrt{1^2 + 2^2 + 3^2 + 4^2} = 5.477$. Astfel, cosinusul unghiului dintre x și y este 0,7875, iar acest unghi este de aproximativ 38 de grade. Cu toate acestea, dacă te uiți la toate 16 vectori diferiți v de lungime 4 care au +1 și −1 ca componente, tu descoperiți că există doar patru dintre acestea ale căror produse dot cu x și y au un semn diferit, și anume v_2, v_3 și complementele lor [+1, −1, +1, −1] și [−1, −1, +1, +1]. Astfel, dacă am fi ales toți cei șaisprezece dintre acești vectori pentru a forma un schiță, estimarea unghiului ar fi fost de 180/4 = 45 de grade. ✷

3.7.4 Familii LSH pentru distanța euclidiană

Acum, să ne întoarcem la distanța euclidiană (secțiunea 3.5.2) și să vedem dacă putem dezvoltă o familie sensibilă la localitate de funcții hash pentru această distanță. Vom începe cu un spațiu euclidian bidimensional. Fiecare funcție hash f din familia noastră

F va fi asociat cu o linie aleasă aleatoriu în acest spațiu. Alegeți o constantă a și împărțiți linia în segmente de lungime a, așa cum este sugerat în Fig. 3.14, unde linia „aleatorie” a fost orientată să fie orizontală.

θ

Puncte la distanță

Găleată lățimea *a*

d

Figura 3.14: Două puncte la distanța d $\gg$ a au o mică șansă de a fi hash la aceeași găleată

Segmentele liniei sunt gălețile în care funcția f are puncte.

Un punct este hash la cupa în care se află proiecția sa pe linie. Dacă distanța d între două puncte este mică în comparație cu a, atunci există un bun șansa celor două puncte hash la aceeași găleată, și astfel funcția hash f va declara cele două puncte egale. De exemplu, dacă d = a / 2, atunci există cel puțin o șansă de 50% ca cele două puncte să cadă în aceeași găleată. De fapt, dacă unghiul θ între linia aleasă aleatoriu și linia care leagă punctele este mare, atunci există o șansă și mai mare ca cele două puncte să cadă la fel găleată. De exemplu, dacă θ are 90 de grade, atunci cele două puncte vor cădea în aceeași găleată.

Cu toate acestea, să presupunem că d este mai mare decât a. Pentru a exista vreo șansă cele două puncte care cad în aceeași găleată, avem nevoie de $d \cos\theta \leq a$. Diagrama de Fig. 3.14 sugerează de ce este valabilă această cerință. Rețineți că, chiar dacă $d \cos\theta \ll a$ it încă nu este sigur că cele două puncte vor cădea în aceeași găleată. In orice caz, putem garanta următoarele. Dacă $d \geq 2a$, atunci nu există mai mult de 1/3 sansa ca cele doua puncte sa cada in aceeasi galeata. Motivul este că pentru $\cos\theta$ to să fie mai mică de 1/2, trebuie să avem θ în intervalul 60-90 de grade. Dacă θ este în variază de la 0 la 60 de grade, atunci $\cos\theta$ este mai mare de 1/2. Dar, din moment ce θ este cel mai mic unghiul dintre două linii alese aleatoriu în plan, θ este de două ori mai probabil să fie între 0 și 60, deoarece trebuie să fie între 60 și 90.

Concluzionăm că familia F tocmai descrisă formează a (a / 2, 2a, 1/2, 1/3) - familie sensibilă de funcții hash. Adică, pentru distanțe de până la a / 2 probabilitatea bilitatea este de cel puțin 1/2 ca două puncte la acea distanță să cadă în aceeași găleată,

în timp ce pentru distanțe de cel puțin 2a punctele de probabilitate la acea distanță vor cădea

Pagina 141

aceeași găleată este cel mult 1/3. Putem amplifica această familie după cum ne place, la fel pentru celelalte exemple de funcții hash sensibile la localitate pe care le-am discutat.

3.7.5 Mai multe familii LSH pentru spații euclidiene

Există ceva nesatisfăcător în familia de funcții hash dezvoltate în secțiunea 3.7.4. În primul rând, tehnica a fost descrisă doar pentru două dimensiuni Spații euclidiene. Ce se întâmplă dacă datele noastre sunt puncte într-un spațiu cu multe dimensiuni? În al doilea rând, pentru distanțele Jaccard și cosinus, am putut să ne dezvoltăm familii sensibile la localitate pentru orice pereche de distanțe d_1 și d_2 atâta timp cât $d_1 < d_2$. În secțiunea 3.7.4 se pare că avem nevoie de condiția mai puternică $d_1 < 4d_2$.
Cu toate acestea, susținem că există o familie de funcții hash sensibile la localitate pentru orice $d_1 < d_2$ și pentru orice număr de dimensiuni. Hash-ul familiei funcțiile derivă încă din linii aleatorii prin spațiu și o dimensiune a cupei a care partiționează linia. Încă avem puncte hash proiectându-le pe linia. Având în vedere că $d_1 < d_2$, este posibil să nu știm care este probabilitatea p_1 ca două puncte la distanța d_1 hash la aceeași găleată, dar putem fi siguri că acesta este mai mare decât p_2, probabilitatea ca două puncte la distanța d_2 hash la aceeasi galeata. Motivul este că această probabilitate crește cu siguranță pe măsură ce distanța se micșorează. Astfel, chiar dacă nu putem calcula cu ușurință p_1 și p_2, știm că acolo este o (d_1, d_2, p_1, p_2) -sensibilă familie de funcții hash pentru orice $d_1 < d_2$ și orice număr dat de dimensiuni.
Folosind tehnicile de amplificare din secțiunea 3.6.3, putem ajusta apoi două probabilități de a înconjura orice valoare anume ne place și de a fi la fel de îndepărtați după cum ne place. Desigur, cu cât vrem să fie mai distanțați, probabil mai mare este numărul de funcții hash de bază în F pe care trebuie să le folosim.

3.7.6 Exerciții pentru secțiunea 3.7

Exercițiul 3.7.1: Să presupunem că construim familia de bază a șase localități sensibile funcții pentru vectori de lungime șase. Pentru fiecare pereche de vectori 000000, 110011, 010101 și 011100, care dintre cele șase funcții îi face candidați?
Exercițiul 3.7.2: Să calculăm schițe folosind următoarele patru „aleatorii” vectori:
$v_1 = [+1, +1, +1, -1]$ $v_2 = [+1, +1, -1, +1]$
$v_3 = [+1, -1, +1, +1]$ $v_4 = [-1, +1, +1, +1]$
Calculați schițele următoarelor vectori.
(a) [2, 3, 4, 5].
(b) [−2, 3, −4, 5].
(c) [2, −3, 4, −5].

Pagina 142

Pentru fiecare pereche, care este unghiul estimat între ele, în funcție de schițe? Care sunt adevăratele unghiuri?
Exercițiul 3.7.3: Să presupunem că formăm schițe folosind toți cei șaisprezece vectori de lungime 4, ale cărei componente sunt fiecare +1 sau −1. Calculați schițele de cei trei vectori din exercițiul 3.7.2. Cum fac estimările unghiurilor dintre fiecare pereche compară cu unghiurile adevărate?
Exercițiul 3.7.4: Să presupunem că formăm schițe folosind cei patru vectori din Exercise 3.7.2.
! (a) Care sunt constrângerile de pe a, b, c și d care vor determina schița lui vectorul [a, b, c, d] să fie [+1, +1, +1, +1]?
!! (b) Luați în considerare doi vectori [a, b, c, d] și [e, f, g, h]. Care sunt condițiile a, b,. . . , h care va face ca schițele acestor doi vectori să fie la fel?
Exercițiul 3.7.5: Să presupunem că avem puncte într-un spațiu euclidian tridimensional: $p_1 = (1, 2, 3)$, $p_2 = (0, 2, 4)$ și $p_3 = (4, 3, 2)$. Luați în considerare cele trei funcții hash definit de cele trei axe (pentru a ne face calculele foarte ușoare). Lasă gălețile să fie de lungime a, cu o găleată intervalul [0, a) (adică setul de puncte x astfel încât $0 \le x < a$), următorul [a, 2a), precedentul [−a, 0) și așa mai departe.
(a) Pentru fiecare dintre cele trei linii, atribuiți fiecare punct la cupe, presupunând a = 1.
(b) Repetați partea (a), presupunând a = 2.
(c) Care sunt perechile candidate pentru cazurile a = 1 și a = 2?

! (d) Pentru fiecare pereche de puncte, pentru ce valori ale unui testament va fi candidat pereche?

3.8 Aplicații de Hashing sensibil la localitate

În această secțiune, vom explora trei exemple despre modul în care LSH este utilizat în practică. În fiecare caz, tehnicile pe care le-am învățat trebuie modificate pentru a îndeplini anumite constrângerile problemei. Cele trei subiecte pe care le abordăm sunt:

1. Rezoluția entității: acest termen se referă la înregistrările de date potrivite la care se referă aceeași entitate din lumea reală, de exemplu, aceeași persoană. Problema principală abordat aici este că asemănarea înregistrărilor nu se potrivește exact fie seturi similare, fie modele de vectori similari de similaritate pe care teoria este construită.
2. Potrivirea amprentelor digitale: Este posibil să se reprezinte amprentele digitale ca seturi. Cu toate acestea, vom explora o altă familie de funcții hash sensibile la localitate acțiuni din cel pe care îl obținem prin minhashing.

Pagina 143

3. Potrivirea articolelor de ziar: Aici, luăm în considerare o noțiune diferită de zoster care concentrează atenția asupra articolului de bază dintr-o știre online pagina Web a hârtiei, ignorând toate materialele străine, cum ar fi reclamele și material specific ziarului.

3.8.1 Rezoluția entității

Este obișnuit să aveți la dispoziție mai multe seturi de date și să știți că se referă la acestea unele dintre aceleași entități. De exemplu, mai multe surse bibliografice diferite furnizați informații despre multe dintre aceleași cărți sau lucrări. În general caz, avem înregistrări care descriu entități de un anumit tip, cum ar fi persoane sau cărți. Înregistrările pot avea toate același format sau pot avea diferite formate, cu diferite tipuri de informații.

Există multe motive pentru care informațiile despre o entitate pot varia, chiar dacă domeniul în cauză ar trebui să fie același. De exemplu, numele pot fi exprimat diferit în diferite înregistrări din cauza greșelilor de ortografie, absenței unui inițială de mijloc, utilizarea unei porecle și multe alte motive. De exemplu, „Bob S. Jomes ”și„ Robert Jones Jr. ” poate fi sau nu aceeași persoană. Dacă înregistrările provin din surse diferite, câmpurile pot diferi și ele. O sursă înregistrările pot avea un câmp „vârstă”, în timp ce altul nu. A doua sursă poate avea un câmp „data nașterii” sau este posibil să nu aibă deloc informații despre când s-a născut o persoană.

3.8.2 Un exemplu de rezoluție a entității

Vom examina un exemplu real al modului în care LSH a fost utilizat pentru a trata cu o entitate-rezolvare problemă. Compania A a fost angajată de Compania B pentru a solicita furnizorii pentru B. Compania B ar plăti A o taxă anuală, atâta timp cât clientul și-au menținut abonamentul. Ulterior s-au certat și nu au fost de acord cu privire la modul în care mulți clienți A i-au furnizat lui B. Fiecare avea aproximativ 1.000.000 de înregistrări, unii dintre care a descris aceiași oameni; aceștia erau clienții pe care A îi oferise către B. Înregistrările aveau câmpuri de date diferite, dar din păcate niciunul dintre acestea câmpurile era „acesta este un client pe care A l-a furnizat lui B.” Astfel, problema trebuia să potrivească înregistrările din cele două seturi pentru a vedea dacă o pereche reprezenta la fel persoană.

Fiecare înregistrare avea câmpuri pentru numele, adresa și numărul de telefon al persoană. Cu toate acestea, valorile din aceste câmpuri pot diferi din mai multe motive. Nu doar au existat greșeli de ortografie și alte diferențe de denumire menționate în Secțiunea 3.8.1, dar au existat și alte oportunități de a nu fi de acord. Un client ar putea da telefonul de acasă lui A și telefonul mobil lui B. Sau ar putea mută și spune-i lui B, dar nu lui A (pentru că nu mai aveau nevoie de o relație cu). Codurile telefonice ale telefoanelor se schimbă uneori.

Strategia de identificare a înregistrărilor a implicat notarea diferențelor în trei câmpuri: nume, adresă și telefon. Pentru a crea un scor care să descrie probabilitatea că două înregistrări, una din A și cealaltă din B, au descris aceeași per-

---

Pagina 144

fiule, 100 de puncte au fost alocate fiecăruia dintre cele trei câmpuri, deci înregistrări cu exact meciurile din toate cele trei domenii au obținut un scor de 300. Cu toate acestea, au existat deduceri pentru

nepotriviri în fiecare dintre cele trei câmpuri. Ca primă aproximare, editați distanța (Secțiunea 3.5.5) a fost utilizată, dar pedeapsa a crescut patru în funcție de distanță.
Apoi, anumite tabele disponibile publicului au fost folosite pentru a reduce penalizarea în situații proprie. De exemplu, „Bill” și „William” au fost tratați ca și cum ar fi ei au diferit într-o singură literă, chiar dacă distanța de editare a acestora este de 5.
Cu toate acestea, nu este fezabil să înscrieți toate un trilion de perechi de înregistrări. Prin urmare, s-a folosit un LSH simplu pentru a se concentra asupra candidaților probabili. Trei „funcții hash” au fost folosite. Primul a trimis înregistrări la aceeași găleată numai dacă au identic nume; al doilea a făcut la fel, dar pentru adrese identice, iar al treilea a făcut-o la fel și pentru numerele de telefon. În practică, nu a existat nici un hashing; mai degrabă înregistrările au fost sortate după nume, astfel încât vor apărea înregistrări cu nume identice consecutiv și obțineți punctaje pentru similaritatea generală a numelui, adresei și telefon. Apoi, înregistrările au fost sortate în funcție de adresă și de cele cu aceeași adresa au fost punctate. În cele din urmă, înregistrările au fost sortate a treia oară prin telefon, și au fost înregistrate înregistrări cu telefoane identice.
Această abordare a ratat o pereche de discuri care reprezenta cu adevărat aceeași persoană dar niciunul dintre cele trei câmpuri nu se potrivea exact. Întrucât scopul era să dovedim o instanță de judecată că persoanele erau aceleași, este puțin probabil ca o astfel de pereche ar fi fost acceptat de un judecător ca fiind suficient de similar oricum.

3.8.3 Validarea meciurilor de înregistrare

Ceea ce rămâne este să determinați cât de mare este un scor care indică faptul că două înregistrări sunt cu adevărat reprezintă același individ. În exemplul de față, a existat un lucru ușor mod de a lua acea decizie, iar tehnica poate fi aplicată în multe similare situații. S-a decis să se analizeze datele de creare a înregistrărilor la îndemână, și să presupunem că 90 de zile a fost o întârziere maximă absolută între timp serviciul a fost cumpărat la Compania A și înregistrat la B. Astfel, s-a propus potrivire între două înregistrări alese la întâmplare, sub rezerva numai a constrângere că data de pe înregistrarea B a fost între 0 și 90 de zile după data înregistrării A, ar avea o întârziere medie de 45 de zile.
S-a constatat că dintre perechile cu un scor perfect de 300, întârzierea medie a fost 10 zile. Dacă presupui că perechile de 300 de puncte sunt cu siguranță meciuri corecte, atunci tu poate privi grupul de perechi cu orice scor dat și poate calcula media întârzierea acelor perechi. Să presupunem că întârzierea medie este x și fracția de potrivirile adevărate dintre acele perechi cu scor s este f. Atunci $x = 10f + 45 (1 - f)$, sau $x = 45 - 35f$. Rezolvând pentru f, găsim că fracția perechilor cu scor s care sunt cu adevărat potrivite este $(45 - x) / 35$.
Același truc poate fi folosit ori de câte ori:
1. Există un sistem de notare utilizat pentru a evalua probabilitatea ca două înregistrări reprezintă aceeași entitate și

---

Când sunt suficient de potrivite înregistrările?

Deși fiecare caz va fi diferit, poate fi de interes să știți cum
Experimentul din secțiunea 3.8.3 sa dovedit pe datele din secțiunea 3.8.2. Pentru scoruri până la 185, valoarea lui x a fost foarte aproape de 10; adică aceste scoruri a indicat faptul că probabilitatea înregistrărilor care reprezintă aceeași persoană a fost în esență 1. Rețineți că un scor de 185 în acest exemplu reprezintă un situație în care un câmp este același (așa cum ar trebui să fie cazul sau înregistrările nu ar fi niciodată marcate), un câmp era complet diferit, iar al treilea câmp avea o mică discrepanță. Mai mult, pentru scoruri cât mai mici 115, valoarea lui x a fost vizibil mai mică de 45, ceea ce înseamnă că unele dintre acestea perechile au reprezentat aceeași persoană. Rețineți că un scor de 115 reprezintă un caz în care un câmp este același, dar există doar o ușoară similitudine în celelalte două domenii.
2. Există un câmp, neutilizat în scor, din care putem deriva un măsură care diferă, în medie, pentru perechile adevărate și perechile false.
De exemplu, să presupunem că a existat un câmp „înălțime” înregistrat de ambele companii A și B în exemplul nostru de rulare. Putem calcula diferența medie în înălțime pentru perechi de înregistrări aleatorii și putem calcula diferența medie în înălțime pentru înregistrările care au un scor perfect (și astfel reprezintă cu siguranță același entități). Pentru un scor dat, putem evalua diferența medie de înălțime a perechi cu acel scor și estimează probabilitatea înregistrărilor care reprezintă aceeași entitate. Adică, dacă h_0 este diferența medie de înălțime pentru perfect meciuri, h_1 este diferența medie de înălțime pentru perechile aleatorii, iar h este diferența medie de înălțime pentru perechile de scoruri, apoi fracția de perechi bune cu scorul s este $(h_1 - h) / (h_1 - h_0)$.

3.8.4 Potrivirea amprentelor digitale

Când amprentele digitale sunt asortate de computer, reprezentarea obișnuită nu este o imagine, dar un set de locații în care sunt localizate detaliile. O minusculă, în contextul descrierilor de amprentă, este un loc unde ceva neobișnuit se întâmplă, cum ar fi fuzionarea a două creste sau un sfârșit de creste. Dacă așezăm o grilă peste o amprentă digitală, putem reprezenta amprenta digitală prin setul de pătrate de rețea în care miniae sunt localizate.

În mod ideal, înainte de suprapunerea grilei, amprentele sunt normalizate pentru dimensiune și orientare, astfel încât dacă am lua două imagini cu același deget, am găsi detalii situate exact în aceleași pătrate ale grilei. Nu vom lua în considerare aici cele mai bune moduri de a normaliza imaginile. Să presupunem că o combinație de tehnici, inclusiv alegerea dimensiunii grilei și plasarea unei amănunte în mai multe adiacente grilele pătrate dacă se află aproape de marginea pătratelor ne permite să presupunem

---

Pagina 146

că pătratele de grilă din două imagini au o probabilitate semnificativ mai mare de de acord în prezența sau absența unei amănunte decât dacă ar fi din imagini de diferite degete.

Astfel, amprentele digitale pot fi reprezentate prin seturi de pătrate ale grilei - cele unde detaliile lor sunt localizate - și comparate ca orice seturi, folosind simularea Jaccard ilaritate sau distanță. Cu toate acestea, există două versiuni de comparare a amprentelor digitale.

- Problema multi-unu este cea la care ne așteptăm de obicei. O amprentă are a fost găsit pe un pistol și vrem să-l comparăm cu toate amprentele digitale într-o bază de date mare, pentru a vedea care se potrivește.
- Versiunea mult-multe a problemei este de a lua întreaga bază de date și vezi dacă există perechi care să reprezinte același individ.

În timp ce versiunea multe-multe se potrivește cu modelul pe care l-am urmat pentru a găsi articole similare, aceeași tehnologie poate fi utilizată pentru a accelera multe-o problemă.

3.8.5 O familie LSH pentru potrivirea amprentelor digitale

Am putea elimina seturile care reprezintă o amprentă și să folosim standardul Tehnica LSH din secțiunea 3.4. Cu toate acestea, întrucât seturile sunt alese dintre a un set relativ mic de puncte de rețea (poate 1000), necesitatea de a le minimiza în semnături mai succinte nu este clar. Vom studia aici o altă formă de hash sensibil la localitate care funcționează bine pentru date de tipul pe care îl discutăm.

Să presupunem, pentru un exemplu, că probabilitatea de a găsi o miniatură în mod aleatoriu pătratul grilei unei amprente aleatorii este de 20%. De asemenea, presupuneți că dacă există două amprente provin de la același deget, iar unul are o miniatură într-un pătrat dat, apoi probabilitatea ca și celălalt să fie de 80%. Putem defini o localitate sensibilă familia de funcții hash după cum urmează. Fiecare funcție f din această familie F este definită de trei pătrate de grilă. Funcția f spune „da” pentru două amprente dacă ambele au detalii în toate cele trei pătrate ale grilei, iar altfel f spune „nu”. Pune altul într-un fel, ne putem imagina că f trimite la o singură găleată toate amprentele digitale pe care le au detalii în toate cele trei puncte ale grilei lui f și își trimite reciproc amprenta digitală către a găleată proprie. În cele ce urmează, ne vom referi la prima dintre aceste găleți ca

,,Găleata” pentru f și ignora gălețile care trebuie să fie singletoni.

Dacă vrem să rezolvăm problema one-one, putem folosi multe funcții din familia F și precalculează gălețile de amprente la care răspund "da." Apoi, având în vedere o nouă amprentă pe care dorim să o potrivim, determinăm cui dintre aceste găleți îi aparține și compară-l cu toate amprentele digitale găsite în oricare dintre găleți. Pentru a rezolva problema multe-multe, calculăm gălețile pentru fiecare dintre funcții și comparați toate amprentele din fiecare dintre găleți.

Să luăm în considerare câte funcții avem nevoie pentru a obține o probabilitate rezonabilă de a prinde un chibrit, fără a fi nevoie să comparați amprenta de pe pistol cu fiecare dintre milioanele de amprente din baza de date. În primul rând, probabilitatea ca

---

Pagina 147

două amprente de la degete diferite ar fi în găleată pentru o funcție f în F este $(0,2)^6 = 0,000064$. Motivul este că ambii vor intra în găleată numai dacă fiecare are câte o minusculă în fiecare dintre cele trei puncte ale grilei asociate

f, iar probabilitatea fiecăruia dintre aceste șase evenimente independente este de 0,2.
Acum, ia în considerare probabilitatea ca două amprente din același deget
înfășurați-vă în găleată pentru f. Probabilitatea pe care o are prima amprentă
detaliile din fiecare dintre cele trei pătrate aparținând lui f sunt $(0,2)^3 = 0,008$. In orice caz,
dacă da, atunci probabilitatea este $(0,8)^3 = 0,512$ decât cealaltă amprentă
va face, de asemenea. Astfel, dacă amprentele sunt de la același deget, există un
$0,008 \times 0,512 = 0,004096$ probabilitate ca ambele să fie în găleată de f.
Asta nu este mult; este aproximativ una din 200. Totuși, dacă folosim multe funcții
din F, dar nu prea multe, atunci putem obține o bună probabilitate de potrivire
amprente de la același deget fără a avea prea multe pozitive false -
amprente care trebuie luate în considerare, dar care nu se potrivesc.
Exemplul 3.23: Pentru un exemplu specific, să presupunem că folosim 1024
funcții alese aleatoriu din F. Apoi, vom construi o nouă familie
ily F_1 prin efectuarea 1024 căi sau pe F. Apoi , probabilitatea ca F_1
va pune amprente de la același deget împreună în cel puțin o găleată este
$1 - (1 - 0,004096)^{1024} = 0,985$. Pe de altă parte, probabilitatea ca
două amprente de la degete diferite vor fi plasate în aceeași găleată
$(1 - (1 - 0,000064)^{1024} = 0,063$. Adică, obținem aproximativ 1,5% false negative
și aproximativ 6,3% fals pozitive. ✷
Rezultatul Exemplului 3.23 nu este cel mai bun pe care îl putem face. În timp ce oferă doar un
1,5% șanse să nu reușim să identificăm amprenta de pe pistol, aceasta forțează
să ne uităm la 6,3% din întreaga bază de date. Creșterea numărului de funcții
de la F va crește numărul de fals pozitivi, cu doar un beneficiu mic
de reducere a numărului de negative negative sub 1,5%. Pe de altă parte, noi
putem folosi, de asemenea, construcția AND și, făcând acest lucru, putem reduce foarte mult
probabilitatea unui fals pozitiv, în timp ce face doar o mică creștere a
rata fals-negativă. De exemplu, am putea lua 2048 de funcții din F în două
grupuri de 1024. Construiți gălețile pentru fiecare dintre funcții. Cu toate acestea, dat
o amprentă P pe pistol:
1. Găsiți gălețile din primul grup din care aparține P și luați
unirea acestor găleți.
2. Faceți același lucru pentru al doilea grup.
3. Luați intersecția celor două uniuni.
4. Comparați P numai cu acele amprente din intersecție.
Rețineți că trebuie să luăm în continuare uniuni și intersecții de seturi mari de
imprimeuri, dar comparăm doar o mică parte din acestea. Este comparația dintre

---

amprente care iau cea mai mare parte a timpului; în pașii (1) și (2) amprente
pot fi reprezentate prin indicii lor întregi în baza de date.
Dacă folosim această schemă, probabilitatea de a detecta o amprentă digitală potrivită
este $(0,985)^2 = 0,970$; adică obținem aproximativ 3% negative negative. Însă
probabilitatea unui fals pozitiv este $(0,063)^2 = 0,00397$. Adică nu trebuie decât să facem asta
examinați aproximativ 1/ 250 din baza de date.

3.8.6 Articole de Ştiri similare

Ultimul nostru studiu de caz se referă la problema organizării unui depozit mare de
articole de știri on-line prin gruparea de pagini web din care au fost derivate
același text de bază. Este comun pentru organizații precum The Associated Press
să producă o știre și să o distribuie în multe ziare. Fiecare ziar
pune povestea în ediția sa online, dar o înconjoară cu informații care sunt
special pentru ziarul respectiv, cum ar fi numele și adresa ziarului,
linkuri către articole similare și linkuri către reclame. În plus, este comun pentru
ziar pentru a modifica articolul, poate lăsând în urmă ultimele paragrafe
sau chiar ștergerea textului din mijloc. Ca urmare, același articol de știri poate
apar destul de diferite pe site-urile web ale diferitelor ziare.
Problema seamănă foarte mult cu cea sugerată în secțiunea 3.4:
găsiți documente ale căror sindrile au o mare asemănare Jaccard. Rețineți că acest lucru
problema este diferită de problema găsirii articolelor de știri care să spună despre
aceleași evenimente. Această din urmă problemă necesită alte tehnici, de obicei examinate
setul de cuvinte importante din documente (un concept pe care l-am discutat pe scurt
în secțiunea 1.3.1) și gruparea acestora pentru a grupa diferite articole despre
același subiect.
Cu toate acestea, sa constatat că există o variație interesantă pe tema șindrilelor
mai eficient pentru datele de tipul descris. Problema este atât de șindrilă
l-am descris în secțiunea 3.2 tratează în mod egal toate părțile unui document. In orice caz,
dorim să ignorăm părți din document, cum ar fi reclame sau titlurile altora

articole la care ziarul a adăugat un link, care nu fac parte din ştiri
articol. Se pare că există o diferenţă vizibilă între textul care
apare în proză şi text care apare în reclame sau titluri. Proza are mult
o frecvenţă mai mare a cuvintelor de oprire, cuvintele foarte frecvente precum „the” sau „şi”.
Numărul total de cuvinte care sunt considerate cuvinte stop variază în funcţie de
aplicaţie, dar este obişnuit să folosiţi o listă de câteva sute dintre cele mai multe
cuvinte frecvente.
Exemplul 3.24: un anunţ tipic ar putea spune pur şi simplu „Cumpăraţi Sudzo”. Pe de altă parte
mână, o versiune în proză a aceluiaşi gând care ar putea apărea într-un articol este
,,Vă recomand să cumpăraţi Sudzo pentru rufe.” În ultima propoziţie, it
ar fi normal să tratezi „eu”, „asta”, „tu”, „pentru” şi „tău” ca pe cuvinte de oprire.
✷
Să presupunem că definim o şindrilă ca fiind un cuvânt de oprire urmat de următorii doi
cuvinte. Apoi, anunţul „Cumpăraţi Sudzo” din Exemplul 3.24 nu are şindrilă şi

---

Pagina 149

nu se va reflecta în reprezentarea paginii Web care conţine acest lucru
anunţ. Pe de altă parte, propoziţia din Exemplul 3.24 ar fi reprezentată
de cinci şindrilă: „Vă recomandăm”, „să cumpăraţi”, „cumpăraţi Sudzo”, „pentru
rufele tale ”şi„ rufele tale x ”, unde x este orice cuvânt urmează
propoziţie.
Să presupunem că avem două pagini web, fiecare dintre ele constând din jumătate de text de ştiri
şi jumătate de reclame sau alt material care are o densitate redusă de cuvinte stop. Dacă veştile
textul este acelaşi, dar materialul din jur este diferit, atunci ne-am aştepta
că o fracţiune mare din zona zoster a celor două pagini ar fi aceeaşi. ei
ar putea avea o asemănare Jaccard de 75%. Cu toate acestea, dacă materialul înconjurător
este acelaşi, dar conţinutul ştirilor este diferit, apoi numărul comun
zona zoster ar fi mică, poate 25%. Dacă ar fi să folosim convenţionalul
zoster, unde zoster este (să zicem) secvenţe de 10 caractere consecutive, noi
s-ar aştepta ca cele două documente să-şi împartă jumătate din zona zoster (adică un Jaccard
similaritate de 1/3), indiferent dacă a fost ştirea sau împrejurimile
material pe care l-au împărtăşit.

3.8.7 Exerciţii pentru secţiunea 3.8

Exerciţiul 3.8.1: Să presupunem că încercăm să realizăm rezoluţia entităţii printre
referinţe bibliografice şi punctăm perechi de referinţe pe baza
titlurile lor, lista autorilor şi locul publicării. Să presupunem şi că
toate referinţele includ un an de publicare, iar anul acesta este la fel de probabil
oricare dintre cei zece ultimii ani. Mai mult, să presupunem că descoperim că printre
perechile de referinţe cu un scor perfect, există o diferenţă medie în
anul publicării de 0,1. [7] Să presupunem că perechile de referinţe cu un anumit
scorurile s au o diferenţă medie în datele de publicare de 2.
Care este fracţiunea de perechi cu scor care reprezintă cu adevărat aceeaşi publicaţie?
licataţie? Notă: Nu faceţi greşeala de a asuma diferenţa medie
în data publicării între perechile aleatorii este 5 sau 5,5. Trebuie să o calculezi
exact şi aveţi suficiente informaţii pentru a face acest lucru.
Exerciţiul 3.8.2: Să presupunem că folosim familia F de funcţii descrise în secţiunea
3.8.5, unde există o şansă de 20% pentru o miniatură într-un pătrat de reţea, un 80%
şansa ca o a doua copie a unei amprente digitale să aibă o amănuntă într-un pătrat de reţea unde
prima copie o face şi fiecare funcţie din F fiind formată din trei pătrate de grilă.
În Exemplul 3.23, am construit familia F_1 utilizând construcţia OR pe
1024 membri ai F. Să presupunem că am folosit în schimb familia F_2 care este un SAU 2048-way
a membrilor F.
(a) Calculaţi ratele falsurilor pozitive şi falselor negative pentru F_2 .
(b) Cum se compară aceste rate cu ceea ce obţinem dacă organizăm aceleaşi
2048 funcţii într - un mod 2 şi ale membrilor F_1 , aşa cum a fost discutat la
sfârşitul secţiunii 3.8.5?

[7] Ne-am putea aştepta ca media să fie 0, dar, în practică, apar erori în anul publicării.

Pagina 150

Exerciţiul 3.8.3: Să presupunem că amprentele digitale au aceleaşi statistici prezentate în Ex.
ercise 3.8.2, dar folosim o familie de bază de funcţii F' definite ca F, dar folosind
doar două pătrate alese aleatoriu. Construiţi un alt set de funcţii F'

$_1$
din F ′ luând modul n OR al funcțiilor din F ′. Ce, în funcție de
n, sunt ratele fals pozitive și fals negative pentru F ′
$_1$?
Exercițiul 3.8.4: Să presupunem că folosim funcțiile F $_1$ din Exemplul 3.23, dar noi
vreau să rezolv problema multe-multe.
(a) Dacă două amprente sunt de la același deget, care este probabilitatea ca
nu vor fi comparate (adică, care este rata fals negativă)?
(b) Ce fracțiune a amprentelor digitale de la diferite degete va fi comparată
(adică, care este rata fals pozitivă)?
! Exercițiul 3.8.5: Să presupunem că avem setul de funcții F ca în Exercițiul 3.8.2,
și construim un nou set de funcții F $_3$ printr-un mod OR de funcții n
F. Pentru ce valoare a lui n este suma ratelor fals pozitive și fals negative
minimizat?

3.9 Metode pentru grade ridicate de similitudine

Metodele bazate pe LSH par cele mai eficiente atunci când gradul de similitudine noi
accept este relativ scăzut. Când vrem să găsim seturi aproape identice,
există alte metode care pot fi mai rapide. Mai mult, aceste metode sunt exacte,
prin faptul că găsesc fiecare pereche de obiecte cu gradul de asemănare dorit. Acolo
nu există negative negative, așa cum se poate întâmpla cu LSH.

3.9.1 Găsirea articolelor identice

Cazul extrem este găsirea unor articole identice, de exemplu, pagini web care sunt
identic, caracter-pentru-caracter. Este simplu să comparăm două documente
menționează și spuneți dacă sunt identice, dar totuși trebuie să evităm să facem acest lucru
comparați fiecare pereche de documente. Primul nostru gând ar fi să hash docu-
menționate pe baza primelor lor caractere și compară doar acele documente
care a căzut în aceeași găleată. Schema respectivă ar trebui să funcționeze bine, cu excepția cazului în care toate
documentele încep cu aceleași caractere, cum ar fi un antet HTML.
Al doilea gând ar fi să folosim o funcție hash care examinează
întregul document. Asta ar funcționa și dacă vom folosi suficiente găleți, ar fi
foarte rar că două documente au intrat în aceeași găleată, dar nu erau identice.
Dezavantajul acestei abordări este că trebuie să examinăm fiecare caracter al fiecăruia
document. Dacă ne limităm examinarea la un număr mic de caractere, atunci
nu trebuie să examinăm niciodată un document unic și care se încadrează într-o găleată de
propria.
O abordare mai bună este alegerea unor poziții aleatorii fixe pentru toate documentele,
și să facă funcția hash să depindă numai de acestea. În acest fel, putem evita

---

o problemă în care există un prefix comun pentru toate sau majoritatea documentelor, totuși noi
nu trebuie să examinați documente întregi decât dacă acestea cad într-o găleată cu altul
document. O problemă cu selectarea pozițiilor fixe este că, dacă unele documente
sunt scurte, este posibil să nu aibă unele dintre pozițiile selectate. Cu toate acestea, dacă suntem
căutând documente foarte similare, nu trebuie să comparăm niciodată două documente
care diferă semnificativ prin lungimea lor. Exploatăm această idee în secțiunea 3.9.3.

3.9.2 Reprezentarea seturilor ca șiruri

Acum, să ne concentrăm pe problema mai dificilă a găsirii, într-o mare colecție de seturi,
toate perechile care au o asemănare mare cu Jaccard, spun cel puțin 0,9. Putem reprezenta
un set prin sortarea elementelor setului universal într-o anumită ordine fixă și
reprezentând orice set prin enumerarea elementelor sale în această ordine. Lista este în esență
un șir de „caractere”, în care personajele sunt elementele universalului
a stabilit. Aceste șiruri sunt neobișnuite, totuși, prin faptul că:
1. Niciun caracter nu apare de mai multe ori într-un șir și
2. Dacă două caractere apar în două șiruri diferite, atunci ele apar în
aceeași ordine în ambele șiruri.
Exemplul 3.25: Să presupunem că setul universal este format din cele 26 de litere mici,
și folosim ordinea alfabetică normală. Apoi mulțimea {d, a, b} este reprezentată
de sfoara abd. ✷
În cele ce urmează, vom presupune că toate șirurile reprezintă seturi în modul corect
descris. Astfel, vom vorbi despre asemănarea Jaccard a șirurilor, când
strict vorbind, ne referim la similitudinea seturilor pe care le reprezintă șirurile.
De asemenea, vom vorbi despre lungimea unui șir, ca un surogat pentru numărul de
elemente din setul pe care îl reprezintă șirul.
Rețineți că documentele discutate în secțiunea 3.9.1 nu se potrivesc exact
acest model, chiar dacă putem vedea documentele ca șiruri. Pentru a se potrivi modelului,

am fi șindrilat documentele, am atribui o ordine șindrilelor și am reprezenta
fiecare document prin lista sa de zona zoster în ordinea selectată.

3.9.3 Filtrare bazată pe lungime

Cel mai simplu mod de a exploata reprezentarea șirului din secțiunea 3.9.2 este să sortați
corzile după lungime. Apoi, fiecare șir s este comparat cu acele șiruri t care
urmați s în listă, dar nu sunt prea lungi. Să presupunem că limita inferioară de pe Jaccard
similitudinea dintre două șiruri este J. Pentru orice șir x, denotați lungimea lui cu L_x.
Rețineți că $L_s \leq L_t$. Intersecția mulțimilor reprezentate de s și t nu poate
au mai mult de L_s membri, în timp ce uniunea lor are cel puțin L_t membri. Prin urmare,
similitudinea Jaccard a lui s și t, pe care o denotăm SIM (s, t), este cel mult L_s / L_t.
Adică, pentru ca s și t să necesite comparație, trebuie să fie că $J \leq L_s / L_t$,
sau echivalent, $L_t \leq L_s / J$.

O comandă mai bună pentru simboluri

În loc să folosiți ordinea evidentă pentru elemente ale setului universal, de exemplu,
ordinea lexicografică pentru zona zoster, putem comanda mai întâi simbolurile cele mai rare. Acea
este, determinați de câte ori apare fiecare element în colecția de
seturi și ordonați-le după acest număr, mai întâi cel mai mic. Avantajul de a face
la fel este că simbolurile din prefixe vor tinde să fie rare. Astfel, vor face
face ca acel șir să fie plasat în găleți index care au relativ puține
membrii. Apoi, când trebuie să examinăm un șir pentru posibile potriviri,
vom găsi câteva alte șiruri care pot fi comparate.

Exemplul 3.26: Să presupunem că s este un șir de lungime 9 și căutăm
șiruri cu cel puțin 0,9 similaritate Jaccard. Atunci nu trebuie decât să comparăm s
cu șiruri care îl urmează în ordinea sortată pe lungime, care au lungimea la
cele mai multe 9 / 0,9 = 10. Adică, comparăm s cu acele șiruri de lungime 9 care
urmați-o în ordine și toate șirurile de lungime 10. Nu avem nevoie să comparăm s
cu orice alt șir.

Să presupunem că lungimea lui s a fost în schimb 8. Atunci s-ar compara cu
urmând șiruri de lungime de până la 8 / 0,9 = 8,89. Adică un șir de lungime 9
ar fi prea lung pentru a avea o similaritate Jaccard de 0,9 cu s, deci trebuie doar să
comparați s cu șirurile care au lungimea 8, dar urmați-o în ordinea sortată.

✷

3.9.4 Indexarea prefixelor

În plus față de lungime, există și alte câteva caracteristici ale șirurilor
exploatat pentru a limita numărul de comparații care trebuie făcute pentru identificare
toate perechile de corzi similare. Cea mai simplă dintre aceste opțiuni este crearea unui index
pentru fiecare simbol; amintește un simbol al unui șir este oricare dintre elementele din
set universal. Pentru fiecare șir s, selectăm un prefix al lui s format din primul p
simboluri ale s. Cât de mare trebuie să fie p depinde de L_s și J, limita inferioară de pe
Similitudine Jaccard. Adăugăm șirul s la index pentru fiecare dintre primele sale simboluri p.
De fapt, indexul pentru fiecare simbol devine o găleată de șiruri care trebuie să fie
comparat. Trebuie să fim siguri că orice alt șir t astfel încât SIM (s, t) $\geq$ J
va avea cel puțin un simbol în prefixul său care apare și în prefixul lui s.

Să presupunem că nu; mai degrabă SIM (s, t) $\geq$ J, dar t nu are nici unul din primele p simboluri ale
s. Atunci cea mai mare similaritate Jaccard pe care o pot avea s și t apare atunci când t este
un sufix al lui s, format din orice, cu excepția primelor p simboluri ale lui s. Jaccard
similaritatea lui s și t ar fi atunci $(L_s - p) / L_s$. Pentru a fi siguri că nu
trebuie să comparăm s cu t, trebuie să fim siguri că $J > (L_s - p) / L_s$. Acea
este, p trebuie să fie cel puțin $\lfloor (1 - J) L_s \rfloor + 1$. Desigur, vrem ca p să fie la fel de mic ca
posibil, deci nu indexăm șirurile în mai multe găleți decât trebuie. Prin urmare,
vom lua în continuare $p = \lfloor (1 - J) L_s \rfloor + 1$ pentru a fi lungimea prefixului care

---

se indexează.

Exemplul 3.27: Să presupunem că J = 0,9. Dacă $L_s = 9$, atunci $p = \lfloor 0.1 \times 9 \rfloor + 1 = \lfloor 0.9 \rfloor + 1 = 1$. Adică, trebuie să indexăm s numai sub primul său simbol. Orice
șirul t care nu are primul simbol al lui s într-o poziție astfel încât t este
indexat cu acel simbol va avea asemănare Jaccard cu s care este mai mică de 0,9.

Să presupunem că s este bcdefghij. Atunci s este indexat numai sub b. Să presupunem că t nu

începe cu b. Există două cazuri de luat în considerare.
1. Dacă t începe cu a și SIM (s, t) ≥ 0,9, atunci poate fi doar că t este
abcdefghij. Dar dacă acesta este cazul, t va fi indexat atât sub a cât și pe
b. Motivul este că L_t = 10, deci t va fi indexat sub simbolurile lui
prefixul său de lungime ⌊0.1 × 10⌋ + 1 = 2.
2. Dacă t începe cu c sau o literă ulterioară, atunci valoarea maximă a cartelei SIM (s, t)
apare când t este cdefghij. Dar apoi SIM (s, t) = 8/9 <0,9.
În general, cu J = 0,9, șirurile de lungime de până la 9 sunt indexate după prima lor
simbol, șirurile de lungimi 10-19 sunt indexate sub primele două simboluri ale acestora,
șirurile de lungime 20-29 sunt indexate sub primele lor trei simboluri și așa mai departe.
✷
Putem folosi schema de indexare în două moduri, în funcție de dacă noi
încearcă să rezolve problema multe-multe sau o problemă multe-unu; amintesc de
distincția a fost introdusă în secțiunea 3.8.4. Pentru problema multora, noi
creați indexul pentru întreaga bază de date. Pentru a căuta potriviri la un set nou
S, convertim setul respectiv într-un șir s, pe care îl numim șir de probă. A determina
lungimea prefixului care trebuie luat în considerare, adică ⌊ (1 - J) L_s ⌋ + 1. Pentru
fiecare simbol care apare într-una din pozițiile prefixului lui s, ne uităm în index
bucket pentru acel simbol și comparăm s cu toate șirurile care apar în acel
găleată.
Dacă vrem să rezolvăm problema multe, începeți cu o bază de date goală
de șiruri și indici. Pentru fiecare set S, tratăm S ca un set nou pentru mulți
problemă. Transformăm S într-un șir s, pe care îl tratăm ca un șir de probă în
multe-o problemă. Cu toate acestea, după ce examinăm o bucket index, adăugăm și s
cu acel cup, deci s va fi comparat cu șirurile ulterioare care ar putea fi potrivite.

3.9.5 Utilizarea informațiilor de poziție

Luați în considerare șirurile s = acdefghijk și t = bcdefghijk și presupuneți J = 0.9.
Deoarece ambele șiruri au lungimea 10, acestea sunt indexate sub primele două
simboluri. Astfel, s este indexat sub a și c, în timp ce t este indexat sub b și c.
Oricare se adaugă ultima o va găsi pe cealaltă în găleată pentru c, și vor fi
comparat. Cu toate acestea, deoarece c este al doilea simbol al ambelor, știm că va exista
fie două simboluri, a și b în acest caz, care se află în uniunea celor două seturi, dar
nu în intersecție. Într-adevăr, chiar dacă s și t sunt identice de la c la

---

Pagina 154

la sfârșit, intersecția lor este de 9 simboluri și unirea lor este de 11; astfel SIM (s, t) = 9/11,
care este mai mic de 0,9.
Dacă ne construim indexul bazându-ne nu numai pe simbol, ci și pe poziția
simbolul din șir, am putea evita compararea lui s și t de mai sus. Acea
este, lăsați indexul nostru să aibă o bucket pentru fiecare pereche (x, i), care conține șirurile care
au simbolul x în poziția i a prefixului lor. Având un șir s, și presupunând că J este
similaritatea minimă dorită a lui Jaccard, ne uităm la prefixul lui s, adică la
pozițiile de la 1 la ⌊ (1 - J) L_s ⌋ + 1. Dacă simbolul din poziția i a prefixului este
x, adăugați s la cupa index pentru (x, i).
Acum considerați s ca un șir de probă. Cu ce găleți trebuie comparat?
Vom vizita simbolurile prefixului lui s din stânga și vom lua
avantajul faptului că trebuie să găsim un posibil șir de potrivire t numai dacă
niciunul dintre cupele anterioare pe care le-am examinat pentru meciurile ținute t. Adică noi
trebuie să găsesc o singură dată un candidat. Astfel, dacă găsim că simbolul ith
lui s este x, atunci trebuie să căutăm în cupa (x, j) anumite valori mici ale lui j.
j
s
t
Simboluri cu siguranță
apărând în
un singur șir
eu
Figura 3.15: Șirurile s și t încep cu i - 1 și j - 1 simboluri unice, respectiv
și apoi sunt de acord dincolo de asta
Pentru a calcula limita superioară pe j, să presupunem că t este un șir din care niciunul nu este primul
j -1 simboluri s-au potrivit cu orice în s, dar simbolul ith al lui s este același cu
al șaselea simbol al lui t. Cea mai mare valoare a cartelei SIM (s, t) apare dacă s și t sunt identice
dincolo de simbolurile lor ith și respectiv jth, așa cum sugerează Fig. 3.15. Dacă
acesta este cazul, dimensiunea intersecției lor este L_s - i + 1, deoarece acesta este
numărul de simboluri ale lui s care ar putea fi în t. Dimensiunea uniunii lor este
cel puțin L_s + j - 1. Adică, s contribuie cu siguranță L_s simboluri la uniune și

există, de asemenea, cel puțin j -1 simboluri ale lui t care nu sunt în s. Raportul dintre dimensiuni intersecției și uniunii trebuie să fie cel puțin J, deci trebuie să avem:

$L_s - i + 1$

$L_s + j - 1$

$\geq J$

Dacă izolăm j în această inegalitate, avem $j \leq (L_s (1 - J) - i + 1 + J) / J$.

Exemplul 3.28: Luați în considerare șirul s = acdefghijk cu J = 0.9 discutat la începutul acestei secțiuni. Să presupunem că s este acum un șir de probă. Noi deja a stabilit că trebuie să luăm în considerare primele două poziții; adică pot fi 1

---

sau 2. Să presupunem că i = 1. Atunci $j \leq (10 \times 0{,}1 - 1 + 1 + 0{,}9) / 0{,}9$. Adică numai noi trebuie să comparați simbolul a cu șiruri în găleată pentru (a, j) dacă $j \leq 2.11$. Astfel, j poate fi 1 sau 2, dar nimic mai mare.

Acum să presupunem că i = 2. Atunci avem nevoie de $j \leq (10 \times 0{,}1 - 2 + 1 + 0{,}9) / 0{,}9$, sau $j \leq 1$. Concluzionăm că trebuie să căutăm în găleți (a, 1), (a, 2) și (c, 1), dar în nicio altă găleată. În comparație, folosind cupele din secțiunea 3.9.4, noi ar căuta în găleți după a și c, ceea ce este echivalent cu privirea pentru toți găleți (a, j) și (c, j) pentru orice j. ✷

3.9.6 Utilizarea poziției și lungimii în indexuri

Când am considerat limita superioară pentru j în secțiunea anterioară, am presupus că ceea ce urmează pozițiilor i și j a fost ca în Fig. 3.15, unde ceea ce a urmat aceste poziții în șiruri s și t s-au potrivit exact. Nu vrem să construim un index care implică fiecare simbol din șiruri, deoarece asta face totalul munca excesiva. Cu toate acestea, putem adăuga la indexul nostru un rezumat al celor ce urmează pozițiile fiind indexate. Acest lucru extinde numărul de găleți, dar nu dincolo de limitele rezonabile și totuși ne permite să eliminăm mulți candidați se potrivește fără a compara șiruri întregi. Ideea este să folosiți găleți index corespunzând unui simbol, unei poziții și lungimii sufixului, adică numărul de simboluri care urmează poziția în cauză.

Exemplul 3.29: Șirul s = acdefghijk, cu J = 0.9, ar fi indexat în găleți pentru (a, 1, 9) și (c, 2, 8). Adică prima poziție a lui s are simbol a, iar sufixul său are lungimea 9. A doua poziție are simbolul c și sufixul său este de lungime 8. ✷

Figura 3.15 presupune că sufixele pentru poziția i a lui s și poziția j a lui t au aceeași lungime. Dacă nu, atunci putem obține fie o margine superioară mai mică mărimea intersecției lui s și t (dacă t este mai scurtă) sau o margine inferioară mai mare pe mărimea uniunii (dacă t este mai lung). Să presupunem că s are lungimea sufixului p și t are lungimea sufixului q.

Cazul 1: $p \geq q$. Aici, dimensiunea maximă a intersecției este

$L_s - i + 1 - (p - q)$

Deoarece $L_s = i + p$, putem scrie expresia de mai sus pentru dimensiunea intersecției ca q + 1. Dimensiunea minimă a uniunii este $L_s + j - 1$, așa cum a fost atunci când nu am făcut-o luați în considerare lungimea sufixului. Astfel, avem nevoie

$q + 1$

$L_s + j - 1$

$\geq J$

ori de câte ori $p \geq q$.

Cazul 2: $p < q$. Aici, dimensiunea maximă a intersecției este $L_s - i + 1$, ca când lungimea sufixului nu a fost luată în considerare. Cu toate acestea, dimensiunea minimă a uniunii este acum $L_s + j - 1 + q - p$. Dacă folosim din nou relația $L_s = i + p$, putem înlocuiți $L_s - p$ cu i și obțineți formula $i + j - 1 + q$ pentru dimensiunea uniunii.

Dacă similitudinea Jaccard este cel puțin J, atunci

$L_s - i + 1$

$i + j - 1 + q$

$\geq J$

ori de câte ori $p < q$.

Exemplul 3.30: Să luăm din nou în considerare șirul s = acdefghijk, dar să facem exemplul arată câteva detalii, să alegem J = 0,8 în loc de 0,9. Noi stim

că $L_s = 10$. Deoarece $\lfloor (1 - J) L_s \rfloor + 1 = 3$, trebuie să luăm în considerare pozițiile prefixului i = 1, 2 și 3 în cele ce urmează. Ca și înainte, fie p să fie lungimea sufixului lui s și q lungimea sufixului lui t.

În primul rând, considerați cazul p ≥ q. Constrângerea suplimentară pe care o avem asupra q și j este (q + 1) / (9 + j) ≥ 0,8. Putem enumera perechile de valori ale lui j și q pentru fiecare i între 1 și 3, după cum urmează.

i = 1: Aici, p = 9, deci q ≤ 9. Să luăm în considerare valorile posibile ale q:

q = 9: Trebuie să avem 10 / (9 + j) ≥ 0,8. Astfel, putem avea j = 1, j = 2, sau j = 3. Rețineți că pentru j = 4, 10/13> 0,8.

q = 8: Trebuie să avem 9 / (9 + j) ≥ 0,8. Astfel, putem avea j = 1 sau j = 2. Pentru j = 3, 9/12> 0,8.

q = 7: Trebuie să avem 8 / (9 + j) ≥ 0,8. Doar j = 1 satisface această inegalitate.

q = 6: Nu există valori posibile pentru j, deoarece 7 / (9 + j)> 0,8 pentru fiecare număr întreg pozitiv j. Același lucru este valabil pentru fiecare valoare mai mică a q.

i = 2: Aici, p = 8, deci avem nevoie de q ≤ 8. Deoarece constrângerea (q + 1) / (9 + j) ≥ 0,8 nu depinde de i, [8] putem folosi analiza din cazul de mai sus, dar excludeți cazul q = 9. Astfel, singurele valori posibile ale j și q când i = 2 sunt

1. q = 8; j = 1.
2. q = 8; j = 2.
3. q = 7; j = 1.

i = 3: Acum, p = 7 și constrângerile sunt q ≤ 7 și (q + 1) / (9 + j) ≥ 0,8. The singura opțiune este q = 7 și j = 1.

În continuare, trebuie să luăm în considerare cazul p <q. Constrângerea suplimentară este

11 - i

i + j + q - 1

≥ 0,8

Din nou, luați în considerare fiecare valoare posibilă a lui i.

[8] Rețineți că i influențează valoarea lui p și, prin p, pune o limită asupra lui q.

---

i = 1: Atunci p = 9, deci avem nevoie de q ≥ 10 și 10 / (q + j) ≥ 0,8. Posibilul valorile lui q și j sunt

1. q = 10; j = 1.
2. q = 10; j = 2.
3. q = 11; j = 1.

i = 2: Acum, p = 8, deci avem nevoie de q ≥ 9 și 9 / (q + j + 1) ≥ 0,8. Din moment ce j trebuie fie un număr întreg pozitiv, singura soluție este q = 9 și j = 1, o posibilitate despre care știam deja.

i = 3: Aici, p = 7, deci avem nevoie de q ≥ 8 și 8 / (q + j + 2) ≥ 0,8. Nu sunt soluții.

qj = 1 j = 2 j = 3
7
X
8
X
X
i = 1
9
X
X
X
10
X
X
11
X
7
X
i = 2
8
X
X
9
X
i = 3

7
X
Figura 3.16: Găleatele care trebuie examinate pentru a găsi posibile potriviri pentru
șirul s = acdefghijk cu J = 0,8 sunt marcate cu un x
Când acumulăm combinațiile posibile de i, j și q, vedem asta
setul de găleți index în care trebuie să ne uităm formează o piramidă. Figura 3.16
arată gălețile în care trebuie să căutăm. Adică trebuie să ne uităm în acestea
găleți (x, j, q) astfel încât simbolul ith al șirului s este x, j este poziția
asociat cu cupa și q lungimea sufixului. ✷

3.9.7 Exerciții pentru secțiunea 3.9

Exercițiul 3.9.1: Să presupunem că setul nostru universal este literele minuscule și
ordinea elementelor este considerată vocalele, în ordine alfabetică, urmată de
consoane în ordine alfabetică inversă. Reprezentați următoarele seturi ca șiruri.
a {q, w, e, r, t, y}.
(b) {a, s, d, f, g, h, j, u, i}.

Exercițiul 3.9.2: Să presupunem că filtrăm perechile de candidați numai pe baza lungimii, ca în
Secțiunea 3.9.3. Dacă s este un șir de lungime 20, cu ce șiruri este s comparat când
J, limita inferioară a similarității Jaccard are următoarele valori: (a) J = 0,85
(b) J = 0,95 (c) J = 0,98?
Exercițiul 3.9.3: Să presupunem că avem un șir de lungime 15 și dorim să indexăm
prefixul acestuia ca în secțiunea 3.9.4.
(a) Câte poziții sunt în prefix dacă J = 0,85?
(b) Câte poziții sunt în prefix dacă J = 0,95?
! (c) Pentru ce interval de valori al lui J va fi indexat sub primele sale patru simboluri,
dar nu mai mult?
Exercițiul 3.9.4: Să presupunem că s este un șir de lungime 12. Cu ce simbol-poziție
perechile vor fi comparate cu dacă folosim abordarea de indexare din secțiunea 3.9.5,
și (a) J = 0,75 (b) J = 0,95?
! Exercițiul 3.9.5: Să presupunem că folosim informații de poziție în indexul nostru, ca în secțiunea
secțiunea 3.9.5. Șirurile s și t sunt ambele alese la întâmplare dintr-un set universal de
100 de elemente. Să presupunem că J = 0,9. Care este probabilitatea ca s și t să fie
comparat dacă
(a) s și t au ambele lungime 9.
(b) s și t au ambele lungime 10.
Exercițiul 3.9.6: Să presupunem că folosim indexuri bazate atât pe poziție, cât și pe sufix
lungime, ca în secțiunea 3.9.6. Dacă s este un șir de lungime 20, cu ce simbol-
triplele poziției-lungime vor fi comparate cu, dacă (a) J = 0,8 (b) J = 0,9?

3.10 Rezumatul capitolului 3

♦ Similitudinea Jaccard: Similitudinea Jaccard a seturilor este raportul dintre dimensiuni
a intersecției mulțimilor la dimensiunea uniunii. Această măsură de
similaritatea este potrivită pentru multe aplicații, inclusiv similaritatea textuală a
documente și similitudinea obiceiurilor de cumpărare ale clienților.

♦ Shingling: Un k-shingle este orice k caractere care apar consecutiv în
un document. Dacă reprezentăm un document prin setul său de k-zoster, atunci
similitudinea Jaccard a seturilor de șindrilă măsoară asemănarea textuală
de documente. Uneori, este util să hash zoster la șiruri de biți de
lungime mai mică și utilizați seturi de valori hash pentru a reprezenta documente.

♦ Minhashing: o funcție minhash pe seturi se bazează pe o permutare a
set universal. Având în vedere o astfel de permutare, valoarea minhash pentru un set este
acel element al setului care apare mai întâi în ordinea permutată.

♦ Semnături Minhash: Putem reprezenta seturi alegând o listă de per-
mutații și calculul pentru fiecare set semnătura sa minhash, care este
secvența valorilor minhash obținute prin aplicarea fiecărei permutări pe
lista cu acel set. Având în vedere două seturi, fracția așteptată a permutațiilor
care va produce aceeași valoare minhash este exact similaritatea Jaccard
a decorurilor.

♦ Minhashing eficient: Deoarece nu este cu adevărat posibil să se genereze aleatorii permutări, este normal să simulezi o permutare prin alegerea unei întâmplări funcția hash și luarea valorii minhash pentru un set să fie cea mai mică hash valoarea oricăruia dintre membrii setului. O eficiență suplimentară poate fi obținută prin restricționarea căutării celei mai mici valori minhash la doar o mică subset al setului universal.

♦ Hashing sensibil la localitate pentru semnături: Această tehnică ne permite evita calculul asemănării fiecărei perechi de seturi sau a minhashului lor naturi. Dacă ni se dau semnături pentru seturi, le putem împărți în și măsoară asemănarea unei perechi de seturi numai dacă sunt identificate cal în cel puțin o bandă. Alegând în mod corespunzător dimensiunea benzilor, noi poate elimina din considerare majoritatea perechilor care nu se întâlnesc cu ale noastre prag de similitudine.

♦ Măsuri de distanță: o măsură de distanță este o funcție pe perechi de puncte în un spațiu care satisface anumite axiome. Distanța dintre două puncte este 0 dacă punctele sunt aceleași, dar mai mari decât 0 dacă punctele sunt diferite. The distanța este simetrică; nu contează în ce ordine luăm în considerare două puncte. O măsură de distanță trebuie să satisfacă inegalitatea triunghiului: distanța dintre două puncte nu este niciodată mai mare decât suma distanțelor între aceste puncte și un al treilea punct.

♦ Distanța euclidiană: cea mai comună noțiune de distanță este cea euclidiană distanță într-un spațiu n-dimensional. Această distanță, uneori numită L_2 -norm, este rădăcina pătrată a sumei pătratelor diferențelor între punctele din fiecare dimensiune. O altă distanță potrivită pentru spațiile clidiene, numite distanța Manhattan sau norma L_{1-} este suma mărimile diferențelor dintre punctele din fiecare dimensiune.

♦ Distanța Jaccard: Un minus similaritatea Jaccard este o măsură a distanței, numită distanța Jaccard.

♦ Distanța cosinusului: Unghiul dintre vectori într-un spațiu vectorial este cosinusul măsura distanței. Putem calcula cosinusul acelui unghi luând produs punct al vectorilor și împărțind la lungimile vectorilor.

♦ Editare distanță: această măsură de distanță se aplică unui spațiu de șiruri și este numărul de inserții și / sau ștergeri necesare pentru a converti un șir în celălalt. Distanța de editare poate fi calculată și ca suma a

---

lungimile corzilor minus de două ori lungimea celei mai lungi comune subsecvența șirurilor.

♦ Distanță de ciocănire: Această măsură a distanței se aplică unui spațiu de vectori. Distanța Hamming între doi vectori este numărul de poziții din de care diferă vectorii.

♦ Hashing generalizat sensibil la localitate: putem începe cu orice colecție de funcții, cum ar fi funcțiile minhash, care pot lua o decizie dacă o pereche de articole ar trebui sau nu să fie candidați la similitudine control. Singura constrângere asupra acestor funcții este că oferă un limita inferioară a probabilității de a spune „da” dacă distanța (conform până la o anumită măsură a distanței) este sub o limită dată și o limită superioară pe probabilitatea de a spune „da” dacă distanța este peste o altă limită dată. Apoi putem crește probabilitatea de a spune „da” pentru articolele din apropiere și în același timp, reduceți probabilitatea de a spune „da” pentru elementele îndepărtate în măsura în care ne dorim, prin aplicarea unei construcții ȘI și o construcție SAU.

♦ Hiperplane aleatorii și LSH pentru distanță cosinus: Putem obține un set de funcții de bază pentru a începe un LSH generalizat pentru măsurarea distanței cosinusului prin identificarea fiecărei funcții cu o listă de vectori aleși aleatoriu. Noi aplicați o funcție unui vector dat v luând produsul punct al lui v cu fiecare vector de pe listă. Rezultatul este o schiță formată din semne (+1 sau −1) din produsele dot. Fracția de poziții în care schițele de doi vectori sunt de acord, înmulțiți cu 180, este o estimare a unghiului dintre cei doi vectori.

♦ LSH pentru distanța euclidiană: Un set de funcții de bază pentru care începe LSH Distanța euclidiană poate fi obținută alegând linii aleatorii și proiectând

introducând puncte pe acele linii. Fiecare linie este împărțită în intervale de lungime fixă, iar funcția răspunde „da” la o pereche de puncte care se încadrează în același interval.

♦ Detecție de înaltă similitudine prin compararea șirurilor: o abordare alternativă la găsirea articolelor similare, când pragul similarității Jaccard este aproape de 1, evită utilizarea minhashing și LSH. Mai degrabă, setul universal este ordonat, iar mulțimile sunt reprezentate prin șiruri, constând din elementele lor în ordine. Cel mai simplu mod de a evita compararea tuturor perechilor de seturi sau a șirurilor lor este de a reținești că seturile foarte asemănătoare vor avea șiruri de aproximativ aceleași lungime. Dacă sortăm șirurile, putem compara fiecare șir cu doar a un număr mic de șiruri imediat următoare.

♦ Indexuri de caractere: Dacă reprezentăm mulțimi prin șiruri și similaritate pragul este aproape de 1, putem indexa toate șirurile după primele lor caractere. Prefixul ale cărui caractere trebuie indexate are aproximativ lungimea din șir de ori distanța maximă Jaccard (1 minus minim Similitudine Jaccard).

♦ Indici de poziție: putem indexa șiruri nu numai pe caracterele din prefixele lor, dar pe poziția acelui caracter în cadrul prefixului. Noi reduceți numărul de perechi de șiruri care trebuie comparate, deoarece dacă două șiruri au un caracter care nu se află în prima poziție în ambele șiruri, atunci știm că fie există unele caractere precedente care sunt în uniune, dar nu în intersecție, sau există un simbol anterior care apare în ambele șiruri.

♦ Indici de sufix: Putem, de asemenea, să indexăm șiruri bazate nu numai pe caractere în prefixele lor și pozițiile acestor caractere, dar pe lungime a sufixului personajului - numărul de poziții care îl urmează în șir. Această structură reduce și mai mult numărul de perechi care trebuie să fie comparativ, deoarece un simbol comun cu lungimi de sufix diferite implică caractere suplimentare care trebuie să fie în uniune, dar nu în intersecție.

3.11 Referințe pentru capitolul 3

Tehnica pe care am numit-o zoster este atribuită [11]. Utilizarea în manieră am discutat aici este din [2].
Minhashing provine din [3]. Îmbunătățirea care evită privirea elementele provin din [10].
Lucrările originale despre hashingul sensibil la localitate au fost [9] și [7]. [1] este un rezumat util al ideilor din acest domeniu.
[4] introduce ideea utilizării hiperplanelor aleatorii pentru a rezuma itemi în un mod care respectă distanța cosinusului. [8] sugerează că hiperplanele aleatorii plus LSH poate fi mai precis la detectarea documentelor similare decât minhashing plus LSH.
Tehnicile pentru rezumarea punctelor într-un spațiu euclidian sunt prezentate în [6].
[12] a prezentat tehnica de șindrilă bazată pe cuvinte stop.
Schemele de indexare bazate pe lungime și prefix pentru o potrivire asemănătoare provine din [5]. Tehnica care implică lungimea sufixului este de la [13].

1. A. Andoni și P. Indyk, „Algoritmi de hashing aproape optimi pentru aproximativ cel mai apropiat vecin în dimensiuni ridicate, ”Com. ACM 51: 1, pp. 117–122, 2008.
2. AZ Broder, „Despre asemănarea și limitarea documentelor”, Proc. Compresia și complexitatea secvențelor, pp. 21-29, Positano Italia, 1997.
3. AZ Broder, M. Charikar, AM Frieze și M. Mitzenmacher, „Min-wise permutări independente ", Simpozionul ACM pe teoria calculelor, pp. 327-336, 1998.
4. MS Charikar, „Tehnici de estimare a similitudinii din rithms, ”ACM Symposium on Theory of Computing, pp. 380–388, 2002.

---

5. S. Chaudhuri, V. Ganti și R. Kaushik, „Un operator primitiv pentru sim-

ilarity se alătură curățării datelor ”, Proc. Intl. Conf. ingineria datelor, 2006.
6. M. Datar, N. Immorlica, P. Indyk și VS Mirrokni, „Locality-sensitive schemă de hashing bazată pe distribuții p-stabile ”, Simpozionul pe Computational Geometry pp. 253–262, 2004.
7. A. Gionis, P. Indyk și R. Motwani, „Căutarea similarității în dimensiuni mari sioni prin hash, ”Proc. Intl. Conf. pe baze de date foarte mari, pp. 518–529, 1999.
8. M. Henzinger, „Găsirea paginilor web aproape duplicate: o evaluare la scară largă de algoritmi, ”Proc. 29 Conf. SIGIR, pp. 284–291, 2006.
9. P. Indyk și R. Motwani. „Aproximativ cel mai apropiat vecin: spre redeplasarea blestemului dimensionalității ", Simpozionul ACM pe teoria compuse, pp. 604–613, 1998.
10. P. Li, AB Owen și CH Zhang. „Un hash de permutare”, Conf. on Neural Information Processing Systems 2012, pp. 3122–3130.
11. U. Manber, „Găsirea fișierelor similare într-un sistem de fișiere mare”, Proc. USENIX Conferință, pp. 1-10, 1994.
12. M. Theobald, J. Siddharth și A. Paepcke, „SpotSigs: robust și eficient detecție aproape dublată în colecții web mari ”, 31-lea ACM anual Conferința SIGIR, iulie 2008, Singapore.
13. C. Xiao, W. Wang, X. Lin și JX Yu, „Similitudinea eficientă se unește pentru aproape de detectarea duplicatului, ”Proc. Conferința WWW, pp. 131-140, 2008.

capitolul 4
Fluxuri de date miniere

Majoritatea algoritmilor descriși în această carte presupun că exploatăm un Bază de date. Adică, toate datele noastre sunt disponibile când și dacă le dorim. In acest capitol, vom face o altă presupunere: datele sosesc într-un flux sau fluxuri, iar dacă nu este procesată imediat sau stocată, atunci se pierde pentru totdeauna. În plus, vom presupune că datele ajung atât de repede încât nu sunt fezabile de stocat totul în stocare activă (adică într-o bază de date convențională) și apoi interacționează cu la momentul ales de noi.
Algoritmii pentru procesarea fluxurilor implică fiecare rezumarea fișierului flux într-un fel. Vom începe prin a lua în considerare cum să realizăm un eșantion util a unui flux și modul de filtrare a unui flux pentru a elimina majoritatea „nedorite” elemente. Arătăm apoi cum să estimăm numărul de elemente diferite din un flux care utilizează mult mai puțin spațiu de stocare decât ar fi necesar dacă am enumera toate elemente pe care le-am văzut.
O altă abordare pentru a rezuma un flux este de a privi doar o lungime fixă

,,Fereastră” constând din ultimele n elemente pentru unele (de obicei mari) n. Noi apoi interogați fereastra ca și cum ar fi o relație într-o bază de date. Dacă există multe fluxuri și / sau n este mare, este posibil să nu putem stoca întreaga fereastră pentru fiecare flux, deci trebuie să rezumăm chiar și ferestrele. Ne adresăm problemă fundamentală a menținerii unui număr aproximativ pe numărul de 1 în fereastra unui flux de biți, în timp ce utilizați mult mai puțin spațiu decât ar fi necesar pentru a stoca întreaga fereastră în sine. Această tehnică se generalizează la aproximare diferite tipuri de sume.

4.1 Modelul de date al fluxului

Să începem discutând elementele fluxurilor și procesarea fluxurilor. Noi explicați diferența dintre fluxuri și baze de date și problemele speciale care apar atunci când avem de-a face cu fluxuri. Unele aplicații tipice în care se va examina modelul fluxului aplicat.

Întrebări
Permanent
Curent
Procesor
. . .

0, 1, 1, 0, 1, 0, 0, 0
q, w, e, r, t, y, u, i, o
1, 5, 2, 7, 4, 0, 3, 5
timp
Fluxuri de ieșire
Depozitare
Depozitare
De arhivă
Fluxuri care intră
Ad-hoc
Întrebări
Limitat
Lucru
Figura 4.1: Un sistem de gestionare a fluxului de date

4.1.1 Un sistem de gestionare a fluxului de date

În analogie cu un sistem de gestionare a bazelor de date, putem vizualiza un procesor de flux ca un fel de sistem de gestionare a datelor, a cărui organizare la nivel înalt este sugerat în Fig. 4.1. Orice număr de fluxuri pot intra în sistem. Fiecare fluxul poate furniza elemente la propriul program; nu trebuie să aibă la fel ratele de date sau tipurile de date, iar timpul dintre elementele unui flux nu este necesar fii uniform. Faptul că rata de sosire a elementelor fluxului nu este sub controlul sistemului distinge procesarea fluxului de procesare de date care se desfășoară în cadrul unui sistem de gestionare a bazelor de date. Ultimul sistem controlează viteza cu care datele sunt citite de pe disc și, prin urmare, nu a fost niciodată să vă faceți griji cu privire la pierderea datelor în timp ce încearcă să execute interogări. Fluxurile pot fi arhivate într-un mare magazin de arhive, dar presupunem că nu este este posibil să răspundeți la întrebări din magazinul de arhive. Poate fi examinat numai în circumstanțe speciale folosind procese de recuperare care consumă mult timp. Există de asemenea, un magazin de lucru, în care pot fi plasate rezumate sau părți ale fluxurilor, și care poate fi folosit pentru a răspunde la întrebări. Magazinul de lucru ar putea fi disc, sau ar putea fi memoria principală, în funcție de cât de repede trebuie să procesăm interogările. În orice caz, are o capacitate suficient de limitată încât nu poate stoca toate date din toate fluxurile.

4.1.2 Exemple de surse de flux

Înainte de a continua, să luăm în considerare câteva dintre modalitățile în care apar datele fluxului natural.

Date senzor

Imaginați-vă un senzor de temperatură care se mișcă în ocean, trimitând înapoi la o stația de bază o citire a temperaturii suprafeței în fiecare oră. Datele produse de acest senzor este un flux de numere reale. Nu este un flux foarte interesant, deoarece rata de date este atât de mică. Nu ar pune accent pe tehnologia modernă și pe întregul flux ar putea fi păstrat în memoria principală, în esență pentru totdeauna. Acum, dați senzorului o unitate GPS și lăsați-l să raporteze înălțimea suprafeței în loc de temperatura. Înălțimea suprafeței variază destul de rapid în comparație cu temperatura Tura, deci s-ar putea ca senzorul să trimită înapoi o citire la fiecare zecime de secundă. Dacă trimite de fiecare dată un număr real de 4 octeți, atunci produce 3,5 megabytes pe zi. Va dura încă ceva timp pentru a umple memoria principală, darămite un singur disc. Dar un senzor ar putea să nu fie atât de interesant. Să învăț ceva despre comportamentul oceanului, s-ar putea să dorim să implementăm un milion de senzori, fiecare trimitând înapoi un flux, cu o rată de zece pe secundă. Un milion de senzori nu sunt foarte mulți; Acolo ar fi unul pentru fiecare 150 de mile pătrate de ocean. Acum avem 3,5 terabytes sosind în fiecare zi și cu siguranță trebuie să ne gândim la ce poate fi păstrat stocarea funcțională și ceea ce poate fi arhivat numai.

Date de imagine

Sateliții trimit adesea pe pământ curenți constând din mai mulți terabyți de imagini pe zi. Camerele de supraveghere produc imagini cu rezoluție mai mică decât sateliții, dar pot fi mulți dintre ei, fiecare producând un flux de imagini la intervale de o secundă. Se spune că Londra are șase milioane de astfel de camere, fiecare producând un flux.

Internet și trafic web

Un nod de comutare din mijlocul internetului primește fluxuri de pachete IP de la multe intrări și le direcționează către ieșirile sale. În mod normal, meseria de comutatorul este de a transmite date și nu de a le păstra sau de a le interoga. Dar există un tendința de a pune mai multă capacitate în comutator, de exemplu, capacitatea de a detecta atacuri de refuz de serviciu sau capacitatea de a redirecționa pachetele pe baza informațiilor

despre congestia din rețea.
Site-urile web primesc fluxuri de diferite tipuri. De exemplu, Google primește șapte aproximativ o sută de milioane de interogări de căutare pe zi. Yahoo! acceptă miliarde de „clicuri” pe zi pe diferitele sale site-uri. Multe lucruri interesante pot fi învățate din acestea pâraie. De exemplu, o creștere a întrebărilor precum „durere în gât” ne permite urmăriți răspândirea virușilor. O creștere bruscă a ratei de clic pentru un link ar putea

---

Pagina 166

indicați câteva știri conectate la pagina respectivă sau ar putea însemna că linkul este rupt și trebuie reparat.

4.1.3 Interogări de flux

Există două moduri în care interogările sunt întrebat despre fluxuri. Arătăm în Fig. 4.1 a loc în procesor unde sunt stocate interogările permanente. Aceste întrebări sunt, într-un anumit sens, execută permanent și produce rezultate la momente adecvate.
Exemplul 4.1: Fluxul produs de temperatura de suprafață a oceanului menționat la începutul secțiunii 4.1.2 ar putea avea o interogare permanentă pentru a emite o alertă ori de câte ori temperatura depășește 25 de grade Celsius. La această interogare se răspunde cu ușurință, deoarece depinde doar de cel mai recent flux element.
Alternativ, am putea avea o interogare permanentă care, de fiecare dată o nouă lectură ajunge, produce media celor mai recente 24 de lecturi. Această interogare, de asemenea poate fi răspuns cu ușurință, dacă stocăm cele mai recente 24 de elemente de flux. Când un sosește un nou element de flux, putem scoate din magazinul de lucru cel mai mult 25 element recent, deoarece nu va mai fi niciodată necesar (dacă nu există altul interogare permanentă care o necesită).
O altă întrebare pe care am putea-o întreba este temperatura maximă înregistrată vreodată de senzorul respectiv. Putem răspunde la această întrebare păstrând un rezumat simplu: maximul tuturor elementelor de flux văzute vreodată. Nu este necesar să înregistrați fișierul întregul flux. Când ajunge un nou element de flux, îl comparăm cu maxim stocat și setați maximul la cel mai mare. Putem atunci răspundeți la interogare producând valoarea curentă a maximului. În mod similar, dacă vrem temperatura medie în toate timpurile, nu trebuie decât să înregistrăm două valori: numărul de citiri trimise vreodată în flux și suma acestora lecturi. Putem ajusta aceste valori cu ușurință de fiecare dată când ajunge o nouă lectură, și putem produce coeficientul lor ca răspuns la interogare. ✷
Cealaltă formă de interogare este ad-hoc, o întrebare adresată o dată despre curent starea unui flux sau fluxuri. Dacă nu stocăm toate fluxurile în întregime, ca în mod normal nu putem, atunci nu ne putem aștepta să răspundem la întrebări arbitrare despre pâraie. Dacă avem o idee despre ce fel de întrebări vor fi întrebat prin intermediul interfață de interogare ad-hoc, apoi ne putem pregăti pentru ele stocând corespunzător părți sau rezumate ale fluxurilor ca în exemplul 4.1.
Dacă dorim ca facilitatea să solicite o gamă largă de întrebări ad-hoc, un lucru comun abordarea este de a stoca o fereastră glisantă a fiecărui flux în magazinul de lucru. A fereastra glisantă poate fi cel mai recent n elemente ale unui flux, pentru unele n sau poate fi toate elementele care au ajuns în ultimele t unități de timp, de exemplu, unul zi. Dacă privim fiecare element de flux ca un tuplu, putem trata fereastra ca pe un relaționează-l și interogează-l cu orice interogare SQL. Desigur, gestionarea fluxului sistemul trebuie să mențină fereastra proaspătă, ștergând cele mai vechi elemente ca altele noi Intrați.

Pagina 167

Exemplul 4.2: site-urile web doresc adesea să raporteze numărul de utilizatori unici ultima lună. Dacă ne gândim la fiecare conectare ca la un element de flux, putem menține o fereastră care este conectată în ultima lună. Trebuie să asociem ora sosirii cu fiecare autentificare, deci știm când nu mai aparține fișierului fereastră. Dacă ne gândim la fereastră ca la o relație Conectări (nume, timp), atunci este simplu să obțineți numărul de utilizatori unici în ultima lună. SQL interogarea este:

```
SELECTAȚI NUMĂRUL (DISTINCT (nume))
FROM Conectări
UNDE timp> = t;
```

Aici, t este o constantă care reprezintă timpul cu o lună înainte de curent

timp.
Reţineţi că trebuie să putem să menţinem întregul flux de conectări pentru
ultima lună în depozit de lucru. Cu toate acestea, chiar şi pentru cele mai mari site-uri, aceste date
nu este mai mult de câţiva terabyţi şi, prin urmare, poate fi stocat pe disc. ✷

4.1.4 Probleme în procesarea fluxului

Înainte de a continua să discutăm algoritmi, să luăm în considerare constrângerile de mai jos
pe care le lucrăm atunci când avem de-a face cu fluxuri. În primul rând, fluxurile oferă adesea elemente
foarte rapid. Trebuie să procesăm elemente în timp real, sau pierdem oportunitatea
pentru a le procesa deloc, fără a accesa stocarea arhivă. Astfel, deseori este
important ca algoritmul de procesare a fluxului să fie executat în memoria principală,
fără acces la stocarea secundară sau cu accesuri rare la secundare
depozitare. Mai mult, chiar şi atunci când fluxurile sunt „lente”, ca în exemplul senzor-date
din secţiunea 4.1.2, pot exista multe astfel de fluxuri. Chiar dacă fiecare flux de unul singur
poate fi procesat folosind o cantitate mică de memorie principală, cerinţele tuturor
fluxurile împreună pot depăşi cu uşurinţă cantitatea de memorie principală disponibilă.
Astfel, multe probleme legate de transmiterea de date ar fi uşor de rezolvat dacă noi
avea suficientă memorie, dar devin destul de dure şi necesită inventarea de noi
tehnici pentru a le executa la o rată realistă pe o maşină de realist
mărimea. Iată două generalizări despre algoritmii de flux care merită să fie luaţi în considerare
în timp ce citiţi acest capitol:

- Adesea, este mult mai eficient să obţinem un răspuns aproximativ la cel al nostru problemă decât o soluţie exactă.
- La fel ca în Capitolul 3, o varietate de tehnici legate de hashing se dovedesc a fi util. În general, aceste tehnici introduc o întâmplare utilă în comportamentul algoritmului, pentru a produce un răspuns aproximativ foarte aproape de rezultatul adevărat.

4.2 Eşantionarea datelor într-un flux

Ca primul nostru exemplu de gestionare a datelor în flux, vom analiza extragerea
mostre de încredere dintr-un flux. La fel ca în cazul multor algoritmi de flux, „trucul”
implică utilizarea hashului într-un mod oarecum neobişnuit.

4.2.1 Un exemplu motivant

Problema generală pe care o vom aborda este selectarea unui subset al unui flux, astfel încât noi
poate pune întrebări despre subsetul selectat şi poate avea răspunsurile statistic
reprezentativ al fluxului în ansamblu. Dacă ştim ce întrebări trebuie să fie
întrebat, atunci există o serie de metode care ar putea funcţiona, dar căutăm
pentru o tehnică care va permite interogări ad-hoc pe eşantion. Ne vom uita la
o problemă particulară, din care va ieşi ideea generală.
Exemplul nostru de rulare este următorul. Un motor de căutare primeşte un flux de
întrebări şi ar dori să studieze comportamentul utilizatorilor tipici. [1] Presupunem că
fluxul este format din tupluri (utilizator, interogare, timp). Să presupunem că vrem să răspundem
interogări precum „Ce fracţie din interogările utilizatorului tipic au fost repetate
ultima lună? ” Să presupunem, de asemenea, că dorim să stocăm doar 1/10 din flux
elemente.
Abordarea evidentă ar fi generarea unui număr aleator, să zicem un număr întreg
de la 0 la 9, ca răspuns la fiecare interogare de căutare. Păstraţi tuplul dacă şi numai dacă
numărul aleator este 0. Dacă facem acest lucru, fiecare utilizator are, în medie, 1/10 din numărul său
interogări stocate. Fluctuaţiile statistice vor introduce ceva zgomot în date,
dar dacă utilizatorii fac multe întrebări, legea numărului mare ne va asigura că
majoritatea utilizatorilor vor avea o fracţiune destul de apropiată de 1/10 din interogările lor stocate.
Cu toate acestea, această schemă ne oferă un răspuns greşit la întrebarea pe care o solicităm
numărul mediu de interogări duplicate pentru un utilizator. Să presupunem că un utilizator a emis
interogări de căutare o dată în ultima lună, d interogări de căutare de două ori şi nr
interogări de căutare de mai mult de două ori. Dacă avem un eşantion 1/10, de întrebări, vom face
vedeţi în eşantion pentru utilizatorul respectiv un s / 10 aşteptat al interogărilor de căutare emise
o singura data. Dintre interogările de căutare emise de două ori, doar d / 100 va apărea de două ori în
probă; acea fracţie este d ori probabilitatea ca ambele apariţii ale
interogarea va fi în eşantionul 1/10. Dintre interogările care apar de două ori în întregime
flux, 18d / 100 va apărea exact o dată. Pentru a vedea de ce, reţineţi că 18/100 este
probabilitatea ca una dintre cele două apariţii să fie în 1/10 din flux
care este selectat, în timp ce celălalt este în 9/10 care nu este selectat.
Răspunsul corect la întrebarea despre fracţiunea de căutări repetate este
d / (s + d). Cu toate acestea, răspunsul pe care îl vom obţine din eşantion este d / (10s + 19d).

Pentru a obține ultima formulă, rețineți că d / 100 apare de două ori, în timp ce s / 10 + 18d / 100 apare o dată. Astfel, fracția care apare de două ori în eșantion este împărțită d / 100

1 În timp ce ne vom referi la „utilizatori”, motorul de căutare primește într-adevăr adrese IP de la care interogarea de căutare a fost emisă. Vom presupune că aceste adrese IP identifică utilizatori unici, care este aproximativ adevărat, dar nu tocmai adevărat.

---

cu d / 100 + s / 10 + 18d / 100. Acest raport este d / (10s + 19d). Fără valori pozitive lui s și d este d / (s + d) = d / (10s + 19d).

4.2.2 Obținerea unui eșantion reprezentativ

Interogarea din secțiunea 4.2.1, la fel ca multe interogări despre statisticile tipice utilizatorilor, nu li se poate răspunde luând un eșantion din interogările de căutare ale fiecărui utilizator. Astfel, trebuie să ne străduim să alegem 1/10 din utilizatori și să le luăm toate căutările eșantion, în timp ce nu efectuează niciuna dintre căutările de la alți utilizatori. Dacă putem stoca o listă cu toți utilizatorii și dacă sunt sau nu în eșantion, atunci am putea urmează următoarele instrucțiuni. De fiecare dată când o interogare de căutare ajunge în flux, ne uităm în sus utilizatorul pentru a vedea dacă sunt sau nu în eșantion. Dacă da, adăugăm această căutare interogați eșantionului și, dacă nu, atunci nu. Cu toate acestea, dacă nu avem nicio evidență Văzând vreodată acest utilizator înainte, atunci generăm un număr întreg aleatoriu între 0 și 9. Dacă numărul este 0, adăugăm acest utilizator la lista noastră cu valoarea „în” și dacă numărul este altul decât 0, adăugăm utilizatorul cu valoarea „out”.

Această metodă funcționează atât timp cât ne permitem să păstrăm lista tuturor utilizatorilor și decizia lor de intrare / ieșire în memoria principală, deoarece nu există timp pentru a merge pe disc pentru fiecare căutare care ajunge. Prin utilizarea unei funcții hash, se poate evita păstrarea fișierului lista utilizatorilor. Adică, hașăm fiecare nume de utilizator la una din cele zece găleți, de la 0 la 9. Dacă utilizatorul hash la bucket 0, atunci acceptați această interogare de căutare pentru eșantion, iar dacă nu, atunci nu.

Rețineți că nu stocăm de fapt utilizatorul în cupă; de fapt, nu există datele din găleți. Efectiv, folosim funcția hash ca o generator de numere, cu proprietatea importantă care, atunci când este aplicată aceluiași utilizator de mai multe ori, obținem întotdeauna același număr „aleatoriu”. Adică fără stocând decizia de intrare / ieșire pentru orice utilizator, putem reconstrui oricare decizie când ajunge o interogare de căutare a utilizatorului respectiv.

Mai general, putem obține un eșantion format din orice fracție rațională a / b a utilizatorilor prin hashing nume de utilizatori la b compartimente, de la 0 la b - 1. Adăugați interogare de căutare către eșantion dacă valoarea hash este mai mică decât a.

4.2.3 Problema generală de eșantionare

Exemplul de rulare este tipic pentru următoarea problemă generală. Fluxul nostru constă din tupluri cu n componente. Un subset de componente sunt cheia componente, pe care se va baza selecția eșantionului. În alergarea noastră de exemplu, există trei componente - utilizator, interogare și timp - dintre care numai utilizatorul este în cheie. Cu toate acestea, am putea lua și un eșantion de interogări făcând interogarea este cheia sau chiar luați un eșantion de perechi utilizator-interogare făcând ambele aceste componente formează cheia.

Pentru a lua un eșantion de dimensiunea a / b, vom hash valoarea cheie pentru fiecare tupl la b și acceptați tuplul pentru eșantion dacă valoarea hash este mai mică decât a.

Dacă cheia constă din mai multe componente, funcția hash trebuie să o facă combinați valorile pentru acele componente pentru a crea o singură valoare hash. The

---

rezultatul va fi un eșantion format din toate tuplurile cu anumite valori cheie. The valorile cheie selectate vor fi aproximativ a / b din toate valorile cheie care apar în curentul.

4.2.4 Variația dimensiunii probei

Adesea, eșantionul va crește pe măsură ce intră mai mult flux în sistem. În a noastră exemplul care rulează, păstrăm toate interogările de căutare din 1/10 din utilizatorii, pentru totdeauna. Odată cu trecerea timpului, vor fi efectuate mai multe căutări pentru aceiași utilizatori acumulate și utilizatorii noi selectați pentru eșantion vor apărea în curent.

Dacă avem un buget pentru câte tupluri din flux pot fi stocate ca eșantionul, atunci fracția valorilor cheie trebuie să varieze, scăzând pe măsură ce trece timpul pe. Pentru a se asigura că, în orice moment, eșantionul constă din toate tuplele de la a

subsetul valorilor cheie, alegem o funcție hash h de la valorile cheie la o foarte
număr mare de valori 0, 1, ..., B − 1. Menținem un prag t, care inițial
poate fi cel mai mare număr de cupă, B - 1. În orice moment, eșantionul constă din
acele tupluri a căror cheie K satisface h (K) ≤ t. Sunt noi tupluri din flux
adăugate la eșantion dacă și numai dacă îndeplinesc aceeași condiție.
Dacă numărul de tupluri stocate ale eșantionului depășește spațiul alocat, noi
coborâți t la t − 1 și scoateți din eșantion toate acele tuple a căror cheie K hashes
la t. Pentru eficiență, putem reduce t cu mai mult de 1 și putem elimina tuplele cu
mai multe dintre cele mai mari valori hash, ori de câte ori trebuie să aruncăm câteva valori cheie
din eșantion. Eficiență suplimentară se obține prin menținerea unui index pe
valoarea hash, astfel încât să putem găsi toate acele tupluri ale căror chei hash la un anumit
valoare rapid.

4.2.5 Exerciții pentru secțiunea 4.2

Exercițiul 4.2.1: Să presupunem că avem un flux de tupluri cu schema
Note (universitate, ID curs, ID student, nota)
Să presupunem că universitățile sunt unice, dar un ID de curs este unic numai în cadrul unei
versitate (de exemplu, universități diferite pot avea cursuri diferite cu același ID,
de exemplu, „CS101”) și, de asemenea, ID-urile student sunt unice numai în cadrul unei universități
(diferite universități pot atribui același ID unor studenți diferiți). Presupune
vrem să răspundem la anumite întrebări aproximativ dintr-un eșantion 1/20 din
date. Pentru fiecare dintre interogările de mai jos, indicați cum ați construi
probă. Adică, spuneți care ar trebui să fie atributele cheie.
(a) Pentru fiecare universitate, estimați numărul mediu de studenți la un curs.
(b) Estimați fracțiunea de studenți care au un GPA de 3,5 sau mai mult.
(c) Estimați fracțiunea de cursuri în care cel puțin jumătate dintre studenți au primit „A.”

4.3 Filtrarea fluxurilor

Un alt proces obișnuit pe fluxuri este selectarea sau filtrarea. Vrem să
acceptați acele tupluri din flux care îndeplinesc un criteriu. Tuplurile acceptate sunt
trecut la un alt proces sub formă de flux, în timp ce alte tupluri sunt abandonate. Dacă
criteriul de selecție este o proprietate a tuplului care poate fi calculată (de exemplu,
prima componentă este mai mică de 10), apoi selectarea este ușor de realizat. Problema
devine mai greu atunci când criteriul implică căutarea apartenenței la un set. Aceasta
este deosebit de greu, atunci când setul este prea mare pentru a fi stocat în memoria principală. In acest
secțiunea, vom discuta despre tehnica cunoscută sub denumirea de „filtrare a florilor” ca modalitate de a
eliminați majoritatea tuplurilor care nu îndeplinesc criteriul.

4.3.1 Un exemplu motivant

Să începem din nou cu un exemplu de execuție care ilustrează problema și
ce putem face în legătură cu asta. Să presupunem că avem un set S de un miliard de e-mailuri permise
adresele - cele pe care le vom permite, deoarece credem că nu
fii spam. Fluxul este format din perechi: o adresă de e-mail și e-mailul în sine.
Deoarece adresa tipică de e-mail este de 20 de octeți sau mai mult, nu este rezonabil să se stocheze
S în memoria principală. Astfel, putem folosi accesul la disc pentru a determina dacă
sau să nu lăsăm să trecem prin orice element de flux dat, sau putem concepe o metodă care
nu necesită mai multă memorie principală decât avem la dispoziție și totuși va filtra cel mai mult
a elementelor de flux nedorite.
Să presupunem, de dragul argumentului, că avem un gigabyte de main disponibil
memorie. În tehnica cunoscută sub numele de filtrare Bloom, folosim acea memorie principală
ca o matrice de biți. În acest caz, avem loc pentru opt miliarde de biți, de la un octet
este egal cu opt biți. Creați o funcție hash h de la adrese de e-mail la opt miliarde
găleți. Hash fiecare membru al lui S la un bit și setați acel bit la 1. Toți ceilalți biți
din matrice rămân 0.
Deoarece există un miliard de membri ai S, aproximativ 1/8 din biți
va fi 1. Fracția exactă de biți setată la 1 va fi puțin mai mică de 1/8,
deoarece este posibil ca doi membri ai lui H să aibă același bit. Vom
discutați fracția exactă a lui 1 în secțiunea 4.3.3. Când ajunge un element de flux,
i-am hash adresa de e-mail. Dacă bitul la care se afișează această adresă de e-mail este 1,
apoi lăsăm e-mailul să treacă. Dar dacă adresa de e-mail se marchează la 0, suntem
sigur că adresa nu este în S, deci putem renunța la acest element de flux.
Din păcate, unele e-mailuri spam vor primi. Aproximativ 1/8 din
elementele fluxului a căror adresă de e-mail nu se află în S se va întâmpla cu hash la a
bit a cărui valoare este 1 și va fi lăsat să treacă. Cu toate acestea, din moment ce majoritatea
e-mailurile sunt spam (aproximativ 80% conform unor rapoarte), eliminând 7/8 din
spamul este un beneficiu semnificativ. Mai mult, dacă vrem să eliminăm orice spam,

trebuie doar să verificăm pentru apartenența la S acele e-mailuri bune și rele care primesc prin filtru. Aceste verificări vor necesita utilizarea memoriei secundare pentru accesează S în sine. Există și alte opțiuni, așa cum vom vedea când vom studia tehnica generală de filtrare a înfloririi. Ca un exemplu simplu, am putea folosi o cascadă

Pagina 172

de filtre, fiecare dintre acestea eliminând 7/8 din spamul rămas.

4.3.2 Filtrul Bloom

Un filtru Bloom constă din:
1. O matrice de n biți, inițial toți 0.
2. O colecție de funcții hash h_1, h_2, ..., h_k. Fiecare hartă funcțională hash Valori „cheie” pentru n cupe, corespunzătoare celor n biți ai bit-array-ului.
3. Un set S de m valori cheie.

Scopul filtrului Bloom este de a permite prin toate elementele fluxului ale căror tastele sunt în S, respingând în același timp majoritatea elementelor fluxului ale căror taste nu sunt în S.
Pentru a inițializa matricea de biți, începeți cu toți biții 0. Luați fiecare valoare cheie în S și hash-l folosind fiecare dintre funcțiile k hash. Setați la 1 fiecare bit care este h_i (K) pentru unele funcții hash h_i și unele valori cheie K în S.
Pentru a testa o cheie K care ajunge în flux, verificați dacă toate
h_1 (K), h_2 (K), ..., h_k (K)
sunt 1 în matricea de biți. Dacă toate sunt 1, atunci lăsați elementul de flux să treacă. Dacă unul sau mai mulți dintre acești biți sunt 0, atunci K nu ar putea fi în S, deci respingeți fluxul element.

4.3.3 Analiza filtrării Bloom

Dacă o valoare cheie este în S, atunci elementul va trece cu siguranță prin Bloom filtru. Cu toate acestea, dacă valoarea cheii nu este în S, s-ar putea să treacă. Avem nevoie să înțelegeți cum să calculați probabilitatea unui fals pozitiv, în funcție de n, lungimea matricei de biți, m numărul de membri ai lui S și k, numărul de funcții hash.
Modelul de utilizat aruncă săgeți asupra țintelor. Să presupunem că avem x ținte și y săgeți. Orice săgeți este la fel de probabil să atingă orice țintă. După aruncarea săgeți, câte ținte ne putem aștepta să fie lovite cel puțin o dată? Analiza este similar cu analiza din secțiunea 3.4.2 și urmează după cum urmează:

- Probabilitatea ca o săgeată dată să nu atingă o țintă dată este (x - 1) / x.
- Probabilitatea ca niciuna dintre săgețile y să nu atingă o țintă dată este $\left(\frac{x-1}{x}\right)^y$.

Putem scrie această expresie ca $\left(1 - \frac{1}{x}\right)^{x\left(\frac{y}{x}\right)}$.

- Folosind aproximarea $(1 - \varrho)^{1/\varrho} = 1 / e$ pentru small mică (reamintim secțiunea 1.3.5), concluzionăm că probabilitatea ca niciuna dintre săgețile y să nu atingă o țintă dată este $e^{-y/x}$.

Pagina 173

Exemplul 4.3: Luați în considerare exemplul curent din secțiunea 4.3.1. Putem folosi calculul de mai sus pentru a obține adevăratul număr așteptat de 1 în matricea de biți. Gândiți-vă la fiecare bit ca la o țintă și la fiecare membru al lui S ca la o săgeată. Apoi probabilitatea ca un bit dat să fie 1 este probabilitatea ca respectivul ținta va fi lovită de una sau mai multe săgeți. Din moment ce există un miliard de membri ai S, avem $y = 10^9$ săgeți. Deoarece există opt miliarde de biți, există $x = 8 \times 10^9$ ținte. Astfel, probabilitatea ca o anumită țintă să nu fie lovită este $e^{-y/x} = e^{-1/8}$ iar probabilitatea ca acesta să fie lovit este $1 - e^{-1/8}$. Această cantitate este de aproximativ 0,1175. În secțiunea 4.3.1 am sugerat că 1/8 = 0,125 este o bună aproximare, care este, dar acum avem calculul exact. ✷
Putem aplica regula situației mai generale, unde mulțimea S are m membri, matricea are n biți și există k funcții hash. Numarul de ținte este x = n, iar numărul de săgeți este y = km. Astfel, probabilitatea că un pic rămâne 0 este $e^{-km/n}$. Vrem ca fracția de 0 biți să fie corectă

mare sau altfel probabilitatea ca un membru al lui S să hash cel puțin o dată
un 0 devine prea mic și există prea multe pozitive false. De exemplu,
am putea alege k, numărul funcțiilor hash să fie n / m sau mai mic. Apoi
probabilitatea unui 0 este de cel puțin e^{-1} sau 37%. În general, probabilitatea unui fals
pozitiv este probabilitatea unui 1 bit, care este $1 - e^{-km/n}$, crescut la kth
putere, adică $(1 - e^{-km/n})^k$.
Exemplul 4.4: În exemplul 4.3 am constatat că fracția de 1 din matricea lui
exemplul nostru de rulare este 0,1175, iar această fracție este, de asemenea, probabilitatea unui fals
pozitiv. Adică, un membru al lui S va trece prin filtru, dacă acesta se marchează
a 1, iar probabilitatea de a face acest lucru este de 0,1175.
Să presupunem că am folosit același S și aceeași matrice, dar am folosit două diferite
funcții hash. Această situație corespunde aruncării a două miliarde de săgeți asupra
opt miliarde de ținte, iar probabilitatea ca un bit să rămână 0 este $e^{-1/4}$. În ordine
pentru a fi un fals pozitiv, un membru al lui S trebuie să hasheze de două ori la biți care sunt 1,
iar această probabilitate este $(1 - e^{-1/4})^2$, sau aproximativ 0,0493. Astfel, adăugând un
a doua funcție hash pentru exemplul nostru de rulare este o îmbunătățire, reducând
rata fals pozitivă de la 0.1175 la 0.0493. ✷

4.3.4 Exerciții pentru secțiunea 4.3

Exercițiul 4.3.1: Pentru situația exemplului nostru de rulare (8 miliarde de biți, 1
miliarde de membri ai setului S), calculați rata fals-pozitivă dacă folosim trei
funcții hash? Ce se întâmplă dacă folosim patru funcții hash?
! Exercițiul 4.3.2: Să presupunem că avem n biți de memorie disponibili și setul nostru S
are m membri. În loc să folosim k funcții hash, am putea împărți n biți
în k matrice și hash o dată la fiecare matrice. În funcție de n, m și k, ce
este probabilitatea unui fals pozitiv? Cum se compară cu utilizarea hashului k?
funcționează într-o singură matrice?

!! Exercițiul 4.3.3: În funcție de n, numărul de biți și m numărul
de membri din setul S, ce număr de funcții hash minimizează falsul-
rata pozitiva?

4.4 Numărarea elementelor distincte într-un flux

În această secțiune ne uităm la un al treilea tip simplu de procesare pe care ne-ar putea dori
face pe un flux. La fel ca în exemplele anterioare - eșantionare și filtrare - este
oarecum complicat să facem ceea ce dorim într-o cantitate rezonabilă de memorie principală,
deci folosim o varietate de hash și un algoritm randomizat pentru a obține aproximativ
ceea ce ne dorim cu puțin spațiu necesar pentru fiecare flux.

4.4.1 Problema distincției

Să presupunem că elementele fluxului sunt alese dintr-un set universal. Am vrea
să știți câte elemente diferite au apărut în flux, numărând
fie de la începutul fluxului, fie de la un moment cunoscut din trecut.
Exemplul 4.5: Ca un exemplu util al acestei probleme, luați în considerare un site web
elaborând statistici despre câți utilizatori unici a văzut în fiecare lună dată. The
setul universal este setul de conectări pentru acel site și se generează un element de flux
de fiecare dată când cineva se conectează. Această măsură este adecvată pentru un site precum Amazon,
unde utilizatorul tipic se conectează cu numele său unic de autentificare.
O problemă similară este un site Web precum Google, care nu necesită autentificare
emite o interogare de căutare și poate identifica utilizatorii numai după adresa IP
de la care trimit interogarea. Există aproximativ 4 miliarde de adrese IP, 2^{32}
secvențele a patru octeți de 8 biți vor servi ca set universal în acest caz. ✷
Modul evident de a rezolva problema este de a păstra în memoria principală o listă a tuturor
elementele văzute până acum în flux. Păstrați-le într-o structură de căutare eficientă
cum ar fi un tabel hash sau un arbore de căutare, astfel încât să puteți adăuga rapid elemente noi și
verificați dacă elementul care tocmai a sosit pe flux era deja sau nu
văzut. Atâta timp cât numărul de elemente distincte nu este prea mare, această structură
se pot potrivi în memoria principală și există puține probleme la obținerea unui răspuns exact
la întrebarea câte elemente distincte apar în flux.
Cu toate acestea, dacă numărul de elemente distincte este prea mare sau dacă există prea multe
multe fluxuri care trebuie procesate simultan (de exemplu, Yahoo! vrea să numere
numărul de utilizatori unici care vizualizează fiecare pagină într-o lună), apoi noi
nu poate stoca datele necesare în memoria principală. Există mai multe opțiuni. Noi
ar putea folosi mai multe mașini, fiecare mașină manipulând doar una sau mai multe dintre
pâraie. Am putea stoca cea mai mare parte a structurii datelor în memoria secundară și
elemente de flux batch, deci ori de câte ori am adus un bloc de disc în memoria principală
ar fi multe teste și actualizări care trebuie efectuate pe datele din acesta

bloc. Sau am putea folosi strategia care va fi discutată în această secțiune, în care noi
[2] Cel puțin așa va fi până când IPv6 va deveni norma.

estimează doar numărul de elemente distincte, dar folosește mult mai puțină memorie decât numărul de elemente distincte.

4.4.2 Algoritmul Flajolet-Martin

Este posibil să se estimeze numărul de elemente distincte prin hashing elementul menționile setului universal la un șir de biți suficient de lung. Lungimea șirul de biți trebuie să fie suficient încât să existe mai multe rezultate posibile ale hashului funcție decât există elemente ale setului universal. De exemplu, 64 de biți este suficient pentru hash URL-uri. Vom alege multe funcții hash diferite și hash fiecare element al fluxului utilizând aceste funcții hash. Proprietatea importantă a unei funcții hash este că, atunci când este aplicată aceluiași element, produce întotdeauna același rezultat. Observați că această proprietate a fost, de asemenea, esențială pentru eșantionare tehnica din secțiunea 4.2.

Ideea din spatele algoritmului Flajolet-Martin este că cu atât mai diferit elementele pe care le vedem în flux, cu atât mai multe valori hash diferite vom vedea. La fel de vedem mai multe valori hash diferite, devine mai probabil ca una dintre acestea valorile vor fi „neobișnuite”. Proprietatea neobișnuită pe care o vom exploata este că valoarea se termină în multe 0, deși există multe alte opțiuni.

Ori de câte ori aplicăm o funcție hash h unui element de flux a, șirul de biți h (a) se va termina cu un număr de 0, posibil niciunul. Numiți acest număr coadă lungimea pentru a și h. Fie R lungimea maximă a cozii oricărei a văzute până acum curentul. Apoi vom folosi estimarea 2^R pentru numărul de elemente distincte văzut în pârâu.

Această estimare are sens intuitiv. Probabilitatea ca un anumit flux elementul a are h (a) care se termină cu cel puțin r 0 este 2^{-r}. Să presupunem că sunt m distincte elemente din flux. Apoi, probabilitatea ca niciunul dintre ei să nu aibă lungimea cozii cel puțin r este $(1 - 2^{-r})^m$. Acest tip de expresie ar trebui să fie familiar până acum.

Îl putem rescrie ca $((1 - 2^{-r})^{2^r})^{m2^{-r}}$. Presupunând că r este destul de mare, expresia interioară are forma $(1 - Q)^{1/Q}$, care este aproximativ 1 / e. Prin urmare, probabilitatea de a nu găsi un element de flux cu câte 0 la sfârșitul valorii sale hash este $e^{-m2^{-r}}$. Putem concluziona:

1. Dacă m este mult mai mare de 2^r, atunci probabilitatea că vom găsi o coadă de lungime cel puțin r se apropie de 1.
2. Dacă m este mult mai mic de 2^r, atunci probabilitatea de a găsi o lungime a cozii la cel puțin r se apropie de 0.

Din aceste două puncte concluzionăm că estimarea propusă pentru m, care este 2^R (amintirea R este cea mai mare lungime a cozii pentru orice element de curent) este puțin probabil să fie fie mult prea sus, fie mult prea jos.

4.4.3 Combinarea estimărilor

Din păcate, există o capcană în ceea ce privește strategia de combinare a estimărilor de m, numărul de elemente distincte, pe care le obținem folosind multe diferite funcții hash. Prima noastră presupunere ar fi că dacă luăm media de valorile 2^R pe care le obținem din fiecare funcție hash, vom obține o valoare care se apropie de adevăratul m, cu atât folosim mai multe funcții hash. Cu toate acestea, nu este așa cazul și motivul are legătură cu influența pe care o are o supraestimare media.

Luați în considerare o valoare a lui r astfel încât 2^r să fie mult mai mare decât m. Este ceva probabilitatea p că vom descoperi că r este cel mai mare număr de 0 la final a valorii hash pentru oricare dintre elementele de flux m. Atunci probabilitatea de găsirea lui r + 1 ca fiind cel mai mare număr de 0 este în schimb cel puțin p / 2. Cu toate acestea, dacă creștem cu 1 numărul de 0 la sfârșitul unei valori hash, valoarea lui 2^R duble. În consecință, contribuția de la fiecare posibil R mare la valoarea așteptată de 2^R crește pe măsură ce R crește, iar valoarea așteptată de 2^R este de fapt infinit. [3]

O altă modalitate de a combina estimările este de a lua mediana tuturor estimărilor.

Mediana nu este afectată de valoarea ocazională supradimensionată de 2^R, deci îngrijorarea descris mai sus pentru medie nu ar trebui să fie transferat la mediană. Nefericit în mod natural, mediana suferă de un alt defect: este întotdeauna o putere de 2. Astfel, indiferent de câte funcții hash folosim, ar trebui să fie valoarea corectă a lui m între două puteri de 2, să zicem 400, atunci va fi imposibil să se obțină o închidere estima.

Cu toate acestea, există o soluție la problemă. Putem combina cele două metode. Mai întâi, grupați funcțiile hash în grupuri mici și luați-le in medie. Apoi, luați mediana mediilor. Este adevărat că un ocazional 2^R supradimensionat va distinge unele grupuri și le va face prea mari. In orice caz, luarea mediană a mediilor de grup va reduce influența acestui efect aproape la nimic. Mai mult, dacă grupurile în sine sunt suficient de mari, atunci mediile pot fi în esență orice număr, ceea ce ne permite să abordăm valoarea adevărată m atâta timp cât folosim suficiente funcții hash. Pentru a garanta că se poate obține orice medie posibilă, grupurile ar trebui să aibă dimensiuni cel puțin a mic multiplu de buşteni 2 m.

4.4.4 Cerințe de spațiu

Observați că, pe măsură ce citim fluxul, nu este necesar să stocați elementele văzut. Singurul lucru pe care trebuie să-l păstrăm în memoria principală este un număr întreg per hash funcție; acest număr întreg înregistrează cea mai mare lungime a cozii văzută până acum pentru acel hash funcție și orice element de flux. Dacă procesăm un singur flux, noi ar putea folosi milioane de funcții hash, ceea ce este mult mai mult decât avem nevoie pentru a obține un

[3] Din punct de vedere tehnic, deoarece valoarea hash este un şir de biți de lungime finită, nu există nicio contribuție la 2^R pentru R care sunt mai mari decât lungimea valorii hash. Cu toate acestea, acest efect nu este suficient pentru a evita concluzia că valoarea aşteptată de 2^R este mult prea mare.

---

estimare apropiată. Doar dacă încercăm să procesăm multe fluxuri în același timp memoria principală ar constrânge numărul de funcții hash pe care le-am putea asocia cu oricare flux. În practică, timpul necesar calculării valorilor hash pentru fiecare element de flux ar fi limitarea mai semnificativă a numărului de funcțiile hash pe care le folosim.

4.4.5 Exerciții pentru secțiunea 4.4

Exercițiul 4.4.1: Să presupunem că fluxul nostru este format din numere întregi 3, 1, 4, 1, 5, 9, 2, 6, 5. Funcțiile noastre hash vor fi toate de forma h (x) = ax + b mod 32 pentru unii a și b. Ar trebui să tratați rezultatul ca un întreg binar pe 5 biți. Determinați lungimea cozii pentru fiecare element de curent și estimarea rezultată a numărului de elemente distincte dacă funcția hash este:

(a) h (x) = 2x +1 mod 32.
(b) h (x) = 3x +7 mod 32.
(c) h (x) = 4x mod 32.

! Exercițiul 4.4.2: vedeți probleme în alegerea funcțiilor hash în Exercițiul 4.4.1? Ce sfaturi ai putea da cuiva care urma să folosească un funcția hash a formei h (x) = ax + b mod 2^k ?

4.5 Estimarea momentelor

În această secțiune considerăm o generalizare a problemei numărării distincte elemente dintr-un flux. Problema, numită „momente” de calcul, implică distribuirea frecvențelor diferitelor elemente în flux. Vom defini momente din toate comenzile și concentrați-vă pe calcularea momentelor secundare, de la care algoritmul general pentru toate momentele este o extensie simplă.

4.5.1 Definiția momentelor

Să presupunem că un flux constă din elemente alese dintr-un set universal. Să presupunem că setul universal este ordonat astfel încât să putem vorbi despre elementul i pentru orice i. Să m_i fie numărul de apariții al elementului i pentru orice i. Apoi ordinul kth momentul (sau doar al k-lea moment) al fluxului este suma peste tot i din $(m_i)^k$.

Exemplul 4.6: Al 0-lea moment este suma de 1 pentru fiecare m_i care este mai mare decât 0. [4] Adică momentul 0 este un număr al numărului de elemente distincte din curentul. Putem folosi metoda Secțiunii 4.4 pentru a estima momentul 0 a unui pârâu.

[4] Din punct de vedere tehnic, deoarece m_i ar putea fi 0 pentru unele elemente din setul universal, trebuie să facem explicit în definiția „momentului” că 0^0 este considerat 0. Pentru momentele 1 și mai sus, contribuția lui m_i care este 0 este cu siguranță 0.

Primul moment este suma lui m_i, care trebuie să fie lungimea lui curent. Astfel, primele momente sunt deosebit de ușor de calculat; numără doar lungimea pârâului văzut până acum.
Al doilea moment este suma pătratelor m_i's. Este oarecum numit numărul surpriză, deoarece măsoară cât de inegală este distribuția ziunea elementelor din flux este. Pentru a vedea distincția, să presupunem că avem o flux de lungime 100, în care apar unsprezece elemente diferite. Cel mai distribuția uniformă a acestor unsprezece elemente ar avea una care să apară de 10 ori iar celelalte zece aparând de 9 ori fiecare. În acest caz, numărul surpriză este $10^2 + 10 \times 9^2 = 910$. La cealaltă extremă, unul dintre cele unsprezece elemente ar putea apar de 90 de ori, iar celelalte zece apar de câte 1 ori fiecare. Apoi, surpriza numărul ar fi $90^2 + 10 \times 1^2 = 8110$. ✷
La fel ca în Secțiunea 4.4, nu există nicio problemă de calculare a momentelor de orice ordine dacă noi își poate permite să păstreze în memoria principală un număr pentru fiecare element care apare în curent. Cu toate acestea, la fel ca în acea secțiune, dacă nu ne permitem să folosim atât de mult memorie, atunci trebuie să estimăm al k-lea moment păstrând un număr limitat valorilor din memoria principală și calculând o estimare din aceste valori. Pentru în cazul elementelor distincte, fiecare dintre aceste valori a fost numărarea celei mai lungi coada produs de o singură funcție hash. Vom vedea o altă formă de valoare care este util pentru momentele secundare și superioare.

4.5.2 Algoritmul Alon-Matias-Szegedy pentru al doilea Momente

Deocamdată, să presupunem că un flux are o anumită lungime n. Vom arăta cum să faceți față fluxurilor în creștere în secțiunea următoare. Să presupunem că nu avem suficient spațiu pentru a număra toate m_i-urile pentru toate elementele fluxului. Putem estimează încă al doilea moment al fluxului folosind o cantitate limitată de spațiu; cu cât vom folosi mai mult spațiu, cu atât va fi mai precisă estimarea. Calculăm un anumit număr de variabile. Pentru fiecare variabilă X, stocăm:
1. Un element particular al setului universal, la care ne referim ca X.element, și
2. Un număr întreg X.value, care este valoarea variabilei. Pentru a determina valoarea unei variabile X, alegem o poziție în flux între 1 și n, uniform și la întâmplare. Setați X.element să fie elementul găsit acolo, și inițializați X.value la 1. Pe măsură ce citim fluxul, adăugați 1 la X.value fiecare când întâlnim o altă apariție a elementului X.
Exemplul 4.7: Să presupunem că fluxul este a, b, c, b, d, a, c, d, a, b, d, c, a, a, b. The lungimea fluxului este n = 15. Deoarece a apare de 5 ori, b apare de 4 ori, și c și d apar de trei ori fiecare, al doilea moment pentru flux este $5^2 + 4^2 + 3^2 + 3^2 = 59$. Să presupunem că păstrăm trei variabile, X_1, X_2 și X_3. De asemenea,

---

presupunem că la „întâmplare” alegem pozițiile 3, 8 și 13 pentru a le defini aceste trei variabile.
Când ajungem la poziția 3, găsim elementul c, deci setăm X_1.element = c și X_1.valoare = 1. Poziția 4 menține b, deci nu schimbăm X_1. De asemenea, nimic nu se întâmplă la pozițiile 5 sau 6. La poziția 7, vedem din nou c, așa că setăm X_1.valoare = 2.
În poziția 8 găsim d, deci setați X_2.element = d și X_2.valor = 1.
Pozițiile 9 și 10 dețin a și b, deci nu afectează X_1 sau X_2. Poziția 11 ține d așa că setăm X_2.value = 2, iar poziția 12 ține c așa că setăm X_1.value = 3.
În poziția 13, găsim elementul a, deci setați X_3.element = a și X_3.valor = 1.
Apoi, la poziția 14 vedem un alt a și așa setăm X_3.value = 2. Poziția 15, cu elementul b nu afectează niciuna dintre variabile, așa că am terminat, cu final valori X_1.valor = 3 și X_2.valor = X_3.valor = 2. ✷
Putem obține o estimare a celui de-al doilea moment din orice variabilă X. Aceasta estimarea este n (2X. valoare - 1).
Exemplul 4.8: Luați în considerare cele trei variabile din exemplul 4.7. Din X_1 noi obțineți estimarea $n (2X_1.\text{valoare} - 1) = 15 \times (2 \times 3 - 1) = 75$. Cealaltă două variabile, X_2 și X_3, fiecare au valoarea 2 la final, deci estimările lor sunt $15 \times (2 \times 2 - 1) = 45$. Amintiți-vă că adevărata valoare a celui de-al doilea moment pentru aceasta fluxul este de 59. Pe de altă parte, media celor trei estimări este de 55, a aproximare destul de apropiată. ✷

4.5.3 De ce funcționează algoritmul Alon-Matias-Szegedy

Putem demonstra că valoarea așteptată a oricărei variabile construite ca în Sec-

secțiunea 4.5.2 este al doilea moment al fluxului din care este construit.
Unele notații vor face argumentul mai ușor de urmărit. Fie e (i) fluxul
element care apare în poziția i în flux și să fie c (i) numărul de
ori elementul e (i) apare în flux între pozițiile i, i + 1, ..., n.
Exemplul 4.9: Luați în considerare fluxul din Exemplul 4.7. e (6) = a, din 6
poziția deține o. De asemenea, c (6) = 4, deoarece a apare la pozițiile 9, 13 și 14, ca
precum și la poziția 6. Rețineți că a apare și la poziția 1, dar acest fapt se întâmplă
nu contribui la c (6). ✷
Valoarea așteptată a n (2X.value - 1) este media pentru toate pozițiile i
între 1 și n din n (2c (i) - 1), adică
E (n (2X. Valoare - 1)) =
1
n
$\sum_{i=1}^{n}$
n (2c (i) - 1)
Putem simplifica cele de mai sus prin anularea factorilor 1 / n și n, pentru a obține
E (n (2X. Valoare - 1)) =
$\sum_{i=1}^{n}$ (2c (i) - 1)

Cu toate acestea, pentru a înțelege formula, trebuie să schimbăm ordinea
însumarea prin gruparea tuturor acelor poziții care au același element. Pentru
de exemplu, concentrați-vă pe un element a care apare de mai multe ori în flux.
Termenul pentru ultima poziție în care apare a trebuie să fie 2 × 1 - 1 = 1.
termenul pentru următoarea poziție în care apare a este 2 × 2 - 1 = 3.
poziții cu un înainte care dau termenii 5, 7 și așa mai departe, până la $2m_a$ - 1, care
este termenul pentru prima poziție în care apare a. Adică formula pentru
valoarea așteptată de 2X.value - 1 poate fi scrisă:
E (n (2X. Valoare - 1)) = Σ_a
$1 + 3 + 5 + \cdots + (2m_a - 1)$
Rețineți că $1 + 3 + 5 + \cdots + (2m_a - 1) = (m_a)^2$. Dovada este o inducție ușoară
pe numărul de termeni din sumă. Astfel, E (n (2X. Valoare - 1)) = $\Sigma_a (m_a)^2$,
care este definiția celui de-al doilea moment.

4.5.4 Momente de ordin superior

Estimăm momentele k, pentru k> 2, în esență în același mod în care estimăm
al doilea moment. Singurul lucru care se schimbă este modul în care obținem o estimare
dintr-o variabilă. În secțiunea 4.5.2 am folosit formula n (2v - 1) pentru a transforma o valoare
v, numărul de apariții al unui anumit element de flux
a, într-o estimare a celui de-al doilea moment. Apoi, în secțiunea 4.5.3 am văzut de ce
această formulă funcționează: termenii 2v - 1, pentru v = 1, 2, ..., m sumă la m^2, unde m
este numărul de apariții a în flux.
Observați că 2v - 1 este diferența dintre v^2 și $(v - 1)^2$. Să presupunem că noi
voiam mai degrabă al treilea moment decât al doilea. Atunci tot ce trebuie să facem este
înlocuiți 2v − 1 cu $v^3 - (v - 1)^3 = 3v^2 - 3v + 1$. Apoi $\Sigma_{v=1}^{m} 3v^2 - 3v + 1 = m^3$, deci noi
putem folosi ca estimare a celui de-al treilea moment formula $n (3v^2 - 3v + 1)$, unde
v = X.value este valoarea asociată cu o variabilă X. Mai general, noi
poate estima kth momente pentru orice k ≥ 2 prin transformarea valorii v = X. valoare în
$n (v^k - (v - 1)^k)$.

4.5.5 Tratarea fluxurilor infinite

Din punct de vedere tehnic, estimarea pe care am folosit-o pentru momentele secundare și superioare presupune că
n, lungimea fluxului, este o constantă. În practică, n crește cu timpul. Acest fapt,
de la sine, nu cauzează probleme, deoarece stocăm numai valorile variabilelor
și înmulțiți o funcție a valorii respective cu n când este timpul să estimați
moment. Dacă numărăm numărul de elemente de flux văzute și stocăm această valoare,
care necesită doar jurnal n biți, atunci avem n disponibile ori de câte ori avem nevoie de el.
O problemă mai gravă este că trebuie să fim atenți la modul în care selectăm pozițiile
pentru variabile. Dacă facem această selecție odată pentru totdeauna, atunci pe măsură ce fluxul devine
mai mult, suntem părtinitori în favoarea pozițiilor timpurii și a estimării momentului
va fi prea mare. Pe de altă parte, dacă așteptăm prea mult pentru a alege poziții, atunci

---

la începutul fluxului, nu avem multe variabile și astfel vom obține un nesigur estima.
Tehnica adecvată este de a menține cât mai multe variabile în care putem stoca tot timpul și să le arunci pe măsură ce pârâul crește. Variabilele aruncate sunt înlocuite cu altele noi, în așa fel încât probabilitatea de a alegerea oricărei poziții pentru o variabilă este aceeași cu alegerea oricărei alte poziții poziție. Să presupunem că avem spațiu pentru stocarea variabilelor. Apoi primele poziții ale fluxului sunt selectate fiecare ca poziția uneia dintre variabilele s.
Inductiv, să presupunem că am văzut n elemente de flux și probabilitatea de orice poziție particulară fiind poziția unei variabile este uniformă, adică s / n.
Când ajunge elementul (n + 1), alegeți poziția respectivă cu probabilitatea s / (n + 1). Dacă nu sunt selectate, variabilele s își păstrează aceleași poziții. Cu toate acestea, dacă (n + 1) se alege prima poziție, apoi aruncați una dintre variabilele curente, cu probabilitate egală. Înlocuiți-o pe cea aruncată de o nouă variabilă al cărei element este cel din poziția n + 1 și a cărui valoare este 1.
Cu siguranță, probabilitatea ca poziția n + 1 să fie selectată pentru o variabilă este ceea ce ar trebui să fie: s / (n + 1). Cu toate acestea, probabilitatea oricărei alte poziții, de asemenea este s / (n + 1), după cum putem demonstra prin inducție pe n. Prin ipoteza inductivă, înainte de sosirea elementului (n + 1) st stream, această probabilitate era s / n.
Cu probabilitatea 1 - s / (n + 1) poziția (n + 1) st nu va fi selectată și probabilitatea fiecăreia dintre primele n poziții rămâne s / n. Cu toate acestea, cu probabilitatea s / (n + 1), se alege prima poziție (n + 1) și probabilitatea pentru fiecare dintre primele n poziții este redusă cu factorul (s - 1) / s. Având în vedere cele două cazuri, probabilitatea de a selecta fiecare dintre primele n poziții este
(1 -
s
n + 1) (
s
n) + (
s
n + 1) (s
- 1
s) (
s
n)
Această expresie simplifică la
(1 -
s
n + 1) (
s
n) + (
s - 1
n + 1) (
s
n)
și apoi la
((1 -
s
n + 1) + (
s - 1
n + 1)) (
s
n)
care la rândul său se simplifică la
(n
n + 1) (
s
n)
=
s
n + 1
Astfel, am arătat prin inducție pe lungimea fluxului n că toate pozițiile au o probabilitate egală s / n de a fi aleasă ca poziția unei variabile.

4.5.6 Exerciții pentru secțiunea 4.5

Exercițiul 4.5.1: Calculați numărul surpriză (al doilea moment) pentru flux 3, 1, 4, 1, 3, 4, 2, 1, 2. Care este al treilea moment al acestui flux?

O problemă generală de eșantionare a fluxului

Observați că tehnica descrisă în secțiunea 4.5.5 rezolvă, de fapt, mai mult problemă generală. Ne oferă o modalitate de a menține un eșantion de flux elemente astfel încât, în orice moment, toate elementele fluxului să fie la fel de probabil selectat pentru eșantion.

Ca exemplu de unde această tehnică poate fi utilă, amintiți-vă că în Secțiunea 4.2 am aranjat să selectăm toate tuplurile unui flux cu cheie valoare într-un subset selectat aleatoriu. Să presupunem că, pe măsură ce trece timpul, acolo sunt prea multe tupluri asociate cu orice tastă. Putem aranja să limităm numărul de tupluri pentru orice cheie K la o constantă fixă s folosind tehnica Secțiunii 4.5.5 ori de câte ori ajunge un nou tuplu pentru cheia K.

! Exercițiul 4.5.2: Dacă un flux are n elemente, dintre care m sunt distincte, care sunt numărul surpriză minim și maxim posibil, în funcție de m și n?

Exercițiul 4.5.3: Să presupunem că ni se dă fluxul exercițiului 4.5.1, la care aplicăm algoritmul Alon-Matias-Szegedy pentru a estima numărul surpriză. Pentru fiecare valoare posibilă a lui i, dacă X_i este o poziție de pornire variabilă i, care este valoarea lui X_i .valor?

Exercițiul 4.5.4: Repetați exercițiul 4.5.3 dacă intenția variabilelor este să pute al treilea moment. Care este valoarea fiecărei variabile la final? Ce estimarea celui de-al treilea moment obțineți de la fiecare variabilă? Cum funcționează media acestor estimări se compară cu valoarea reală a celui de-al treilea moment?

Exercițiul 4.5.5: Dovediți prin inducție pe m că $1 + 3 + 5 + \cdots + (2m - 1) = m^2$.

Exercițiul 4.5.6: Dacă am vrea să calculăm al patrulea moment, cum am face converti valoarea X la o estimare a celui de-al patrulea moment?

4.6 Numărarea celor într-o fereastră

Ne îndreptăm acum atenția asupra numărării problemelor pentru fluxuri. Să presupunem că avem o fereastră de lungime N pe un flux binar. Vrem în orice moment să putem răspundeți la întrebări de forma „câte 1 există în ultimii k biți?” pentru orice $k \leq N$. La fel ca în secțiunile anterioare, ne concentrăm asupra situației în care nu putem permiteți-vă să stocați întreaga fereastră. După ce a arătat un algoritm aproximativ pentru în cazul binar, discutăm cum această idee poate fi extinsă la sumarea numerelor.

4.6.1 Costul numărărilor exacte

Pentru început, să presupunem că vrem să putem conta exact numărul de 1 în ultimii k biți pentru orice $k \leq N$. Apoi susținem că este necesar să stocăm tot N biți ai ferestrei, așa cum ar putea orice reprezentare care utilizează mai puțin de N biți nu functioneaza. În dovadă, să presupunem că avem o reprezentare care folosește mai puțin de N biți pentru a reprezenta N biți din fereastră. Deoarece există 2^N secvențe de N biți, dar mai puțin de 2^N reprezentări, trebuie să existe două șiruri de biți diferiți w și x care au aceeași reprezentare. Deoarece w = x, acestea trebuie să difere în cel puțin un bit. Fie ca ultimii k - 1 biți de w și x să fie de acord, dar să difere bitul k de la capătul drept.

Exemplul 4.10: Dacă w = 0101 și x = 1010, atunci k = 1, de la scanarea din dreapta, mai întâi nu sunt de acord în poziția 1. Dacă w = 1001 și x = 0101, atunci k = 3, deoarece mai întâi nu sunt de acord cu a treia poziție din dreapta. ✷

Să presupunem că datele care reprezintă conținutul ferestrei sunt indiferent de secvența de biți reprezintă atât w cât și x. Întrebați întrebarea „în câte sunt 1 ultimii k biți? ” Algoritmul de răspuns la interogare va produce același answer, indiferent dacă fereastra conține w sau x, deoarece algoritmul poate vedea numai reprezentarea lor. Dar răspunsurile corecte sunt cu siguranță diferite pentru aceste două șiruri de biți. Astfel, am demonstrat că trebuie să folosim cel puțin N biți pentru a răspunde interogări despre ultimii k biți pentru orice k.

De fapt, avem nevoie de N biți, chiar dacă singura interogare pe care o putem întreba este „câte 1 sunt în întreaga fereastră cu lungimea N? ” Argumentul este similar cu acela folosit mai sus. Să presupunem că folosim mai puțin de N biți pentru a reprezenta fereastra și prin urmare putem găsi w, x și k ca mai sus. S-ar putea ca w și x să aibă același număr de 1, așa cum au făcut-o în ambele cazuri din exemplul 4.10. In orice caz,

dacă urmărim fereastra curentă cu orice N - k biți, vom avea o situație
unde conținutul adevărat al ferestrei rezultat din w și x este identic cu excepția
bitul din stânga și, prin urmare, numărul lor de 1 este inegal. Cu toate acestea, din moment ce
reprezentările lui w și x sunt aceleași, reprezentarea ferestrei
trebuie să fie la fel dacă alimentăm aceeași secvență de biți la aceste reprezentări.
Astfel, putem forța răspunsul la întrebarea „câte 1 sunt în fereastră?” la
să fie incorect pentru unul dintre cele două posibile conținuturi ale ferestrei.

4.6.2 Algoritmul Datar-Gionis-Indyk-Motwani

Vom prezenta cel mai simplu caz al unui algoritm numit DGIM. Această versiune de
algoritmul folosește O (log 2 N) biți pentru a reprezenta o fereastră de N biți și permite
să estimăm numărul de 1 din fereastră cu o eroare de cel mult
50%. Mai târziu, vom discuta despre o îmbunătățire a metodei care limitează
eroare la orice fracție ϱ> 0 și încă folosește numai biți O (log 2 N) (deși cu un
factor constant care crește pe măsură ce ϱ se micșorează).
Pentru început, fiecare bit al fluxului are un timestamp, poziția în care acesta
ajunge. Primul bit are timestamp 1, al doilea are timestamp 2 și așa mai departe.

Deoarece trebuie doar să distingem pozițiile din fereastra de lungime N, noi
vor reprezenta marcaje de timp modulo N, deci pot fi reprezentate prin log 2 N
biți. Dacă stocăm și numărul total de biți văzuți vreodată în flux (adică
cel mai recent timestamp) modulo N, atunci putem determina dintr-un timestamp
modulo N unde în fereastra curentă este bitul cu acel timestamp.
Împărțim fereastra în găleți, 5 constând din:
1. Marcajul de timp al finalului său drept (cel mai recent).
2. Numărul de 1 din găleată. Acest număr trebuie să aibă o putere de 2 și
ne referim la numărul de 1 ca mărimea cupei.
Pentru a reprezenta o găleată, avem nevoie de log 2 N biți pentru a reprezenta marca de timp (modulo
N) de capătul său drept. Pentru a reprezenta numărul de 1 avem nevoie doar de log 2 log 2 N
biți. Motivul este că știm că acest număr i este o putere de 2, să zicem 2 j , deci noi
poate reprezenta i codând j în binar. Deoarece j este cel mult log 2 N, necesită
log 2 log 2 N biți. Astfel, O (log N) biți sunt suficienți pentru a reprezenta o bucket.
Există șase reguli care trebuie respectate atunci când se reprezintă un flux de către
găleți.

- Capătul drept al unei găleți este întotdeauna o poziție cu 1.
- Fiecare poziție cu 1 este într-o găleată.
- Nicio poziție nu se află în mai mult de o găleată.
- Există una sau două găleți de orice dimensiune dată, până la o dimensiune maximă.
- Toate dimensiunile trebuie să aibă o putere de 2.
- Cupele nu pot scădea în dimensiune pe măsură ce ne deplasăm spre stânga (înapoi în timp).

mărimea 1
Două dintre
Una dintre
mărimea 2
Cel puțin unul
de mărimea 8
Două de mărimea 4

```
. . 1 0 1 1 0 1 1 0 0 0 1 0 1 1 1 0 1 1 0 0 1 0 1 1 0
1 0 1 1 0 0 0 1 0 1 1 1 0 1 1 0 0 1
. . . 1 0 1
0 1 1 0
```

Figura 4.2: Un flux de biți împărțit în cupe după regulile DGIM

5 Nu confundați aceste „găleți” cu „gălețile” discutate în legătură cu hashing.

Exemplul 4.11: Figura 4.2 arată un flux de biți împărțit în găleți într-un fel
care îndeplinește regulile DGIM. La capătul din dreapta (cel mai recent) vedem două găleți
de mărimea 1. În stânga sa vedem o găleată de mărimea 2. Rețineți că această găleată acoperă
patru poziții, dar doar două dintre ele sunt 1. Continuând la stânga, vedem două găleți
de mărimea 4 și vă sugerăm că există o găleată de mărimea 8 mai la stânga.
Observați că este OK ca unii 0 să se așeze între găleți. De asemenea, observați din

Fig. 4.2 că găleţile nu se suprapun; există una sau două din fiecare dimensiune până la cea mai mare dimensiune, iar dimensiunile cresc doar deplasându-se spre stânga. ✷

În secţiunile următoare, vom explica următoarele despre ceva DGIM-ritm:

1. De ce numărul de găleţi care reprezintă o fereastră trebuie să fie mic.
2. Cum se estimează numărul de 1 din ultimii k biţi pentru orice k, cu un eroare nu mai mare de 50%.
3. Cum se menţin condiţiile DGIM pe măsură ce noii biţi intră în flux.

4.6.3 Cerinţe de stocare pentru algoritmul DGIM

Am observat că fiecare cupă poate fi reprezentată prin O (log N) biţi. Dacă fereastra are lungimea N, atunci nu există mai mult decât N 1, cu siguranţă. Să presupunem că cea mai mare găleată are dimensiunea 2^j . Atunci j nu poate depăşi $\log_2 N$, altfel sunt mai multe 1 este în această găleată decât există 1 în întreaga fereastră. Astfel, există la cele mai multe două găleţi de toate dimensiunile, de la jurnal $_2$ N până la 1, şi nu găleţi mai mari dimensiuni.

Concluzionăm că există găleţi O (log N). Deoarece fiecare găleată poate fi reprezentat în O (log N) biţi, spaţiul total necesar pentru toate găleţile reprezentate trimiterea unei ferestre de dimensiunea N este O ($\log_2 N$).

4.6.4 Răspunsul la interogare în algoritmul DGIM

Să presupunem că suntem întrebaţi câte 1 există în ultimii k biţi ai ferestrei, pentru aproximativ $1 \le k \le N$. Găsiţi cupa b cu cea mai timpurie marcaj de timp care include cel puţin unele dintre cele mai recente biţi k. Estimează numărul de 1 până la fie suma mărimilor tuturor găleţilor din dreapta (mai recentă) decât găleţii b, plus jumătate din dimensiunea lui b în sine.

Exemplul 4.12: Să presupunem că fluxul este cel din Fig. 4.2 şi k = 10. Apoi interogarea solicită numărul de 1 din cei zece biţi din dreapta, care se întâmplă să fie 0110010110. Fie că marca de timp curentă (ora bitului din dreapta) este t. Apoi cele două găleţi cu unul 1, având marcaje de timp t - 1 şi t - 2 sunt complet inclus în răspuns. Găleată de mărimea 2, cu timestamp t - 4, este, de asemenea complet inclus. Cu toate acestea, cea mai dreaptă găleată de mărimea 4, cu marcaj de timp t - 8 este inclus doar parţial. Ştim că este ultima găleată care contribuie la răspundeţi, deoarece următoarea găleată din stânga are marcaj de timp mai mic de t - 9 şi

---

astfel este complet afară din fereastră. Pe de altă parte, cunoaştem găleţile în dreapta sa se află complet în raza de interogare din cauza existenţei dintr-o găleată la stânga cu marcaj de timp t - 9 sau mai mare.

Estimarea noastră a numărului de 1 în ultimele zece poziţii este astfel 6. Aceasta numărul este cele două găleţi de mărimea 1, găleată de mărimea 2 şi jumătate din găleată de mărimea 4 care se află parţial în raza de acţiune. Desigur, răspunsul corect este 5. ✷

Să presupunem că estimarea de mai sus a răspunsului la o interogare implică o găleată b de dimensiunea 2^j care se află parţial în intervalul interogării. Să ne gândim cum departe de răspunsul corect c estimarea noastră ar putea fi. Există două cazuri: estimarea ar putea fi mai mare sau mai mică decât c.

Cazul 1: estimarea este mai mică decât c. În cel mai rău caz, toate 1 ale lui b sunt de fapt, în raza de acţiune a interogării, deci estimării îi lipseşte jumătate de găleată b sau 2^{j-1} 1. Dar în acest caz, c este de cel puţin 2^j ; de fapt este cel puţin 2^{j+1} - 1, deoarece există cel puţin o găleată din fiecare dintre dimensiunile 2^{j-1} , 2^{j-2} , ..., 1. Încheiem că estimarea noastră este de cel puţin 50% din c.

Cazul 2: estimarea este mai mare decât c. În cel mai rău caz, doar cel din dreapta bucata de găleată b se află în raza de acţiune şi există o singură găleată din fiecare dintre dimensiuni mai mic decât b. Atunci $c = 1 + 2^{j-1} + 2^{j-2} + \cdots + 1 = 2^j$ şi estimarea vom da este $2^{j-1} + 2^{j-1} + 2^{j-2} + \cdots + 1 = 2^j + 2^{j-1} - 1$. Vedem că estimarea nu este cu mai mult de 50% mai mare decât c.

4.6.5 Menţinerea condiţiilor DGIM

Să presupunem că avem o fereastră de lungime N reprezentată corespunzător prin găleţi care să îndeplinească condiţiile DGIM. Când apare un bit nou, este posibil să fie nevoie să modificăm găleţile, astfel încât acestea să continue să reprezinte fereastra şi să satisfacă în continuare condiţiile DGIM. În primul rând, ori de câte ori intră un bit nou:

- Verificaţi cel mai stâng (cel mai devreme) cupă. Dacă marca sa de timp a ajuns acum marcajul de timp actual minus N, atunci această găleată nu mai are nimic 1 e în fereastră. Prin urmare, aruncaţi-l din lista găleţilor.

Acum, trebuie să ne gândim dacă bitul nou este 0 sau 1. Dacă este 0, atunci nu este necesară schimbarea ulterioară a cupelor. Cu toate acestea, dacă noul bit este 1, este posibil trebuie să facă mai multe modificări. Primul:

• Creați o găleată nouă cu marcajul de timp actual și dimensiunea 1.
Dacă a existat o singură găleată de mărimea 1, atunci nu mai trebuie făcut nimic.
Cu toate acestea, dacă există acum trei găleți de mărimea 1, aceasta este una prea multe. Remediem
această problemă prin combinarea celor mai stângi (cele mai vechi) două găleți de mărimea 1.

• Pentru a combina oricare două găleți adiacente de aceeași dimensiune, înlocuiți-le cu
o găleată de două ori mai mare. Marcajul de timp al noii cupe este
marca de timp a celei mai drepte (mai târziu în timp) dintre cele două găleți.

Pagina 187

Combinarea a două găleți de mărimea 1 poate crea o a treia găleță de mărimea 2. Dacă da,
combinăm cele două găleți din stânga de mărimea 2 într-o găleată de mărimea 4. Asta, în
la rândul său, se poate crea o a treia găleată de mărimea 4 și, în caz afirmativ, combinăm cele două din stânga
într-o găleată de mărimea 8. Acest proces se poate derula prin dimensiunile găleatei, dar
există cel mult $\log_2 N$ dimensiuni diferite și combinația a două adiacente
gălețile de aceeași dimensiune necesită doar timp constant. Ca urmare, orice bit nou
poate fi procesat în timp O (log N).
Exemplul 4.13: Să presupunem că începem cu gălețile din Fig. 4.2 și intră un 1.
În primul rând, găleata din stânga evident nu a căzut pe fereastră, așa că noi
nu aruncați găleți. Creăm o nouă găleată de mărimea 1 cu cea actuală
timestamp, spuneți t. Acum există trei găleți de mărimea 1, așa că combinăm
cele mai la stânga două. Acestea sunt înlocuite cu o singură găleată de mărimea 2. Marcajul său de timp este
t - 2, marca de timp a cupei din dreapta (adică cea mai dreaptă cupă care
apare de fapt în Fig. 4.2.

```
Cel puțin unul
de mărimea 8
Două de mărimea 4
mărimea 2
Două dintre
Una dintre
mărimea 1
10
1 1
0
1 0 0 1
1 1 1 0 1
0
1 0 1 1 0 0 0 1
. . 1 0 1
```

Figura 4.3: Cupe modificate după ce un nou 1 ajunge în flux
Acum există două găleți de dimensiunea 2, dar acest lucru este permis de regulile DGIM.
Astfel, secvența finală a găleților după adăugarea 1 este așa cum se arată în
Fig. 4.3. ✷

4.6.6 Reducerea erorii

În loc să permiteți una sau două dintre fiecare găleată de dimensiune, să presupunem că permitem oricare
r - 1 sau r din fiecare dintre mărimile 1, 2, 4, ... în creștere exponențială, pentru un număr întreg
r> 2. Pentru a reprezenta orice număr posibil de 1, trebuie să relaxăm acest lucru
condiție pentru gălețile de mărimea 1 și gălețile de cea mai mare mărime prezente; Acolo
poate fi orice număr, de la 1 la r, de găleți de aceste dimensiuni.
Regula pentru combinarea găleților este în esență aceeași ca în secțiunea 4.6.5. Dacă
obținem r + 1 găleți de mărimea 2^j, combinăm cele două din stânga într-o găleată de mărime
2^{j+1}. Acest lucru poate determina, la rândul său, să existe găleți r + 1 de dimensiunea 2^{j+1} și, dacă da
continuăm să combinăm găleți de dimensiuni mai mari.
Argumentul utilizat în secțiunea 4.6.4 poate fi folosit și aici. Cu toate acestea, pentru că
există mai multe găleți de dimensiuni mai mici, putem obține o legătură mai puternică asupra erorii.
Am văzut acolo că cea mai mare eroare relativă apare atunci când doar un 1 din
găleată cea mai stângă b se află în intervalul de interogare și, prin urmare, supraestimăm
adevărat număr. Să presupunem că găleata b are dimensiunea 2^j. Atunci adevăratul număr este cel puțin

---

Pagina 188

Dimensiunile cupei și adăugătoarele de transport

Există un model de distribuție a dimensiunilor cupei pe măsură ce executăm
algoritmul de bază al secțiunii 4.6.5. Gândiți-vă la două găleți de dimensiunea 2^j ca la „1”
în poziția j și o găleată de dimensiunea 2^j ca „0” în acea poziție. Apoi

pe măsură ce sosesc 1 în flux, dimensiunile cupei după fiecare 1 se formează consecutiv numere întregi binare. Secvențele lungi ocazionale de combinații de cupe sunt analoage cu ochiurile lungi ocazionale ale transporturilor pe măsură ce trecem de la un număr întreg, cum ar fi 101111 până la 110000.

$1+ (r - 1) (2^{j-1} + 2^{j-2} + \cdots +1) = 1 + (r - 1) (2^j - 1)$. Supraevaluarea este $2^{j-1} - 1$. Astfel, eroarea fracțională este

$$\frac{2^{j-1} - 1}{1+ (r - 1) (2^j - 1)}$$

Indiferent de ce este j, această fracție este delimitată cu 1 / (r - 1). Astfel, prin alegând r suficient de mare, putem limita eroarea la orice desired> 0 dorit.

4.6.7 Extensii la numărarea celor

Este firesc să ne întrebăm dacă putem extinde tehnica acestei secțiuni la gestionați agregările mai generale decât numărarea 1 într-un flux binar. Un direcția evidentă de a privi este să luăm în considerare fluxurile de numere întregi și să întrebăm dacă putem estimează suma ultimelor k numere întregi pentru orice $1 \leq k \leq N$, unde N, ca de obicei, este dimensiunea ferestrei.

Este puțin probabil să putem folosi abordarea DGIM pentru fluxurile care conțin atât întregi pozitive, cât și negative. Am putea avea un flux care să le conțină pe amândouă numere întregi pozitive foarte mari și numere întregi negative foarte mari, dar cu o sumă în fereastra care este foarte aproape de 0. Orice imprecizie în estimarea valorilor de aceste numere întregi mari ar avea un efect uriaș asupra estimării sumei și deci eroarea fracționară ar putea fi nelimitată.

De exemplu, să presupunem că am rupt fluxul în găleți așa cum am făcut, dar a reprezentat galeata prin suma numerelor întregi, mai degrabă decât numărul din 1. Dacă b este bucketul care se află parțial în intervalul de interogare, ar putea fi acela b are, în prima sa jumătate, numere întregi negative foarte mari și în a doua jumătate, în mod egal numere întregi pozitive mari, cu o sumă de 0. Dacă estimăm contribuția lui b cu jumătate din suma sa, această contribuție este în esență 0. Dar contribuția efectivă a acea parte a cupei b care se află în intervalul de interogare ar putea fi de la 0 la suma tuturor numerelor întregi pozitive. Această diferență ar putea fi mult mai mare decât răspunsul real la interogare, astfel încât estimarea nu ar avea sens.

Pe de altă parte, funcționează și alte extensii care implică numere întregi. Cina-presupuneți că fluxul este format din numere întregi pozitive în intervalul $1 - 2^m$ pentru

---

unele m. Putem trata fiecare dintre m biții fiecărui număr întreg ca și cum ar fi un separat curent. Apoi folosim metoda DGIM pentru a număra 1 în fiecare bit. Presupune numărul bitului ith (presupunând că numărul de biți este de la capătul de ordine joasă, începând de la la 0) este c_i . Atunci suma numerelor întregi este

$$\sum_{i=0}^{m-1} c_i 2^i$$

Dacă folosim tehnica Secțiunii 4.6.6 pentru a estima fiecare c_i cu eroare fracționată cel mult ϱ, atunci estimarea sumei adevărate are eroare cel mult ϱ. Cel mai rau cazul apare atunci când toate c_i sunt supraestimate sau toate sunt subestimate de aceeași fracțiune.

4.6.8 Exerciții pentru secțiunea 4.6

Exercițiul 4.6.1: Să presupunem că fereastra este așa cum se arată în Fig. 4.2. Estimează numărul de 1 este ultimele k poziții, pentru k = (a) 5 (b) 15. În fiecare caz, cum departe de valoarea corectă este estimarea ta?

! Exercițiul 4.6.2: Există mai multe moduri în care fluxul de biți 1001011011101 ar putea fi împărțite în găleți. Găsește-le pe toate.

Exercițiul 4.6.3: Descrieți ce se întâmplă cu gălețile dacă mai intră încă 1 fereastra reprezentată de Fig. 4.3. Puteți presupune că niciunul dintre cele 1 nu este afișat pleacă de la fereastră.

4.7 Windows în descompunere

Am presupus că și o fereastră glisantă deține o anumită coadă a pârâului cele mai recente N elemente pentru N fix sau toate elementele care au ajuns după ceva timp în trecut. Uneori nu vrem să facem o distincție clară între elementele recente și cele din trecutul îndepărtat, dar doresc să pondereze elementele recente mai puternic. În această secțiune, considerăm „exponențial ferestre în descompunere ”și o aplicație în care sunt destul de utile: găsirea cele mai comune elemente „recente”.

4.7.1 Problema elementelor cele mai comune

Să presupunem că avem un flux ale cărui elemente sunt toate biletele de film cumpărate
peste tot în lume, cu numele filmului ca parte a elementului. Noi vrem
pentru a păstra un rezumat al fluxului care este cel mai popular film „în prezent”.
În timp ce noțiunea de „în prezent” este imprecisă, intuitiv, vrem să reducem
popularitatea unui film precum Star Wars – Episodul 4, care a vândut multe bilete,
dar cele mai multe dintre acestea au fost vândute cu zeci de ani în urmă. Pe de altă parte, un film care s-a vândut

---

Pagina 190

n biletele din fiecare din ultimele 10 săptămâni sunt probabil mai populare decât un film
care a vândut 2n bilete săptămâna trecută, dar nimic în săptămânile anterioare.
O soluție ar fi să ne imaginăm un flux mic pentru fiecare film. Cel de-al treilea bit
are valoarea 1 dacă biletul ith este pentru filmul respectiv și 0 în caz contrar. Alegeți o fereastră
mărimea N, care este numărul celor mai recente bilete care ar fi luate în considerare
în evaluarea popularității. Apoi, utilizați metoda secțiunii 4.6 pentru a estima
numărul de bilete pentru fiecare film și clasificați filmele după numărul lor estimat.
Această tehnică ar putea funcționa pentru filme, deoarece există doar mii de
filme, dar ar eșua dacă am înregistra în schimb popularitatea articolelor
vândut la Amazon sau rata la care difuzează diferiți utilizatori de Twitter, deoarece
există prea multe produse Amazon și prea multe tweetere. Mai mult decât atât
oferă răspunsuri aproximative.

4.7.2 Definiția ferestrei în descompunere

O abordare alternativă este redefinirea întrebării astfel încât să nu ne punem
pentru un număr de 1 într-o fereastră. Mai degrabă, să calculăm o agregare lină a
toate cele 1 văzute vreodată în flux, cu greutăți în descompunere, deci cu atât mai în spate
în flux, cu atât este mai mică greutatea. În mod formal, să permită un flux să conste în prezent
dintre elementele a_1, a_2, ..., a_t, unde a_1 este primul element care ajunge și a_t este
elementul actual. Fie c o mică constantă, cum ar fi 10^{-6} sau 10^{-9}. Defini
fereastra în descreștere exponențială pentru ca acest flux să fie suma

$$\sum_{i=0}^{t-1} a_{t-i}(1-c)^i$$

Efectul acestei definiții este de a răspândi greutățile fluxului
elemente la fel de îndepărtate în timp pe măsură ce curge curentul. În schimb, o fereastră fixă
cu aceeași sumă a greutăților, 1 / c, ar pune greutate egală 1 pe fiecare dintre
cele mai recente elemente 1 / c care ajung și cântăresc 0 pe toate elementele anterioare. The
distincția este sugerată de Fig. 4.4.

Fereastra de
lungime 1 / c

Figura 4.4: O fereastră în descompunere și o fereastră cu lungime fixă cu greutate egală

Este mult mai ușor să ajustați suma într-o fereastră exponențială în descompunere
decât într-o fereastră glisantă de lungime fixă. În fereastra glisantă, trebuie
vă faceți griji cu privire la elementul care cade din fereastră de fiecare dată când un element nou
ajunge. Asta ne obligă să păstrăm exact elementele împreună cu suma sau să le folosim

---

Pagina 191

o schemă de aproximare precum DGIM. Cu toate acestea, atunci când un element nou a_{t+1}
ajunge la intrarea fluxului, tot ce trebuie să facem este:

1. Înmulțiți suma curentă cu 1 - c.
2. Adăugați un $_{t+1}$.

Motivul pentru care funcționează această metodă este că fiecare dintre elementele anterioare le are acum
s-a deplasat o poziție mai departe de elementul curent, astfel încât greutatea sa este înmulțită
de 1 - c. Mai mult, greutatea elementului curent este $(1-c)^0 = 1$, deci adăugând
a_{t+1} este modul corect de a include contribuția noului element.

4.7.3 Găsirea celor mai populare elemente

Să ne întoarcem la problema găsirii celor mai populare filme într-un flux de
vânzări de bilete. [6] Vom folosi o fereastră exponențială în descompunere cu o constantă
c, la care s-ar putea crede că este 10^{-9}. Adică, aproximăm o fereastră glisantă
deținând ultimele miliarde de vânzări de bilete. Pentru fiecare film, ne imaginăm un separat
flux cu 1 de fiecare dată când un bilet pentru filmul respectiv apare în flux și un
0 de fiecare dată când ajunge un bilet pentru alt film. Suma în descompunere a lui 1
măsoară popularitatea actuală a filmului.

Ne imaginăm că numărul de filme posibile în flux este imens, așa că noi
nu vreau să înregistrați valori pentru filmele nepopulare. Prin urmare, stabilim
un prag, să zicem 1/2, astfel încât dacă scorul de popularitate pentru un film să fie mai mic decât acesta
numărul, scorul său este scăzut de la numărare. Din motive care vor deveni
evident, pragul trebuie să fie mai mic de 1, deși poate fi cu orice număr mai mic
decât 1. Când un nou bilet ajunge în flux, efectuați următoarele:
1. Pentru fiecare film al cărui scor îl menținem în prezent, înmulțiți-l
scor de (1 - c).
2. Să presupunem că noul bilet este pentru filmul M. Dacă există în prezent un scor pentru M,
adăugați 1 la acel scor. Dacă nu există scor pentru M, creați unul și inițializați-l
la 1.
3. Dacă vreun scor este sub pragul 1/2, renunțați la acel scor.
Este posibil să nu fie evident faptul că numărul de filme ale căror scoruri sunt principale
menținută în orice moment este limitată. Cu toate acestea, rețineți că suma tuturor scorurilor este de 1 / c.
Nu pot exista mai mult de filme 2 / c cu scor de 1/2 sau mai mult, altfel
suma scorurilor ar depăși 1 / c. Astfel, 2 / c este o limită a numărului de
filmele fiind numărate în orice moment. Desigur, în practică, vânzările de bilete ar fi
fi concentrat doar pe un număr mic de filme în orice moment, deci numărul
din filmele numărate activ ar fi mult mai mici de 2 / c.

6 Acest exemplu ar trebui luat cu un bob de sare, deoarece, așa cum am subliniat, acolo
nu sunt suficiente filme diferite pentru ca această tehnică să fie esențială. Imaginați-vă, dacă vreți, asta
numărul de filme este extrem de mare, deci nu contează vânzările de bilete pentru fiecare dintre ele separat
fezabil.

4.8 Rezumatul capitolului 4

♦ Modelul de date în flux: acest model presupune că datele ajung la o procesare
motor la o viteză care face imposibil de stocat totul în activ
depozitare. O strategie pentru gestionarea fluxurilor este menținerea rezumatelor
dintre fluxuri, suficient pentru a răspunde la întrebările așteptate despre date.
O a doua abordare este menținerea unei ferestre glisante a celei mai recente
date sosite.

♦ Eșantionarea fluxurilor: pentru a crea un eșantion dintr-un flux care este utilizabil pentru
o clasă de interogări, identificăm un set de atribute cheie pentru flux. De
bifând cheia oricărui element de flux care ajunge, putem folosi valoarea hash
pentru a decide în mod consecvent dacă toate sau niciunul dintre elementele cu cheia respectivă
va deveni parte a eșantionului.

♦ Filtre Bloom: Această tehnică ne permite să filtrăm fluxurile astfel încât elementele care
aparțin unui anumit set sunt permise, în timp ce majoritatea celor care nu sunt membri
sunt șterse. Folosim o matrice mare de biți și mai multe funcții hash. Membri
din setul selectat sunt hash la cupe, care sunt biți din matrice și
acei biți sunt setați la 1. Pentru a testa un element de flux pentru calitatea de membru, avem hash
elementul la un set de biți folosind fiecare dintre funcțiile hash și numai
acceptați elementul dacă toți acești biți sunt 1.

♦ Numărarea elementelor distincte: pentru a estima numărul de elemente diferite
apărând într-un flux, putem hash elemente la numere întregi, interpretate ca
numere binare. 2 ridicat la puterea care este cea mai lungă secvență de 0
văzut în valoarea hash a oricărui element de flux este o estimare a numărului
a diferitelor elemente. Prin utilizarea mai multor funcții hash și combinarea acestora
estimări, mai întâi luând medii în cadrul grupurilor și apoi luând
mediana mediilor, obținem o estimare fiabilă.

♦ Momente de fluxuri: al k-lea moment al unui flux este suma celui de-al zecelea
puterile numărărilor fiecărui element care apare cel puțin o dată în
curent. Momentul 0 este numărul de elemente distincte, iar primul
momentul este lungimea cursului.

♦ Estimarea celui de-al doilea moment: o estimare bună pentru al doilea moment sau
numărul surpriză, se obține alegând o poziție aleatorie în flux,
luând de două ori de câte ori apare acest element în flux
această poziție înainte, scăzând 1 și înmulțind cu lungimea lui
curentul. Multe variabile aleatorii de acest tip pot fi combinate ca
estimările pentru numărarea numărului de elemente distincte, pentru a produce un
estimare de încredere a celui de-al doilea moment.

♦ Estimarea momentelor superioare: tehnica pentru momentele secundare funcționează
și pentru momentele k, atâta timp cât înlocuim formula 2x - 1 (unde

x este de câte ori apare elementul la sau după selectat
poziție) cu $x_k - (x - 1)_k$.

Pagina 193

♦ Estimarea numărului de 1 într-o fereastră: Putem estima numărul de 1 într-o fereastră de 0 și 1 prin gruparea 1 în găleți. Fiecare cupa are un număr de 1, care este o putere de 2; sunt unul sau doi gălețile de fiecare dimensiune și dimensiunile nu scad niciodată pe măsură ce ne întoarcem în timp. Dacă înregistrăm doar poziția și dimensiunea găleților, putem reprezenta conținutul unei ferestre de dimensiunea N cu spațiul O ($\log_2 N$).

♦ Răspuns la întrebări despre numere de 1: Dacă vrem să știm aproximativ imitați numere de 1 în cele mai recente k elemente ale unui flux binar, găsim cea mai timpurie găleată B care se află cel puțin parțial în ultimul k pozițiile ferestrei și estimați numărul de 1 pentru a fi suma dimensiunile fiecărei găleți mai recente plus jumătate din mărimea lui B. Aceasta estimarea nu poate fi niciodată scăzută cu mai mult de 50% din numărul real de 1.

♦ Aproximări mai apropiate de numărul de 1: Prin schimbarea regulii pentru câte găleți de o dimensiune dată pot exista în reprezentarea unui fereastră binară, astfel încât să existe fie r, fie r - 1 de o dimensiune dată, putem asigurați-vă că aproximarea la numărul real de 1 nu este niciodată dezactivată de mai mult de 1 / r.

♦ Windows în descompunere exponențială: noi, mai degrabă decât să reparăm dimensiunea unei ferestre îmi pot imagina că fereastra constă din toate elementele care au ajuns vreodată în flux, dar cu elementul care a sosit cu unități de timp în urmă ponderate prin e^{-ct} pentru o constantă de timp c. Făcând acest lucru ne permite să ne menținem anumite rezumate ale unei ferestre exponențial descompuse cu ușurință. De exemplu, suma ponderată a elementelor poate fi recomputată, când ajunge un element nou, înmulțind vechea sumă cu 1 - c și apoi adăugând noul element.

♦ Menținerea elementelor frecvente într-o fereastră în descompunere exponențială: Ne putem imagina că fiecare element este reprezentat de un flux binar, unde 0 înseamnă că elementul nu a fost elementul care a ajuns la un moment dat și 1 înseamnă că a fost. Putem găsi elementele a căror sumă binară fluxul este de cel puțin 1/2. Când ajunge un element nou, înmulțiți toate înregistrările sumează cu 1 minus constanta de timp, adăugați 1 la numărul articolului care tocmai a sosit și ștergeți din înregistrare orice articol a cărui sumă a scăzut sub 1/2.

4.9 Referințe pentru capitolul 4

Multe idei asociate cu gestionarea fluxurilor apar în „date cronice model ”din [8]. O cercetare timpurie a cercetării în sistemele de gestionare a fluxurilor este [2]. De asemenea, [6] este o carte recentă despre subiectul gestionării fluxurilor. Tehnica de eșantionare din secțiunea 4.2 este de la [7]. Filtrul Bloom este atribuit în general lui [3], deși în esență a apărut aceeași tehnică ca

,,Coduri suprapuse” în [9].

Pagina 194

Algoritmul pentru numărarea elementelor distincte este în esență cel al lui [5], al-deși metoda specială pe care am descris-o apare în [1]. Acesta din urmă este, de asemenea sursa pentru algoritmul de calcul al numărului surpriză și mai mare momente. Cu toate acestea, tehnica de menținere a unei probe alese uniform de poziții în flux se numește „eșantionare rezervor” și provine din [10]. Tehnica pentru numărarea aproximativă a 1 într-o fereastră este de la [4].

1. N. Alon, Y. Matias și M. Szegedy, „Complexitatea spațială a aprox. momente de frecvență de împerechere ", al 28-lea Simpozion ACM despre teoria com-puting, pp. 20-29, 1996.
2. B. Babcock, S. Babu, M. Datar, R. Motwani și J. Widom, „Models și probleme în sistemele de flux de date ", Simpozionul pe principiile bazei de date Sisteme, pp. 1-16, 2002.
3. BH Bloom, „Compensări spațiu / timp în codificarea hash cu erori permise” Com. ACM 13: 7, pp. 422-426, 1970.
4. M. Datar, A. Gionis, P. Indyk și R. Motwani, „Maintaining stream

statistici asupra ferestrelor glisante ", SIAM J. Computing 31, pp. 1794–1813, 2002.
5. P. Flajolet și GN Martin, „Numărarea probabilistică pentru aplicațiile bazei de date ,,al 24-lea Simpozion privind fundamentele științei computerizate, pp. 76–82, 1983.
6. M. Garofalakis, J. Gehrke și R. Rastogi (editori), Data Stream Management, Springer, 2009.
7. PB Gibbons, „Eșantionare distinctă pentru răspunsuri extrem de precise la diferite interogări de valori și rapoarte de evenimente ", Intl. Conf. pe baze de date foarte mari, pp. 541-550, 2001.
8. HV Jagadish, IS Mumick și A. Silberschatz, „Vezi întreținerea probleme pentru modelul de date cronice, ”Proc. ACM Symp. pe Principiile de Sisteme de baze de date, pp. 113-124, 1995.
9. WH Kautz și RC Singleton, „Coduri binare neadaptative suprapuse” IEEE Transactions on Information Theory 10, pp. 363–377, 1964.
10. J. Vitter, „Eșantionare aleatorie cu un rezervor”, ACM Transactions on Software matematic 11: 1, pp. 37-57, 1985.

capitolul 5
Analiza legăturilor

Una dintre cele mai mari schimbări din viața noastră în deceniul următor secolul a fost disponibilitatea căutării eficiente și precise pe web, prin intermediul motoare de căutare precum Google. Deși Google nu a fost primul motor de căutare, acesta a fost a fost primul care a reușit să învingă spammerii care au făcut căutarea aproape inutilă. Mai mult, inovația oferită de Google a fost o tehnologie non-trivială avans, numit „PageRank”. Vom începe capitolul explicând ce PageRank este și cum este calculat eficient.

Cu toate acestea, războiul dintre cei care doresc să facă internetul util și cei cine l-ar exploata în scopuri proprii nu s-a terminat niciodată. Când a fost PageRank stabilit ca o tehnică esențială pentru un motor de căutare, au inventat spammerii modalități de a manipula PageRank-ul unei pagini Web, denumit adesea link spam. [1] Această dezvoltare a condus la răspunsul TrustRank și a altor tehnici pentru împiedicând spammerii să atace PageRank. Vom discuta despre TrustRank și alte abordări de detectare a spamului de link.

În cele din urmă, acest capitol acoperă, de asemenea, unele variații ale PageRank. Aceste tehnologii niques includ PageRank sensibil la subiect (care poate fi, de asemenea, adaptat pentru combaterea de spam de link) și abordarea HITS sau „hub-uri și autorități” pentru evaluare pagini de pe Web.

5.1 PageRank

Începem cu o parte din istoria motoarelor de căutare, pentru a motiva definiția PageRank [2] , un instrument pentru evaluarea importanței paginilor web într-un mod în care nu este ușor de păcălit. Introducem ideea de „surferi aleatori”, pentru a explica de ce este eficient PageRank. Apoi introducem tehnica „impozitului-acțiune ”sau reciclarea navigatorilor aleatori, pentru a evita anumite structuri web

[1] Link-ul spammers încearcă uneori să facă etica lor mai puțin evidentă făcând referire la ceea ce fac ca „optimizarea motorului de căutare”.

[2] Termenul PageRank provine de la Larry Page, inventatorul ideii și fondatorul Google.

care prezintă probleme pentru versiunea simplă a PageRank.

5.1.1 Motoare de căutare timpurie și spam de termeni

Înainte de Google existau multe motoare de căutare. În mare parte, au funcționat Internetul și enumerarea termenilor (cuvinte sau alte șiruri de caractere altele decât spațiul alb) găsit în fiecare pagină, într-un index inversat. Un index inversat este o structură de date care face mai ușor, dat un termen, să găsești (indicatori către) toate locurile în care apare termenul respectiv.

Când a fost emisă o interogare de căutare (lista termenilor), paginile cu acești termeni au fost extrase din indexul inversat și clasate într-un mod care reflectă

utilizarea termenilor din pagină. Astfel, prezența unui termen într-un antet al
pagina a făcut pagina mai relevantă decât ar fi prezența termenului în
textul obișnuit și un număr mare de apariții ale termenului s-ar adăuga la
relevanța asumată a paginii pentru interogarea de căutare.
Pe măsură ce oamenii au început să folosească motoarele de căutare pentru a-și găsi drumul pe Web,
oamenii lipsiți de etică au văzut ocazia de a păcăli motoarele de căutare în oameni de frunte
la pagina lor. Astfel, dacă ați vinde cămăși pe web, tot ce ținea la tine
a fost că oamenii îți vor vedea pagina, indiferent de ceea ce căutau.
Astfel, ați putea adăuga un termen ca „film” la pagina dvs. și să-l faceți cu mii de
de multe ori, astfel încât un motor de căutare ar crede că ești o pagină teribil de importantă
filme. Când un utilizator a emis o interogare de căutare cu termenul „film”, căutarea
motorul ar afișa pagina dvs. mai întâi. Pentru a preveni miile de apariții ale
,,Filmul” de la apariția pe pagina dvs., i-ați putea da aceeași culoare ca
fundal. Și dacă simpla adăugare a „filmului” la pagina dvs. nu a făcut trucul,
apoi puteți merge la motorul de căutare, dați-i interogarea „film” și vedeți ce
pagina a revenit ca prima alegere. Apoi, copiați acea pagină în a dvs.,
din nou folosind culoarea de fundal pentru ao face invizibilă.
Tehnici pentru a păcăli motoarele de căutare să creadă că pagina dvs. este despre
lucru care nu este, se numesc termen spam. Abilitatea de a opera termenii spammeri
atât de ușor de redat motoarele de căutare timpurii aproape inutile. Pentru a combate termenul spam,
Google a introdus două inovații:
1. PageRank a fost folosit pentru a simula unde navigatorii web, începând la întâmplare
pagină, ar tinde să se adune dacă ar urma link-uri alese aleatoriu
de pe pagina la care se aflau în prezent și acest proces a fost
permis să itereze de multe ori. Pagini care ar avea un număr mare de
surferii erau considerați mai „importanți” decât paginile care ar fi rareori
să fie vizitat. Google preferă paginile importante decât paginile neimportante atunci când
să decidem ce pagini să afișăm mai întâi ca răspuns la o interogare de căutare.
2. Conținutul unei pagini a fost judecat nu numai prin termenii care apar pe aceasta
pagină, dar prin termenii utilizați în sau lângă linkurile către pagina respectivă. Rețineți că
deși este ușor pentru un spammer să adauge termeni falși pe o pagină pe care o controlează,
nu pot obține la fel de ușor adăugarea de termeni falși pe paginile care leagă de ei
propria pagină, dacă nu controlează acele pagini.

PageRank simplificat nu funcționează

După cum vom vedea, calculul PageRank prin simularea surferilor aleatori este
un proces care consumă mult timp. S-ar putea crede că pur și simplu numărând
numărul de link-uri pentru fiecare pagină ar fi o bună aproximare a locului
surferii întâmplători s-ar termina. Cu toate acestea, dacă doar asta am făcut, atunci
ipoteticul vânzător de cămăși ar putea crea pur și simplu o „fermă de spam” de un milion
pagini, fiecare dintre ele legându-se la pagina cămășii sale. Apoi, pagina cămășii arată
foarte important și un motor de căutare ar fi păcălit.
Aceste două tehnici împreună o îngreunează foarte mult pentru cămașa ipotetică
vânzător pentru a păcăli Google. În timp ce vânzătorul de cămăși poate adăuga în continuare „film” la pagina sa,
faptul că Google a crezut ceea ce spun alte pagini despre el, peste ceea ce spune el
despre sine ar nega utilizarea termenilor falși. Contramăsura evidentă
este ca vânzătorul de cămăși să își creeze multe pagini și să se conecteze la cămașă
pagina de vânzare cu un link pe care scrie „film”. Dar acele pagini nu ar fi date
multă importanță de PageRank, deoarece alte pagini nu ar face legătura cu acestea. The
vânzătorul de cămăși ar putea crea multe linkuri între propriile sale pagini, dar niciuna dintre acestea
paginile ar avea multă importanță în conformitate cu algoritmul PageRank și
prin urmare, tot nu ar fi în stare să-l păcălească pe Google să creadă că pagina lui era
despre filme.
Este rezonabil să ne întrebăm de ce ar trebui să ne permită simularea surferilor aleatori
aproximează noțiunea intuitivă de „importanță” a paginilor. Sunt două
motivații conexe care au inspirat această abordare.

- Utilizatorii Web „votează cu picioarele”. Au tendința de a plasa linkuri către pagini
ei consideră că sunt pagini bune sau utile de privit, mai degrabă decât rele sau inutile
pagini.
- Comportamentul unui surfer aleatoriu indică ce pagini sunt utilizatorii de pe Web
sunt susceptibile de a vizita. Utilizatorii au mai multe șanse să viziteze pagini utile decât inutile
pagini.

Dar, indiferent de motiv, măsura PageRank a fost dovedită empiric
să funcționeze și așa vom studia în detaliu modul în care este calculat.

5.1.2 Definiția PageRank

PageRank este o funcție care atribuie un număr real fiecărei pagini de pe Web (sau cel puțin către acea porțiune a Web-ului care a fost accesat cu crawlere și la linkurile sale descoperit). Intenția este că cu cât este mai mare PageRank-ul unei pagini, cu atât mai mult ,,Important” este. Nu există un algoritm fix pentru atribuirea PageRank, și, de fapt, variațiile ideii de bază pot modifica PageRank-ul relativ al oricărui două pagini. Începem prin a defini PageRank-ul de bază, idealizat, și îl urmăm

---

Pagina 198

prin modificări care sunt necesare pentru a face față unor probleme din lumea reală referitoare la structura Web-ului.

Gândiți-vă la Web ca la un grafic direcționat, unde paginile sunt nodurile și acolo este un arc de la pagina p_1 la pagina p_2 dacă există unul sau mai multe linkuri de la p_1 la p_2. Figura 5.1 este un exemplu de versiune mică a Web-ului, unde există doar patru pagini. Pagina A are legături către fiecare dintre celelalte trei pagini; pagina B are legături către Numai A și D; pagina C are o legătură doar către A, iar pagina D are legături către B și C numai.

B
A
C
D

Figura 5.1: Un exemplu ipotetic al Web-ului

Să presupunem că un surfer aleator începe la pagina A din Fig. 5.1. Există legături către B, C și D, deci acest surfer va fi în continuare la fiecare dintre acele pagini cu probabilitate 1/3 și are zero probabilitate de a fi la A. Un surfer aleator la B are, la pasul următor, probabilitatea 1/2 de a fi la A, 1/2 de a fi la D și 0 de a fi la B sau C.

În general, putem defini matricea de tranziție a Webului pentru a descrie ce se întâmplă surferilor întâmplători după un pas. Această matrice M are n rânduri și coloane, dacă există n pagini. Elementul m_{ij} din rândul i și coloana j are valoare 1 / k dacă pagina j are k arcuri afară, iar una dintre ele este pagina i. În caz contrar, $m_{ij} = 0$.

Exemplul 5.1: Matricea de tranziție pentru rețeaua din Fig. 5.1 este

M =
□
□
0
1/2 1
0
1/3
0
0 1/2
1/3
0
0 1/2
1/3 1/2 0
0
□
□
□
□

În această matrice, ordinea paginilor este cea naturală, A, B, C și D. Astfel, prima coloană exprimă faptul, deja discutat, că un surfer la A are un 1/3 probabilitate ca următoarea să fie la fiecare dintre celelalte pagini. A doua coloană exprimă faptul că un surfer la B are 1/2 probabilitate de a fi următorul la A și același lucru cu a fi la D. Cea de-a treia coloană spune că un surfer la C este sigur că fii la A next. Ultima coloană spune că un surfer la D are o probabilitate de 1/2 de a fi următorul la B și la fel la C. ✷

Pagina 199

Distribuția probabilității pentru locația unui surfer aleator poate fi

descris de un vector coloană a cărui componentă j este probabilitatea ca
surfer este la pagina j. Această probabilitate este funcția PageRank (idealizată).
Să presupunem că începem un surfer aleatoriu la oricare dintre cele n pagini de pe Web cu
probabilitate egală. Apoi vectorul inițial v_0 va avea 1 / n pentru fiecare componentă.
Dacă M este matricea de tranziție a Web-ului, atunci după un pas, distribuția
al surferului va fi Mv_0, după doi pași va fi $M(Mv_0) = M^2 v_0$ și așa
pe. În general, înmulțind vectorul inițial v_0 cu M un total de i ori va fi
dați-ne distribuția surferului după i pași.
Pentru a vedea de ce multiplicarea unui vector de distribuție v cu M dă distribuția
x = Mv la pasul următor, motivăm după cum urmează. Probabilitatea x_i ca a
surferul aleatoriu va fi la nodul i la pasul următor, este $\Sigma_j m_{ij} v_j$. Aici, m_{ij} este
probabilitatea ca un surfer la nodul j să se deplaseze la nodul i la pasul următor (adesea
0 deoarece nu există nicio legătură de la j la i), iar v_j este probabilitatea ca surferul
a fost la nodul j la pasul anterior.
Acest tip de comportament este un exemplu al teoriei antice a proceselor Markov.
Se știe că distribuția surferului se apropie de o distribuție limitativă
v care satisface v = Mv, cu condiția să fie îndeplinite două condiții:
1. Graficul este puternic conectat; adică este posibil să obțineți de la oricare
nod către orice alt nod.
2. Nu există fundături: noduri care nu au arcuri afară.
Rețineți că Fig. 5.1 îndeplinește ambele condiții.
Limita este atinsă atunci când înmulțiți distribuția cu M altă dată
nu modifică distribuția. În alți termeni, v-ul limitativ este un eigenvec-
torul lui M (un vector propriu al unei matrice M este un vector v care satisface $v = \lambda Mv$ pentru
unele valori proprii constante λ). De fapt, pentru că M este stocastic, ceea ce înseamnă că este
fiecare coloană adaugă câte 1, v este vectorul propriu principal (propriul său asociat
valoarea este cea mai mare dintre toate valorile proprii). De asemenea, rețineți că, deoarece M este stocastic,
valoarea proprie asociată cu vectorul propriu principal este 1.
Vectorul propriu principal al lui M ne spune unde este cel mai probabil surferul
fie după mult timp. Amintiți-vă că intuiția din spatele PageRank este că
cu cât este mai probabil ca un surfer să fie la o pagină, cu atât este mai importantă pagina. Noi
poate calcula vectorul propriu principal al lui M începând cu vectorul inițial
v_0 și înmulțind cu M de câteva ori, până când vectorul pe care îl obținem arată
mică schimbare la fiecare rundă. În practică, pentru Web în sine, 50-75 de iterații sunt
suficient pentru a converge în limitele de eroare ale aritmeticii cu precizie dublă.
Exemplul 5.2: Să presupunem că aplicăm procesul descris mai sus matricei
M din exemplul 5.1. Deoarece există patru noduri, vectorul inițial v_0 are patru
componente, fiecare 1/4. Secvența de aproximări la limita pe care noi

---

Pagina 200

Rezolvarea ecuațiilor liniare

Dacă vă uitați la „Web” cu 4 noduri din Exemplul 5.2, ați putea crede că
modalitatea de a rezolva ecuația v = Mv este prin eliminare gaussiană. Intr-adevar,
în acest exemplu, am argumentat care ar fi limita, în esență, făcând
asa de. Cu toate acestea, în exemple realiste, unde există zeci sau sute de
miliarde de noduri, eliminarea Gaussiană nu este fezabilă. Motivul este că
Eliminarea gaussiană necesită timp cubic în numărul de ecuații.
Astfel, singura modalitate de a rezolva ecuații pe această scară este să iterați ca și noi
au sugerat. Chiar și această iterație este pătratică la fiecare rundă, dar noi
o poate accelera profitând de faptul că matricea M este foarte
rar; există în medie aproximativ zece linkuri pe pagină, adică zece diferite de zero
intrări pe coloană.
Mai mult, există o altă diferență între calculul PageRank
și rezolvarea ecuațiilor liniare. Ecuația v = Mv are un număr infinit
de soluții, deoarece putem lua orice soluție v, înmulțim componentele sale cu
orice constantă fixă c și obțineți o altă soluție la aceeași ecuație. Cand
includem constrângerea că suma componentelor este 1, așa cum am avut
gata, atunci vom obține o soluție unică.
obține înmulțind la fiecare pas cu M este:

□

□

□

□

1/4
1/4

1/4
1/4
□
□
□
□
,
□
□
24 septembrie
24.05
24.05
24.05
□
□
□
□
,
□
□
15/48
48.11.11
48.11.11
48.11.11
□
□
□
□
,
□
□
11/32
32.07
32.07
32.07
□
□
□
□
, ...,
□
□
3/9
2/9
2/9
2/9
□
□
□
□
Observați că în acest exemplu, probabilitățile pentru B, C și D rămân
la fel. Este ușor de văzut că B și C trebuie să aibă întotdeauna aceleași valori
iterație, deoarece rândurile lor din M sunt identice. Pentru a arăta că valorile lor sunt
De asemenea, la fel ca valoarea pentru D, o dovadă inductivă funcționează și o lăsăm ca
un exercitiu. Având în vedere că ultimele trei valori ale vectorului limitativ trebuie să fie
la fel, este ușor să descoperiți limita secvenței de mai sus. Primul rând al
M ne spune că probabilitatea lui A trebuie să fie 3/2 celelalte probabilități, deci
limita are probabilitatea A egală cu 3/9 sau 1/3, în timp ce probabilitatea pentru
alte trei noduri sunt 2/9.
Această diferență de probabilitate nu este mare. Dar în web-ul real, cu miliarde
de noduri de importanță foarte variabilă, adevărata probabilitate de a fi la un nod

ca www.amazon.com este ordinele de mărime mai mari decât probabilitatea de
noduri tipice. ✷

Pagina 201

5.1.3 Structura Web-ului

Ar fi frumos dacă Web-ul ar fi conectat puternic, precum Fig. 5.1. Cu toate acestea, ea nu este, în practică. Un studiu timpuriu al Web-ului a constatat că are structura prezentat în Fig. 5.2. A existat o componentă mare puternic conectată (SCC), dar au existat alte câteva porţii care erau aproape la fel de mari.

1. Componenta integrată, formată din pagini care ar putea ajunge la SCC după link-uri lowing, dar nu au putut fi accesate de la SCC.
2. Componenta exterioară, formată din pagini accesibile din SCC, dar neadecvate capabil să ajungă la CSC.
3. Varcuri, care sunt de două tipuri. Unele grinzi constau din pagini accesibile de la componentă, dar nu este capabil să ajungă la componentă. The alte fluturi pot ajunge la componenta exterioară, dar nu sunt accesibile componenta exterioară.

Componenta
Tuburi
Puternic
Conectat
Componenta
În
Componenta
Afară
Afară
Varcuri
În
Varcuri
Deconectat
Componente

Figura 5.2: Imaginea „papion” a Web-ului

În plus, au existat un număr mic de pagini găsite fie în

Pagina 202

(a) Tuburi, care sunt pagini accesibile din componentă și care pot ajunge componenta exterioară, dar incapabilă să ajungă la SCC sau să poată fi atinsă de la SCC.
(b) Componente izolate care nu pot fi atinse de componentele mari (SCC, componente de intrare și de ieșire) și nu pot atinge aceste componente.

Mai multe dintre aceste structuri încalcă ipotezele necesare pentru iterația procesului pentru a converge la o limită. De exemplu, atunci când un surfer aleatoriu intră în componenta exterioară, nu pot pleca niciodată. Ca urmare, surferii încep fie în SCC, fie în componentă, se vor termina fie în exterior componentă sau o tendrilă în afara componentei. Astfel, nicio pagină din SCC sau componenta se termină cu orice probabilitate ca un surfer să fie acolo. Dacă interpretăm această probabilitate ca măsurare a importanței unei pagini, apoi concluzionăm în mod fals că nimic din SCC sau din componentă nu are nicio importanță.

Ca rezultat, PageRank este de obicei modificat pentru a preveni astfel de anomalii. Acolo sunt într-adevăr două probleme pe care trebuie să le evităm. În primul rând este impasul, o pagină care nu are linkuri. Surferii care ajung la o astfel de pagină dispar, iar rezultatul este acela în limită, nicio pagină care poate ajunge la un punct mort nu poate avea deloc niciun PageRank.

A doua problemă este grupurile de pagini care au toate outlinkuri, dar nu au niciodată link către orice alte pagini. Aceste structuri se numesc capcane pentru păianjeni. [3] Ambele acestea problemele sunt rezolvate printr-o metodă numită „impozitare”, unde presupunem o întâmplare surferul are o probabilitate finită de a părăsi Web-ul la orice pas și noi surferi sunt începute la fiecare pagină. Vom ilustra acest proces pe măsură ce studiem fiecare cele două cazuri problematice.

5.1.4 Evitarea impasurilor

Amintiți-vă că o pagină fără niciun link se numește un punct mort. Dacă permitem morții se termină, matricea de tranziție a Webului nu mai este stocastică, deoarece unele dintre coloanele vor însuma 0 mai degrabă decât 1. O matrice ale cărei sume de coloane sunt la majoritatea 1 se numește substochastic. Dacă calculăm $M^i v$ pentru puteri crescânde ale lui a

matricea substocastică M, apoi unele sau toate componentele vectorului merg la 0. Adică, importanța „se scurge” de pe web și nu obținem informații despre importanța relativă a paginilor.

Exemplul 5.3: În Fig. 5.3 am modificat Fig. 5.1 prin eliminarea arcului din C la A. Astfel, C devine o fundătură. În ceea ce privește surferii aleatori, când un surfer ajunge la C, ei dispar la următoarea rundă. Matricea M care descrie Fig. 5.3 este

M =

□

□

0

1/2 0

0

1/3

0

0 1/2

1/3

0

0 1/2

1/3 1/2 0

0

□

□

□

□

[3] Acestea se numesc așa deoarece programele care accesează cu crawlere pe Web, înregistrarea paginilor și a linkurilor, sunt adesea denumiți „păianjeni”. Odată ce un păianjen intră într-o capcană pentru păianjen, nu poate pleca niciodată.

B

A

C

D

Figura 5.3: C este acum o fundătură

Rețineți că este substocastic, dar nu stocastic, deoarece suma celui de-al treilea coloana, pentru C, este 0, nu 1. Iată secvența vectorilor care rezultă prin pornire cu vectorul cu fiecare componentă 1/4 și înmulțind în mod repetat vectorul de M:

□

□

□

□

1/4

1/4

1/4

1/4

□

□

□

□

,

□

□

3/24

24.05

24.05

24.05

□

□

□

□
,
□
□
5/48
7/48
7/48
7/48
□
□
□
□
,
□
□
21/288
31/288
31/288
31/288
□
□
□
□
, ...,
□
□
0
0
0
0
□
□
□
□
După cum vedem, probabilitatea ca un surfer să fie oriunde merge la 0, ca număr
de trepte cresc. ✷
Există două abordări pentru a face față fundurilor.
1. Putem scoate fundăturile din grafic și, de asemenea, arunca intrările lor
arcuri. Dacă faceți acest lucru, se pot crea mai multe fundături, care trebuie, de asemenea, abandonate,
recursiv. Cu toate acestea, în cele din urmă, ajungem la o conexiune puternică
componentă, nici unul dintre ale cărui noduri nu au fundătură. În ceea ce privește Fig. 5.2,
ștergerea recursivă a fundurilor va elimina părțile componentei exterioare,
vânturi și tuburi, dar părăsesc SCC și componenta integrală, precum și
părți ale oricăror componente mici izolate. [4]
2. Putem modifica procesul prin care se presupune că se deplasează surferii aleatori
despre Web. Această metodă, la care ne referim ca „impozitare”, se rezolvă și ea
problema capcanelor păianjen, așa că o vom amâna la Secțiunea 5.1.5.
Dacă folosim prima abordare, ștergerea recursivă a fundurilor, atunci rezolvăm
graficul rămas G prin orice mijloace sunt adecvate, inclusiv impozitul
metoda dacă ar putea exista capcane de păianjen în G. Apoi, restaurăm graficul, dar păstrăm
[4] S-ar putea să presupuneți că întreaga componentă exterioară și toate gârlele vor fi eliminate, dar
amintiți-vă că pot avea în interior componente mai mici puternic conectate, inclusiv
capcane pentru păianjen, care nu pot fi șterse.

---

valorile PageRank pentru nodurile lui G. Noduri nu în G, ci cu predecesori
toți cei din G pot avea PageRank-ul lor calculat prin însumare, peste toți predecesorii
p, PageRank-ul lui p împărțit la numărul succesorilor lui p din graficul complet.
Acum pot exista și alte noduri, nu în G, care au PageRank-ul tuturor
predecesorii au calculat. Acestea pot avea propriul PageRank calculat de
același proces. În cele din urmă, toate nodurile din afara G vor avea PageRank-ul lor

calculat; ele pot fi cu siguranță calculate în ordinea opusă celei în care
au fost șterse.
B
A
C
D
E
Figura 5.4: Un grafic cu două niveluri de fundături
Exemplul 5.4: Figura 5.4 este o variație din Fig. 5.3, unde am introdus
un succesor E pentru C. Dar E este o fundătură, iar când îl eliminăm, și
intrând din C, descoperim că C este acum o fundătură. După îndepărtarea C, nu
mai multe noduri pot fi eliminate, deoarece fiecare dintre A, B și D au arcuri care pleacă. The
graficul rezultat este prezentat în Fig. 5.5.
Matricea pentru graficul din Fig. 5.5 este
M =
□
0
1/2 0
1/2
0
1
1/2 1/2 0
□
□
Rândurile și coloanele corespund A, B și D în această ordine. Pentru a obține
PageRanks pentru această matrice, începem cu un vector cu toate componentele egale
la 1/3 și înmulțit în mod repetat cu M. Succesiunea vectorilor pe care o obținem este
□
□
1/3
1/3
1/3
□
□
,
□
1/6
3/6
2/6
□
□
,
□
3/12
5/12
4/12
□
□
,
□
24.05
24.11
24.08
□
□
, ...,
□
2/9
4/9
3/9
□
□
Acum știm că PageRank-ul lui A este 2/9, PageRank-ul lui B este 4/9,
iar PageRank-ul lui D este 3/9. Mai trebuie să calculăm PageRanks pentru C

B
A
D

Figura 5.5: Graficul redus fără fundături

și E, și o facem în ordinea opusă celei în care au fost șterse.

De când C a fost șters ultima dată, știm că toți predecesorii săi au PageRanks calculat. Acești predecesori sunt A și D. În Fig. 5.4, A are trei succesori, deci contribuie cu 1/3 din PageRank la C. Pagina D are doi succesori în Fig. 5.4, deci contribuie cu jumătate din PageRank la C. Astfel, PageRank-ul lui C este

$$\frac{1}{3} \times \frac{2}{9} + \frac{1}{2} \times \frac{3}{9} = 13/54.$$

Acum putem calcula PageRank pentru E. Acest nod are o singură predecesor, C și C are un singur succesor. Astfel, PageRank-ul lui E este la fel ca cea a lui C. Rețineți că sumele din PageRanks depășesc 1, iar ele nu mai reprezintă distribuția unui surfer aleatoriu. Cu toate acestea, ele reprezintă estimări decente ale importanței relative a paginilor. ✷

5.1.5 Capcane pentru păianjen și impozitare

După cum am menționat, o capcană de păianjen este un set de noduri fără fundături, dar fără arcuri afară. Aceste structuri pot apărea intenționat sau neintenționat pe web, și fac ca calculul PageRank să plaseze tot PageRank în capcane păianjen.

Exemplul 5.5: Luați în considerare figura 5.6, care este figura 5.1 cu arcul în afara lui C schimbat pentru a indica C însuși. Această schimbare face din C o simplă capcană de păianjen nodul. Rețineți că, în general, capcanele pentru păianjen pot avea multe noduri, și așa cum vom avea vezi în secțiunea 5.4, există capcane pentru păianjen cu milioane de noduri pe care spammerii construiți intenționat.

Matricea de tranziție pentru Fig. 5.6 este

M =

□

□

0
1/2 0
0
1/3
0
0 1/2
1/3
0
1 1/2
1/3 1/2 0
0

□

□

□

□

Dacă efectuăm iterația obișnuită pentru a calcula PageRank-ul nodurilor, noi

B
A
C
D

Figura 5.6: Un grafic cu o capcană de păianjen cu un singur nod

obține

□
□
□
□
1/4
1/4
1/4
1/4
□
□
□
□
,
□
□
3/24
24.05
24.11
24.05
□
□
□
□
,
□
□
5/48
7/48
29/48
7/48
□
□
□
□
,
□
□
21/288
31/288
205/288
31/288
□
□
□
□
, ...,
□
□
0
0
1
0
□
□
□
□
Aşa cum s-a prezis, tot PageRank-ul se află la C, întrucât o dată acolo există un surfer aleatoriu nu pleca niciodata. ✷
Pentru a evita problema ilustrată în Exemplul 5.5, modificăm calculul

din PageRank permițând fiecărui surfer aleatoriu o mică probabilitate de teleportare
la o pagină aleatorie, mai degrabă decât să urmeze un out-link de la pagina lor curentă.
Pasul iterativ, în care calculăm o nouă estimare vectorială a PageRanks v ′
din estimarea PageRank actuală v și matricea de tranziție M este
v ′ = βMv + (1 - β)e / n
unde βeste o constantă aleasă, de obicei în intervalul 0,8 - 0,9, e este un vector al tuturor
1 are numărul corespunzător de componente, iar n este numărul de noduri
în graficul web. Termenul βMv reprezintă cazul în care, cu probabilitate
β,surferul aleator decide să urmeze un out-link din pagina lor actuală. The
termenul (1 - β)e / n este un vector fiecare dintre ale cărui componente are valoare (1 - β)/ n și
reprezintă introducerea, cu probabilitatea 1 - β,a unui nou surfer aleator la
o pagină aleatorie.
Rețineți că, dacă graficul nu are fundături, atunci probabilitatea de a introduce a
noul surfer aleator este exact egal cu probabilitatea ca surferul aleator să o facă
decide să nu urmeze un link din pagina lor curentă. În acest caz, este rezonabil
pentru a vizualiza surferul ca hotărând fie să urmeze un link, fie să se teleporteze la o întâmplare
pagină. Cu toate acestea, dacă există funduri, există o a treia posibilitate, care
este că surferul nu merge nicăieri. Deoarece termenul (1 - β)e / n nu depinde de
suma componentelor vectorului v, va exista întotdeauna o fracțiune

---

Pagina 207

a unui surfer care operează pe web. Adică, atunci când există fundaturi, suma
dintre componentele lui v poate fi mai mică de 1, dar nu va ajunge niciodată la 0.
Exemplul 5.6: Să vedem modul în care noua abordare a calculului PageRank
tarifele pe graficul din Fig. 5.6. Vom folosi β= 0,8 în acest exemplu. Astfel,
ecuația pentru iterație devine
v ′ =
□
□
0
2/5
0
0
15.04
0
0
2/5
15.04
0
4/5 2/5
15.04.2 / 5
0
0
□
□
□
□
v +
□
□
1/20
1/20
1/20
1/20
□
□
□
□
Observați că am încorporat factorul βîn M prin înmulțirea fiecăruia dintre
elementele sale de 4/5. Componentele vectorului (1 - β)e / n sunt fiecare 1/20,
din moment ce 1 - β= 1/5 și n = 4. Iată primele iterații:
□

□
□
□
1/4
1/4
1/4
1/4
□
□
□
□
,
□
□
9/60
13/60
25/60
13/60
□
□
□
□
,
□
□
41/300
53/300
153/300
53/300
□
□
□
□
,
□
□
543/4500
707/4500
2543/4500
707/4500
□
□
□
□
, ...,
□
□
15/148
19/148
95/148
19/148
□
□
□
□
Fiind o capcană pentru păianjen, C a reușit să obțină mai mult de jumătate din PageRank pentru sine. Cu toate acestea, efectul a fost limitat și fiecare dintre noduri obține unele din PageRank. ✷

5.1.6 Utilizarea PageRank într-un motor de căutare

După ce am văzut cum se calculează vectorul PageRank pentru porțiunea Web
că un motor de căutare a accesat cu crawlere, ar trebui să examinăm cum sunt aceste informații
folosit. Fiecare motor de căutare are o formulă secretă care decide ordinea în care
pentru a afișa pagini utilizatorului ca răspuns la o interogare de căutare constând dintr-una sau
mai mulți termeni de căutare (cuvinte). Se spune că Google folosește peste 250 de proprietăți diferite
de pagini, din care se decide o ordine liniară a paginilor.
În primul rând, pentru a fi luată în considerare pentru clasament, o pagină trebuie să aibă la
cel puțin unul dintre termenii de căutare din interogare. În mod normal, ponderarea proprietăților
este astfel încât, dacă nu sunt prezenți toți termenii de căutare, o pagină are foarte puține șanse
de a fi în top zece care sunt afișate în mod normal utilizatorului. Printre
pagini calificate, se calculează un scor pentru fiecare și o componentă importantă a
acest scor este PageRank-ul paginii. Alte componente includ prezența
sau absența termenilor de căutare în locuri proeminente, cum ar fi anteturile sau linkurile către
pagina în sine.

5.1.7 Exerciții pentru secțiunea 5.1

Exercițiul 5.1.1: Calculați PageRank-ul fiecărei pagini din Fig. 5.7, presupunând
fără impozitare.

Pagina 208

188
CAPITOLUL 5. ANALIZA LEGĂTURILOR
A
b
c
Figura 5.7: Un exemplu de grafic pentru exerciții
Exercițiul 5.1.2: Calculați PageRank-ul fiecărei pagini din Fig. 5.7, presupunând
β= 0,8.
! Exercițiul 5.1.3: Să presupunem că Web-ul este format dintr-o clică (set de noduri cu toate
arce posibile de la unul la altul) de n noduri și un singur nod suplimentar care
este succesorul fiecăruia dintre nodurile din clică. Figura 5.8 prezintă acest grafic
pentru cazul n = 4. Determinați PageRank-ul fiecărei pagini, în funcție de n
și β.
Figura 5.8: Exemplu de grafice discutate în exercițiul 5.1.3
!! Exercițiul 5.1.4: Construiți, pentru orice număr întreg n, un web astfel încât, în funcție de
β,oricare dintre cele n noduri poate avea cel mai mare PageRank dintre cele n. Este
a permis să existe și alte noduri pe Web în afară de aceste n.
! Exercițiul 5.1.5: Arătați prin inducție pe n că dacă al doilea, al treilea și al patrulea
componentele unui vector v sunt egale, iar M este matricea de tranziție a examenului-
p. 5.1, apoi a doua, a treia și a patra componentă sunt, de asemenea, egale în $M^n v$
pentru orice $n \geq 0$.

Pagina 209

5.2. CALCULUL EFICIENT AL PAGERANK
189
. . .
Figura 5.9: Un lanț de fundături
Exercițiul 5.1.6: Să presupunem că eliminăm recursiv fundăturile din grafic,
rezolvați graficul rămas și estimați PageRank-ul pentru paginile fără fund
așa cum este descris în secțiunea 5.1.4. Să presupunem că graficul este un lanț de fundături, cu capul
de un nod cu o buclă de sine, așa cum este sugerat în Fig. 5.9. Care ar fi pagina-
Rangul atribuit fiecăruia dintre noduri?
Exercițiul 5.1.7: Repetați exercițiul 5.1.6 pentru arborele fundurilor sugerate de
Fig. 5.10. Adică, există un singur nod cu o buclă de sine, care este și rădăcina
a unui arbore binar complet de n niveluri.
. . .
. . .
. . .
. . .
Figura 5.10: Un copac de fundături

5.2 Calcul eficient al PageRank

Pentru a calcula PageRank pentru un grafic mare reprezentând Web, avem
pentru a efectua o multiplicare matrice-vector de ordinul a 50 de ori, până la
vectorul este aproape neschimbat la o singură iterație. La o primă aproximare,
Metoda MapReduce dată în secțiunea 2.3.1 este potrivită. Cu toate acestea, trebuie să ne ocupăm
cu două probleme:
1. Matricea de tranziție a Web M este foarte rară. Astfel, reprezentând

prin toate elementele sale este extrem de ineficient. Mai degrabă vrem să reprezentăm
matricea prin elementele sale nenule.
2. Este posibil să nu folosim MapReduce sau, din motive de eficiență, ne-am dori
pentru a utiliza un combinator (a se vedea secțiunea 2.2.4) cu sarcinile Hărți pentru a reduce
cantitatea de date care trebuie transmise de la sarcini de hartă pentru a reduce sarcinile. În
în acest caz, abordarea de striping discutată în secțiunea 2.3.1 nu este suficientă
pentru a evita utilizarea grea a discului (thrashing).
Discutăm soluția la aceste două probleme în această secțiune.

Pagina 210

5.2.1 Reprezentarea matricilor de tranziție

Matricea de tranziție este foarte redusă, deoarece pagina web medie are aproximativ 10
link-uri externe. Dacă, să zicem, analizăm un grafic de zece miliarde de pagini, atunci doar unul
într-un miliard de intrări nu este 0. Modul corect de reprezentare a oricărei matrici rare este
pentru a lista locațiile intrărilor diferite de zero și valorile acestora. Dacă folosim 4 octeți
numere întregi pentru coordonatele unui element și un număr de 8 octeți cu precizie dublă
pentru valoare, atunci avem nevoie de 16 octeți pe intrare diferită de zero. Adică spațiul
necesar este liniar în numărul de intrări diferite de zero, mai degrabă decât pătratic în
latura matricei.
Cu toate acestea, pentru o matrice de tranziție a Web-ului, există o altă compresie
că putem face. Dacă enumerăm intrările diferite de zero după coloană, atunci știm ce
fiecare intrare diferită de zero este; este împărțit la gradul exterior al paginii. Putem
reprezintă astfel o coloană cu un întreg pentru gradul exterior și un întreg
pe intrare diferită de zero din acea coloană, indicând numărul rândului unde se află acea intrare
este localizat. Astfel, avem nevoie de puțin mai mult de 4 octeți pe intrare diferită de zero
reprezintă o matrice de tranziție.
Exemplul 5.7: Să reluăm exemplul graficului web din Fig. 5.1, a cărui
matricea de tranziție este
M =
□
□
0
1/2 1
0
1/3
0
0 1/2
1/3
0
0 1/2
1/3 1/2 0
0
□
□
□
□
Amintiți-vă că rândurile și coloanele reprezintă nodurile A, B, C și D, în acel
Ordin. În Fig. 5.11 este o reprezentare compactă a acestei matrice. 5
Destinații ale gradului sursă
A
3
B, C, D
B
2
ANUNȚ
C
1
A
D
2
B, C
Figura 5.11: Reprezentați o matrice de tranziție după gradul exterior al fiecărui nod și
lista succesorilor săi
De exemplu, intrarea pentru A are gradul 3 și o listă de trei succesori.

Din acel rând din Fig. 5.11 putem deduce că coloana pentru A din matricea M
are 0 în rând pentru A (deoarece nu este pe lista destinațiilor) și 1/3 în
rândurile pentru B, C și D. Știm că valoarea este 1/3 deoarece gradul
coloana din Fig. 5.11 ne spune că există trei legături din A. ✷

[5] Deoarece M nu este rar, această reprezentare nu este foarte utilă pentru M. Cu toate acestea, Exemplul ilustrează procesul de reprezentare a matricelor în general, iar matricea este mai rară este, cu atât această reprezentare va economisi mai mult.

Pagina 211

5.2.2 Iterarea PageRank folosind MapReduce

O iterație a algoritmului PageRank implică luarea unei estimări de pagină
Clasează vectorul v și calculează următoarea estimare v′ cu
$v' = \beta Mv + (1 - \beta)e / n$
Reamintim β este o constantă puțin mai mică de 1, e este un vector al tuturor 1 și n este
numărul de noduri din graficul pe care îl reprezintă matricea de tranziție M.
Dacă n este suficient de mic încât fiecare sarcină de hartă să poată stoca întregul vector v în main
memorie și, de asemenea, au loc în memoria principală pentru vectorul de rezultat v′, apoi acolo
este puțin mai mult aici decât o multiplicare matrice-vector. Pașii suplimentari
sunt să înmulțească fiecare componentă a Mv cu constanta β și să adauge $(1 - \beta)/ n$ la
fiecare componentă.
Cu toate acestea, este probabil, având în vedere dimensiunea Web-ului de astăzi, că v este prea mult
mare pentru a se potrivi în memoria principală. Așa cum am discutat în secțiunea 2.3.1, metoda
de dungi, unde rupem M în dungi verticale (vezi Fig. 2.4) și rupem v
în dungi orizontale corespunzătoare, ne va permite să executăm MapReduce
procesați eficient, fără mai mult de v la o sarcină de hartă decât poate convenabil
încap în memoria principală.

5.2.3 Utilizarea combinatorilor pentru a consolida vectorul de rezultate

Există două motive pentru care metoda secțiunii 5.2.2 ar putea să nu fie adecvată.
1. S-ar putea să dorim să adăugăm termeni pentru v'_i, a i-a componentă a vectorului de rezultat
v, la sarcinile Hartă. Această îmbunătățire este aceeași cu utilizarea unui combinator,
deoarece funcția Reduce adaugă pur și simplu termeni cu o cheie comună. Reamintim
că pentru o implementare MapReduce a multiplicării matrice-vector,
cheia este valoarea lui i pentru care este destinat un termen $m_{ij} v_j$.
2. S-ar putea să nu folosim deloc MapReduce, ci mai degrabă să executăm iter-
pas de acțiune la o singură mașină sau o colecție de mașini.
Vom presupune că încercăm să implementăm un combinator împreună
cu o sarcină de hartă; al doilea caz folosește în esență aceeași idee.
Să presupunem că folosim metoda stripe pentru a partiționa o matrice și
vector care nu se încadrează în memoria principală. Apoi o bandă verticală din matrice
M și o bandă orizontală din vectorul v vor contribui la toate componentele
a vectorului de rezultat v′. Deoarece acel vector are aceeași lungime ca v, nu va avea
încap în memoria principală fie. Mai mult, deoarece M este stocat coloană cu coloană pentru
Din motive de eficiență, o coloană poate afecta oricare dintre componentele lui v′. Ca rezultat,
este puțin probabil ca atunci când trebuie să adăugăm un termen la o componentă v'_{eu}, asta
componenta va fi deja în memoria principală. Astfel, majoritatea termenilor vor necesita
ca o pagină să fie adusă în memoria principală pentru ao adăuga la componenta corespunzătoare.
Această situație, numită thrashing, necesită ordine de mărime prea mult timp
fezabil.

Pagina 212

O strategie alternativă se bazează pe partiționarea matricei în k^2 blocuri,
în timp ce vectorii sunt încă partiționați în k dungi. O imagine, care arată
diviziunea pentru k = 4, este în Fig. 5.12. Rețineți că nu am arătat multiplica-
ziunea matricei prin β sau adaosul de $(1 - \beta)e / n$, deoarece acești pași sunt
simplă, indiferent de strategia pe care o folosim.

v
v
v
v
4
1
2

3
v
v
v
v
4
1
2
3
M
M
M
M
M
M
M
M
M
M
M
M
M
M
11
M
12
13
14
21
22
23
24
31
32
33
34
41
42
43
'
'
'
'
44
=

Figura 5.12: Partiționarea unei matrice în blocuri pătrate

În această metodă, folosim sarcini k 2 Map. Fiecare sarcină primește un pătrat din matricea M, să spunem M_{ij} și o bandă a vectorului v, care trebuie să fie v_j . Înștiințare că fiecare bandă a vectorului este trimisă la k sarcini de hartă diferite; v_j este trimis la gestionarea sarcinii M_{ij} pentru fiecare dintre cele k valori posibile ale lui i. Astfel, v este transmis prin rețea de k ori. Cu toate acestea, fiecare bucată din matrice este trimisă o singură dată. Deoarece dimensiunea matricei, codificată corect așa cum este descris în secțiunea 5.2.1, poate este de așteptat să fie de câteva ori mai mare decât vectorul, costul transmisiei este nu mult mai mare decât minimul posibil. Și pentru că facem asta combinând considerabil la sarcinile Hărții, salvăm pe măsură ce datele sunt transmise din Asociați sarcinile la sarcinile Reduceți.

Avantajul acestei abordări este că putem păstra ambele benzi a v și banda i de v' în memoria principală pe măsură ce procesăm M_{ij} . Rețineți că toate termenii generați din M_{ij} și v_j contribuie la v'_i și nici o altă bandă de v'.

5.2.4 Reprezentarea blocurilor matricei de tranziție

Deoarece reprezentăm matricile de tranziție în modul special descris în Secțiunea 5.2.1, trebuie să luăm în considerare modul în care sunt reprezentate blocurile din Fig. 5.12. Din păcate, spațiul necesar pentru o coloană de blocuri (o „bandă” așa cum am numit-o mai devreme) este mai mare decât spațiul necesar benzii în ansamblu, dar nu prea mult mai mare.

Pentru fiecare bloc, avem nevoie de date despre toate acele coloane care au cel puțin una intrare diferită de zero în cadrul blocului. Dacă k, numărul de dungi din fiecare dimensiune, este mare, atunci majoritatea coloanelor nu vor avea nimic în majoritatea blocurilor din banda sa. Pentru un bloc dat, nu trebuie doar să listăm acele rânduri care au o intrare diferită de zero coloana respectivă, dar trebuie să repetăm gradul de ieșire pentru nodul reprezentat de coloana. În consecință, este posibil ca gradul extern să se repete ca de multe ori ca și gradul în sine. Această observație se limitează de deasupra

spațiul necesar pentru a stoca blocurile unei benzi la dublul spațiului necesar pentru a stoca banda în ansamblu.
ABCD
A
B
C
D
Figura 5.13: Un grafic cu patru noduri este împărțit în patru blocuri 2-pe-2
Exemplul 5.8: Să presupunem că matricea din Exemplul 5.7 este partiționată în blocuri, cu k = 2. Adică, cadranul din stânga sus reprezintă legături de la A sau De la B la A sau B, cadranul din dreapta sus reprezintă legături de la C sau D la A sau B, și așa mai departe. Se pare că în acest mic exemplu, singura intrare pe care noi poate evita este intrarea pentru C în M_{22} , deoarece C nu are arcuri nici la C, nici la D. Tabelele care reprezintă fiecare dintre cele patru blocuri sunt prezentate în Fig. 5.14. Dacă examinăm Fig. 5.14 (a), vedem reprezentarea quad-stânga sus declama. Observați că gradele pentru A și B sunt aceleași ca în Fig. 5.11, deoarece trebuie să cunoaștem întregul număr de succesori, nu numărul succesorilor în cadrul blocului relevant. Cu toate acestea, fiecare succesor al lui A sau B este reprezentat în Fig. 5.14 (a) sau Fig. 5.14 (c), dar nu ambele. Observați, de asemenea, că în Fig. 5.14 (d), nu există nicio intrare pentru C, deoarece nu există succesori ai lui C în partea inferioară jumătate din matrice (rândurile C și D). ✷

5.2.5 Alte abordări eficiente ale iterației PageRank

Algoritmul discutat în secțiunea 5.2.3 nu este singura opțiune. Vom discuta alte câteva abordări care utilizează mai puține procesoare. Acești algoritmi partajează cu algoritmul din Secțiunea 5.2.3 proprietatea bună că matricea M este doar citită o dată, deși vectorul v este citit de k ori, unde se alege parametrul k astfel încât 1 / kth din vectorii v și v' poate fi ținut în memoria principală. Reamintim că algoritmul din Secțiunea 5.2.3 folosește k^2 procesoare, presupunând că toate sarcinile Hărții sunt executat în paralel la diferite procesoare.
Putem atribui toate blocurile dintr-un rând de blocuri unei singure sarcini de hartă și reduceți astfel numărul de sarcini de hartă la k. De exemplu, în Fig. 5.12, M_{11} , M_{12} , M_{13} și M_{14} vor fi atribuite unei singure sarcini de hartă. Dacă reprezentăm blocuri ca în Fig. 5.14, putem citi blocurile dintr-un rând de blocuri unul câte unul, deci matricea nu consumă o cantitate semnificativă de memorie principală. La în același timp în care citim M_{ij} , trebuie să citim banda vectorială v_j . Ca rezultat, fiecare dintre sarcinile k Hartă citește întregul vector v, împreună cu 1 / kth din matrice.

Destinații ale gradului sursă
A
3
B
B
2
A
(a) Reprezentarea lui M_{11} conectând A și B la A și B
Destinații ale gradului sursă
C
1
A
D
2
B
(b) Reprezentarea lui M_{12} conectând C și D la A și B
Destinații ale gradului sursă
A
3
C, D
B
2
D
(c) Reprezentarea lui M_{21} conectând A și B la C și D

Destinații ale gradului sursă
D
2
C
(d) Reprezentarea lui M_{22} conectând C și D la C și D
Figura 5.14: Reprezentare rară a blocurilor unei matrice
Lucrarea care citește M și v este deci aceeași ca și pentru algoritmul Sec-
5.2.3, dar avantajul acestei abordări este că fiecare sarcină de hartă se poate combina
toți termenii pentru porțiunea v'_i pentru care este exclusiv responsabil. În altele
cuvintele, sarcinile de reducere nu au altceva de făcut decât să concateneze bucățile de v'
primit de la k sarcini Harta.
Putem extinde această idee la un mediu în care MapReduce nu este utilizat.
Să presupunem că avem un singur procesor, cu M și v stocate pe discul său, folosind
aceeași reprezentare rară pentru M pe care am discutat-o. Mai întâi putem simula
prima sarcină Map, cea care folosește blocurile M_{11} până la M_{1k} și toate de la v la
calculează v'_1. Apoi simulăm cea de-a doua sarcină a hărții, citind M_{21} până la M_{2k}
și tot din v pentru a calcula v'_2 și așa mai departe. În ceea ce privește algoritmii anteriori, noi astfel
citiți M o dată și vk ori. Putem face k cât mai mic posibil, sub rezerva
constrângere că există suficientă memorie principală pentru a stoca 1 / kth din v și 1 / kth din
v', împreună cu o porțiune cât mai mică de M pe care o putem citi de pe disc (de obicei, una
bloc de disc).

5.2.6 Exerciții pentru secțiunea 5.2

Exercițiul 5.2.1: Să presupunem că dorim să stocăm o matrice booleană n × n (0 și
1 numai elemente). L-am putea reprezenta prin biți înșiși sau am putea
reprezintă matricea prin listarea pozițiilor 1 ca perechi de numere întregi, fiecare
întreg care necesită $\lceil \log_2 n \rceil$ biți. Primul este potrivit pentru matrici dense; the
acesta din urmă este potrivit pentru matrici rare. Cât de rar trebuie să fie matricea (adică ce
fracțiunea din elemente ar trebui să fie 1) pentru salvarea reprezentării rare
spațiu?
Exercițiul 5.2.2: Folosind metoda secțiunii 5.2.1, reprezentați tranziția
matrici ale următoarelor grafice:
(a) Figura 5.4.
(b) Figura 5.7.
Exercițiul 5.2.3: Folosind metoda secțiunii 5.2.4, reprezentați tranziția
matricile grafului din Fig. 5.3, presupunând că blocurile au latura 2.
Exercițiul 5.2.4: Luați în considerare un grafic web care este un lanț, cum ar fi Fig. 5.9, cu
n noduri. Ca o funcție a lui k, pe care o puteți presupune că împarte n, descrieți
reprezentarea matricei de tranziție pentru acest grafic, folosind metoda
Secțiunea 5.2.4

5.3 PageRank sensibil la subiect

Există mai multe îmbunătățiri pe care le putem aduce la PageRank. Unul, de studiat
în această secțiune, este că putem cântări mai mult anumite pagini din cauza lor
subiect. Mecanismul pentru aplicarea acestei ponderări este de a modifica modul aleatoriu
surferii se comportă, făcându-i să prefere să aterizeze pe o pagină despre care se știe că acoperă
subiect ales. În secțiunea următoare, vom vedea cum poate ideea sensibilă la subiect
de asemenea, pentru a anula efectele unui nou tip de spam, numit „'link spam”,
care sa dezvoltat pentru a încerca să păcălească algoritmul PageRank.

5.3.1 Motivația pentru Page Rank-ul sensibil la subiect

Oamenii diferiți au interese diferite și, uneori, interese diferite sunt
exprimat folosind același termen într-o interogare. Exemplul canonic este căutarea
interogare jaguar, care s-ar putea referi la animal, automobil, o versiune a
Sistem de operare MAC sau chiar o consolă de jocuri antică. Dacă un motor de căutare
poate deduce că utilizatorul este interesat de automobile, de exemplu, atunci poate
faceți o treabă mai bună de a returna paginile relevante utilizatorului.
În mod ideal, fiecare utilizator ar avea un vector PageRank privat care oferă
importanța fiecărei pagini pentru utilizatorul respectiv. Nu este fezabil să stocați un vector de
lungime de multe miliarde pentru fiecare dintre un miliard de utilizatori, deci trebuie să facem ceva

---

mai simplu. Abordarea PageRank sensibilă la subiect creează un vector pentru fiecare dintre ele un număr mic de subiecte, influențând PageRank pentru a favoriza paginile acelui subiect. Ne străduim apoi să clasificăm utilizatorii în funcție de gradul de interes al acestora fiecare dintre subiectele selectate. Deși cu siguranță pierdem o anumită acuratețe, beneficiul este că stocăm doar un vector scurt pentru fiecare utilizator, mai degrabă decât un vector enorm pentru fiecare utilizator.

Exemplul 5.9: Un set de subiecte utile îl constituie cele 16 categorii de nivel superior (sport, medicină) icine etc.) din Open Directory (DMOZ). [6]

Am putea crea 16 PageRank vectori, câte unul pentru fiecare subiect. Dacă am putea stabili dacă utilizatorul este interesat într-unul dintre aceste subiecte, poate prin conținutul paginilor pe care le au recent vizualizat, atunci am putea folosi vectorul PageRank pentru subiectul respectiv atunci când decidem pe clasamentul paginilor. ✷

5.3.2 Plimbări aleatorii părtinitoare

Să presupunem că am identificat câteva pagini care reprezintă un subiect precum „sportul”. Pentru a crea un PageRank sensibil la subiect pentru sport, putem aranja acest lucru aleatoriu surferii sunt introduși doar într-o pagină sportivă aleatorie, mai degrabă decât într-o întâmplare pagină de orice fel. Consecința acestei alegeri este că sunt surferi aleatorii probabil să se afle pe o pagină de sport identificată sau pe o pagină accesibilă pe o cale scurtă dintr-una dintre aceste pagini de sport cunoscute. Intuiția noastră este că paginile legate de după paginile de sport sunt ele însele susceptibile să fie despre sport. Paginile pe care le leagă să fie, de asemenea, probabil să fie despre sport, deși probabilitatea de a fi despre sportul scade cu siguranță pe măsură ce crește distanța de la o pagină de sport identificată.

Formularea matematică pentru iterația care dă sensibilitate la subiect PageRank este similar cu ecuația pe care am folosit-o pentru PageRank general. Singurul diferența este modul în care adăugăm noii surferi. Să presupunem că S este un set de numere întregi constând din numerele de rând / coloană pentru paginile pe care le-am identificat ca aparținând unui anumit subiect (numit set de teleport). Fie e s un vector care are 1 în componente în S și 0 în alte componente. Apoi, pagina sensibilă la subiect Rangul pentru S este limita iterației

$v' = \beta M v + (1 - \beta) e_S / |S|$

Aici, ca de obicei, M este matricea de tranziție a Webului și | S | este dimensiunea setului S.

Exemplul 5.10: Să reconsiderăm graficul web original pe care l-am folosit în Fig. 5.1, pe care le reproducem ca Fig. 5.15. Să presupunem că folosim β= 0,8. Apoi tranziția matricea pentru acest grafic, înmulțită cu β,este

βM=

□

□

0
2/5 4/5
0
15.04
0
0
2/5
15.04
0
0
2/5
15.04.2 / 5
0
0

□

□

□

□

[6] Acest director, aflat la www.dmoz.org, este o colecție de pagini web clasificate de oameni.

B

A
C
D
Figura 5.15: Repetați un exemplu de grafic web
Să presupunem că tema noastră este reprezentată de setul de teleportare S = {B, D}. Apoi vectorul (1 - β)e s / | S | are 1/10 pentru a doua și a patra componentă și 0 pentru celelalte două componente. Motivul este că 1 - β= 1/5, dimensiunea lui S este 2 și e s are 1 în componentele pentru B și D și 0 în componentele pentru A și C. Astfel, ecuația care trebuie iterată este
v ′ =
□
□
0
2/5 4/5
0
15.04
0
0
2/5
15.04
0
0
2/5
15.04.2 / 5
0
0
□
□
□
□
v +
□
□
0
1/10
0
1/10
□
□
□
□
Iată primele iterații ale acestei ecuații. Am început și cu surferii doar la paginile din setul de teleportare. Deși distribuția inițială nu are niciun efect asupra limitei, poate ajuta calculul să convergă mai repede.
□
□
□
□
0/2
1/2
0/2
1/2
□
□
□
□
,
□
□
2/10
3/10
2/10
3/10

□
□
□
□
,
□
□
42/150
41/150
26/150
41/150
□
□
□
□
,
□
□
62/250
71/250
46/250
71/250
□
□
□
□
, ...,
□
□
54/210
59/210
38/210
59/210
□
□
□
□
Observați că datorită concentrației de surferi la B și D, aceste noduri se obțin
un PageRank mai mare decât în Exemplul 5.2. În acel exemplu, A a fost
nodul celui mai înalt PageRank. ✷

5.3.3 Utilizarea PageRank sensibil la subiect

Pentru a integra PageRank sensibil la subiect într-un motor de căutare, trebuie:
1. Decideți subiectele pentru care vom crea vecini specializate PageRank
tori.
2. Alegeți un set de teleport pentru fiecare dintre aceste subiecte și utilizați setul respectiv pentru a calcula
vectorul PageRank sensibil la subiect pentru subiectul respectiv.

---

3. Găsiți o modalitate de a determina subiectul sau setul de subiecte care sunt cele mai relevante
pentru o anumită interogare de căutare.
4. Folosiți vectorii PageRank pentru subiectul sau subiectele respective în ordonarea
răspunsuri la interogarea de căutare.
Am menționat un mod de a selecta setul de subiecte: utilizați subiectele de nivel superior
din Open Directory. Sunt posibile și alte abordări, dar probabil că există o
nevoie de clasificare umană a cel puțin unor pagini.
Al treilea pas este probabil cel mai dificil și au existat mai multe metode
propus. Câteva posibilități:
(a) Permiteți utilizatorului să selecteze un subiect dintr-un meniu.
(b) deduceți subiectul (subiectele) prin cuvintele care apar recent în paginile web

căutat de utilizator sau interogări recente emise de utilizator. Avem nevoie să
discutăm despre cum se trece de la o colecție de cuvinte la un subiect și vom face asta
faceți acest lucru în secțiunea 5.3.4
(c) deduceți subiectul (subiectele) prin informații despre utilizator, de exemplu, marcajele acestuia sau
interesele lor declarate pe Facebook.

5.3.4 deducerea subiectelor din cuvinte

Problema clasificării documentelor pe subiecte este un subiect care a fost studiat
timp de decenii și nu vom intra în detalii mari aici. Este suficient să spunem asta
subiectele sunt caracterizate de cuvinte care apar surprinzător de des în documente
pe acest subiect. De exemplu, nici fullback, nici rujeola nu apar foarte des în
documente de pe web. Dar fundașul va apărea mult mai des decât media
în paginile despre sport, iar rujeola va apărea mult mai des decât media în
pagini despre medicină.
Dacă examinăm întregul web sau un eșantion mare și aleatoriu de web, noi
poate obține frecvența de fundal a fiecărui cuvânt. Să presupunem că mergem apoi la un mare
eșantion de pagini despre care se știe despre un anumit subiect, spun paginile clasificate
sub sport de Open Directory. Examinați frecvențele cuvintelor din
proba sportivă și identifică cuvintele care apar semnificativ mai frecvent
în proba sportivă decât în fundal. În a face această judecată, noi
trebuie să fie atent pentru a evita un cuvânt extrem de rar care apare în sport
eșantion cu frecvență relativ mai mare. Acest cuvânt este probabil o greșeală de ortografie
care s-a întâmplat să apară doar într-una sau câteva dintre paginile de sport. Astfel, noi
probabil că doriți să puneți un etaj pe de câte ori apare un cuvânt, înainte de acesta
poate fi considerat caracteristic unui subiect.
Odată ce am identificat o mare colecție de cuvinte care apar mult mai mult
frecvent în eșantionul de sport decât în fundal și facem același lucru
pentru toate subiectele de pe lista noastră, putem examina alte pagini și le putem clasifica după
subiect. Iată o abordare simplă. Să presupunem că S_1, S_2, ..., S_k sunt mulțimile de
cuvinte care au fost determinate ca fiind caracteristice fiecăruia dintre subiectele de pe

---

lista noastră. Fie P setul de cuvinte care apar într-o pagină dată P. Calculați
asemănarea Jaccard (reamintim secțiunea 3.1.1) dintre P și fiecare dintre S_i.
Clasificați pagina ca subiect cu cea mai mare similaritate Jaccard. Rețineți că
toate asemănările Jaccard pot fi foarte mici, mai ales dacă dimensiunile seturilor S_i sunt
mic. Astfel, este important să alegeți seturi destul de mari S_i pentru a vă asigura că
acoperim toate aspectele subiectului reprezentat de set.
Putem folosi această metodă sau o serie de variante pentru a clasifica paginile
utilizatorul a recuperat cel mai recent. Am putea spune că utilizatorul este interesat de subiect
în care se încadrează cel mai mare număr din aceste pagini. Sau am putea amesteca subiectul-
vectori PageRank sensibili proporțional cu fracțiunea acestor pagini care
cad în fiecare subiect, construind astfel un singur vector PageRank care reflectă
combinația actuală de interese a utilizatorului. De asemenea, am putea folosi aceeași procedură pe
paginile pe care utilizatorul le are în prezent marcate sau combină marcajele
pagini cu paginile vizualizate recent.

5.3.5 Exerciții pentru secțiunea 5.3

Exercițiul 5.3.1: Calculați PageRank sensibil la subiect pentru graficul din Fig.
5.15, presupunând că setul de teleportare este:
(a) Numai A.
(b) A și C.

5.4 Link spam

Când a devenit evident că PageRank și alte tehnici utilizate de Google
a făcut ca termenul de spam să fie ineficient, spammerii au apelat la metode concepute pentru a păcăli
algoritmul PageRank în supraevaluarea anumitor pagini. Tehnicile pentru
creșterea artificială a PageRank-ului unei pagini se numește colectiv spam spam.
În această secțiune, vom examina mai întâi modul în care spammerii creează linkuri spam și
apoi vedeți mai multe metode pentru scăderea eficacității acestor spam
tehnici, inclusiv TrustRank și măsurarea masei de spam.

5.4.1 Arhitectura unei ferme de spam

O colecție de pagini al căror scop este de a crește PageRank-ul unei anumite
pagina sau paginile se numește o fermă de spam. Figura 5.16 prezintă cea mai simplă formă de
ferma de spam. Din punctul de vedere al spamului, Web-ul este împărțit în
trei părți:
1. Pagini inaccesibile: paginile pe care spamul nu le poate afecta. Majoritatea
Web-ul se află în această parte.

2. Pagini accesibile: acele pagini care, deși nu sunt controlate de
spammer, poate fi afectat de spam.

Pagina 220

3. Pagini proprii: paginile pe care spamul le deține și le controlează.
Accesibil
Pagini
Proprie
Pagini
Inaccesibil
Pagini
Țintă
Pagină
Figura 5.16: Web-ul din punctul de vedere al link-ului spammer
Farmul de spam conține pagini proprii ale spamului, organizate într-un special
așa cum se vede în partea dreaptă, și câteva link-uri din paginile accesibile către
paginile spammerului. Fără unele legături din exterior, ferma de spam ar fi
fi inutil, deoarece nici măcar nu ar fi accesat cu crawlere de un motor de căutare tipic.
În ceea ce privește paginile accesibile, ar putea părea surprinzător faptul că se poate
fect o pagină fără a o deține. Cu toate acestea, astăzi există multe site-uri, cum ar fi
bloguri sau ziare care îi invită pe alții să-și posteze comentariile pe site. În
pentru a obține cât mai mult PageRank care curge către propriile sale pagini din exterior,
spammerul postează multe comentarii precum „Sunt de acord. Vă rugăm să consultați articolul meu la
www.mySpamFarm.com. ”
În ferma de spam, există o pagină t, pagina țintă, la care spammerul
încearcă să plaseze cât mai mult PageRank. Există un număr mare
m de pagini suport, care acumulează porțiunea de PageRank care este
distribuite în mod egal tuturor paginilor (fracțiunea 1 - βa PageRank-ului care reprezintă
trimite surferii care merg la o pagină aleatorie). Paginile de sprijin previn, de asemenea
PageRank de la pierderea, în măsura posibilului, deoarece unele vor fi impozitate
la fiecare rundă. Observați că t are un link către fiecare pagină de sprijin și
fiecare pagină de sprijin se leagă numai către t.

Pagina 221

5.4.2 Analiza unei ferme de spam
Să presupunem că PageRank este calculat folosind un parametru de impozitare β,de obicei
în jur de 0,85. Adică, βeste fracțiunea din PageRank-ul unei pagini care se dezvăluie
omagiat succesorilor săi la următoarea rundă. Să existe n pagini pe Web
în total, și să fie unii dintre ei o fermă de spam de forma sugerată în Fig. 5.16,
cu o pagină țintă t și m care susțin paginile. Fie x suma de PageRank
contribuit de paginile accesibile. Adică, x este suma, peste toate accesibile
paginile p cu un link către t, din PageRank de p ori β,împărțit la număr
de succesori ai p. În cele din urmă, să fie y PageRank necunoscut al lui t. Vom rezolva
pentru tine.
În primul rând, PageRank-ul fiecărei pagini suport este
$\beta y/ m + (1 - \beta)/ n$
Primul termen reprezintă contribuția din t. PageRank y de t este impozitat,
deci numai βyeste distribuit succesorilor lui t. Că PageRank este împărțit în mod egal
printre m paginile de sprijin. Al doilea termen este cota paginii de sprijin
din fracțiunea 1 - βa PageRank care este împărțită în mod egal între toate paginile
pe internet.
Acum, să calculăm PageRank y al paginii țintă t. PageRank-ul său vine
din trei surse:
1. Contribuția x din exterior, așa cum am presupus.
2. βori PageRank-ul fiecărei pagini suport; acesta este,
$\beta(\beta y / m + (1 - \beta)/ n)$
3. $(1 - \beta)/ n$, ponderea fracției $1 - \beta$din PageRank care aparține
t. Această sumă este neglijabilă și va fi abandonată pentru a simplifica analiza.
Astfel, din (1) și (2) de mai sus, putem scrie
$y = x + \beta m(\beta ym +$
$1 - \beta$
$n)$
$= x + \beta^2 y + \beta(1 - \beta)$

m
n
Putem rezolva ecuația de mai sus pentru y, cedând
y =
X
1 - β_2
+ c
m
n
unde c = $\beta(1 - \beta)/(1 - \beta_2) = \beta/(1 + \beta)$.
Exemplul 5.11: Dacă alegem β= 0,85, atunci 1 / (1 - β_2) = 3,6 și c =
$\beta/(1 + \beta)$= 0,46. Adică structura a amplificat PageRank-ul extern
contribuție cu 360% și, de asemenea, a obținut o sumă de PageRank care este de 46%
din fracțiunea Web, m / n, care se află în ferma de spam. ✷

Pagina 222

5.4.3 Combaterea spamului prin link

A devenit esențial ca motoarele de căutare să detecteze și să elimine spamul pe linkuri,
la fel cum a fost necesar în deceniul anterior pentru a elimina termenul spam. Acolo
sunt două abordări pentru a lega spamul. Una este să căutați structuri precum
5.16, unde o pagină se leagă de un număr foarte mare de pagini,
fiecare dintre ele se leagă înapoi de el. Motoarele de căutare caută cu siguranță astfel de structuri
și eliminați acele pagini din index. Acest lucru determină dezvoltarea de spammeri
structuri diferite care au în esență același efect al capturării PageRank
pentru o pagină sau pagini țintă. În esență, nu există sfârșit al variațiilor din Fig. 5.16,
deci acest război dintre spammeri și motoarele de căutare va continua probabil
o perioadă lungă de timp.
Cu toate acestea, există o altă abordare a eliminării spamului de linkuri care nu există
mizați-vă pe localizarea fermelor de spam. Mai degrabă, un motor de căutare își poate modifica definițiile
PageRank pentru a reduce automat rangul paginilor de link-spam. Vom
ia în considerare două formule diferite:
1. TrustRank, o variantă a PageRank sensibil la subiect, concepută pentru a reduce nivelul
scor de pagini spam.
2. Masa de spam, un calcul care identifică paginile care ar putea fi
spam și permite motorului de căutare să elimine acele pagini sau să coboare
PageRank-ul lor puternic.

5.4.4 TrustRank

TrustRank este PageRank sensibil la subiect, unde „topic” este un set de pagini
cred că sunt de încredere (nu spam). Teoria este că în timp ce o pagină de spam
se poate face cu ușurință un link către o pagină de încredere, este puțin probabil ca o
o pagină demnă ar fi legată de o pagină de spam. Zona limită este un site cu
bloguri sau alte oportunități pentru spammeri de a crea linkuri, așa cum sa discutat în
Secțiunea 5.4.1. Aceste pagini nu pot fi considerate demne de încredere, chiar dacă sunt proprii
conținutul este extrem de fiabil, așa cum ar fi cazul unui ziar de renume care
le-a permis cititorilor să posteze comentarii.
Pentru a implementa TrustRank, trebuie să dezvoltăm un set adecvat de teleportare a
pagini de încredere. Două abordări care au fost încercate sunt:
1. Lăsați oamenii să examineze un set de pagini și să decidă care dintre ele sunt de încredere
vrednic. De exemplu, am putea alege paginile cu cel mai înalt nivel de PageRank
examinați, pe baza teoriei că, în timp ce spam-ul link-ului poate ridica rangul unei pagini de la
de jos până la mijlocul ambalajului, este în esență imposibil de dat
o pagină spam un PageRank aproape de partea de sus a listei.
2. Alegeți un domeniu al cărui apartenență este controlată, presupunând că acesta
este greu pentru un spammer să-și introducă paginile în aceste domenii. De exemplu,
am putea alege domeniul .edu, deoarece este puțin probabil ca paginile universității să fie
ferme de spam. De asemenea, am putea alege .mil sau .gov. Cu toate acestea, problema

---

Pagina 223

cu aceste alegeri specifice este că acestea sunt aproape exclusiv site-uri din SUA. La
obțineți o distribuție bună a paginilor Web de încredere, ar trebui să includem
site-uri similare din țări străine, de exemplu, ac.il sau edu.sg.
Este probabil ca motoarele de căutare să implementeze astăzi o strategie de al doilea tip

în mod obișnuit, astfel încât ceea ce credem că este PageRank într-adevăr este o formă de TrustRank.

5.4.5 Masa spam

Ideea din spatele masei de spam este că măsurăm pentru fiecare pagină fracțiunea acesteia PageRank care provine din spam. O facem calculând atât obișnuitul PageRank și TrustRank se bazează pe un set de teleporturi de pagini de încredere. Să presupunem că pagina p are PageRank r și TrustRank t. Apoi, masa spam a lui p este (r - t) / r. O masă de spam pozitivă negativă sau mică înseamnă că p nu este probabil o pagină de spam, în timp ce o masă de spam apropiată de 1 sugerează că pagina este probabil spam. Este posibil să eliminați paginile cu o masă mare de spam din index de pagini Web utilizate de un motor de căutare, eliminând astfel o mare parte din link spam fără a fi nevoie să identifice anumite structuri pe care agricultorii de spam le utilizează.

Exemplul 5.12: Să luăm în considerare atât PageRank, cât și Page-topic Rang care a fost calculat pentru graficul din Fig. 5.1 din Exemplele 5.2 și 5.10, respectiv. În acest din urmă caz, setul de teleportare a fost nodurile B și D, așa că presupunem că acestea sunt paginile de încredere. Figura 5.17 tabelează PageRank, TrustRank și masa spam pentru fiecare dintre cele patru noduri.

Nod PageRank TrustRank Spam Mass
A
3/9
54/210
0,229
B
2/9
59/210
-0.264
C
2/9
38/210
0,186
D
2/9
59/210
-0.264

Figura 5.17: Calculul masei de spam

În acest exemplu simplu, singura concluzie este că nodurile B și D, care au fost a priori determinate să nu fie spam, au masă de spam negativă și sunt deci nu spam. Celelalte două noduri, A și C, au fiecare un spam pozitiv masă, deoarece PageRanks lor sunt mai mari decât TrustRanks. De exemplu, masa de spam a lui A se calculează luând diferența 3/9 - 54/210 = 8/105 și împărțind 8/105 la PageRank 3/9 pentru a obține 8/35 sau aproximativ 0,229. In orice caz, masa lor de spam este încă mai aproape de 0 decât de 1, deci este probabil să nu fie spam. ✷

5.4.6 Exerciții pentru secțiunea 5.4

Exercițiul 5.4.1: În secțiunea 5.4.2 am analizat ferma de spam din Fig. 5.16, unde fiecare pagină de sprijin se leagă înapoi la pagina țintă. Repetați analiza pentru a

---

fermă de spam în care:
(a) Fiecare pagină de sprijin se leagă de sine în loc de pagina țintă.
(b) Fiecare pagină de sprijin nu leagă nicăieri.
(c) Fiecare pagină de sprijin se leagă atât la ea însăși, cât și la pagina țintă.

Exercițiul 5.4.2: Pentru graficul web original din Fig. 5.1, presupunând că numai B este a pagina de încredere:
(a) Calculați TrustRank-ul fiecărei pagini.
(b) Calculați masa de spam a fiecărei pagini.

! Exercițiul 5.4.3: Să presupunem că doi fermieri de spam sunt de acord să-și conecteze fermele de spam. Cum ați conecta paginile pentru a crește cât mai mult posibil PageRank al paginii țintă a fiecărei ferme de spam? Există un avantaj în legătură ferme de spam?

5.5 Huburi și autorități

O idee numită „hub-uri și autorități” a fost propusă la scurt timp după ce a fost PageRank prima implementare. Algoritmul pentru hub-urile de calcul și autoritățile poartă o oarecare asemănare cu calculul PageRank, deoarece se ocupă și de calcul iterativ al unui punct fix care implică matrice repetată - vector multiplicatie. Cu toate acestea, există și diferențe semnificative între cele două idei,

și nici unul nu-l poate substitui pe celălalt.
Acest algoritm hub-și-autorități, numit uneori HITS (hyperlink-
căutare subiect indusă), a fost inițial concepută nu ca un pas de preprocesare anterior
gestionarea interogărilor de căutare, așa cum este PageRank, dar ca un pas de făcut împreună
procesarea unei interogări de căutare, pentru a clasifica doar răspunsurile la această interogare. Noi
va descrie, totuși, ca o tehnică de analiză a întregului web sau a
porțiune accesată cu crawlere de un motor de căutare. Există motive să credem că ceva
ca această abordare este, de fapt, utilizată de motorul de căutare Ask.

5.5.1 Intuitia din spatele HITS

În timp ce PageRank își asumă o noțiune unidimensională de importanță pentru pagini,
HITS consideră că paginile importante au două arome de importanță.
1. Anumite pagini sunt valoroase deoarece oferă informații despre un
subiect. Aceste pagini se numesc autorități.
2. Alte pagini sunt valoroase nu pentru că oferă informații despre oricare
subiect, ci pentru că îți spun unde să mergi pentru a afla despre subiectul respectiv.
Aceste pagini se numesc hub-uri.

Pagina 225

Exemplul 5.13: Un departament tipic la o universitate menține o pagină web
listând toate cursurile oferite de departament, cu link-uri către o pagină pentru fiecare
curs, povestind despre curs - instructorul, textul, o schiță a
conținutul cursului și așa mai departe. Dacă vrei să afli despre un anumit curs, tu
nevoie de pagina pentru cursul respectiv; lista cursurilor departamentale nu va funcționa. The
pagina cursului este o autoritate pentru cursul respectiv. Cu toate acestea, dacă doriți să aflați
ce cursuri oferă departamentul, nu este util să le căutați pe fiecare
pagina cursurilor; mai întâi aveți nevoie de pagina cu lista cursurilor. Această pagină este un hub
pentru informații despre cursuri. ✷
La fel cum PageRank folosește definiția recursivă a importanței că „o pagină
este important dacă paginile importante se leagă de acesta ", HITS folosește un recursiv reciproc
definirea a două concepte: „o pagină este un hub bun dacă se leagă de autorități bune,
iar o pagină este o autoritate bună dacă este legată de hub-uri bune. ”

5.5.2 Formalizarea interesului și autorității

Pentru a oficializa intuiția de mai sus, vom atribui două scoruri fiecărei pagini web.
Un scor reprezintă interesul unei pagini - adică gradul în care aceasta
este un hub bun, iar al doilea scor reprezintă gradul în care pagina
este o bună autoritate. Presupunând că paginile sunt enumerate, le reprezentăm
scoruri după vectori h și a. Componenta a i-a a h oferă aglomerația
ith page, iar a i-a componentă a a oferă autoritatea aceleiași pagini.
În timp ce importanța este împărțită între succesorii unei pagini, așa cum este exprimată de
matricea de tranziție a Web-ului, modalitatea normală de a descrie calculul
de hubbiness și autoritate este de a adăuga autoritatea succesorilor de a estima
pentru a estima autoritatea. Daca asta
este tot ceea ce am făcut, atunci valorile autorității și autorității ar crește de obicei
dincolo de limite. Astfel, scalăm în mod normal valorile vectorilor h și a so
că cea mai mare componentă este 1. O alternativă este de a scala astfel încât suma de
componentele este 1.
Pentru a descrie calculul iterativ al lui h și a formal, folosim linkul
matricea Web, L. Dacă avem n pagini, atunci L este o matrice $n \times n$ și $L_{ij} = 1$
dacă există un link de la pagina i la pagina j, iar $L_{ij} = 0$ dacă nu. Vom avea și noi
nevoie de L^T, transpunerea lui L. Adică $L^T_{ij} = 1$ dacă există un link din pagină
j la pagina i și $L^T_{ij} = 0$ altfel. Observați că L^T este similar cu matricea M
pe care l-am folosit pentru PageRank, dar unde L^T are 1, M are o fracție - 1 împărțită
după numărul de out-link-uri din pagina reprezentată de acea coloană.
Exemplul 5.14: Pentru un exemplu de execuție, vom folosi rețeaua din Fig. 5.4,
pe care le reproducem aici ca Fig. 5.18. O observație importantă este că mortul
capetele sau capcanele păianjen nu împiedică iterația HITS să convergă la a
pereche semnificativă de vectori. Astfel, putem lucra direct cu Fig. 5.18, cu
nu este necesară „impozitare” sau modificare a graficului. Matricea de legătură L și a sa
transpunerea sunt prezentate în Fig. 5.19. ✷

Pagina 226

B
A
C
D
E
Figura 5.18: Exemple de date utilizate pentru exemplele HITS
L =
□
□
□
□
□
□
0 1 1 1 0
1 0 0 1 0
0 0 0 0 1
0 1 1 0 0
0 0 0 0 0
□
□
□
□
□
□
L T =
□
□
□
□
□
□
0 1 0 0 0
1 0 0 1 0
1 0 0 1 0
1 1 0 0 0
0 0 1 0 0
□
□
□
□
□
□
Figura 5.19: Matricea de legătură pentru rețeaua din Fig. 5.18 și transpunerea acesteia
Faptul că limbajul unei pagini este proporțional cu suma autoritatea succesorilor săi este exprimată prin ecuația h = λLa, unde λeste o constantă necunoscută reprezentând factorul de scalare necesar. La fel, faptul că autoritatea unei pagini este proporțională cu suma afacerilor din predecesorii săi se exprimă prin a = µL T h, unde µ este o altă constantă de scalare. Aceste ecuații ne permit să calculăm independența și autoritatea ușor, prin substituirea unei ecuații în cealaltă, ca:

- h = λµLL T h.
- a = λµL T La.

Cu toate acestea, deoarece LL T și L T L nu sunt la fel de rare ca L și L T , suntem de obicei mai bine calculează h și a într-o adevărată recursivitate reciprocă. Adică începeți cu h un vector al tuturor celor 1.
1. Calculați a = L T h și apoi scalați astfel încât cea mai mare componentă să fie 1.
2. Apoi, calculați h = La și scalați din nou.

Acum, avem o nouă h şi putem repeta paşii (1) şi (2) până la o anumită iteraţie modificările celor doi vectori sunt suficient de mici încât să le putem opri şi accepta valorile curente ca limită.

□
□
□
□
□
□
1
1
1
1
1
□
□
□
□
□
□
□
□
□
□
□
□
□
1
2
2
2
1
□
□
□
□
□
□
□
□
□
□
□
□
1/2
1
1
1
1/2
□
□
□
□
□
□
□

□
□
□
□
□
3
3/2
1/2
2
0
□
□
□
□
□
□
□
□
□
□
□
□
1
1/2
1/6
2/3
0
□
□
□
□
□
□
h
L т h
A
La
h
□
□
□
□
□
□
1/2
5/3
5/3
3/2
1/6
□
□
□
□
□
□
□
□

□
□
□
□
3/10
1
1
9/10
1/10
□
□
□
□
□
□
□
□
□
□
□
□
29/10
6/5
1/10
2
0
□
□
□
□
□
□
□
□
□
□
□
□
1
29.12
1/29
20/29
0
□
□
□
□
□
□
L T h
A
La
h

Figura 5.20: Primele două iterații ale algoritmului HITS

Exemplul 5.15: Să efectuăm primele două iterații ale algoritmului HITS pe web din Fig. 5.18. În Fig. 5.20 vedem succesiunea vectorilor calculați. Prima coloană este h-ul inițial, toate 1-urile. În a doua coloană, am estimat autoritatea relativă a paginilor calculând L T h, oferind astfel fiecărei pagini suma imuziunilor predecesorilor săi. Cea de-a treia coloană ne oferă prima

estimare a. Se calculează prin scalarea celei de-a doua coloane; în acest caz noi au împărțit fiecare componentă la 2, deoarece aceasta este cea mai mare valoare din a doua coloană.
Cea de-a patra coloană este La. Adică, am estimat interesul fiecăruia pagina prin însumarea estimării autorităților fiecăruia dintre succesorii săi. Apoi, a cincea coloană scalează a patra coloană. În acest caz, împărțim la 3, din moment ce aceasta este cea mai mare valoare din a patra coloană. Coloanele de la șase la nouă se repetă procesul descris în explicațiile noastre pentru coloanele două până la cinci, dar cu cea mai bună estimare a nebuniei dată de coloana a cincea.
Este posibil ca limita acestui proces să nu fie evidentă, dar poate fi calculată de un program simplu. Limitele sunt:
h =
□
□
□
□
□
□
1
0,3583
0
0,7165
0
□
□
□
□
□
□
a =
□
□
□
□
□
□
0,2087
1
1
0,7913
0
□
□
□
□
□
□

Acest rezultat are sens. În primul rând, observăm că hubbiness-ul lui E este cu siguranță 0, întrucât nu duce nicăieri. Distrugerea lui C depinde numai de autoritatea lui E și invers, deci nu ar trebui să ne surprindă că ambele sunt 0. A este cel mai mare hub, deoarece se leagă de cele mai mari trei autorități, B, C și D. De asemenea, B și C sunt cele mai mari autorități, deoarece acestea sunt legate de cele mai mari două hub-uri, A și D.
Pentru graficele de dimensiuni web, singurul mod de a calcula soluția la hub-uri și-ecuațiile autorităților este iterativ. Cu toate acestea, pentru acest mic exemplu, noi poate calcula soluția rezolvând ecuații. Vom folosi ecuațiile

h = λμLL T h. În primul rând, LL T este
LL T =
□
□
□
□
□
□
3 1 0 2 0
1 2 0 0 0
0 0 1 0 0
2 0 0 2 0
0 0 0 0 0
□
□
□
□
□
□
Fie ν= 1 / (λ μ) și să fie componentele lui h pentru nodurile A până la E să fie a
e, respectiv. Atunci se pot scrie ecuațiile pentru h
νa= 3a + b + 2d νb= a + 2b
νc= c
νd= 2a + 2d
νe= 0
Ecuația pentru b ne spune b = a / (ν - 2) și ecuația pentru d ne spune d =
2a / (ν - 2). Dacă înlocuim aceste expresii cu b și d în ecuația cu a,
obținem νa= a (3 + 5 / (ν − 2)). Din această ecuație, deoarece a este un factor al ambelor părți,
rămânem cu o ecuație pătratică pentru νcare simplifică la ν2 - 5ν + 1 = 0.
Rădăcina pozitivă este ν= (5 + √21) /2=4.791. Acum, că știm νnu este nici una, nici alta
0 sau 1, ecuațiile pentru c și e ne spun imediat că c = e = 0.
În cele din urmă, dacă recunoaștem că a este cea mai mare componentă a lui h și setăm a = 1,
obținem b = 0,3583 și d = 0,7165. Împreună cu c = e = 0, aceste valori ne dau
valoarea limitativă a h. Valoarea lui poate fi calculată din h prin multiplicare
prin L T și scalare. ✷

5.5.3 Exerciții pentru secțiunea 5.5

Exercițiul 5.5.1: Calculați grosimea și autoritatea fiecăruia dintre nodurile din
graficul nostru web original din Fig. 5.1.
! Exercițiul 5.5.2: Să presupunem că graficul nostru este un lanț de n noduri, așa cum a fost sugerat de
Fig. 5.9. Calculați hub-urile și vectorii de autorități, în funcție de n.

5.6 Rezumatul capitolului 5

♦ Termen Spam: motoarele de căutare timpurii nu au reușit să furnizeze rezultate relevante
deoarece erau vulnerabili la termenul spam - introducerea în Web
pagini de cuvinte care au denaturat în mod greșit despre ce era pagina.

Pagina 229

♦ Soluția Google pentru Termenul Spam: Google a reușit să contracareze termenul
spam prin două tehnici. Mai întâi a fost algoritmul PageRank pentru determinarea
exploatarea importanței relative a paginilor de pe web. Al doilea a fost un
strategia de a crede ce au spus alte pagini despre o anumită pagină, în sau în apropiere
linkurile lor către acea pagină, mai degrabă decât să creadă doar ceea ce spunea pagina
despre sine.

♦ PageRank: PageRank este un algoritm care atribuie un număr real, numit
PageRank-ul său, către fiecare pagină de pe Web. PageRank-ul unei pagini este un
măsura cât de importantă este pagina sau cât de probabil este să fie un bun
răspuns la o interogare de căutare. În forma sa cea mai simplă, PageRank este o soluție
la ecuația recursivă „o pagină este importantă dacă paginile importante se leagă de
aceasta."

♦ Matricea de tranziție a Web-ului: Reprezentăm linkuri pe Web printr-o matrice
al cărui al doilea rând și coloana a XI-a reprezintă pagina a XI-a a Web-ului. În cazul în care există

sunt unul sau mai multe linkuri de la pagina j la pagina i, apoi intrarea din rândul i și coloana j este 1 / k, unde k este numărul de pagini către care se leagă pagina j. Alte intrări ale matricei de tranziție sunt 0.

♦ Computarea PageRank pe graficele web puternic conectate: Pentru puternic grafice web conectate (cele în care orice nod poate ajunge la orice alt nod), PageRank este vectorul propriu principal al matricei de tranziție. Putem calculați PageRank începând cu orice vector diferit de zero și în mod repetat înmulțind vectorul curent cu matricea de tranziție, pentru a obține o mai bună estima. [7] După aproximativ 50 de iterații, estimarea va fi foarte aproape de limita, care este adevăratul PageRank.

♦ Modelul de surfer aleator: se poate gândi la calculul PageRank ca simulând comportamentul multor surferi aleatori, care încep fiecare de la pagina aleatorie și la orice pas treceți, la întâmplare, la una dintre pagini la pe care le leagă pagina curentă. Probabilitatea limitativă ca un surfer să fie la o anumită pagină este PageRank-ul acelei pagini. Intuiția este că oamenii tind să creeze legături către paginile pe care le consideră utile, deci surferi aleatori va tinde să fie la o pagină utilă.

♦ Impas: o fundătură este o pagină web fără linkuri. Prezenta lui impasurile vor face ca PageRank-ul pentru unele sau pentru toate paginile să ajungă la 0 în calculul iterativ, inclusiv pagini care nu sunt impas. Noi poate elimina toate fundurile înainte de a efectua un calcul PageRank prin renunțarea recursivă a nodurilor fără arcuri. Rețineți că renunțarea la unul nodul poate face ca un altul, care se leagă numai de acesta, să devină o fundătură, deci procesul trebuie să fie recursiv.

[7] Din punct de vedere tehnic, condiția ca această metodă să funcționeze este mai restrânsă decât pur și simplu „puternic conectat." Cu toate acestea, celelalte condiții necesare vor fi îndeplinite cu siguranță de orice marime puternică componentă conectată a Web-ului care nu a fost construită artificial.

Pagina 230

♦ Capcane pentru păianjen: o capcană pentru păianjen este un set de noduri la care, deși se pot lega reciproc, nu au legături către alte noduri. Într-un calcul iterativ din PageRank, prezența capcanelor pentru păianjeni determină ca tot PageRank să fie capturat în cadrul acelui set de noduri.

♦ Scheme de impozitare: pentru a contracara efectul capcanelor de păianjen (și ale fundurilor, dacă nu le eliminăm), PageRank este normal calculat într-un fel care modifică multiplicarea iterativă simplă cu matricea de tranziție. Se alege un parametru β,de obicei în jur de 0,85. Având în vedere o estimare a PageRank, următoarea estimare este calculată prin înmulțirea estimării cu βori matricea de tranziție, apoi adăugând (1 - β)/ n la estimare pentru fiecare pagină, unde n este numărul total de pagini.

♦ Impozitare și surferi aleatori: calculul PageRank folosind taxoni- parametrul βpoate fi considerat ca oferind fiecărui surfer aleatoriu o probă capacitatea 1 - βde a părăsi webul și de a introduce un număr echivalent de surferi în mod aleatoriu pe web.

♦ Reprezentarea eficientă a matricilor de tranziție: de la o matrice de tranziție este foarte rar (aproape toate intrările sunt 0), economisește atât timp, cât și spațiu pentru a-l reprezenta prin listarea intrărilor sale diferite de zero. Cu toate acestea, pe lângă fiind rare, intrările diferite de zero au o proprietate specială: acestea sunt toate la fel în orice coloană dată; valoarea fiecărei intrări diferite de zero este inversă a numărului de intrări diferite de zero din acea coloană. Astfel, preferatul reprezentarea este coloană cu coloană, unde reprezentarea unei coloane este numărul de intrări diferite de zero, urmat de o listă a rândurilor în care acestea intrările apar.

♦ Matricea la scară foarte mare - Multiplicarea vectorială: pentru grafice de dimensiuni web, aceasta s-ar putea să nu fie fezabil să stochezi întregul vector estimativ PageRank în memoria principală a unei mașini. Astfel, putem sparge vectorul în k segmente și rupe matricea de tranziție în k^2 pătrate, numite blocuri, atribuind fiecare pătrat unei singure mașini. Segmentele vectoriale sunt trimise fiecare la k mașini, deci există un cost suplimentar mic în replicarea vectorului.

♦ Reprezentarea blocurilor unei matrice de tranziție: Când împărțim o tranziție matrice în blocuri pătrate, coloanele sunt împărțite în k segmente. La reprezintă un segment al unei coloane, nu este nevoie de nimic dacă nu există nule intrări în acel segment. Cu toate acestea, dacă există una sau mai multe intrări diferite de zero, atunci trebuie să reprezentăm segmentul coloanei cu numărul total

de intrări diferite de zero în coloană (deci putem spune ce valoare are zero
intrările au) urmată de o listă a rândurilor cu intrări diferite de zero.

♦ PageRank sensibil la subiect: dacă știm că interogatorul este interesat de o cerere
În acest subiect, este logic să părtinim PageRank în favoarea paginilor
pe acest subiect. Pentru a calcula această formă de PageRank, identificăm un set de
pagini despre care se știe că se află pe subiectul respectiv și îl folosim ca „set de teleportare”. The

---

Pagina 231

Calculul PageRank este modificat astfel încât numai paginile din teleport
setului li se acordă o cotă din impozit, mai degrabă decât distribuirea impozitului între
toate paginile de pe Web.

♦ Crearea seturilor de teleportare: Pentru a funcționa PageRank sensibil la subiect, trebuie
identificați paginile care sunt foarte susceptibile să fie despre un anumit subiect. O abordare
este să începeți cu paginile cu care directorul deschis (DMOZ) se identifică
acel subiect. Un altul este acela de a identifica cuvintele despre care se știe că sunt asociate cu
subiect și selectați pentru teleport set acele pagini care au un neobișnuit
numărul mare de apariții ale unor astfel de cuvinte.

♦ Link Spam: pentru a păcăli algoritmul PageRank, actorii fără scrupule au
a creat ferme de spam. Acestea sunt colecții de pagini al căror scop este
concentrați PageRank ridicat pe o anumită pagină țintă.

♦ Structura unei ferme de spam: de obicei, o fermă de spam este formată dintr-o țintă
pagină și foarte multe pagini de sprijin. Pagina țintă ține legătura cu toate fișierele
paginile de sprijin, iar paginile de sprijin se leagă numai de pagina țintă.
În plus, este esențial ca unele legături din afara fermei de spam să fie
creată. De exemplu, spammer-ul ar putea introduce link-uri către ținta sa
pagină scriind comentarii în blogurile altor persoane sau în grupurile de discuții.

♦ TrustRank: o modalitate de a ameliora efectul spamului link-ului este de a calcula
un PageRank sensibil la subiect numit TrustRank, unde setul de teleportare este un
colecție de pagini de încredere. De exemplu, paginile de start ale universităților
ar putea servi drept set de încredere. Această tehnică evită împărțirea impozitului în
calculul PageRank cu un număr mare de pagini suport în
ferme de spam și, prin urmare, își reduce preferențial PageRank-ul.

♦ Spam Mass: Pentru a identifica fermele de spam, putem calcula atât convențiile
PageRank internațional și TrustRank pentru toate paginile. Acele pagini care au
TrustRank mult mai scăzut decât PageRank ar putea face parte dintr-un spam
fermă.

♦ Huburi și autorități: în timp ce PageRank oferă o viziune unidimensională
din importanța paginilor, un algoritm numit HITS încearcă să măsoare
două aspecte diferite ale importanței. Autoritățile sunt acele pagini care
conțin informații valoroase. Huburile sunt pagini care, deși nu
ele însele conțin informații, link către locuri unde informațiile
poate fi găsit.

♦ Formularea recursivă a algoritmului HITS: calculul hub-urilor
și scorurile autorităților pentru pagini depind de rezolvarea echivalenței recursive

,,un hub se leagă de multe autorități, iar o autoritate este legată de
de multe hub-uri. ” Soluția la aceste ecuații este în esență un iter-
matrice ated-multiplicare vectorială, la fel ca PageRank. Însă
existența unor fundături sau capcane pentru păianjen nu afectează soluția la

---

Pagina 232

HITS ecuații în modul în care o fac pentru PageRank, deci nu există un sistem de impozitare
este necesar.

5.7 Referințe pentru capitolul 5

Algoritmul PageRank a fost exprimat pentru prima dată în [1]. Experimentele pe
structura Web-ului, pe care am folosit-o pentru a justifica existența unor fundături și
capcane pentru păianjeni, au fost descrise în [2]. Metoda cu bandă de bloc pentru efectuarea
Iterația PageRank este preluată din [5].
PageRank sensibil la subiect este preluat din [6]. TrustRank este descris în [4],
iar ideea de masă spam este preluată din [3].

Ideea HITS (hub-uri și autorități) a fost descrisă în [7].
1. S. Brin și L. Page, „Anatomia unei căutări hipertextuale pe scară largă pe web motor, ”Proc. A 7-a Intl. Conferință World-Wide-Web, pp. 107-117, 1998.
2. A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins și J. Weiner, „Structura graficului pe web”, Computer Networks 33: 1-6, pp. 309-320, 2000.
3. Z. Gyöngi, P. Berkhin, H. Garcia-Molina și J. Pedersen, „Link spam detectarea pe baza estimării masei, ”Proc. 32 Intl. Conf. pe Foarte mare Baze de date, pp. 439-450, 2006.
4. Z. Gyöngi, H. Garcia-Molina și J. Pedersen, „Combating link link spam cu trustrank, ”Proc. 30 Intl. Conf. pe baze de date foarte mari, pp. 576–587, 2004.
5. TH Haveliwala, „Calcul eficient al PageRank”, Stanford Univ. Raport tehnic al Departamentului de Informatică, septembrie 1999. Disponibil ca http://infolab.stanford.edu/~taherh/papers/efficient-pr.pdf
6. TH Haveliwala, „PageRank sensibil la subiect”, Proc. Intl. 11 Lume-Wide-Web Conference, pp. 517-526, 2002
7. JM Kleinberg, „Surse autoritare într-un mediu hiperlegat”, J. ACM 46: 5, pp. 604-632, 1999.

Capitolul 6
Seturi obiecte frecvente

Trecem în acest capitol la una dintre marile familii de tehnici pentru caracterizare date izante: descoperirea unor obiecte frecvente. Această problemă este adesea privită ca descoperirea „regulilor de asociere”, deși aceasta din urmă este un caracter mai complex activizarea datelor, a căror descoperire depinde fundamental de descoperire de obiecte frecvente.

Pentru început, introducem modelul de date „market-basket”, care este esențial o relație mult-multe între două tipuri de elemente, numite „articole” și „coșuri”, dar cu unele presupuneri despre forma datelor. The problema seturilor de obiecte frecvente este aceea de a găsi seturi de elemente care apar în (sunt legat de) multe dintre aceleași coșuri.

Problema găsirii unor obiecte frecvente diferă de căutarea similarității discutată în capitolul 3. Aici ne interesează numărul absolut de coșuri care conțin un anumit set de articole. În capitolul 3 am vrut articole care au o mare parte din coșurile lor în comun, chiar dacă numărul absolut de coșurile sunt mici.

Diferența duce la o nouă clasă de algoritmi pentru a găsi elemente frecvente-seturi. Începem cu algoritmul A-Priori, care funcționează prin eliminarea celor mai multe seturi mari ca candidați, examinând mai întâi seturi mai mici și recunoscând că a setul mare nu poate fi frecvent decât dacă toate subseturile sale sunt. Luăm apoi în considerare diverse îmbunătățiri ale ideii de bază A-Priori, concentrându-se pe seturi de date foarte mari care stresează memoria principală disponibilă.

În continuare, considerăm algoritmi aproximativi care funcționează mai repede, dar nu sunt garantat pentru a găsi toate seturile obiecte frecvente. De asemenea, în această clasă de algoritmi sunt cele care exploatează paralelismul, inclusiv paralelismul pe care îl putem obține o formulare MapReduce. În cele din urmă, discutăm pe scurt cum să găsim frecvente seturi de articole într-un flux de date.

6.1 Modelul Market-Basket

Modelul de piață-coș de date este utilizat pentru a descrie o formă comună de multe-multe relații între două tipuri de obiecte. Pe de o parte, avem articole, iar pe de altă parte avem coșuri, uneori numite „tranzacții”.

Fiecare coș este format dintr-un set de articole (un set de articole) și, de obicei, presupunem că numărul articolelor dintr-un coș este mic - mult mai mic decât numărul total de articole. Numărul coșurilor este de obicei presupus a fi foarte mare, mai mare decât ceea ce se poate încadra în memoria principală. Se presupune că datele sunt reprezentate într-un fișier format dintr-o succesiune de coșuri. În ceea ce privește sistemul de fișiere distribuit

descrise în secțiunea 2.1, coșurile sunt obiectele fișierului și ale fiecărui coș
este de tip „set de articole”.

6.1.1 Definiția seturilor de articole frecvente

Intuitiv, un set de articole care apare în multe coșuri se spune că este „frecvent”.
Pentru a fi formal, presupunem că există un număr s, numit pragul de sprijin. Dacă
I este un set de articole, suportul pentru I este numărul de coșuri pentru care eu sunt
subset. Spunem că sunt frecventă dacă suportul său este mai mare sau mai mare.
Exemplul 6.1: În Fig. 6.1 sunt seturi de cuvinte. Fiecare set este un coș, iar
cuvintele sunt elemente. Am luat aceste seturi cu un câine de pisică Googling și luând fragmente
din cele mai bine clasate pagini. Nu vă îngrijorați dacă apare un cuvânt de două ori
într-un coș, deoarece coșurile sunt seturi și, în principiu, articolele pot apărea o singură dată.
De asemenea, ignorați majusculele.
1. {Pisica și câinele mușcă}
2. {Yahoo, știri, revendicări, o, pisică, împerecheată, cu, un, câine și, produsă, viabilă,
descendenți}
3. {Pisică, ucigașă, probabil, este, un, câine mare}
4. {Consultanță profesională, gratuită, pe, câine, antrenament, cățeluș, antrenament}
5. {Pisică și pisoi, antrenament și comportament}
6. {Dog, &, Cat, oferă, câine, antrenament, în, Eugene, Oregon}
7. {„Câine și pisică”, este, un, argou, termen, folosit, de, polițiști, ofițeri, pentru, un, bărbat -
femeie, relație}
8. {Cumpărați, pentru, spectacol, dvs., câine, îngrijire și, animale de companie, rechizite}
Figura 6.1: Iată opt coșuri, fiecare constând din articole care sunt cuvinte
Deoarece setul gol este un subset al oricărui set, suportul pentru ∅ este 8. Cu toate acestea,
în general nu ne vom preocupa de setul gol, din moment ce acesta ne spune

---

nimic.
Dintre seturile singleton, evident {cat} și {dog} sunt destul de frecvente.
,,Câine” apare în toate, cu excepția coșului (5), deci suportul său este 7, în timp ce „pisică” apare în
toate cu excepția (4) și (8), deci suportul său este 6. Cuvântul „și” este, de asemenea, destul de frecvent;
apare în (1), (2), (5), (7) și (8), deci suportul său este 5. Cuvintele „a” și
,,Antrenament” apare în trei seturi, în timp ce „pentru” și „este” apar în două. Nu
alt cuvânt apare de mai multe ori.
Să presupunem că ne setăm pragul la s = 3. Apoi sunt cinci frecvente
seturi de articole singleton: {câine}, {pisică}, {și}, {a} și {antrenament}.
Acum, să ne uităm la dubletoni. Un dubleton nu poate fi frecvent decât dacă
ambele elemente din set sunt frecvente de la sine. Astfel, sunt doar zece
posibile dubletonuri frecvente. Fig. 6.2 este un tabel care indică ce coșuri conțin
care dubleton.
instruire a
și
pisică
câine
4, 6
2, 3, 7 1, 2, 7, 8 1, 2, 3, 6, 7
pisică
5, 6
2, 3, 7 1, 2, 5, 7
și
5
2, 7
A
nici unul
Figura 6.2: Aparițiile dubletonelor
De exemplu, vedem din tabelul din Fig. 6.2 că dubleton {câine, antrenament}
apare doar în coșurile (4) și (6). Prin urmare, suportul său este 2 și nu este
frecvent. Există cinci dubletoni frecvenți dacă s = 3; sunt
{câine, un} {câine și} {câine, pisică}
{pisică, a}
{pisică și}
Fiecare apare de cel puțin trei ori; de exemplu, {câine, pisică} apare de cinci ori.
În continuare, să vedem dacă există frecvent triple. Pentru a fi un triplu frecvent,
fiecare pereche de elemente din set trebuie să fie un dubleton frecvent. De exemplu,
{câine, a și} nu poate fi un set de articole frecvente, pentru că dacă ar fi, atunci cu siguranță {a,

și} ar fi frecvent, dar nu este. Triplul {câine, pisică și} ar putea fi
frecvent, deoarece fiecare dintre subgrupurile sale dubleton este frecvent. Din păcate,
trei cuvinte apar împreună numai în coșurile (1) și (2), deci nu este o frecvență
triplu. Triplul {câine, pisică, a} ar putea fi frecvent, deoarece dubletonele sale sunt toate
frecvent. De fapt, toate cele trei cuvinte apar în coșurile (2), (3) și (7), deci
este un triplu frecvent. Niciun alt triplu de cuvinte nu este nici măcar un candidat pentru a fi
un triplu frecvent, deoarece pentru nici un alt triplu al cuvintelor nu sunt cele trei dubletonuri ale sale
subseturi frecvente. Deoarece există un singur triplu frecvent, nu poate fi frecvent
cvadrupluri sau seturi mai mari. ✷

Pagina 236

Vânzare cu amănuntul on-line versus cărămidă

Am sugerat în secțiunea 3.1.3 că un comerciant cu amănuntul online ar folosi similaritatea
măsuri pentru ca obiectele să găsească perechi de obiecte care, deși ar putea să nu fie
cumpărate de mulți clienți, au avut o fracțiune semnificativă din clienții lor
in comun. Un comerciant cu amănuntul online ar putea apoi să facă publicitate unui articol din pereche
către câțiva clienți care cumpăraseră celălalt obiect al perechii. Acest
metodologia nu are sens pentru un comerciant cu amănuntul de cărămizi, deoarece
mai puțini oameni cumpără un articol, nu poate fi rentabil să faceți publicitate unui
vânzare pe articol. Astfel, tehnicile din capitolul 3 nu sunt adesea utile
pentru comercianții cu amănuntul de cărămidă și mortar.
Dimpotrivă, comerciantul cu amănuntul online nu are mare nevoie de analiza pe care o prezentăm
în acest capitol, deoarece este conceput pentru a căuta seturi de articole care apar
frecvent. Dacă vânzătorul cu amănuntul online era limitat la obiecte frecvente, acestea
ar pierde toate oportunitățile care sunt prezente în „coada lungă”
selectați reclame pentru fiecare client individual.

6.1.2 Aplicații de obiecte frecvente

Aplicarea inițială a modelului de coș de piață a fost în analiza adevăratului
coșuri de piață. Adică, supermarketurile și lanțurile de magazine înregistrează conținutul
din fiecare coș de piață (coș de cumpărături fizic) adus în registru pentru
verifică. Aici „articolele” sunt diferitele produse pe care magazinul le vinde și
,,coșurile” sunt seturile de articole dintr-un coș de piață unică. Un lanț major
ar putea vinde 100.000 de articole diferite și ar putea colecta date despre milioane de piețe
coșuri.
Găsind obiecte frecvente, un comerciant cu amănuntul poate afla ce se cumpără în mod obișnuit
împreună. Deosebit de importante sunt perechile sau seturile mai mari de articole care apar mult
mai frecvent decât s-ar fi așteptat au fost articolele cumpărate independent.
Vom discuta acest aspect al problemei în secțiunea 6.1.3, dar pentru moment
să luăm în considerare pur și simplu căutarea unor seturi de articole frecvente. Vom descoperi prin aceasta
analiză că mulți oameni cumpără pâine și lapte împreună, dar asta este de puțin
interes, deoarece știam deja că acestea erau articole populare în mod individual. Noi
ar putea descoperi că mulți oameni cumpără hot dog și muștar împreună. Acea,
din nou, nu ar trebui să fie o surpriză pentru persoanele cărora le plac hot dog-urile, dar le oferă
supermarketului o oportunitate de a face niște marketing inteligent. Ei pot face reclama
o vânzare la hot dog și crește prețul muștarului. Când oamenii vin la
magazin pentru hot-dog-urile ieftine, de multe ori își vor aminti că au nevoie de muștar,
și cumpără și asta. Fie nu vor observa că prețul este ridicat, fie motivează
că nu merită să te chinui să mergi în altă parte pentru muștar mai ieftin.
Celebrul exemplu de acest tip este „scutece și bere”. Cu greu s-ar putea
așteptați ca aceste două elemente să fie corelate, dar prin analiza datelor un lanț de magazine
a descoperit că persoanele care cumpără scutece sunt neobișnuit de susceptibile să cumpere bere. The

---

Pagina 237

teoria este că, dacă cumperi scutece, probabil că ai un copil acasă și dacă ai
ai un copil, atunci este puțin probabil să bei la un bar; de aceea ești mai mult
probabil să aducă bere acasă. Același tip de truc de marketing pe care l-am sugerat
pentru hot dog și muștar ar putea fi folosite pentru scutece și bere.
Cu toate acestea, aplicațiile de analiză a elementelor frecvente nu se limitează la piață
coșuri. Același model poate fi folosit pentru a extrage multe alte tipuri de date. niste
exemple sunt:
1. Concepte conexe: Fie ca articolele să fie cuvinte, iar coșurile să fie documente
(de exemplu, pagini web, bloguri, tweets). Un coș / document le conține

elemente / cuvinte care sunt prezente în document. Dacă căutăm seturi de
cuvinte care apar împreună în multe documente, seturile vor fi
indicat de cele mai frecvente cuvinte (cuvinte stop), așa cum am văzut în exemplul 6.1.
Acolo, deși intenția era să găsim fragmente care să vorbească despre pisici
și câini, cuvintele de oprire „și” și „a” erau proeminente printre
seturi de obiecte. Cu toate acestea, dacă ignorăm toate cele mai frecvente cuvinte, atunci
am spera să găsim printre perechile frecvente câteva perechi de cuvinte
care reprezintă un concept comun. De exemplu, ne-am aștepta la o pereche de genul
{Brad, Angelina} să apară cu o frecvență surprinzătoare.
2. Plagiat: Lăsați articolele să fie documente, iar coșurile să fie propoziții.
Un articol / document este „în” un coș / propoziție dacă propoziția se află în
document. Acest aranjament apare înapoi, dar este exact ce
avem nevoie și ar trebui să ne amintim că relația dintre articole
iar coșurile sunt o relație arbitrară multe-multe. Adică „în” nevoie
nu au semnificația sa convențională: „parte din”. În această aplicație, noi
căutați perechi de articole care apar împreună în mai multe coșuri. Dacă găsim
o astfel de pereche, atunci avem două documente care împărtășesc mai multe propoziții
uzual. În practică, chiar una sau două propoziții în comun este un bun
indicator al plagiatului.
3. Biomarkeri: Să fie elementele de două tipuri - biomarkeri, cum ar fi genele
sau proteine din sânge și boli. Fiecare coș este setul de date despre
un pacient: analiza genomului și a chimiei sângelui, precum și a lor
istoricul medical al bolii. Un set frecvent de articole care constă dintr-o singură boală
și unul sau mai mulți biomarkeri sugerează un test pentru boală.

6.1.3 Reguli de asociere

În timp ce subiectul acestui capitol extrage seturi frecvente de articole din date,
aceste informații sunt adesea prezentate ca o colecție de reguli if – then, numite asociate
reguli de acțiune. Forma unei reguli de asociere este I → j, unde I este un set de itemi
iar j este un element. Implicația acestei reguli de asociere este că dacă toate
articolele din I apar într-un coș, apoi j este „probabil” să apară în coșul respectiv ca
bine.

Formalizăm noțiunea de „probabil” definind încrederea regulii
I → j să fie raportul dintre suportul pentru I ∪ {j} și suportul pentru I. Adică,
încrederea regulii este fracția coșurilor cu tot I care, de asemenea
conține j.
Exemplul 6.2: Luați în considerare coșurile din Fig. 6.1. Încrederea regulii
{pisică, câine} → și este 3/5. Cuvintele „pisică” și „câine” apar în cinci coșuri:
(1), (2), (3), (6) și (7). Dintre acestea, „și” apare în (1), (2) și (7) sau 3/5
a coșurilor.
Pentru o altă ilustrație, încrederea lui {pisică} → pisoi este 1/6. Cuvantul
,,Pisică” apare în șase coșuri, (1), (2), (3), (5), (6) și (7). Dintre acestea, doar (5)
are cuvântul „pisoi”. ✷
Încrederea singură poate fi utilă, cu condiția să sprijine partea stângă a
regula este destul de mare. De exemplu, nu trebuie să știm că oamenii sunt
neobișnuit de probabil să cumpere muștar atunci când cumpără hot dog, atâta timp cât știm
că mulți oameni cumpără hot dog și mulți oameni cumpără hot dog și
muștar. Putem folosi în continuare trucul vânzării la hot-dogs discutat în secțiunea 6.1.2.
Cu toate acestea, există adesea o valoare mai mare pentru o regulă de asociere dacă aceasta reflectă o adevărată
relație, în care elementul sau articolele din stânga afectează cumva elementul de pe
dreapta.
Astfel, definim interesul unei reguli de asociere I → j pentru a fi diferența
între încrederea sa și fracția de coșuri care conțin j. Acesta este,
dacă nu am nicio influență asupra lui j, atunci ne-am aștepta ca fracția coșurilor
inclusiv I care conține j ar fi exact la fel ca fracțiunea dintre toate
coșuri care conțin j. O astfel de regulă are interes 0. Cu toate acestea, este interesant, în
atât sensul informal, cât și cel tehnic, dacă o regulă are fie un interes ridicat, fie un sens
că prezența lui I într-un coș provoacă cumva prezența lui j sau foarte mare
interes negativ, adică prezența lui I descurajează prezența lui j.
Exemplul 6.3: Povestea despre bere și scutece este într-adevăr o afirmație că
regula ciației {scutece} → berea are un interes ridicat. Adică, fracția de scutec-
cumpărătorii care cumpără bere este semnificativ mai mare decât fracția dintre toți clienții
care cumpără bere. Un exemplu de regulă cu dobândă negativă este {cocs} → pepsi.
Adică, este puțin probabil ca oamenii care cumpără Coca-Cola să cumpere și Pepsi
o fracțiune bună din toți oamenii cumpără Pepsi - oamenii preferă de obicei unul sau unul

altele, dar nu ambele. În mod similar, se poate aștepta ca regula {pepsi} → cocs
au interes negativ.
Pentru unele calcule numerice, să ne întoarcem la datele din Fig. 6.1. The
regula {câine} → pisica are încredere 5/7, deoarece „câine” apare în șapte coșuri, de
care cinci au „pisică”. Cu toate acestea, „pisica” apare în șase din cele opt coșuri,
deci ne-am aștepta ca 75% din cele șapte coșuri cu „câine” să aibă „pisică”
de asemenea. Astfel, interesul regulii este 5 / 7-3 / 4 = -0,036, care este în esență
0. Regula {pisică} → pisoi are interes 1/6 - 1/8 = 0,042. Justificarea este
că unul din cele șase coșuri cu „pisică” are și „pisoi”, în timp ce „pisoi”

Pagina 239

apare doar într-unul din cele opt coșuri. Acest interes, deși pozitiv, este aproape
la 0 și, prin urmare, indică faptul că regula asocierii nu este foarte „interesantă”. ✷

6.1.4 Găsirea regulilor de asociere cu mare încredere

Identificarea regulilor de asociere utile nu este mult mai dificilă decât găsirea frecventă
seturi de articole. Vom aborda problema găsirii unor obiecte frecvente în
echilibrul acestui capitol, dar pentru moment, presupunem că este posibil să le găsim
obiecte frecvente al căror suport este la sau peste un prag de sprijin s.
Dacă căutăm reguli de asociere I → j care se aplică într-un mod rezonabil
fracțiune din coșuri, atunci sprijinul lui I trebuie să fie destul de ridicat. În
practică, cum ar fi pentru comercializarea în magazinele de cărămidă și mortar, „destul de ridicată”
este adesea în jur de 1% din coșuri. Vrem și încrederea regulii
să fie rezonabil de ridicat, poate 50%, altfel regula are un efect practic redus. La fel de
ca rezultat, setul I ∪ {j} va avea, de asemenea, suport destul de mare.
Să presupunem că am găsit toate seturile de articole care îndeplinesc un prag de asistență și
că avem suportul exact calculat pentru fiecare dintre aceste seturi de articole. Putem
găsiți în ele toate regulile de asociere care au atât sprijin ridicat, cât și mare
încredere. Adică, dacă J este un set de n elemente care se dovedește a fi frecvent, există
numai n posibile reguli de asociere care implică acest set de itemi, și anume J - {j} → j
pentru fiecare j din J. Dacă J este frecvent, J - {j} trebuie să fie cel puțin la fel de frecvent. Astfel, ea
De asemenea, este un set de articole frecvent și am calculat deja suportul ambelor J
și J - {j}. Raportul lor este încrederea regulii J - {j} → j.
Trebuie să presupunem că nu există prea multe seturi de articole frecvente și astfel
nu prea mulți candidați pentru reguli de asociere de înaltă încredere.
Motivul este că fiecare găsit trebuie acționat. Dacă dăm magazinul
gestionați un milion de reguli de asociere care ne îndeplinesc pragurile de asistență și
încredere, nici măcar nu le pot citi, darămite să acționeze asupra lor. La fel, dacă noi
producem un milion de candidați pentru biomarkeri, nu ne putem permite să conducem
sentimentele necesare pentru a le verifica. Astfel, este normal să reglați suportul
prag, astfel încât să nu primim prea multe seturi obiecte frecvente. Această presupunere
în secțiunile ulterioare, consecințele importante asupra eficienței
algoritmi pentru găsirea unor obiecte frecvente.

6.1.5 Exerciții pentru secțiunea 6.1

Exercițiul 6.1.1: Să presupunem că există 100 de articole, numerotate de la 1 la 100 și, de asemenea, 100
coșuri, de asemenea numerotate de la 1 la 100. Elementul i se află în coșul b dacă și numai dacă i împarte b
fără rest. Astfel, articolul 1 este în toate coșurile, articolul 2 este în toate cele cincizeci din
coșuri cu număr par și așa mai departe. Coșul 12 este format din articolele {1, 2, 3, 4, 6, 12},
deoarece acestea sunt toate numerele întregi care împart 12. Răspundeți la următoarele întrebări:
(a) Dacă pragul de sprijin este 5, care elemente sunt frecvente?
! (b) Dacă pragul de sprijin este 5, ce perechi de articole sunt frecvente?

Pagina 240

! (c) Care este suma dimensiunilor tuturor coșurilor?
! Exercițiul 6.1.2: Pentru datele coșului articolului din Exercițiul 6.1.1, care este coșul
cel mai mare?
Exercițiul 6.1.3: Să presupunem că există 100 de articole, numerotate de la 1 la 100 și, de asemenea, 100
coșuri, de asemenea numerotate de la 1 la 100. Elementul i se află în coșul b dacă și numai dacă b împarte i
fără rest. De exemplu, coșul 12 constă din articole
{12, 24, 36, 48, 60, 72, 84, 96}
Repetați exercițiul 6.1.1 pentru aceste date.
! Exercițiul 6.1.4: Această întrebare implică date din care nimic interesant
pot fi învățate despre obiecte frecvente, deoarece nu există seturi de articole care

sunt corelate. Să presupunem că articolele sunt numerotate de la 1 la 10 și fiecare coș este construit prin includerea elementului i cu probabilitatea 1 / i, fiecare decizie fiind luată independent de toate celelalte decizii. Adică, toate coșurile conțin elementul 1, jumătate conțin articolul 2, o a treia conține articolul 3 și așa mai departe. Să presupunem numărul de coșurile sunt suficient de mari încât coșurile să se comporte colectiv așa cum s-ar face așteptați statistic. Lăsați pragul de sprijin să fie 1% din coșuri. Găsi obiecte frecvente.

Exercițiul 6.1.5: Pentru datele din Exercițiul 6.1.1, care este încrederea respectând regulile de asociere?

(a) {5, 7} → 2.

(b) {2, 3, 4} → 5.

Exercițiul 6.1.6: Pentru datele din Exercițiul 6.1.3, care este încrederea respectând regulile de asociere?

(a) {24, 60} → 8.

(b) {2, 3, 4} → 5.

!! Exercițiul 6.1.7: Descrieți toate regulile de asociere care au încredere de 100% pentru datele coșului de piață ale:

(a) Exercițiul 6.1.1.

(b) Exercițiul 6.1.3.

! Exercițiul 6.1.8: Dovediți că în datele din Exercițiul 6.1.4 nu există niciun interes-regulile de asociere; adică interesul fiecărei reguli de asociere este 0.

6.2 Coșuri de piață și algoritmul A-Priori

Vom începe acum o discuție despre cum să găsim obiecte sau informații frecvente derivate din acestea, cum ar fi regulile de asociere cu sprijin și încredere ridicate. Îmbunătățirea inițială a algoritmilor evidenți, cunoscuți ca „A-Priori”, din care au fost dezvoltate multe variante, vor fi tratate aici. Următoarele două secțiunile vor discuta anumite îmbunătățiri suplimentare. Înainte de a discuta despre A-Algoritmul priori în sine, începem secțiunea cu o schiță a ipotezelor despre modul în care sunt stocate și manipulate datele atunci când căutați obiecte frecvente.

6.2.1 Reprezentarea datelor din coșul de piață

După cum am menționat, presupunem că datele despre coșul de piață sunt stocate într-un coș de fișiere. coș de cumpărături. Eventual, datele se află într-un sistem de fișiere distribuit ca în secțiunea 2.1, iar coșurile sunt obiectele pe care le conține fișierul. Sau datele pot fi stocate într-un fișier convențional, cu un cod de caractere pentru a reprezenta coșurile și a lor obiecte.

Exemplul 6.4: Ne-am putea imagina că un astfel de fișier începe:

{23.456.1001} {3.18.92.145} {...

Aici, personajul {începe un coș și personajul} îl termină. Articolele din un coș sunt reprezentate prin numere întregi și sunt separate prin virgule. Astfel, primul coș conține articolele 23, 456 și 1001; al doilea coș conține obiecte 3, 18, 92 și 145. ✷

Este posibil ca o mașină să primească întregul fișier. Sau am putea folosi MapReduce sau un instrument similar pentru a împărți munca între mai multe procesoare, în caz în care fiecare procesor primește doar o parte din fișier. Se pare că combinând munca procesorelor paralele pentru a obține colecția exactă de articole care îndeplinesc un prag global de asistență este greu și vom aborda această întrebare numai în secțiunea 6.4.4.

De asemenea, presupunem că dimensiunea fișierului coșurilor este suficient de mare încât să fie nu se încadrează în memoria principală. Astfel, un cost major al oricărui algoritm este timpul este nevoie pentru a citi coșurile de pe disc. Odată ce un bloc de disc plin de coșuri este în memorie principală, o putem extinde, generând toate subseturile de mărime k. De la unul dintre ipotezele modelului nostru este că dimensiunea medie a unui coș este mică, generarea tuturor perechilor din memoria principală ar trebui să dureze mult mai puțin timp decât timpul necesar citirii coșului de pe disc. De exemplu, dacă sunt 20 obiecte într-un coș, apoi există $\binom{20}{2}$ = 190 perechi de articole în coș și acestea pot fi generate cu ușurință într-o pereche de bucle imbricate.

Pe măsură ce dimensiunea subseturilor pe care dorim să le generăm crește, timpul necesar crește mai mare; de fapt durează aproximativ timp $n^k / k!$ pentru a genera toate subseturile de mărimea k pentru un coș cu n articole. În cele din urmă, de data aceasta domină timpul necesare pentru a transfera datele de pe disc. In orice caz:

---

1. Adesea, avem nevoie doar de mici obiecte frecvente, astfel încât k nu crește niciodată dincolo de 2 sau 3.
2. Și când avem nevoie de seturi de articole pentru o dimensiune mare k, de obicei este posibil pentru a elimina multe dintre articolele din fiecare coș, deoarece nu pot participa într-un set de articole frecvente, astfel încât valoarea lui n scade pe măsură ce k crește.

Concluzia pe care am vrea să o tragem este că lucrarea de examinare a fiecăruia dintre coșurile pot fi de obicei presupuse proporționale cu dimensiunea fișierului. Putem măsurați astfel timpul de rulare al unui algoritm de obiecte frecvente cu numărul de câte ori este citit fiecare bloc de disc al fișierului de date.

Mai mult, toți algoritmii pe care îi discutăm au proprietatea pe care o citesc fișier coș secvențial. Astfel, algoritmii pot fi caracterizați prin număr de trece prin dosarul coșului pe care îl fac și timpul lor de funcționare este proporțional cu produsul numărului de treceri pe care le fac prin fișier coș de ori dimensiunea fișierului respectiv. Deoarece nu putem controla cantitatea de date, contează doar numărul de treceri luate de algoritm, și asta este aspect al algoritmului pe care ne vom concentra la măsurarea rulării timpul unui algoritm de obiecte frecvente.

6.2.2 Utilizarea memoriei principale pentru numărarea seturilor de articole

Cu toate acestea, trebuie să examinăm o a doua problemă legată de date. Toate algoritmii de obiecte frecvente ne impun să menținem numeroase numere diferite facem o trecere prin date. De exemplu, ar putea fi necesar să numărăm de câte ori apare fiecare pereche de articole în coșuri. Dacă nu avem suficientă memorie principală pentru a stoca fiecare dintre numere, apoi adăugând 1 la o întâmplare numărarea va necesita cel mai probabil să încărcăm o pagină de pe disc. În acest caz, algoritmul va bate și va rula mai multe ordine de mărime mai lent decât dacă am fi sigur că va găsi fiecare număr în memoria principală. Concluzia este că nu putem numărați orice nu se potrivește în memoria principală. Astfel, fiecare algoritm are un limită cu câte articole se poate ocupa.

Exemplul 6.5: Să presupunem că un anumit algoritm trebuie să numere toate perechile de elemente, și există n elemente. Avem astfel nevoie de spațiu pentru stocare ($_{n}$ $_{2}$) numere întregi, sau aproximativ n $_{2}$ /2 numere întregi. Dacă numerele întregi iau 4 octeți, avem nevoie de 2n $_{2}$ octeți. Dacă mașina noastră are 2 gigaocteți sau 2 $_{31}$ octeți de memorie principală, atunci avem nevoie de n ≤ 2 $_{15}$ sau aproximativ n <33.000. ✷

Nu este banal să stochezi ($_{n}$ $_{2}$) contează într-un mod care îl face ușor de găsit numărul pentru o pereche {i, j}. În primul rând, nu am presupus nimic despre cum elementele sunt reprezentate. Ar putea fi, de exemplu, sfori ca „pâinea”. Aceasta este mai eficient din punct de vedere spațial pentru a reprezenta elemente prin numere întregi consecutive de la 1 la n, unde n este numărul de elemente distincte. Cu excepția cazului în care elementele sunt deja reprezentate în acest fel, avem nevoie de un tabel hash care să traducă elementele pe măsură ce apar în fișier la numere întregi. Adică, de fiecare dată când vedem un element în fișier, îl hashăm. Dacă este

---

deja în tabelul hash, putem obține codul său întreg din intrarea sa în masa. Dacă articolul nu este acolo, îi atribuim următorul număr disponibil (de la numărul numărului de articole distincte văzute până acum) și introduceți elementul și elementele acestuia cod în tabel.

Metoda Matricii Triunghiulare

Chiar și după codificarea articolelor ca numere întregi, avem totuși problema pe care trebuie să o avem numărați o pereche {i, j} într-un singur loc. De exemplu, am putea comanda perechea astfel că i <j și folosesc doar intrarea a [i, j] într-o matrice bidimensională a. Acea strategia ar face inutilă jumătate din matrice. O modalitate mai eficientă din punct de vedere spațial este de a utilizați o matrice triunghiulară unidimensională. Stocăm într-un [k] numărul pentru pereche {i, j}, cu 1 ≤ i <j ≤ n, unde

k = (i - 1) (n -
eu
2)
+ j - i

Rezultatul acestui aspect este că perechile sunt stocate în ordine lexicografică, adică este mai întâi {1, 2}, {1, 3}, ..., {1, n}, apoi {2, 3}, {2, 4}, ..., {2, n} și așa mai departe , jos la {n - 2, n - 1}, {n - 2, n} și în final {n - 1, n}.

Metoda Triplurilor
Există o altă abordare a stocării numărărilor care poate fi mai adecvată, în funcție de fracțiunea posibilelor perechi de articole care apar efectiv în ceva coș. Putem stoca numărul ca triplu [i, j, c], adică numărul de perechea {i, j}, cu i <j, este c. O structură de date, cum ar fi un tabel hash cu i și j ca cheie de căutare, este utilizată, astfel încât să putem afla dacă există un triplu pentru un i dat și j și, dacă da, să-l găsesc rapid. Numim această abordare metoda triplelor de stocare contează.
Spre deosebire de matricea triunghiulară, metoda triplurilor nu ne impune stocarea orice dacă numărul pentru o pereche este 0. Pe de altă parte, metoda triplelor ne cere să stocăm trei numere întregi, mai degrabă decât unul, pentru fiecare pereche care o face apar într-un coș. În plus, există spațiul necesar pentru hash tabel sau altă structură de date utilizată pentru a sprijini recuperarea eficientă. Concluzia este că matricea triunghiulară va fi mai bună dacă cel puțin 1/3 din $\binom{n}{2}$ posibil perechile apar de fapt în unele coșuri, în timp ce dacă sunt semnificativ mai mici de 1/3 din apar perechi posibile, ar trebui să luăm în considerare utilizarea metodei triplelor.
Exemplul 6.6: Să presupunem că există 100.000 de articole și 10.000.000 de coșuri de 10 elemente fiecare. Apoi metoda matricei triunghiulare necesită $\binom{100000}{2} = 5 \times 10^9$ (aproximativ) număr întreg. [1] Pe de altă parte, numărul total de perechi dintre toate coșurile este $10^7 \binom{10}{2} = 4{,}5 \times 10^8$. Chiar și în cazul extrem că fiecare pereche de articole a apărut o singură dată, ar putea exista doar $4{,}5 \times 10^8$ perechi cu

[1] Aici și pe tot parcursul capitolului, vom folosi aproximarea că $\binom{n}{2} = n^2/2$ pentru mare n.

---

conturi nenule. Dacă am folosi metoda triplelor pentru a stoca numărul, am avea nevoie doar de trei ori numărul de numere întregi sau $1{,}35 \times 10^9$ numere întregi. Astfel, în aceasta în cazul în care metoda triplelor va ocupa cu siguranță mult mai puțin spațiu decât triunghiulara matrice.
Cu toate acestea, chiar dacă erau de zece sau o sută de ori mai multe coșuri, aceasta ar fi normal să existe o distribuție suficient de inegală a articolelor care s-ar putea să fim mai bine folosind metoda triplelor. Adică ar face unele perechi au un număr foarte mare și numărul de perechi diferite care au apărut într-una sau mai multe coșuri ar fi mult mai mici decât numărul maxim teoretic de astfel de perechi. ✷

6.2.3 Monotonicitatea seturilor de articole

O mare parte din eficacitatea algoritmilor pe care îi vom discuta este condusă de un singur observație, numită monotonitate pentru seturile de obiecte:

- Dacă un set I de articole este frecvent, la fel este și fiecare subset al lui I.

Motivul este simplu. Să fie J ⊆ I. Apoi fiecare coș care conține toate articolele în I conține cu siguranță toate articolele din J. Astfel, numărul pentru J trebuie să fie cel puțin la fel de mare ca numărul pentru I și dacă numărul pentru I este cel puțin s, atunci numărul pentru J este cel puțin s. Deoarece J poate fi cuprins în unele coșuri care lipsesc unul sau mai multe elemente din I - J, este complet posibil ca numărul pentru J să fie strict mai mare decât numărul pentru I.
Pe lângă faptul că algoritmul A-Priori funcționează, monotonicitatea ne oferă o modalitate de compactare a informațiilor despre obiecte frecvente. Dacă ni se dă un prag de suport s, atunci spunem că un set de articole este maxim dacă nu există un superset frecvent. Dacă enumerăm numai seturile maxime de articole, atunci știm că toate subseturile ale unui set maxim de articole sunt frecvente și niciun set care să nu fie un subset al unora setul maxim de articole poate fi frecvent.
Exemplul 6.7: Să reconsiderăm datele din exemplul 6.1 cu ajutorul vechi s = 3. Am constatat că există cinci singletoni frecvenți, cei cu cuvinte ,,Pisică”, „câine”, „a”, „și” și „antrenament”. Fiecare dintre acestea este conținută într-un dubleton frecvent, cu excepția „antrenamentului”, deci un set de obiecte frecvente maxim este {Instruire}. Există, de asemenea, cinci dubletoni frecvenți cu s = 3, și anume
{câine, un} {câine și} {câine, pisică}
{pisică, a}
{pisică și}
Am găsit, de asemenea, un triplu frecvent, {câine, pisică, a} și nu există niciun frecvent mai mare seturi de articole. Astfel, acest triplu este maxim, dar cele trei frecvente îl dublonează

conține nu sunt maxime. Celelalte doubletoni frecvenți, {câine și} și {pisică, și}, sunt maxime. Observați că putem deduce din dubletonii frecvenți că singletonii ca {câine} sunt frecvenți. ✷

6.2.4 Tirania numărării perechilor

După cum probabil ați observat, ne-am concentrat pe problema numărării perechilor în discuția de până acum. Există un motiv bun pentru a face acest lucru: în practică cel mai principal este necesară memorie pentru determinarea perechilor frecvente. Numărul de articole, deși este foarte mare, rareori este atât de mare încât nu putem număra toate singletonul setează în memoria principală în același timp.

Dar seturile mai mari - triplu, cvadruplu și așa mai departe? Amintiți-vă că, în ordine pentru ca analiza seturilor frecvente să aibă sens, rezultatul trebuie să fie un număr mic de seturi, sau nu putem nici măcar să le citim pe toate, să nu mai vorbim de semnificația lor. Astfel, în practică, pragul de sprijin este stabilit suficient de mare încât să fie doar unul rar set care este frecvent. Monotonitatea ne spune că dacă există un triplu frecvent, atunci există trei perechi frecvente conținute în el. Și, desigur, poate exista perechi frecvente conținute în niciun triplu frecvent. Astfel, ne așteptăm să găsim perechi mai frecvente decât tripluri frecvente, tripluri mai frecvente decât frecvente cvadrupluri și așa mai departe.

Acest argument nu ar fi suficient dacă ar fi imposibil să se evite numărarea toate triplele, deoarece există mult mai multe tripluri decât perechi. Este treaba Algoritmul A-Priori și algoritmi asociați pentru a evita numărarea multor triple sau seturi mai mari și, după cum vom vedea, sunt eficiente în acest sens. Astfel, în ce în continuare, ne concentrăm pe algoritmi pentru calculul perechilor frecvente.

6.2.5 Algoritmul A-Priori

Pentru moment, să ne concentrăm pe găsirea numai a perechilor frecvente. Daca avem suficientă memorie principală pentru a număra toate perechile, folosind oricare dintre metodele discutate în secțiunea 6.2.2 (matrice triunghiulară sau triple), atunci este o chestiune simplă de citit fișierul coșurilor într-o singură trecere. Pentru fiecare coș, folosim o buclă dublă pentru generează toate perechile. De fiecare dată când generăm o pereche, adăugăm 1 la numărul ei. La sfârșit, examinăm toate perechile pentru a vedea care au numere egale cu sau mai mare decât pragul de sprijin s; acestea sunt perechile frecvente.

Cu toate acestea, această abordare simplă eșuează dacă există prea multe perechi de elemente pentru a numărați-le pe toate în memoria principală. Algoritmul A-Priori este conceput pentru a reduce numărul de perechi care trebuie numărate, în detrimentul efectuării a două trece peste date, mai degrabă decât o trecere.

Prima trecere a lui A-Priori

În prima trecere, creăm două tabele. Primul tabel, dacă este necesar, se traduce numele articolelor în numere întregi de la 1 la n, așa cum este descris în secțiunea 6.2.2. Celălalt tabelul este o serie de numărătoare; elementul matricii ith contează aparițiile articol numerotat i. Inițial, conturile pentru toate articolele sunt 0.

Pe măsură ce citim coșuri, ne uităm la fiecare articol din coș și îl traducem nume într-un număr întreg. Apoi, vom folosi acel număr întreg pentru a indexa în matricea lui contează și adăugăm 1 la numărul întreg găsit acolo.

Între trecătoarele lui A-Priori

După prima trecere, examinăm numărul articolelor pentru a determina care dintre ele ele sunt frecvente ca singletoni. Ar putea părea surprinzător faptul că mulți singletoni nu sunt frecvente. Dar amintiți-vă că am stabilit pragul suficient de ridicat că nu primim prea multe seturi frecvente; un s tipic ar fi 1% din coșuri. Dacă ne gândim la propriile noastre vizite la un supermarket, cu siguranță cumpărăm anumite lucruri mai mult de 1% din timp: poate lapte, pâine, Cola sau Pepsi, și așa mai departe. Putem chiar crede că 1% dintre clienți cumpără scutece, chiar deși este posibil să nu o facem. Cu toate acestea, multe dintre articolele de pe rafturi sunt cu siguranță nu cumpărat de 1% din clienți: Cremă Caesar Salad Dressing de exemplu.

Pentru a doua trecere a lui A-Priori, creăm o nouă numerotare de la 1 la m pentru doar articolele frecvente. Acest tabel este o matrice indexată de la 1 la n și intrarea pentru i este fie 0, dacă elementul i nu este frecvent, fie un număr întreg unic în intervalul 1 până la m dacă articolul i este frecvent. Ne vom referi la acest tabel drept tabelul cu articole frecvente.

A doua trecere a lui A-Priori

În timpul celei de-a doua treceri, numărăm toate perechile care constau din două frecvente obiecte. Reamintim din secțiunea 6.2.3 că o pereche nu poate fi frecventă decât dacă ambele sale membrii sunt frecvenți. Astfel, nu ne lipsesc perechile frecvente. Spațiul necesar pe a doua trecere este de $2m^2$ octeți, mai degrabă decât $2n^2$ octeți, dacă folosim triunghiularul-metoda matricei pentru numărare. Observați că renumerotarea doar a celor frecvente elemente este necesar dacă dorim să utilizăm o matrice triunghiulară de mărimea potrivită. The set complet de structuri de memorie principală utilizate în prima și a doua trecere este prezentat în Fig. 6.3.

De asemenea, observați că beneficiul eliminării articolelor rare este amplificat; dacă doar jumătate din articole sunt frecvente, avem nevoie de un sfert din spațiu pentru a conta. La fel, dacă folosim metoda triplelor, trebuie să numărăm doar acele perechi de două obiecte frecvente care apar în cel puțin un coș.

Mecanica celei de-a doua treceri este următoarea.

1. Pentru fiecare coș, căutați în tabelul cu articole frecvente pentru a vedea care dintre articolele sale sunt frecvente.
2. Într-o buclă dublă, generați toate perechile de obiecte frecvente din coșul respectiv.
3. Pentru fiecare astfel de pereche, adăugați una la numărul său în structura de date obișnuită numărul de magazine.

În cele din urmă, la sfârșitul celei de-a doua treceri, examinați structura numărărilor până la determinați ce perechi sunt frecvente.

6.2.6 A-Priori pentru toate seturile obiecte frecvente

Aceeași tehnică utilizată pentru găsirea perechilor frecvente fără a număra toate perechile ne permite să găsim seturi obiecte frecvente mai mari, fără un număr exhaustiv al tuturor seturilor. În

---

1
2
Articol
nume
la
numere întregi
n
1
2
Articol
nume
la
numere întregi
n
contează
Articol
obiecte
quent
Fre−
Neutilizat
de perechi
pentru numărătoare
Structură de date
Trecerea 1
Trecerea 2

Figura 6.3: Schema utilizării memoriei principale în timpul celor două treceri ale A-Priori Algoritm

Algoritmul A-Priori, se ia o trecere pentru fiecare set-size k. Dacă nu frecvent sunt găsite seturi de articole de o anumită dimensiune, apoi monotonitatea ne spune că nu poate exista seturi obiecte frecvente mai mari, astfel încât să ne putem opri.

Modelul de deplasare de la o dimensiune k la următoarea dimensiune k + 1 poate fi sumar-marizat astfel. Pentru fiecare dimensiune k, există două seturi de articole:

1. C_k este setul de articole candidate de mărime k - seturile de articole pe care trebuie să le avem numărare pentru a determina dacă acestea sunt de fapt frecvente.
2. L_k este setul de obiecte cu adevărat frecvente de mărimea k.

Modelul de a trece de la un set la următorul și o dimensiune la următoarea este sugerat de Fig. 6.4.

Filtru
C
Construi
L
Filtru
C
Construi
L

Filtru
C
Construi
L
1
1
2
2
3
3
. . .
Perechi de
frecvent
obiecte
Toate
obiecte
obiecte
Frecvent
perechi
Frecvent
Figura 6.4: Algoritmul A-Priori alternează între construirea candidatului seturi și filtrare pentru a găsi cele care sunt cu adevărat frecvente

---

Pagina 248

228
CAPITOLUL 6. ARTICOLE FRECVENTE
Începem cu C_1, care reprezintă toate seturile de articole singulare, adică articolele în sine. Adică, înainte de a examina datele, orice element ar putea fi frecvent până la noi știu. Primul pas de filtrare este de a număra toate articolele și cele a căror numărare este la cel puțin pragul de susținere formează setul L_1 al articolelor frecvente.
Setul C_2 al perechilor candidate este setul de perechi ale căror elemente sunt în L_1; adică sunt obiecte frecvente. Rețineți că nu construim C_2 explicit. Mai degrabă folosim definiția lui C_2 și testăm calitatea de membru în C_2 cu testând dacă ambii membrii săi se află în L_1. A doua trecere a A-Priori Algoritmul numără toate perechile de candidați și determină care apar cel puțin de ori. Aceste perechi formează L_2, perechile frecvente.
Putem urma acest tipar pe cât dorim. Setul C_3 al candidatului triplurile sunt construite (implicit) ca ansamblul triplurilor, dintre care oricare două este a pereche în L_2. Presupunerea noastră despre raritatea seturilor de articole frecvente, subliniată în secțiunea 6.2.4 implică faptul că nu vor exista prea multe perechi frecvente, deci ele poate fi listat într-un tabel de memorie principală. La fel, nu vor fi prea mulți tripluri candidate, deci toate acestea pot fi numărate printr-o generalizare a triplurilor metodă. Adică, în timp ce triplele sunt folosite pentru a număra perechi, am folosi cvadrupluri, constând din cele trei coduri de articole și numărul asociat, atunci când dorim numără tripluri. În mod similar, putem număra seturi de dimensiuni k folosind tupluri cu k + 1 componente, dintre care ultimul este numărul, iar primul k al cărui element este coduri, în ordine sortată.
Pentru a găsi L_3 facem o a treia trecere prin fișierul coșului. Pentru fiecare coș, trebuie doar să ne uităm la acele elemente care se află în L_1. Din aceste elemente, putem examinați fiecare pereche și determinați dacă acea pereche este sau nu în L_2. Orice articol a coșului care nu apare în cel puțin două perechi frecvente, ambele constau din articole din coș, nu pot face parte dintr-un triplu frecvent pe care coșul conține. Astfel, avem o căutare destul de limitată de triple care sunt ambele conținute în coș și sunt candidați în C_3. Orice astfel de tripluri găsite au Am adăugat la numărul lor.
Exemplul 6.8: Să presupunem că coșul nostru este format din articolele de la 1 la 10. Din acestea, 1 până la 5 s-au dovedit a fi obiecte frecvente, iar următoarele perechi au au fost găsite frecvente: {1, 2}, {2, 3}, {3, 4} și {4, 5}. La început, eliminăm elemente care nu sunt frecvente, lăsând doar 1 până la 5. Cu toate acestea, 1 și 5 apar numai în o pereche frecventă în setul de articole și, prin urmare, nu poate contribui la o frecvență triplu conținut în coș. Astfel, trebuie să luăm în considerare adăugarea numărului de tripluri care sunt conținute în {2, 3, 4}. Desigur, există doar un astfel de triplu.
Cu toate acestea, nu o vom găsi în C_3, deoarece {2, 4} evident nu este frecventă. ✷
Construirea colecțiilor de obiecte și candidați frecvenți mai mari continuă în esență în același mod, până când la un pas nu găsim nimic nou obiecte frecvente și opriți. Acesta este:
1. Definiți C_k pentru a fi toate acele seturi de mărimi k, fiecare k - 1 dintre care este un set de articole în L_{k-1}.

Pagina 249

6.2. COȘURI DE PIAȚĂ ȘI ALGORITMUL A-PRIORI
229

2. Găsiți L_k făcând o trecere prin coșuri și numărând toate și numai
seturile de mărimi k care se află în C_k . Acele seturi de articole care au contat la
cel mai puțin s sunt în L_k .

6.2.7 Exerciții pentru secțiunea 6.2

Exercițiul 6.2.1: Dacă folosim o matrice triunghiulară pentru a număra perechi și n, numărul
un număr de articole, este 20, ce număr de perechi este într-un [100]?
! Exercițiul 6.2.2: În descrierea noastră a metodei matricei triunghiulare din secțiunea
cțiunea 6.2.2, formula pentru k implică împărțirea unui număr întreg i arbitrar la 2. Cu toate acestea
trebuie să avem k întotdeauna un număr întreg. Dovediți că k va fi, de fapt, un număr întreg.
! Exercițiul 6.2.3: Să existe I articole într-un set de date din coșul de piață al B coșuri.
Să presupunem că fiecare coș conține exact K articole. În funcție de I, B,
și K:
(a) Cât spațiu ocupă metoda matricei triunghiulare pentru a stoca
contează toate perechile de articole, presupunând patru octeți per element matrice?
(b) Care este cel mai mare număr posibil de perechi cu un număr diferit de zero?
(c) În ce circumstanțe putem fi siguri că metoda triplelor va fi
folosiți mai puțin spațiu decât matricea triunghiulară?
!! Exercițiul 6.2.4: Cum ați număra toate seturile de elemente de mărimea 3 printr-o generalizare
a metodei matricei triunghiulare? Adică, aranjați-l într-o singură dimensiune
matrice există exact un element pentru fiecare set de trei elemente.
! Exercițiul 6.2.5: Să presupunem că pragul de sprijin este 5. Găsiți frecvența maximă
seturi de articole pentru datele de:
(a) Exercițiul 6.1.1.
(b) Exercițiul 6.1.3.
Exercițiul 6.2.6: Aplicați algoritmul A-Priori cu pragul de sprijin 5 la
datele despre:
(a) Exercițiul 6.1.1.
(b) Exercițiul 6.1.3.
! Exercițiul 6.2.7: Să presupunem că avem coșuri de piață care satisfac următoarele
ipoteze:
1. Pragul de suport este de 10.000.
2. Există un milion de articole, reprezentate de numerele întregi 0, 1, ..., 999999.

Pagina 250

3. Există N obiecte frecvente, adică obiecte care apar de 10.000 ori sau
Mai Mult.
4. Există un milion de perechi care apar de 10.000 de ori sau mai mult.
5. Există 2 milioane de perechi care apar exact o singură dată. Dintre aceste perechi, M constă din
două obiecte frecvente; celelalte M au fiecare cel puțin un articol non-frecvent.
6. Nu apar deloc alte perechi.
7. Numerele întregi sunt întotdeauna reprezentate de 4 octeți.
Să presupunem că rulăm algoritmul A-Priori și putem alege pentru a doua trecere
între metoda triunghiulară-matrice pentru numărarea perechilor candidate și un hash
tabel cu triple-element-item-count. Neglijați, în primul caz, spațiul necesar
traduceți între numerele originale ale articolelor și numerele pentru articolele frecvente,
iar în al doilea caz neglijează spațiul necesar pentru masa hash. Ca o funcție
din N și M, care este numărul minim de octeți de memorie principală necesar
executați algoritmul A-Priori pe aceste date?

6.3 Gestionarea seturilor de date mai mari în memoria principală

Algoritmul A-Priori este bine atâta timp cât pasul cu cea mai mare cerință
pentru memoria principală - de obicei numărarea perechilor candidate C_2 - are suficient
memorie că poate fi realizată fără zdrobire (mișcare repetată a
date între disc și memorie principală). Au fost propuși mai mulți algoritmi
pentru a reduce dimensiunea setului de candidați C_2 . Aici, considerăm PCY
Algoritm, care profită de faptul că în prima trecere a lui A-Priori
există de obicei o mulțime de memorie principală care nu este necesară pentru numărarea single-ului
obiecte. Apoi ne uităm la algoritmul cu mai multe etape, care folosește trucul PCY
și introduce, de asemenea, treceri suplimentare pentru a reduce în continuare dimensiunea C_2 .

6.3.1 Algoritmul lui Park, Chen și Yu

Acest algoritm, pe care îl numim PCY după autorii săi, exploatează observația
că poate exista mult spațiu neutilizat în memoria principală la prima trecere. În cazul în care există
sunt un milion de articole și gigaocteți de memorie principală, nu avem nevoie de mai mult de
10% din memoria principală pentru cele două tabele sugerate în Fig. 6.3 - o traducere
tabel de la nume de articole la numere întregi mici și o matrice pentru a număra aceste numere întregi.
Algoritmul PCY folosește acel spațiu pentru o serie de numere întregi care generalizează

ideea unui filtru Bloom (vezi secțiunea 4.3). Ideea este prezentată schematic în
Fig. 6.5.
Gândiți-vă la această matrice ca la un tabel hash, ale cărui găleți conțin mai degrabă numere întregi decât
seturi de taste (ca într-un tabel hash obișnuit) sau biți (ca într-un filtru Bloom). Perechi de
articolele sunt hashate în găleți ale acestui tabel hash. În timp ce examinăm un coș în timpul
prima trecere, nu numai că adăugăm 1 la numărul pentru fiecare articol din coș, ci

---

1
2
Articol
nume
la
numere întregi
n
1
2
Articol
nume
la
numere întregi
n
contează
Articol
obiecte
quent
Fre−
Trecerea 1
Trecerea 2
de perechi
pentru numărătoare
Structură de date
Bitmap
contează
pentru găleată
Masă Hash
Figura 6.5: Organizarea memoriei principale pentru primele două treceri ale PCY
Algoritm
generăm toate perechile, folosind o buclă dublă. Hashem fiecare pereche și adăugăm
1 la găleată în care se îmbracă perechea respectivă. Rețineți că perechea în sine nu
du-te în găleată; perechea afectează numai singurul întreg din cupă.
La sfârșitul primei treceri, fiecare găleată are un număr, care este suma
numărătoarea tuturor perechilor care au hash la găleată. Dacă numărul unei găleți
este cel puțin la fel de mare ca pragul de sprijin s, se numește cupă frecventă.
Nu putem spune nimic despre perechile care sunt hash la o găleată frecventă; ei ar putea
toate să fie perechi frecvente din informațiile disponibile. Dar dacă numărul de
găleată este mai mică decât s (o găleată rară), nu cunoaștem nicio pereche care să hasheze
la această găleată poate fi frecventă, chiar dacă perechea constă din două obiecte frecvente.
Acest fapt ne oferă un avantaj la a doua pasă. Putem defini setul de
perechile candidate C_2 să fie acele perechi {i, j} astfel încât:
1. i și j sunt obiecte frecvente.
2. {i, j} hashează la o găleată frecventă.
Este a doua condiție care distinge PCY de A-Priori.
Exemplul 6.9: În funcție de date și de cantitatea de memorie principală disponibilă
ory, poate fi sau nu un avantaj al utilizării tabelului hash la pasul 1. În
în cel mai rău caz, toate gălețile sunt frecvente, iar algoritmul PCY contează exact
aceleași perechi ca și A-Priori la a doua pasă. Cu toate acestea, uneori, noi
se poate aștepta ca majoritatea găleților să fie rare. În acest caz, PCY reduce
cerințele de memorie ale celei de-a doua treceri.

Să presupunem că avem un gigabyte de memorie principală disponibil pentru tabelul hash
la prima trecere. Să presupunem, de asemenea, că fișierul de date are câte un miliard de coșuri, fiecare
cu zece obiecte. O bucket este un număr întreg, de obicei 4 octeți, deci putem menține un
un sfert de miliard de găleți. Numărul de perechi din toate coșurile este de $10^9 \times \binom{10}{}$

^{2})
sau 4,5 × 10 ^{10} perechi; acest număr este, de asemenea, suma numărărilor din găleți.
Astfel, numărul mediu este de 4,5 × 10 ^{10} / 2,5 × 10 ^{8} , sau 180. Dacă pragul de sprijin
s este în jur de 180 sau mai puțin, ne-am putea aștepta ca puține găleți să fie rare. In orice caz,
dacă s este mult mai mare, să zicem 1000, atunci trebuie să fie că marea majoritate a
gălețile sunt rare. Cel mai mare număr posibil de găleți frecvente este
4,5 x 10 ^{10} /1000, sau 45 de milioane din cele 250 de milioane de găleți. ✷
Între trecerile PCY, tabelul hash este rezumat ca o bitmap, cu
câte un bit pentru fiecare găleată. Bitul este 1 dacă cupa este frecventă și 0 dacă nu. Prin urmare
numerele întregi de 32 de biți sunt înlocuite cu biți simpli, iar harta de biți afișată în
a doua trecere din Fig. 6.5 ocupă doar 1/32 din spațiul care altfel ar fi
disponibil pentru a stoca numărul. Cu toate acestea, dacă majoritatea găleților sunt rare, ne așteptăm
că numărul de perechi numărate la a doua trecere va fi mult mai mic
decât numărul total de perechi de articole frecvente. Astfel, PCY poate gestiona unele
seturi de date fără a bate în timpul celei de-a doua pase, în timp ce A-Priori ar alerga
din memoria principală și thrash.
Există o altă subtilitate cu privire la a doua trecere a PCY care afectează
cantitatea de spațiu necesară. În timp ce am reușit să folosim matricea triunghiulară
metodă la a doua trecere a lui A-Priori, dacă am dori, deoarece elementele frecvente
poate fi renumerotat de la 1 la câțiva m, nu putem face acest lucru pentru PCY. Motivul
este că sunt plasate perechile de articole frecvente pe care PCY ne permite să le evităm
aleatoriu în matricea triunghiulară; sunt perechile care se întâmplă cu hash
la o găleată rară la prima trecere. Nu există un mod cunoscut de compactare
matricea pentru a evita lăsarea spațiului pentru perechile nenumărate.
În consecință, suntem forțați să folosim metoda triplelor în PCY. Care re-
strict ția nu poate conta dacă fracția de perechi de obiecte frecvente care de fapt
apar în găleți erau mici; am vrea apoi să folosim triple pentru A-Priori
oricum. Cu toate acestea, dacă cele mai multe perechi de articole frecvente apar împreună în cel puțin
o găleată, atunci suntem obligați în PCY să folosim triple, în timp ce A-Priori poate folosi a
matricea triunghiulară. Astfel, dacă PCY nu ne permite să numărăm cel puțin 2/3 din
perechi de obiecte frecvente, nu putem câștiga folosind PCY în loc de A-Priori.
În timp ce descoperirea perechilor frecvente de către PCY diferă semnificativ de A-
Priori, etapele ulterioare, unde găsim tripluri frecvente și seturi mai mari, dacă se dorește,
sunt în esență aceleași ca A-Priori. Această afirmație este valabilă și pentru fiecare dintre ele
îmbunătățirile aduse A-Priori pe care le acoperim în această secțiune. Drept urmare, noi
va acoperi doar construcția perechilor frecvente de aici înainte.

6.3.2 Algoritmul cu mai multe etape

Algoritmul cu mai multe etape îmbunătățește PCY utilizând mai multe hash-uri succesive
tabele pentru a reduce în continuare numărul de perechi de candidați. Compensația este că

Multistage necesită mai mult de două pase pentru a găsi perechile frecvente. O schiță a
Algoritmul cu mai multe etape este prezentat în Fig. 6.6.
1
2
Articol
nume
la
numere întregi
n
obiecte
quent
Fre−
1
2
Articol
nume
la
numere întregi
n
contează
Articol
Trecerea 1
contează
pentru găleată
Masă Hash
1
2

Articol
nume
la
numere întregi
n
obiecte
quent
Fre−
Bitmap 1
Bitmap 2
Structură de date
pentru numărătoare
de perechi
Trecerea 2
Bitmap 1
contează
pentru găleată
Al doilea
masa hash
Trecerea 3
Figura 6.6: Algoritmul cu mai multe etape utilizează tabele hash suplimentare pentru a reduce
numărul de perechi de candidați
Prima trecere a Multistage este aceeași cu prima trecere a PCY. După care
treceți, gălețile frecvente sunt identificate și sintetizate printr-o hartă bit, din nou
la fel ca în PCY. Dar a doua trecere a Multistage nu contează
perechi candidate. Mai degrabă, folosește memoria principală disponibilă pentru un alt hash
tabel, folosind o altă funcție hash. De când bitmap-ul din primul tabel hash
ocupă 1/32 din memoria principală disponibilă, al doilea tabel hash are aproape
la fel de multe găleți ca prima.
La a doua trecere a Multistage, trecem din nou prin fișierul coșurilor.
Nu este nevoie să numărăm din nou articolele, deoarece avem aceste numere de la
prima trecere. Cu toate acestea, trebuie să păstrăm informațiile despre care sunt articolele
frecvente, deoarece avem nevoie atât la a doua, cât și la a treia trecere. In timpul
a doua trecere, vom hash anumite perechi de articole la găleți din al doilea tabel hash.
O pereche este hashată numai dacă îndeplinește cele două criterii pentru a fi numărată în a doua
trecere de PCY; adică, vom hash {i, j} dacă și numai dacă i și j sunt ambii frecvenți,
iar perechea a ajuns la o găleată frecventă la prima trecere. Ca urmare, suma
dintre numărul din al doilea tabel hash ar trebui să fie semnificativ mai mic decât suma
pentru prima trecere. Rezultatul este că, chiar dacă al doilea tabel hash are
ne așteptăm doar 31/32 din numărul de găleți pe care le are primul tabel
să fie mult mai puține găleți frecvente în al doilea tabel hash decât în primul.
După a doua trecere, al doilea tabel hash este, de asemenea, rezumat ca o hartă de biți,
iar acea bitmap este stocată în memoria principală. Cele două bitmap-uri împreună ocupă

O eroare subtilă în mai multe etape

Ocazional, o implementare încearcă să elimine a doua cerință
pentru ca {i, j} să fie un candidat - că se hash la o găleată frecventă pe primul
trece. Raționamentul (fals) este că, dacă nu a fost hash la o găleată frecventă
la prima pasă, nu ar fi fost deloc hashată la a doua pasă,
și astfel nu ar contribui la numărarea găleții sale pe a doua
trece. Deși este adevărat că perechea nu este luată în considerare la a doua trecere, asta
nu înseamnă că n-ar fi ajuns la o găleată frecventă dacă ar fi fost
hash. Astfel, este complet posibil ca {i, j} să fie format din două frecvente
obiecte și hashuri la o găleată frecventă la a doua trecere, dar nu a făcut-o
hash la o găleată frecventă la prima trecere. Prin urmare, toate cele trei condiții
trebuie verificat pe trecerea de numărare a Multistage.
puțin mai puțin de 1/16 din memoria principală disponibilă, deci există încă o mulțime
spațiu pentru a număra perechile candidate la a treia trecere. O pereche {i, j} este în C_2 dacă
și numai dacă:
1. i și j sunt ambele elemente frecvente.
2. {i, j} hash la o cupă frecventă în primul tabel hash.
3. {i, j} hash la o cupă frecventă în al doilea tabel hash.
A treia condiție este distincția între Multistage și PCY.
Ar putea fi evident că este posibil să introduceți orice număr de treceri între
primul și ultimul din algoritmul cu mai multe etape. Există un factor limitativ care
fiecare trecere trebuie să stocheze bitmap-urile din fiecare dintre trecerile anterioare. În cele din urmă,
nu mai este suficient spațiu în memoria principală pentru a face numărările. Nu conteaza

câte treceri folosim, cuplurile cu adevărat frecvente vor fi întotdeauna hash la un frecvent
găleată, deci nu există nicio modalitate de a evita numărarea lor.

6.3.3 Algoritmul Multihash

Uneori, putem obține cea mai mare parte a beneficiilor permiselor suplimentare de pe mai multe etape
Algoritm într-o singură trecere. Această variație a PCY se numește Multihash
Algoritm. În loc să utilizați două tabele hash diferite pe două pase succesive,
utilizați două funcții hash și două tabele hash separate care partajează memoria principală
la prima trecere, așa cum sugerează Fig. 6.7.
Pericolul utilizării a două tabele hash pe o singură trecere este că fiecare tabel hash are
jumătate de găleată decât cea mare masă de hash din PCY. Atâta timp cât media
Numărul unei cupe pentru PCY este mult mai mic decât pragul de sprijin, putem
operați două mese de hașe pe jumătate și vă așteptați în continuare la majoritatea găleților ambelor
tabelele hash să fie rare. Astfel, în această situație am putea alege foarte bine
abordare multihash.

1
2
Articol
nume
la
numere întregi
n
1
2
Articol
nume
la
numere întregi
n
contează
Articol
obiecte
quent
Fre−
Trecerea 1
Trecerea 2
de perechi
pentru numărătoare
Structură de date
Bitmap 1 Bitmap 2
Hash Tabelul 2
Hash Tabelul 1
Figura 6.7: Algoritmul Multihash folosește mai multe tabele hash într-o singură trecere
Exemplul 6.10: Să presupunem că, dacă rulăm PCY, bucket-ul mediu va avea un
număr s / 10, unde s este pragul de sprijin. Apoi, dacă am folosit Multihash
abordare cu două tabele hash de jumătate, numărul mediu ar fi s / 5. La fel de
un rezultat, cel mult 1/5 din gălețile din oricare dintre tabelele de hash ar putea fi frecvente,
iar o pereche aleatorie rare are cel mult probabilitatea $(1/5)^2 = 0,04$ de a fi în
o găleată frecventă în ambele mese de hash.
Prin același raționament, limita superioară a perechii rare fiind într-un
găleată frecventă în tabelul de hash PCY este cel mult 1/10. Adică am putea
așteptați să numărați de 2,5 ori mai multe perechi rare în PCY decât în
versiunea Multihash sugerată mai sus. Prin urmare, ne-am aștepta ca Multihash să o facă
au o cerință de memorie mai mică pentru a doua trecere decât ar fi PCY.
Dar aceste limite superioare nu spun povestea completă. Pot exista
mult mai puține găleți frecvente decât maximul pentru oricare algoritm, din moment ce
prezența unor perechi foarte frecvente va distorsiona distribuția numărului de găleți.
Cu toate acestea, această analiză sugerează posibilitatea ca pentru unele date și
acceptă praguri, putem face mai bine executând mai multe funcții hash în principal
memoria deodată. ✷
Pentru a doua trecere a Multihash, fiecare tabel hash este convertit într-o hartă de biți,
ca de obicei. Rețineți că cele două bitmap-uri pentru cele două funcții hash din Fig. 6.7
ocupă exact la fel de mult spațiu ca o singură bitmap pentru a doua trecere a
algoritmul PCY. Condițiile pentru ca o pereche {i, j} să fie în C_2 și astfel
pentru a solicita un număr pe a doua trecere, sunt aceleași ca pentru a treia trecere de
Multietaj: i și j trebuie să fie frecvente, iar perechea trebuie să aibă hash la
o găleată frecventă conform ambelor tabele hash.

Așa cum Multistage nu se limitează la două tabele hash, putem împărți memoria principală disponibilă în câte tabele hash dorim în prima trecere a Multihash. Riscul este că ar trebui să folosim prea multe tabele hash, în medie numărarea pentru o găleată va depăși pragul de sprijin. În acel moment, poate să fie foarte puține găleți rare în oricare dintre mesele hash. Chiar dacă o pereche s-ar putea să găsim un hash la o găleată frecventă în fiecare tabel de hash care trebuie numărat că probabilitatea ca o pereche rară să fie un candidat crește, mai degrabă decât cade, dacă adăugăm un alt tabel hash.

6.3.4 Exerciții pentru secțiunea 6.3

Exercițiul 6.3.1: Iată o colecție de douăsprezece coșuri. Fiecare conține trei dintre cele șase elemente de la 1 la 6.

{1, 2, 3} {2, 3, 4} {3, 4, 5} {4, 5, 6}
{1, 3, 5} {2, 4, 6} {1, 3, 4} {2, 4, 5}
{3, 5, 6} {1, 2, 4} {2, 3, 5} {3, 4, 6}

Să presupunem că pragul de suport este 4. La prima trecere a algoritmului PCY folosim o masă hash cu 11 găleți, iar setul {i, j} este hash la cupa i × j mod 11.

(a) Prin orice metodă, calculați suportul pentru fiecare articol și fiecare pereche de articole.
(b) Ce perechi de hash la ce găleți?
(c) Care găleți sunt frecvente?
(d) Care perechi sunt numărate la a doua trecere a algoritmului PCY?

Exercițiul 6.3.2: Să presupunem că rulăm algoritmul cu mai multe etape pe datele de Exercițiul 6.3.1, cu același prag de sprijin de 4. Prima trecere este aceeași ca în acel exercițiu, și pentru a doua trecere, vom hash perechi la nouă găleți, folosind funcția hash care hashes {i, j} la cupa i + j mod 9. Determinați numărătoarele găleților la a doua trecere. A doua trecere reduce set de perechi de candidați? Rețineți că toate articolele sunt frecvente, deci singurul motiv a perechea nu ar fi hash la a doua trecere, dacă este hash la o rară găleată la prima trecere.

Exercițiul 6.3.3: Să presupunem că rulăm algoritmul Multihash pe datele de Exercițiul 6.3.1. Vom folosi două mese hash cu câte cinci găleți. Pentru un, setul {i, j}, este hash la cupa 2i + 3j + 4 mod 5, iar pentru cealaltă, setul este hash to i + 4j mod 5. Deoarece aceste funcții hash nu sunt simetrice în i și j, ordonați elementele astfel încât i <j când evaluați fiecare funcție hash. A determina numărul fiecăreia dintre cele 10 găleți. Cât de mare este pragul de sprijin trebuie să fie pentru ca algoritmul cu mai multe etape să elimine mai multe perechi decât PCY Algoritmul ar folosi, folosind tabelul hash și funcția descrisă în exercițiul 6.3.1?

! Exercițiul 6.3.4: Să presupunem că executăm algoritmul PCY pentru a găsi frecvent perechi, cu date despre coșul de piață care îndeplinesc următoarele specificații:

1. Pragul de suport este de 10.000.
2. Există un milion de articole, reprezentate de numerele întregi 0, 1, ..., 999999.
3. Există 250.000 de articole frecvente, adică articole care apar de 10.000 de ori sau mai mult.
4. Există un milion de perechi care apar de 10.000 de ori sau mai mult.
5. Există P perechi care apar exact o dată și constau din două frecvente obiecte.
6. Nu apar deloc alte perechi.
7. Numerele întregi sunt întotdeauna reprezentate de 4 octeți.
8. Când vom împerechea perechi, acestea se distribuie între găleți la întâmplare, dar ca cât mai uniform posibil; adică, puteți presupune că fiecare găleată devine exact a sa cota echitabilă a perechilor P care apar o singură dată.

Să presupunem că există S octeți ai memoriei principale. Pentru a rula algoritmul PCY cu succes, numărul de găleți trebuie să fie suficient de mare încât majoritatea găleților nu sunt frecvente. În plus, la a doua trecere, trebuie să fie suficient spațiu să numere toate perechile candidate. În funcție de S, care este cea mai mare valoare din P pentru care putem rula cu succes algoritmul PCY pe aceste date?

! Exercițiul 6.3.5: În conformitate cu ipotezele date în exercițiul 6.3.4, Mul-Algoritmul tihash reduce cerințele de memorie principală pentru a doua trecere?

În funcție de S și P, care este numărul optim de tabele hash de utilizat
la prima trecere?
! Exercițiul 6.3.6: Să presupunem că executăm algoritmul cu mai multe etape în 3 treceri pentru a găsi
perechi frecvente, cu date despre coșul de piață care îndeplinesc următoarele specificații:
1. Pragul de suport este de 10.000.
2. Există un milion de articole, reprezentate de numerele întregi 0, 1, ..., 999999.
Toate articolele sunt frecvente; adică apar de cel puțin 10.000 de ori.
3. Există un milion de perechi care apar de 10.000 de ori sau mai mult.
4. Există P perechi care apar exact o singură dată.
5. Nu apar deloc alte perechi.
6. Numerele întregi sunt întotdeauna reprezentate de 4 octeți.

Pagina 258

7. Când vom împerechea perechi, acestea se distribuie între găleți la întâmplare, dar ca
cât mai uniform posibil; adică, puteți presupune că fiecare găleată devine exact a sa
cota echitabilă a perechilor P care apar o singură dată.
8. Funcțiile hash utilizate la primele două treceri sunt complet independente
adâncitură.
Să presupunem că există S octeți ai memoriei principale. În funcție de S și P, ce
este numărul așteptat de perechi de candidați la a treia trecere a Multistage
Algoritm?

6.4 Algoritmi de trecere limitată

Algoritmii pentru seturile de articole frecvente discutate până acum utilizează o trecere pentru fiecare dimensiune
a obiectelor pe care le investigăm. Dacă memoria principală este prea mică pentru a păstra datele și fișierul
spațiul necesar pentru a număra obiecte frecvente de o mărime, nu pare să existe
orice modalitate de a evita k trece pentru a calcula colecția exactă de obiecte frecvente.
Cu toate acestea, există multe aplicații în care nu este esențial să descoperiți fiecare
obiecte frecvente. De exemplu, dacă căutăm articole cumpărate împreună de la
un supermarket, nu vom desfășura o vânzare bazată pe fiecare set de articole frecvente
găsim, deci este destul de suficient să găsim majoritatea, dar nu toate seturile obișnuite.
În această secțiune explorăm câțiva algoritmi care au fost propuși să fie găsiți
toate sau cele mai frecvente seturi de articole folosind cel mult două pase. Începem cu
abordare evidentă a utilizării unui eșantion de date, mai degrabă decât a întregului set de date.
Un algoritm numit SON folosește două treceri, primește răspunsul exact și împrumută
la implementarea prin MapReduce sau un alt regim de calcul paralel.
În cele din urmă, algoritmul lui Toivonen folosește în medie două pase, primește un răspuns exact,
dar, rareori, nu se poate termina într-o anumită perioadă de timp.

6.4.1 Algoritmul simplu, randomizat

În loc să folosim întregul fișier de coșuri, am putea alege un subset aleatoriu de
coșurile și pretindeți că este întregul set de date. Trebuie să ajustăm suportul
prag pentru a reflecta numărul mai mic de coșuri. De exemplu, dacă suportul
pragul pentru setul de date complet este s și alegem un eșantion de 1% din coșuri,
atunci ar trebui să examinăm eșantionul pentru seturile de articole care apar în cel puțin s / 100
a coșurilor.
Cel mai sigur mod de a alege proba este de a citi întregul set de date și pentru fiecare
coș, selectați coșul respectiv pentru eșantion cu o probabilitate fixă p. Presupune
există m coșuri în întregul fișier. La sfârșit, vom avea un eșantion al cărui
dimensiunea este foarte aproape de coșurile pm. Cu toate acestea, dacă avem motive să credem că
coșurile apar deja în ordine aleatorie în fișier, apoi nici măcar nu avem
pentru a citi întregul fișier. Putem selecta primele coșuri pm pentru eșantionul nostru. Sau daca
fișierul face parte dintr-un sistem de fișiere distribuit, putem alege câteva bucăți la întâmplare
să servească drept probă.

Pagina 259

De ce nu alegeți doar prima parte a fișierului?

Riscul în selectarea unui eșantion dintr-o porțiune a unui fișier mare este ca
datele nu sunt distribuite uniform în fișier. De exemplu, să presupunem că fișierul
au fost o listă cu adevărat conținut de coș de piață la un magazin universal, organizat
după data vânzării. Dacă ați lua doar primele coșuri din fișier, ați face-o
au date vechi. De exemplu, nu ar exista iPod-uri în aceste coșuri,
chiar dacă iPod-urile ar fi putut deveni un element popular mai târziu.

Ca un alt exemplu, luați în considerare un dosar de teste medicale efectuate la
diferite spitale. Dacă fiecare bucată provine dintr-un spital diferit, atunci
alegerea bucăților la întâmplare ne va oferi un eșantion extras doar dintr-un mic
subset al spitalelor. Dacă spitalele efectuează diferite teste sau le efectuează
în diferite moduri, datele pot fi extrem de părtinitoare.
După ce am selectat eșantionul coșurilor, folosim o parte din memoria principală
pentru a depozita aceste cosuri. Echilibrul memoriei principale este folosit pentru a executa
unul dintre algoritmii pe care i-am discutat, cum ar fi A-Priori, PCY, Multistage,
sau Multihash. Cu toate acestea, algoritmul trebuie să ruleze treceri peste memoria principală
eșantion pentru fiecare dimensiune set de articole, până când găsim o dimensiune fără elemente frecvente. Acolo
nu sunt necesare accesări pe disc pentru a citi eșantionul, deoarece acesta se află în memoria principală.
Pe măsură ce sunt descoperite obiecte frecvente de fiecare dimensiune, acestea pot fi scrise pe disc;
această operație și citirea inițială a eșantionului de pe disc sunt singurul disc
Algoritmul face I / O.
Desigur, algoritmul va eșua dacă oricare dintre metodele din secțiunea 6.2 sau 6.3
alegem că nu poate fi rulat în cantitatea de memorie principală rămasă după stocarea fișierului
probă. Dacă avem nevoie de mai multă memorie principală, atunci o opțiune este să citim eșantionul
de pe disc pentru fiecare trecere. Deoarece eșantionul este mult mai mic decât setul de date complet,
evităm totuși majoritatea discurilor I / O pe care le-au discutat algoritmii anterior
ar folosi.

6.4.2 Evitarea erorilor în algoritmii de eșantionare

Ar trebui să fim conștienți de problema cu algoritmul simplu din secțiunea 6.4.1:
nu se poate baza nici pentru a produce toate elementele care sunt frecvente în
întregul set de date și nici nu va produce numai seturi de articole care sunt frecvente în
întreg. Un set de articole care este frecvent în ansamblu, dar nu în eșantion, este fals
negativ, în timp ce un set de articole care este frecvent în eșantion, dar nu în ansamblu, este un
fals pozitiv.
Dacă eșantionul este suficient de mare, este puțin probabil să existe erori grave. Acesta este,
un set de articole al cărui suport este mult mai mare decât pragul va fi aproape sigur
să fie identificat dintr-un eșantion aleatoriu și un set de articole al cărui suport este mult mai mic
decât pragul este puțin probabil să apară frecvent în eșantion. Cu toate acestea, un
elemente al căror sprijin în ansamblu este foarte aproape de prag este la fel de probabil

---

să fie frecvent în eșantion ca nu.
Putem elimina falsurile pozitive făcând o trecere prin setul de date complet
și numărarea tuturor seturilor de articole care au fost identificate ca frecvente în eșantion.
Păstrați la fel de frecvent numai acele seturi de articole care au fost frecvente în eșantion și
de asemenea frecvent în ansamblu. Rețineți că această îmbunătățire va elimina toate falsurile
pozitive, dar un fals negativ nu este numărat și, prin urmare, rămâne nedescoperit.
Pentru a îndeplini această sarcină într-o singură trecere, trebuie să putem conta toate
obiecte frecvente de toate dimensiunile simultan, în memoria principală. Dacă am putea
rulați algoritmul simplu cu memoria principală disponibilă, apoi
există mari șanse să putem conta toate seturile de articole frecvente la
o dată, pentru că:
(a) Singletonii și perechile frecvente ar putea domina colecția de
toate obiecte frecvente și a trebuit să le numărăm pe toate într-o singură trecere.
(b) Acum avem toată memoria principală disponibilă, deoarece nu trebuie să stocăm
proba din memoria principală.
Nu putem elimina complet negativele false, dar le putem reduce
numărul dacă memoria principală o permite. Am presupus că dacă s este
pragul de suport, iar eșantionul este fracțiunea p din întregul set de date, atunci noi
utilizați ps ca prag de susținere pentru eșantion. Cu toate acestea, putem folosi ceva
mai mic decât cel al pragului pentru eșantion, un astfel de 0.9ps. Având o mai mică
prag înseamnă că vor trebui numărate mai multe seturi de articole din fiecare dimensiune, deci
cerința de memorie principală crește. Pe de altă parte, dacă este suficient
memorie principală, atunci ne vom identifica ca având suport de cel puțin 0.9ps în
eșantion aproape toate acele seturi de articole care au suport cel puțin s este întregul. Dacă
facem apoi o trecere completă pentru a elimina acele seturi de articole care au fost identificate
la fel de frecvente în eșantion, dar nu sunt frecvente în ansamblu, nu avem fals
pozitive și, sperăm, să nu aibă niciunul sau foarte puține negative false.

6.4.3 Algoritmul lui Savasere, Omiecinski și Navathe

Următoarea noastră îmbunătățire evită atât falsurile negative, cât și falsele pozitive, la
costul efectuării a două treceri complete. Se numește algoritmul SON după autori.

Ideea este să împărțiți fișierul de intrare în bucăți (care pot fi „bucăți” în
sensul unui sistem de fișiere distribuit sau pur și simplu o bucată din fișier). Tratează-le pe fiecare
bucată ca un eșantion și rulați algoritmul Secțiunii 6.4.1 pe acea bucată. Noi
folosiți ps ca prag, dacă fiecare bucată este fracția p din întregul fișier și s este
pragul de sprijin. Stocați pe disc toate seturile de articole frecvente găsite pentru fiecare
bucată mare.
Odată ce toate bucățile au fost procesate în acest fel, luați uniunea de
toate seturile de articole care au fost găsite frecvente pentru una sau mai multe bucăți. Aceste
sunt elementele candidate. Observați că dacă un set de articole nu este frecvent în niciunul
bucată, atunci suportul său este mai mic de ps în fiecare bucată. Deoarece numărul de

---

Pagina 261

bucăți este 1 / p, concluzionăm că suportul total pentru acel set de articole este mai mic de
(1 / p) ps = s. Astfel, fiecare set de articole care este frecvent în ansamblu este frecvent în
cel puțin o bucată și putem fi siguri că toate elementele cu adevărat frecvente sunt
printre candidați; adică, nu există negative negative.
Am făcut un total de o trecere prin date pe măsură ce citim fiecare bucată
și a procesat-o. Într-o a doua trecere, numărăm toate seturile de articole candidate și
selectați cele care au suport cel puțin s ca elemente frecvente.

6.4.4 Algoritmul SON și MapReduce

Algoritmul SON se pretează bine unui mediu de calcul paralel. Fiecare
din bucăți pot fi procesate în paralel, iar elementele frecvente din fiecare
bucăți combinate pentru a forma candidații. Putem distribui candidații către
mulți procesoare, solicitați fiecărui procesor să numere suportul pentru fiecare candidat
într-un subset de coșuri și, în cele din urmă, rezumați acele suporturi pentru a obține sprijinul
pentru fiecare set de articole candidate din întregul set de date. Acest proces nu are
să fie implementat în MapReduce, dar există un mod natural de exprimare
fiecare dintre cele două treceri ca o operațiune MapReduce. Vom rezuma acest lucru
Secvența MapReduce-MapReduce de mai jos.
Funcția Prima hartă: luați subsetul atribuit al coșurilor și găsiți
elemente frecvente în subset folosind algoritmul Secțiunii 6.4.1. Ca de-
înscris acolo, reduceți pragul de asistență de la s la ps dacă fiecare sarcină Hărți devine
fracțiunea p din totalul fișierului de intrare. Ieșirea este un set de perechi cheie-valoare (F, 1),
unde F este un set frecvent de articole din eșantion. Valoarea este întotdeauna 1 și este
irelevant.
Prima funcție de reducere: fiecărei sarcini de reducere i se atribuie un set de taste, care
sunt seturi de articole. Valoarea este ignorată, iar sarcina Reduce le produce pur și simplu
taste (seturi de articole) care apar de una sau mai multe ori. Astfel, ieșirea primului
Funcția de reducere este setul de articole candidate.
A doua funcție de hartă: sarcinile de hartă pentru a doua funcție de hartă sunt luate
toate ieșirile din prima funcție de reducere (seturile de articole candidate) și a
porțiunea din fișierul de date de intrare. Fiecare activitate Harta contează numărul de apariții
a fiecăruia dintre elementele candidate dintre coșurile din porțiunea setului de date
că a fost atribuit. Ieșirea este un set de perechi cheie-valoare (C, v), unde C este una
dintre seturile de candidați și v este suportul pentru acel set de articole printre coșuri
care au fost introduse în această sarcină de hartă.
A doua funcție de reducere: sarcinile de reducere iau seturile de elemente care sunt
date ca chei și sumează valorile asociate. Rezultatul este sprijinul total
pentru fiecare set de articole pe care sarcina Reduce a fost atribuit să îl gestioneze. Acestea
seturile de articole a căror sumă de valori este cel puțin s sunt frecvente în întregul set de date, deci

---

Pagina 262

sarcina Reduce produce aceste seturi de elemente cu numărul lor. Seturi de articole care nu
au suport total cel puțin s nu sunt transmise la ieșirea Reducerii
sarcină. 2

6.4.5 Algoritmul lui Toivonen

Acest algoritm utilizează întâmplarea într-un mod diferit de eșantionarea simplă
algoritmul secțiunii 6.4.1. Algoritmul lui Toivonen, având în vedere suficiente
ory, va folosi o trecere peste un eșantion mic și o trecere completă peste date. Aceasta
nu va da nici negative negative, nici pozitive, dar există un mic, dar diferit de zero
probabilitatea că nu va reuși să producă niciun răspuns. În acest caz, are nevoie
să se repete până dă un răspuns. Cu toate acestea, numărul mediu de treceri

necesar înainte de a produce toate și numai seturile obiecte frecvente este o mică constantă.
Algoritmul lui Toivonen începe prin selectarea unui mic eșantion din setul de date de intrare, și găsind din acesta candidatul frecvent seturi de articole. Procesul este exact așa din secțiunea 6.4.1, cu excepția faptului că este esențial ca pragul să fie setat la ceva mai mică decât valoarea sa proporțională. Adică, dacă pragul de sprijin pentru întreg setul de date este s, iar dimensiunea eșantionului este fracția p, atunci când se caută frecvent seturi de elemente din eșantion, utilizați un prag cum ar fi 0.9ps sau 0.8ps. Cu cât noi suntem mai mici face pragul, cu atât avem nevoie de mai multă memorie principală pentru calculul tuturor seturilor de articole care sunt frecvente în eșantion, dar cu cât avem mai multe șanse să evităm situația unde algoritmul nu reușește să ofere un răspuns.
După ce am construit colecția de obiecte frecvente pentru eșantion, noi apoi construiți marginea negativă. Aceasta este colecția de articole care nu sunt frecvente în eșantion, dar toate subseturile lor imediate (subseturi construite prin ștergerea exactă a unui articol) sunt frecvente în eșantion.
Exemplul 6.11: Să presupunem că elementele sunt {A, B, C, D, E} și am găsit următoarele seturi de articole să fie frecvente în eșantion: {A}, {B}, {C}, {D}, {B, C}, {C, D}. Rețineți că ∅ este, de asemenea, frecvent, atât timp cât există cel puțin la fel de multe coșuri ca prag de sprijin, deși din punct de vedere tehnic algoritmii pe care îi avem descrise omit acest fapt evident. În primul rând, {E} se află în marginea negativă, deoarece nu este frecvent în eșantion, dar singurul său subgrup imediat, ∅, este frecvent.
Seturile {A, B}, {A, C}, {A, D} și {B, D} se află în marginea negativă.
Niciunul dintre aceste seturi nu este frecvent și fiecare are două subseturi imediate, ambele care sunt frecvente. De exemplu, {A, B} are subseturi imediate {A} și {B}.
Dintre celelalte șase dubletoni, niciunul nu se află în granița negativă. Seturile {B, C} și {C, D} nu se află în granița negativă, deoarece sunt frecvente. The cele patru perechi rămase sunt fiecare E împreună cu un alt element, iar acestea nu în marginea negativă, deoarece au un subset imediat {E} care nu este frecvent.
Niciunul dintre tripluri sau seturi mai mari nu se află în marginea negativă. De exemplu, {B, C, D} nu se află în marginea negativă, deoarece are un subset imediat

2 Strict vorbind, funcția Reduce trebuie să producă o valoare pentru fiecare tastă. Poate produce 1 ca valoare pentru seturile de articole găsite frecvente și 0 pentru cele care nu sunt frecvente.

{B, D} care nu este frecvent. Astfel, chenarul negativ este format din cinci seturi: {E}, {A, B}, {A, C}, {A, D} și {B, D}. ✷
Pentru a completa algoritmul lui Toivonen, facem o trecere prin întreaga set, numărând toate seturile de elemente care sunt frecvente în eșantion sau se află în chenar negativ. Există două rezultate posibile.
1. Niciun membru al chenarului negativ nu este frecvent în întregul set de date. În în acest caz, setul corect de elemente frecvente este exact acele seturi de articole din proba care s-a dovedit a fi frecventă în ansamblu.
2. Unul dintre membrii frontierei negative este frecvent în ansamblu. Atunci noi nu pot fi sigur că nu există niște seturi chiar mai mari, nici în chenar negativ și nici colecția de obiecte frecvente pentru eșantion, care sunt, de asemenea, frecvente în ansamblu. Astfel, nu putem oferi niciun răspuns la acest lucru timp și trebuie să repete algoritmul cu un nou eșantion aleatoriu.

6.4.6 De ce funcționează algoritmul lui Toivonen

În mod clar, algoritmul lui Toivonen nu produce niciodată un fals pozitiv, întrucât porturi la fel de frecvente acele seturi de articole care au fost numărate și care s-au dovedit a fi frecvente în ansamblu. Pentru a argumenta că nu produce niciodată un fals negativ, trebuie să arătăm că atunci când niciun membru al frontierei negative nu este frecvent în ansamblu, atunci nu poate exista niciun set de elemente care să fie:
1. Frecvent în ansamblu, dar
2. În nici granița negativă și nici colecția de obiecte frecvente pentru proba.
Să presupunem contrariul. Adică, există un set S care este frecvent în întreg, dar nu în granița negativă și nu frecvent în eșantion. De asemenea, această rundă a algoritmului lui Toivonen a produs un răspuns, care ar fi cu siguranță nu includeți S printre seturile de articole frecvente. Prin monotonie, toate subseturile lui S sunt, de asemenea, frecvente în ansamblu. Fie T un subset al lui S care este cel mai mic dimensiunea posibilă între toate subseturile de S care nu sunt frecvente în eșantion.
Susținem că T trebuie să se afle în granița negativă. Sigur T întâlnește unul dintre condiții pentru a fi în granița negativă: nu este frecventă în eșantion.
De asemenea, îndeplinește cealaltă condiție pentru a fi în granița negativă: fiecare dintre ei subseturile imediate sunt frecvente în eșantion. Căci dacă vreun subset imediat de

T nu au fost frecvente în eșantion, atunci ar exista un subset de S, adică
mai mic decât T și nu frecvent în eșantion, contrazicând selecția noastră de T
ca un subset de S care nu a fost frecvent în eșantion, totuși la fel de mic ca oricare altul
a stabilit.
Acum vedem că T este atât la granița negativă, cât și frecvent în ansamblu
set de date. În consecință, această rundă a algoritmului lui Toivonen nu a produs un
Răspuns.

Pagina 264

6.4.7 Exerciții pentru secțiunea 6.4

Exercițiul 6.4.1: Să presupunem că există opt elemente, A, B, ..., H și următoarele
sunt seturile de articole frecvente maxime: {A, B}, {B, C}, {A, C}, {A, D}, {E} și
{F}. Găsiți marginea negativă.
Exercițiul 6.4.2: Aplicați algoritmul lui Toivonen la datele din exercițiul 6.3.1,
cu un prag de sprijin de 4. Luați ca eșantion primul rând de coșuri:
{1, 2, 3}, {2, 3, 4}, {3, 4, 5} și {4, 5, 6}, adică o treime din fișier. Scara noastră
suportul jos va fi 1.
(a) Care sunt seturile de articole frecvente în eșantion?
(b) Care este granița negativă?
(c) Care este rezultatul trecerii prin setul de date complet? Sunt oricare dintre
seturi de articole în granița negativă frecvente în ansamblu?
!! Exercițiul 6.4.3: Să presupunem că elementul i apare exact de câte ori într-un fișier de n coșuri,
unde s este pragul de sprijin. Dacă luăm un eșantion de n / 100 coșuri și
reduceți pragul de sprijin pentru eșantion la s / 100, care este probabilitatea
că voi fi găsit a fi frecvent? Puteți presupune că atât s, cât și n sunt
divizibil cu 100.

6.5 Numărarea elementelor frecvente într-un flux

Să presupunem că în loc de un fișier de coșuri avem un flux de coșuri, de la
pe care vrem să-l exploatăm frecvent. Reamintim din capitolul 4 că
diferența dintre un flux și un fișier de date este că elementele fluxului sunt doar
disponibile când sosesc și, de obicei, rata de sosire este atât de mare încât noi
nu poate stoca întregul flux într-un mod care permite interogarea ușoară. Mai departe, ea
este obișnuit ca fluxurile să evolueze în timp, astfel încât elementele care sunt frecvente în
fluxul de astăzi s-ar putea să nu fie frecvent mâine.
O distincție clară între fluxuri și fișiere, atunci când sunt obiecte frecvente
considerat, este că nu există un sfârșit al unui flux, deci în cele din urmă un set de articole merge
să depășească pragul de suport, atâta timp cât apare în mod repetat în flux.
Ca rezultat, pentru fluxuri, trebuie să ne gândim la pragul de sprijin s ca la o fracțiune de
coșurile în care trebuie să apară un set de articole pentru a fi considerat frecvent.
Chiar și cu această ajustare, avem încă mai multe opțiuni cu privire la porțiune
a fluxului peste care se măsoară acea fracțiune.
În această secțiune, vom discuta mai multe moduri pe care le-am putea extrage frecvent
seturi de articole dintr-un flux. În primul rând, luăm în considerare modalitățile de utilizare a tehnologiei de eșantionare
niques din secțiunea anterioară. Apoi, luăm în considerare modelul ferestrei în descompunere
din secțiunea 4.7 și extindeți metoda descrisă în secțiunea 4.7.3 pentru găsire
Obiecte „populare”.

Pagina 265

6.5.1 Metode de eșantionare pentru fluxuri

În cele ce urmează, vom presupune că elementele fluxului sunt coșuri de articole.
Poate cea mai simplă abordare pentru a menține o estimare actuală a frecvenței
seturile de articole într-un flux este să colecteze un anumit număr de coșuri și să le stocheze ca fișier.
În același timp, rulați unul dintre algoritmii frecvenți-set discutați în acest capitol
ignorarea elementelor de flux care sosesc sau stocarea lor ca alt fișier care urmează să fie
analizate ulterior. Când algoritmul de obiecte frecvente se termină, avem un
estimarea seturilor frecvente de articole din flux. Avem apoi mai multe opțiuni.
1. Putem folosi această colecție de obiecte frecvente pentru orice aplicație este
la îndemână, dar începeți să rulați o altă iterație a setului de obiecte frecvente alese
algoritm imediat. Acest algoritm poate:
(a) Utilizați fișierul care a fost colectat în timpul primei iterații a algoritmului
rithm alerga. În același timp, colectați încă un fișier care să fie

utilizat la o altă iterație a algoritmului, atunci când această iterație curentă se termină.
(b) Începeți acum să colectați un alt fișier de coșuri și rulați algoritmul când a fost colectat un număr adecvat de coșuri.
2. Putem continua să numărăm numărul de apariții al fiecăreia dintre acestea seturi obiecte frecvente, împreună cu numărul total de coșuri văzute în de la începutul numărării. Dacă se descoperă că există un set de articole într-o fracțiune din coșuri care este semnificativ sub fracțiunea de prag s, atunci acest set poate fi eliminat din colecția de obiecte frecvente. Când calculați fracția, este important să includeți aparițiile din fișierul original al coșurilor, din care erau obiecte frecvente derivat. Dacă nu, riscăm să ne confruntăm cu o perioadă scurtă de timp care un set de articole cu adevărat frecvent nu apare suficient de frecvent și arunc-o afara. De asemenea, ar trebui să permitem un anumit mod pentru seturile de articole noi frecvente pentru a fi adăugat la colecția curentă. Posibilitățile includ:
(a) Adunați periodic un nou segment al coșurilor în flux și folosiți-l ca fișier de date pentru o altă iterație a frecvenței alese algoritmul seturilor de articole. Se formează noua colecție de articole frecvente din rezultatul acestei iterații și din seturile de articole frecvente din colecția anterioară care a supraviețuit posibilității de a fi fost șters pentru a deveni rare.
(b) Adăugați câteva seturi de elemente aleatorii la colecția curentă și numărați-le fracțiune de apariții pentru o vreme, până când cineva are o idee bună despre indiferent dacă sunt sau nu frecvente în prezent. Mai degrabă decât să alegi elemente noi complet la întâmplare, s-ar putea concentra pe seturi cu elemente care apar în multe seturi de elemente deja cunoscute ca fiind frecvente. De exemplu, o alegere bună este să alegeți seturi de elemente noi din negativ frontieră (secțiunea 6.4.5) a setului curent de obiecte frecvente.

6.5.2 Seturi de articole frecvente în Windows în descompunere

Amintiți-vă din secțiunea 4.7 că o fereastră în descompunere pe un flux se formează prin picking o mică constantă c și care dă elementul ith anterior elementului cel mai recent greutatea $(1 - c)^i$, sau aproximativ e^{-ci}. Secțiunea 4.7.3 a prezentat efectiv un metoda de calcul a articolelor frecvente, cu condiția ca pragul de suport să fie definit într-un mod oarecum diferit. Adică am considerat, pentru fiecare articol, un flux care avea 1 dacă elementul a apărut la un anumit element de flux și 0 dacă nu. Am definit „scorul” pentru acel element ca fiind suma ponderilor în care elementul său de flux era 1. Am fost constrânși să înregistrăm toate articolele al căror scor a fost cel puțin 1/2. Nu putem folosi un prag de scor peste 1, pentru că nu inițiază un număr pentru un articol până când acesta apare în flux și primul când apare, scorul său este doar 1 (din moment ce 1, sau $(1 - c)^0$, este greutatea element curent).
Dacă dorim să adaptăm această metodă la fluxurile de coșuri, există două modificări cationi pe care trebuie sa ii facem. Primul este simplu. Elementele fluxului sunt mai degrabă coșuri decât elementele individuale, pot apărea atât de multe elemente la un anumit element de flux. Tratați fiecare dintre aceste elemente ca și cum ar fi elementul „curent” și adăugați 1 la al lor scor după înmulțirea tuturor scorurilor actuale cu 1 - c, așa cum este descris în secțiunea 4.7.3. Dacă unele articole dintr-un coș nu au scor curent, inițializați scorurile acestora elemente la 1.
A doua modificare este mai complicată. Vrem să găsim toate seturile de articole frecvente, nu doar seturi de articole singulare. Dacă ar fi să inițializăm un număr pentru un set de articole ori de câte ori am văzut-o, am avea prea multe conturi. De exemplu, un coș din 20 de articole are peste un milion de subseturi și toate acestea ar trebui inițiate pentru un coș. Pe de altă parte, după cum am menționat, dacă folosim o cerință peste 1 pentru inițierea punctării unui set de articole, atunci nu am primi niciodată seturile de articole au început și metoda nu ar funcționa.
O modalitate de a rezolva această problemă este de a începe să notezi anumite seturi de articole ca de îndată ce vom vedea o singură instanță, dar fiți prudenți cu privire la ce seturi de articole începem. Este posibil să împrumutăm din trucul A-Priori și să începem un set de articole I numai dacă este complet subseturile adecvate imediate sunt deja înscrise. Consecința acestui lucru restricția este că, dacă eu sunt cu adevărat frecvent, în cele din urmă vom începe să-l numărăm, dar nu începem niciodată un set de articole decât dacă ar fi cel puțin un candidat în sens folosit în algoritmul A-Priori.
Exemplul 6.12: Să presupunem că I este un set mare de articole, dar acesta apare în flux periodic, o dată la 1 / 2c coșuri. Apoi scorul său și cel al subseturilor sale,

nu scade niciodată sub $e^{-1/2}$, care este mai mare de 1/2. Astfel, odată creat un scor
pentru un anumit subset al lui I, acel subset va continua să fie punctat pentru totdeauna. Primul
când apare I, numai subseturile sale individuale vor avea scoruri create pentru ele.
Cu toate acestea, data viitoare când voi apărea, fiecare dintre subseturile sale dubleton va începe
punctaj, deoarece fiecare dintre subseturile imediate ale acelor dubletoni este deja în curs
marcat. La fel, a zecea oară când apar I, subseturile sale de mărime k - 1 sunt toate
punctat, așa că inițiam scoruri pentru fiecare dintre subseturile sale de mărime k. În cele din urmă, noi
atinge dimensiunea | I |, moment în care începem să notăm eu însuși. ✷

6.5.3 Metode hibride

Abordarea secțiunii 6.5.2 oferă anumite avantaje. Este nevoie de o limită
cantitatea de muncă de fiecare dată când ajunge un element de flux și oferă întotdeauna
o imagine actualizată a ceea ce este frecvent în fereastra în descompunere. Este mare
dezavantajul este că ne cere să menținem scorurile pentru fiecare set de articole cu
un scor de cel puțin 1/2. Putem limita numărul de elemente care sunt punctate
prin creșterea valorii parametrului c. Dar cu cât c este mai mare, cu atât este mai mic
fereastră în descompunere este. Astfel, am putea fi obligați să acceptăm informații care urmăresc
fluctuațiile locale de frecvență prea strâns, decât să se integreze peste o
perioada lunga.
Putem combina ideile din secțiunile 6.5.1 și 6.5.2. De exemplu, noi
ar putea rula un algoritm standard pentru obiecte frecvente pe un eșantion de flux,
cu un prag convențional pentru sprijin. Seturile de articole care se găsesc frecvent
prin acest algoritm va fi tratat ca și cum toți au ajuns la ora curentă. Acea
adică fiecare obține un scor egal cu o fracție fixă din numărul lor.
Mai exact, să presupunem că eșantionul inițial are b coșuri, c este decăderea
constantă pentru fereastra în descompunere și scorul minim pe care dorim să-l acceptăm
pentru un set de obiecte frecvente în fereastra în descompunere este s. Apoi pragul de sprijin
pentru rularea inițială a algoritmului de obiecte frecvente este bcs. Dacă un set de articole I este
sa constatat că are suport t în eșantion, apoi i se acordă inițial un scor de t / (bc).
Exemplul 6.13: Să presupunem $c = 10^{-6}$ și scorul minim pe care dorim să-l acceptăm
în fereastra în descompunere este 10. Să presupunem că luăm un eșantion de 10^{8} coșuri
din pârâu. Apoi, atunci când analizăm eșantionul respectiv, folosim un prag de suport
de $10^{8} \times 10^{-6} \times 10 = 1000$.
Luați în considerare un set de articole I care are suport 2000 în eșantion. Apoi inițiala
scorul pe care îl folosim pentru I este $2000 / (10^{8} \times 10^{-6}) = 20$. După acest pas de inițiere, fiecare
când un coș ajunge în flux, scorul curent va fi înmulțit cu
1 - c = 0,999999. Dacă I este un subset al coșului curent, atunci adăugați 1 la scor.
Dacă scorul pentru I scade sub 10, atunci se consideră că nu mai este frecvent,
deci este eliminat din colecția de obiecte frecvente. ✷
Din păcate, nu avem un mod rezonabil de a iniția scorul de noi
seturi de articole. Dacă nu avem niciun scor pentru itemul I, iar 10 este scorul minim pe care îl avem noi
doresc să mențină, nu există nicio modalitate prin care un singur coș să-și poată sări scorul
0 la ceva mai mult decât 1. Cea mai bună strategie pentru adăugarea de seturi noi este să rulați un
noi calcule frecvente de articole pe un eșantion din flux și adăugați la
colecția de seturi de articole fiind punctată pe oricare care îndeplinește pragul pentru eșantionul respectiv
dar nu au fost marcate anterior.

6.5.4 Exerciții pentru secțiunea 6.5

!! Exercițiul 6.5.1: Să presupunem că numărăm obiecte frecvente într-un câștig în descompunere
dow cu o constantă de descompunere c. Să presupunem, de asemenea, că, cu probabilitatea p, un dat

---

elementul stream (coș) conține atât articolele i, cât și j. În plus, cu
abilitatea p coșul conține i dar nu j, iar cu probabilitatea p conține j
dar nu eu. În funcție de c și p, care este fracțiunea de timp în care vom fi
marcând perechea {i, j}?

6.6 Rezumatul capitolului 6

♦ Date Market-Basket: Acest model de date presupune că există două tipuri de date
entități: articole și coșuri. Există o relație între mulți și mulți
obiecte și coșuri. De obicei, coșurile sunt legate de seturi mici de articole,
în timp ce articolele pot fi legate de multe coșuri.

♦ Seturi obiecte frecvente: suportul pentru un set de articole este numărul de coșuri care conține toate acele articole. Elemente cu suport care este cel puțin unii pragul sunt numite obiecte frecvente.

♦ Reguli de asociere: Acestea sunt implicații pe care dacă un coș conține un anumite seturi de elemente I, atunci este probabil să conțină un alt element particular j, de asemenea. Se numește probabilitatea ca j să fie și într-un coș care conține I încrederea regulii. Dobânda regulii este suma cu care încrederea se abate de la fracția tuturor coșurilor care conțin j.

♦ Gâtul de sticlă de numărare a perechilor: pentru a găsi obiecte frecvente, trebuie examinați toate coșurile și numărați numărul de apariții ale seturilor de a anumite dimensiuni. Pentru date tipice, cu scopul de a produce un număr mic de seturi de articole care sunt cele mai frecvente dintre toate, partea care ia adesea cea mai mare memorie principală este numărarea perechilor de articole. Astfel, metodele pentru găsirea de obiecte frecvente se concentrează în mod obișnuit pe modul de a minimiza memoria principală necesară numărării perechilor.

♦ Matrici triunghiulare: În timp ce s-ar putea folosi o matrice bidimensională pentru numărați perechi, astfel pierdeți jumătate din spațiu, deoarece nu este nevoie numărați perechea {i, j} atât în elementele matricei ij, cât și ji. Aranjând perechi (i, j) pentru care i <j în ordine lexicografică, putem stoca numai conturile necesare într-o matrice unidimensională fără spațiu irosit și totuși să puteți accesa eficient numărul pentru orice pereche.

♦ Stocarea perechilor se consideră triplă: dacă mai puțin de 1/3 din perechile posibile apar de fapt în coșuri, atunci este mai eficient spațiul pentru a stoca numărul de perechi ca triple (i, j, c), unde c este numărul perechii {i, j} și i <j. O structură de index, cum ar fi un tabel hash, ne permite să găsim triplul pentru (i, j) eficient.

♦ Monotonicitatea seturilor obiecte frecvente: O proprietate importantă a seturilor de obiecte este că dacă un set de articole este frecvent, atunci la fel sunt toate subseturile sale. Exploatăm această proprietate pentru a elimina necesitatea de a număra anumite seturi de articole folosind-o contrapozitiv: dacă un set de articole nu este frecvent, atunci nici supersetele sale nu sunt.

Pagina 269

♦ Algoritmul A-Priori pentru perechi: Putem găsi toate perechile frecvente după făcând două pase peste coșuri. La prima trecere, numărăm articolele și apoi stabiliți ce elemente sunt frecvente. Pe al doilea treceți, numărăm doar perechile de articole care se găsesc frecvente la prima trecere. Monotonitatea ne justifică ignorarea altor perechi.

♦ Găsirea unor obiecte frecvente mai mari: A-Priori și mulți alți algoritmi permiteți-ne să găsim obiecte frecvente mai mari decât perechile, dacă facem o singură trecere peste coșurile pentru fiecare set de mărimi, până la o anumită limită. Pentru a găsi obiecte frecvente de mărimea k, monotonitatea ne permite să ne restrângem atenția numai acele seturi de articole astfel încât toate subseturile lor de dimensiunea k - 1 au deja a fost găsit frecvent.

♦ Algoritmul PCY: Acest algoritm se îmbunătățește pe A-Priori prin crearea un tabel hash la prima trecere, folosind tot spațiul de memorie principal care nu este necesare pentru a număra articolele. Perechile de articole sunt hash și tabelul hash gălețile sunt utilizate ca număr întreg de câte ori o pereche a hash la găleată. Apoi, la a doua trecere, trebuie doar să numărăm perechi de obiecte frecvente care au ajuns la o găleată frecventă (una a cărei numărare este la cel puțin pragul de sprijin) la prima trecere.

♦ Algoritmul cu mai multe etape: Putem insera treceri suplimentare între prima și a doua trecere a algoritmului PCY la hash perechi cu alte, în-tabele hash dependente. La fiecare trecere intermediară, nu trebuie decât să hash perechi de obiecte frecvente care s-au mărunțit la găleți frecvente pe toate multe pase.

♦ Algoritmul Multihash: Putem modifica prima trecere a PCY Algo-rithm pentru a împărți memoria principală disponibilă în mai multe tabele hash. Pe a doua trecere, trebuie să contorizăm o pereche de obiecte frecvente numai dacă sunt hash pentru a frecventa gălețile în toate mesele hash.

♦ Algoritmi aleatori: în loc să facă treceri prin toate datele, putem alege un eșantion aleatoriu de coșuri, suficient de mic încât să fie este posibil să stocați atât eșantionul, cât și numărul necesar de articole în

memoria principala. Pragul de sprijin trebuie redus proporțional.
Putem găsi apoi seturile de articole frecvente pentru eșantion și sperăm că acesta este o bună reprezentare a datelor ca întreg. În timp ce această metodă folosește la majoritatea trec prin întregul set de date, sunt supuse unor fals pozitive (articole care sunt frecvente în eșantion, dar nu în ansamblu) și false negative (elemente care sunt frecvente în ansamblu, dar nu eșantionul).

♦ Algoritmul SON: o îmbunătățire a algoritmului simplu randomizat rithm este de a împărți întregul fișier de coșuri în segmente suficient de mici că toate seturile obiecte frecvente pentru segment pot fi găsite în memoria principală. Seturile de articole candidate sunt cele găsite frecvent pentru cel puțin un segment. A

Pagina 270

a doua trecere ne permite să numărăm toți candidații și să găsim exact colecția alegerea de obiecte frecvente. Acest algoritm este adecvat în special într-un Setare MapReduce.

♦ Algoritmul lui Toivonen: Acest algoritm începe prin găsirea unor obiecte frecvente într-un eșantion, dar cu pragul coborât, deci există puține șanse lipsesc un set de articole care este frecvent în ansamblu. Apoi, vom examina întregul fișier de coșuri, numărând nu numai seturile de articole frecvente în eșantionul, dar și marginea negativă - elemente care nu au fost găsit frecvent, dar toate subseturile lor imediate sunt. Dacă niciun membru al frontiera negativă se găsește frecventă în ansamblu, atunci răspunsul este exact. Dar dacă un membru al frontierei negative este găsit frecvent, atunci întregul procesul trebuie să se repete cu un alt eșantion.

♦ Elementele frecvente în fluxuri: dacă folosim o fereastră în descompunere cu constantă c, atunci putem începe să numărăm un articol ori de câte ori îl vedem într-un coș. Noi începeți să numărați un set de articole dacă îl vedem cuprins în coșul curent, și toate subseturile sale imediate adecvate sunt deja numărate. Dupa cum fereastra se descompune, înmulțim toate numărările cu 1 - c și le eliminăm care sunt mai mici de 1/2.

6.7 Referințe pentru capitolul 6

Modelul de date al coșului de piață, inclusiv regulile de asociere și A-Priori Algoritm, sunt din [1] și [2].
Algoritmul PCY este de la [4]. Algoritmii multietape și multihash se găsesc în [3].
Algoritmul SON este din [5]. Algoritmul lui Toivonen apare în [6].

1. R. Agrawal, T. Imielinski și A. Swami, „Asociațiile miniere între seturi de articole din baze de date masive, ”Proc. ACM SIGMOD Intl. Conf. pe Managementul datelor, pp. 207-216, 1993.
2. R. Agrawal și R. Srikant, „Algoritmi rapizi pentru regulile asociației miniere” Intl. Conf. pe baze de date foarte mari, pp. 487–499, 1994.
3. M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani și JD Ull-omule, „Computer iceberg interogă eficient”, Intl. Conf. pe Foarte mare Baze de date, pp. 299-310, 1998.
4. JS Park, M.-S. Chen și PS Yu, „Un algoritm eficient bazat pe hash pentru regulile asociației miniere, ”Proc. ACM SIGMOD Intl. Conf. pe Man-de date, pp. 175–186, 1995.
5. A. Savasere, E. Omiecinski și SB Navathe, „Un algoritm eficient pentru reguli de asociere minieră în baze de date mari ”, Intl. Conf. pe Foarte mare Baze de date, pp. 432-444, 1995.

Pagina 271

6. H. Toivonen, „Eșantionarea bazelor de date mari pentru reguli de asociere”, Intl. Conf. pe baze de date foarte mari, pp. 134-145, 1996.

Pagina 272

Capitolul 7
Clustering

Clusterizarea este procesul de examinare a unei colecții de „puncte” și de grupare punctele în „clustere” în funcție de o anumită măsură a distanței. Scopul este acela punctele din același grup au o distanță mică una de cealaltă, în timp ce punctele în diferite clustere sunt la o distanță mare una de cealaltă. O sugestie de cum ar putea arăta clusterele a fost văzut în Fig. 1.1. Cu toate acestea, există intenția a fost că existau trei clustere în jurul a trei intersecții de drumuri diferite, dar două dintre clustere s-au amestecat unul în celălalt pentru că nu erau suficient separat.

Scopul nostru în acest capitol este să oferim metode pentru descoperirea clusterelor în date. Ne interesează în special situațiile în care datele sunt foarte mari și / sau unde spațiul este fie de înaltă dimensiune, fie spațiul nu este euclidian toate. Prin urmare, vom discuta mai mulți algoritmi care presupun că datele sunt valabile nu se încadrează în memoria principală. Cu toate acestea, începem cu elementele de bază: cele două generale abordări ale clusterizării și metodele de tratare a clusterelor într-un Spațiul euclidian.

7.1 Introducere în tehnicile de clusterizare

Începem prin a revizui noțiunile de măsuri de distanță și spații. Cei doi abordările majore ale grupării - ierarhice și atribuirea punctelor - sunt amendat. Trecem apoi la o discuție despre „blestemul dimensionalității”, care face dificilă aglomerarea în spații cu dimensiuni ridicate, dar și, așa cum vom vedea, permite unele simplificări dacă sunt utilizate corect într-un algoritm de grupare.

7.1.1 Puncte, spații și distanțe

Un set de date adecvat pentru grupare este o colecție de puncte, care sunt obiecte aparținând unor spații. În sensul său cel mai general, un spațiu este doar un universal set de puncte, din care sunt extrase punctele din setul de date. Cu toate acestea, noi ar trebui să țină cont de cazul obișnuit al unui spațiu euclidian (a se vedea secțiunea 3.5.2),

care are o serie de proprietăți importante utile pentru grupare. În special, punctele unui spațiu euclidian sunt vectori de numere reale. Lungimea vectorului este numărul de dimensiuni ale spațiului. Componentele vectorului sunt denumite în mod obișnuit coordonatele punctelor reprezentate.

Toate spațiile pentru care putem efectua o grupare au o măsură a distanței, oferind o distanță între oricare două puncte din spațiu. Am introdus distanțe în secțiunea 3.5. Distanța euclidiană comună (rădăcină pătrată a sumelor din pătrate ale diferențelor dintre coordonatele punctelor din fiecare dimen- sion) servește pentru toate spațiile euclidiene, deși am menționat și altele opțiuni pentru măsurarea distanței în spațiile euclidiene, inclusiv în Manhattan distanța (suma mărimilor diferențelor în fiecare dimensiune) și L ∞-distanță (magnitudinea maximă a diferenței în orice dimensiune).

Beagles
Greutate
Înălțime
Chihuahua
Teckelii

Figura 7.1: Înălțimile și greutățile câinilor preluați din trei soiuri

Exemplul 7.1: Aplicațiile clasice ale clusterizării implică adesea dimensiuni reduse spații euclidiene secționale. De exemplu, Fig. 7.1 prezintă înălțimea și greutatea asigurări de câini de mai multe soiuri. Fără a ști care câine este din care varietate, putem vedea doar uitându-ne la diagramă că câinii cad în trei clustere, iar acele clustere se întâmplă să corespundă a trei soiuri. Cu mici cantități de date, orice algoritm de grupare va stabili clustere corecte și simpla reprezentare a punctelor și „globul ocular” al complotului vor fi de asemenea suficiente. ✷

Cu toate acestea, problemele moderne de grupare nu sunt atât de simple. Ele pot implica Spații euclidiene de dimensiuni foarte mari sau spații care nu sunt euclidiene toate. De exemplu, este dificil să grupați documentele după subiectul lor, pe baza apariția unor cuvinte obișnuite, neobișnuite în documente. Este provocator să grupeze cinefilii după tipul sau tipurile de filme care le plac.

De asemenea, am luat în considerare în secțiunea 3.5 măsuri la distanță pentru spații non-euclidee.

ces. Acestea includ distanța Jaccard, distanța cosinusului, distanța Hamming,

---

Pagina 275

și editați distanța. Amintiți-vă că cerințele pentru o funcție pe perechi de puncte
a fi o măsură de distanță sunt că
1. Distanțele sunt întotdeauna nenegative și numai distanța dintre un punct
și în sine este 0.
2. Distanța este simetrică; nu contează în ce ordine consideri
puncte atunci când calculează distanța lor.
3. Măsurile de distanță respectă inegalitatea triunghiului; distanța de la x la y
la z nu este niciodată mai mică decât distanța directă de la x la z.

7.1.2 Strategii de clusterizare

Putem împărți (cluster!) Algoritmi de grupare în două grupuri care urmează două
strategii fundamental diferite.
1. Algoritmii ierarhici sau aglomerativi încep cu fiecare punct al său
grup. Clusterele sunt combinate pe baza „apropierii” lor, folosind una dintre
multe definiții posibile ale „aproape”. Combinația se oprește când este mai departe
combinația duce la clustere care nu sunt de dorit pentru una din mai multe reacții
fii. De exemplu, ne putem opri atunci când avem un număr prestabilit de
sau putem folosi o măsură de compacitate pentru clustere și deșeuri
pentru a construi un cluster prin combinarea a două clustere mai mici dacă rezultatul
clusterul are puncte care se întind pe o regiune prea mare.
2. Cealaltă clasă de algoritmi implică atribuirea punctelor. Punctele sunt con-
considerate într-o anumită ordine și fiecare este alocată clusterului în care
se potrivește cel mai bine. Acest proces este în mod normal precedat de o scurtă fază în care
clusterele inițiale sunt estimate. Variațiile permit combinarea ocazională sau
împărțirea clusterelor sau poate permite neasignarea punctelor, dacă acestea sunt
valori aberante (puncte prea departe de oricare dintre grupurile actuale).
Algoritmii pentru grupare se pot distinge și prin:
(a) Dacă algoritmul presupune un spațiu euclidian sau dacă al-
goritmul funcționează pentru o măsură de distanță arbitrară. Vom vedea că a
distincția cheie este că într-un spațiu euclidian este posibil să se rezume
o colecție de puncte de către centroul lor - media punctelor. Într-o
spațiul non-euclidian, nu există noțiunea de centroid și suntem forțați
pentru a dezvolta un alt mod de a rezuma clusterele.
(b) Dacă algoritmul presupune că datele sunt suficient de mici pentru a se potrivi
memorie principală sau dacă datele trebuie să locuiască în memoria secundară,
marily. Algoritmii pentru cantități mari de date trebuie adesea să ia comenzi rapide,
întrucât este imposibil să te uiți la toate perechile de puncte, de exemplu. Este, de asemenea
este necesar pentru a rezuma grupurile din memoria principală, deoarece nu putem reține
toate punctele tuturor clusterelor din memoria principală în același timp.

Pagina 276

7.1.3 Blestemul dimensionalității

Spațiile euclidiene de înaltă dimensiune au o serie de proprietăți neintuitive care
sunt denumite uneori „blestemul dimensionalității”. Neeuclidian
spațiile împărtășesc de obicei și aceste anomalii. O manifestare a „blestemului”
este că în dimensiuni ridicate, aproape toate perechile de puncte sunt la fel de departe de
unul pe altul. O altă manifestare este că aproape orice doi vectori sunt aproape
ortogonală. Vom explora pe rând fiecare dintre acestea.
Distribuția distanțelor într-un spațiu de înaltă dimensiune
Să luăm în considerare un spațiu euclidian d-dimensional. Să presupunem că alegem n aleatoriu
punctele din cubul unității, adică punctele [x_1, x_2, ..., x_d], unde fiecare x_i se află în intervalul
De la 0 la 1. Dacă d = 1, plasăm puncte aleatorii pe o linie de lungime 1. Ne așteptăm
că unele perechi de puncte vor fi foarte apropiate, de exemplu, puncte consecutive pe
linia. De asemenea, ne așteptăm ca unele puncte să fie foarte îndepărtate - cele aflate în apropiere
capetele opuse ale liniei. Distanța medie între o pereche de puncte este de 1/3. [1]
Să presupunem că d este foarte mare. Distanța euclidiană între două aleatorii
punctele [x_1, x_2, ..., x_d] și [y_1, y_2, ..., y_d] este
√
√

$\sqrt{}$
$\sqrt{}$
d
$\sum$
i = 1
$(x_i - y_i)^2$
Aici, fiecare x_i și y_i este o variabilă aleatorie aleasă uniform în intervalul 0 la
1. Deoarece d este mare, ne putem aștepta ca pentru unii i, $|x_i - y_i|$ va fi aproape de 1.
Aceasta pune o limită inferioară de 1 la distanța dintre aproape oricare două aleatorii
puncte. De fapt, un argument mai atent poate pune o limită inferioară mai puternică
distanța dintre toate, cu excepția unei fracțiuni foarte mici din perechile de puncte.
Cu toate acestea, distanța maximă dintre două puncte este $\sqrt{d}$și se poate argumenta
că toate, cu excepția unei fracțiuni foarte mici din perechi, nu au o distanță
aproape de această limită superioară. De fapt, aproape toate punctele vor avea o distanță apropiată de
distanța medie.
Dacă în esență nu există perechi de puncte apropiate, este greu de construit
deloc clustere. Există puține justificări pentru gruparea unei perechi de puncte și
nu altul. Desigur, datele pot să nu fie aleatorii și pot fi utile
clustere, chiar și în spații foarte înalte. Cu toate acestea, argumentul despre
datele aleatorii sugerează că va fi greu să găsești aceste grupuri printre atât de mulți
perechi care sunt toate la aproximativ aceeași distanță.
Unghiuri între vectori
Să presupunem din nou că avem trei puncte aleatorii A, B și C într-o dimensiune d
spațiu, unde d este mare. Aici, nu presupunem că punctele se află în cubul unității;
1 Puteți demonstra acest fapt evaluând o integrală dublă, dar nu vom face calculul
aici, deoarece nu este esențial pentru discuție.

---

pot fi oriunde în spațiu. Ce este unghiul ABC? Putem presupune că
A este punctul $[x_1, x_2, ..., x_d]$ și C este punctul $[y_1, y_2, ..., y_d]$, în timp ce B este
origine. Reamintim din secțiunea 3.5.4 că cosinusul unghiului ABC este punctul
produsul lui A și C împărțit la produsul lungimilor vectorilor A și
C. Adică cosinusul este
$\sum_{i=1}^{d} x_i y_i$
$\sqrt{}$
$\sum_{i=1}^{d} x_i^2$
$\sqrt{}$
$\sum_{i=1}^{d} y_i^2$
Pe măsură ce d crește, numitorul crește liniar în d, dar numeratorul este o sumă
de valori aleatorii, care sunt la fel de probabil să fie pozitive ca negative. Astfel,
valoarea așteptată a numărătorului este 0 și, pe măsură ce d crește, deviația standard
crește doar ca $\sqrt{d}$. Astfel, pentru d mare, cosinusul unghiului dintre oricare două
vectorii sunt aproape siguri că sunt aproape de 0, ceea ce înseamnă că unghiul este aproape de 90
grade.
O consecință importantă a faptului că vectorii aleatori sunt ortogonali este că dacă noi
au trei puncte aleatorii A, B și C și știm distanța de la A la B
este d_1, în timp ce distanța de la B la C este d_2, putem presupune distanța de la A
la C este aproximativ $\sqrt{d_1 + d_2}$. Această regulă nu se menține, nici măcar aproximativ,
dacă numărul dimensiunilor este mic. Ca caz extrem, dacă d = 1, atunci
distanța de la A la C ar fi neapărat $d_1 + d_2$ dacă A și C ar fi opuse
laturile lui B sau $|d_1 - d_2|$ dacă ar fi de aceeași parte.

7.1.4 Exerciții pentru secțiunea 7.1

! Exercițiul 7.1.1: Dovediți că dacă alegeți două puncte în mod uniform și independent
ușor pe o linie de lungime 1, atunci distanța așteptată între puncte este
1/3.
!! Exercițiul 7.1.2: Dacă alegeți două puncte în mod uniform în pătratul unității, ce
este distanța lor euclidiană așteptată?
! Exercițiul 7.1.3: Să presupunem că avem un spațiu euclidian d-dimensional. Considera
vectori ale căror componente sunt doar +1 sau −1 în fiecare dimensiune. Rețineți că

fiecare vector are lungimea $\sqrt{d}$, deci produsul lungimilor lor (numitor în formula pentru cosinusul unghiului dintre ele) este d. Dacă le-am alege pe fiecare componentă independent, iar o componentă este la fel de probabil să fie +1 ca -1, ce este distribuția valorii numărătorului formulei (adică suma din produsele componentelor corespunzătoare din fiecare vector)? Ce poate spuneți despre valoarea așteptată a cosinusului unghiului dintre vectori, pe măsură ce d crește mare?

7.2 Clusterizarea ierarhică

Începem prin a lua în considerare gruparea ierarhică într-un spațiu euclidian. Acest algoritmul poate fi folosit doar pentru seturi de date relativ mici, dar chiar și așa, acolo

---

Pagina 278

sunt unele eficiențe pe care le putem realiza printr-o implementare atentă. Când spațiul este non-euclidian, există probleme suplimentare asociate cu ierarhizarea grupare. Prin urmare, considerăm „clustroizi” și modul în care putem reprezenta un cluster atunci când nu există nici un punct central sau mediu într-un cluster.

7.2.1 Clusterizarea ierarhică într-un spațiu euclidian

Orice algoritm ierarhic de clusterizare funcționează după cum urmează. Începem cu fiecare punct în propriul cluster. Pe măsură ce timpul trece, clustere mai mari vor fi construite de combinând două clustere mai mici și trebuie să decidem din timp:

1. Cum vor fi reprezentate clusterele?
2. Cum vom alege ce două clustere să fuzionăm?
3. Când vom opri combinarea clusterelor?

Odată ce avem răspunsuri la aceste întrebări, algoritmul poate fi descris succesiv cinctly ca:

CÂND nu este timpul să te oprești DO
alege cele mai bune două clustere pentru a fuziona;
combinați cele două clustere într-un singur cluster;
SFÂRȘIT;

Pentru început, vom presupune că spațiul este euclidian. Asta ne permite să reprezentăm un cluster după centroul său sau media punctelor din cluster. Rețineți că într-un grup de un punct, acel punct este centroidul, deci putem inițializa grupează direct. Putem apoi folosi regula de fuziune care arată distanța între oricare două clustere este distanța euclidiană dintre centroidele lor, și ar trebui să alegem cele două grupuri la cea mai mică distanță. Alte modalități de a este posibilă definirea distanței intercluster și putem alege, de asemenea, cea mai bună pereche de clustere pe altă bază decât distanța lor. Vom discuta câteva opțiuni în Secțiunea 7.2.3.

Exemplul 7.2: Să vedem cum ar funcționa gruparea ierarhică de bază datele din Fig. 7.2. Aceste puncte trăiesc într-un spațiu euclidian bidimensional și fiecare punct este numit prin coordonatele sale (x, y). Inițial, fiecare punct se află într-un cluster de la sine și este centroidul acelui cluster. Dintre toate perechile de puncte, acolo sunt două perechi care sunt cele mai apropiate: (10,5) și (11,4) sau (11,4) și (12,3). Fiecare este la distanță $\sqrt{2}$. Să rupem legăturile în mod arbitrar și să decidem să combinăm (11,4) cu (12,3). Rezultatul este prezentat în Fig. 7.3, inclusiv centroidul noului cluster, care este la (11,5, 3,5).

S-ar putea să credeți că (10,5) se combină cu noul cluster în continuare, din moment ce acesta este atât de aproape de (11,4). Dar regula noastră la distanță ne impune să comparăm doar clusterul centroide, iar distanța de la (10,5) la centroul noului cluster este $1.5\sqrt{2}$, care este puțin mai mare decât 2. Astfel, acum sunt cele două cele mai apropiate clustere

---

Pagina 279

(4,10)
(7,10)
(9,3)
(11,4)
(12,6)
(5,2)
(2,2)
(3,4)
(6,8)
(4,8)
(10,5)
(12,3)

Figura 7.2: Doisprezece puncte care trebuie grupate ierarhic

cele ale punctelor (4,8) și (4,10). Le combinăm într-un singur cluster cu

centroid (4,9).
În acest moment, cei mai apropiați doi centroizi sunt (10,5) și (11,5, 3,5), deci noi combina aceste două clustere. Rezultatul este un grup de trei puncte (10,5), (11,4), și (12,3). Centroidul acestui cluster este (11,4), care se întâmplă să fie unul dintre punctele clusterului, dar această situație este întâmplătoare. Starea clustere este prezentat în Fig. 7.4.
Acum, există mai multe perechi de centroizi care se află la distanță $\sqrt{5}$, iar acestea sunt cei mai apropiați centroizi. Arătăm în Fig. 7.5 rezultatul alegerii a trei dintre aceste:
1. (6,8) este combinat cu grupul de două elemente având centroid (4,9).
2. (2,2) este combinat cu (3,4).
3. (9,3) este combinat cu grupul de trei elemente având centroid (11,4).
Putem continua să combinăm clusterele în continuare. Vom discuta oprirea alternativă ping reguli în continuare. ✷
Există mai multe abordări pe care le-am putea folosi pentru a opri procesul de clusterizare încetează.
1. Ne-ar putea fi spus sau avem convingeri despre câte grupuri există datele. De exemplu, dacă ni se spune că datele despre câini sunt luate de la Chihuahuas, Dachshunds și Beagles, atunci știm să ne oprim când au rămas trei clustere.
2. Am putea să nu mai combinăm când la un moment dat cea mai bună combinație de clusterele existente produc un cluster care este inadecvat. Vom discuta

---

Pagina 280

(4,10)
(7,10)
(9,3)
(11,4)
(12,6)
(5,2)
(2,2)
(3,4)
(6,8)
(4,8)
(10,5)
(12,3)
(11,5,3,5)
Figura 7.3: Combinarea primelor două puncte într-un cluster
diferite teste pentru adecvarea unui cluster în secțiunea 7.2.3, dar pentru un de exemplu, am putea insista ca orice grup să aibă o distanță medie între centroidul și punctele sale nu mai mari decât o anumită limită. Această abordare este numai sensibil dacă avem un motiv să credem că niciun cluster nu se extinde peste prea mult din spațiu.
3. Am putea continua gruparea până când există un singur cluster. In orice caz, nu are sens să returnăm un singur cluster format din toate punctele.
Mai degrabă, returnăm arborele reprezentând modul în care toate punctele au fost combinate. Această formă de răspuns are sens în unele aplicații acțiuni, cum ar fi una în care punctele sunt genomi de specii diferite, iar măsurarea distanței reflectă diferența în genom. [2] Apoi, arborele reprezintă evoluția acestor specii, adică ordinea probabilă în care două specii s-au ramificat dintr-un strămoș comun.
Exemplul 7.3: Dacă finalizăm gruparea datelor din Fig. 7.2, arborele descrierea modului în care grupările au fost grupate este arborele prezentat în Fig. 7.6. ✷

7.2.2 Eficiența clusterizării ierarhice

Algoritmul de bază pentru clusterizarea ierarhică nu este foarte eficient. La fiecare pas, trebuie să calculăm distanțele dintre fiecare pereche de clustere, pentru a găsiți cea mai bună fuziune. Pasul inițial necesită timp $O(n^2)$, dar etapele ulterioare ia timp proporțional cu $(n - 1)^2$, $(n - 2)^2$, ... Suma pătratelor până la n este $O(n^3)$, deci acest algoritm este cubic. Astfel, nu poate fi rulat decât pentru destul de mic numere de puncte.
[2] Acest spațiu nu ar fi euclidian, desigur, ci principiile ierarhice clusterizarea continuă, cu unele modificări, la clusterizarea non-euclidiană.

Pagina 281

(7,10)
(9,3)
(11,4)
(12,6)
(5,2)

(2,2)
(3,4)
(6,8)
(12,3)
(4,8)
(4,10)
(4,9)
(10,5)

Figura 7.4: Clustering după doi pași suplimentari

Cu toate acestea, există o implementare oarecum mai eficientă a căreia noi ar trebui să fie conștient.

1. Începem, așa cum trebuie, calculând distanțele dintre toate perechile de puncte, iar acest pas este O (n^2).
2. Formați perechile și distanțele lor într-o coadă prioritară, astfel încât să putem oricând găsiți cea mai mică distanță într-un singur pas. Această operație este, de asemenea, O (n^2).
3. Când decidem să îmbinăm două clustere C și D, eliminăm toate intrările în coada prioritară care implică unul dintre aceste două clustere; asta necesită funcționează O (n log n) deoarece există cel mult 2n ștergeri care trebuie efectuate și ștergerea cozii de prioritate poate fi efectuată în timp O (jurnal n).
4. Calculăm apoi toate distanțele dintre noul cluster și re-grupuri principale. Această lucrare este, de asemenea, O (n log n), deoarece există cel mult n intrările care trebuie inserate în coada de prioritate și inserarea într-o prioritate coada poate fi făcută și în timp O (jurnal n).

Deoarece ultimii doi pași sunt executați de cel mult n ori, iar primii doi pași sunt executat o singură dată, timpul total de rulare al acestui algoritm este O (n^2 jurnal n).

Aceasta este mai bună decât O (n^3), dar pune totuși o limită puternică asupra cât de mare poate fi n fie înainte de a deveni imposibil de utilizat această abordare de grupare.

7.2.3 Reguli alternative pentru controlul ierarhic Clustering

Am văzut o regulă pentru alegerea celor mai bune clustere pentru a fuziona: găsiți perechea cu cea mai mică distanță dintre centroizii lor. Alte opțiuni sunt:

Pagina 282

(7,10)
(9,3)
(11,4)
(12,6)
(5,2)
(2,2)
(3,4)
(12,3)
(4,8)
(4,10)
(4,9)
(10,5)
(2,5, 3)
(6,8)
(4.7, 8.7)
(10,5, 3,8)

Figura 7.5: Încă trei pași ai grupării ierarhice

1. Luați distanța dintre două clustere pentru a fi minimul distanțelor între oricare două puncte, unul ales din fiecare grup. De exemplu, în Fig. 7.3 am alege apoi să grupăm punctul (10,5) cu clusterul de două puncte, deoarece (10,5) are distanța $\sqrt{2}$și nici o altă pereche de necluse punctul este atât de aproape. Rețineți că, în exemplul 7.2, am făcut această combinație națiune în cele din urmă, dar nu până când am combinat o altă pereche de puncte. În general, este posibil ca această regulă să conducă la o complet diferită grupându-se din cea obținută folosind regula distanței-centroidilor.
2. Luați distanța dintre două clustere pentru a fi distanța medie a tuturor perechi de puncte, câte unul din fiecare grup.

(2,2) (3,4) (5,2) (4,8) (4,10) (6,8) (7,10) (11,4) (12,3) (10,5) (9,3) (12,6)

Figura 7.6: Arborele care prezintă gruparea completă a punctelor din Fig. 7.2

Pagina 283

3. Raza unui cluster este distanța maximă dintre toate punctele iar centroidul. Combinați cele două clustere al căror cluster rezultat are raza cea mai mică. O ușoară modificare este combinarea clusterelor ale căror rezultatul are cea mai mică distanță medie între un punct și centroid. O altă modificare este de a utiliza suma pătratelor distanțelor între puncte și centroid. În unele algoritmi, vom găsi

aceste definiții variante ale „razei” denumite „rază”.
4. Diametrul unui cluster este distanța maximă dintre oricare două punctele clusterului. Rețineți că raza și diametrul unui cluster nu sunt legate direct, deoarece sunt într-un cerc, dar există o tendință pentru ca acestea să fie proporționale. Putem alege să fuzionăm acele clustere al cărui cluster rezultat are cel mai mic diametru. Variante ale acestei reguli, analog cu regula pentru raza, sunt posibile.
Exemplul 7.4: Să luăm în considerare grupul format din cele cinci puncte la dreapta din Fig. 7.2. Centroidul acestor cinci puncte este (10.8, 4.2). Există o cravată pentru cele două puncte cele mai îndepărtate de centroid: (9,3) și (12,6), ambele la distanță $\sqrt{4.68} = 2.16$. Astfel, raza este de 2,16. Pentru diametru, îi găsim pe cei doi punctele din grup care au cea mai mare distanță. Acestea sunt din nou (9,3) și (12,6). Distanța lor este $\sqrt{18} = 4,24$, deci acesta este diametrul. Observați că diametrul nu este exact de două ori mai mare decât raza, deși este aproape în acest caz. The motivul este că centroidul nu este pe linia dintre (9,3) și (12,6). ✷
De asemenea, avem opțiuni pentru a stabili când să oprim procesul de fuzionare. Noi menționat deja „oprește-te când avem k clustere” pentru unele k predeterminate. Iată câteva alte opțiuni.
1. Opriți-vă dacă diametrul clusterului care rezultă din cea mai bună fuziune plafonează un prag. Putem baza, de asemenea, această regulă pe rază sau pe oricare dintre variantele razei menționate mai sus.
2. Opriți-vă dacă densitatea clusterului care rezultă din cea mai bună fuziune este sub un anumit prag. Densitatea poate fi definită în multe moduri diferite. Aproximativ, ar trebui să fie numărul de puncte cluster pe unitate de volum a grup. Acest raport poate fi estimat la numărul de puncte împărțit la o oarecare putere a diametrului sau razei clusterului. Puterea corectă ar putea fi numărul de dimensiuni ale spațiului. Uneori, 1 sau 2 este ales ca putere, indiferent de numărul de dimensiuni.
3. Opriți-vă când există dovezi că următoarea pereche de clustere care trebuie combinate produce un cluster rău. De exemplu, am putea urmări diametrul mediu din toate grupurile actuale. Atâta timp cât combinăm puncte care într-adevăr aparțin unui grup, această medie va crește treptat. Cu toate acestea, dacă combinăm două clustere care chiar nu merită să fie combinate, apoi media diametrul va face un salt brusc.

Pagina 284

Exemplul 7.5: Să reconsiderăm Fig. 7.2. Are trei clustere naturale. Noi a calculat diametrul celui mai mare - cele cinci puncte din dreapta - în Exemplu 7,4; este 4.24. Diametrul grupului cu 3 noduri din stânga jos este 3, distanța dintre (2,2) și (5,2). Diametrul clusterului cu 4 noduri la stânga sus este $\sqrt{13} = 3,61$. Diametrul mediu, 3,62, a fost atins începând de la 0 după nouă fuziuni, deci creșterea este evident lentă: aproximativ 0,4 pe fuziune. Dacă suntem obligați să fuzionăm două dintre aceste clustere naturale, putem face cel mai bine este fuzionează cele două din stânga. Diametrul acestui cluster este $\sqrt{89} = 9,43$; acesta este distanța dintre cele două puncte (2,2) și (7,10). Acum, media diametrul este (9,43 + 4,24) / 2 = 6,84. Această medie a sărit aproape la fel de mult într-un singur pas ca în toți cei nouă pași anteriori. Această comparație indică faptul că Ultima fuziune nu a fost recomandată și ar trebui să o retragem și să o oprim. ✷

7.2.4 Clusterizarea ierarhică în spații neuclidiene

Când spațiul este neeuclidian, trebuie să folosim o anumită măsură a distanței este calculat din puncte, cum ar fi Jaccard, cosinus sau distanță de editare. Adică noi nu poate baza distanțele pe „locația” punctelor. Algoritmul secțiunii 7.2.1 necesită calcularea distanțelor dintre puncte, dar probabil avem o mod de a calcula acele distanțe. O problemă apare atunci când trebuie să reprezentăm un cluster, deoarece nu putem înlocui o colecție de puncte cu centroul lor.
Exemplul 7.6: Problema apare pentru oricare dintre distanțele neeuclidiene am discutat, dar pentru a fi concret, să presupunem că folosim distanța de editare și noi decideți să îmbinați șirurile abcd și aecdb. Acestea au distanța de editare 3 și s-ar putea fuziona. Cu toate acestea, nu există niciun șir care să le reprezinte media, sau care ar putea fi considerat că se află în mod natural între ei. Am putea lua una dintre șirurile prin care am putea trece atunci când transformăm un șir în cealaltă prin inserții sau ștergeri unice, cum ar fi aebcd, dar există multe astfel de opțiuni. Mai mult, atunci când clusterele sunt formate din mai mult de două șiruri, noțiunea de „pe calea dintre” încetează să mai aibă sens. ✷
Având în vedere că nu putem combina puncte într-un cluster atunci când spațiul este non-Euclidian, singura noastră alegere este să alegem unul dintre punctele clusterului în sine

reprezintă clusterul. În mod ideal, acest punct este aproape de toate punctele clusterului, deci, într-un anumit sens, se află în „centru”. Numim punctul reprezentativ clustroid. Putem selecta clustroidul în diferite moduri, fiecare conceput astfel încât, în unele sens, minimizați distanțele dintre clustroid și celelalte puncte din grupul. Alegerile obișnuite includ selectarea drept clustroid a punctului care minimizează:

1. Suma distanțelor față de celelalte puncte din cluster.
2. Distanța maximă până la un alt punct din cluster.
3. Suma pătratelor distanțelor față de celelalte puncte din cluster.

Exemplul 7.7: Să presupunem că folosim distanța de editare și un cluster constă din cele patru puncte abcd, aecdb, abecb și ecdab. Distanțele lor se găsesc în următorul tabel:

ecdab abecb aecdb
abcd
5
3
3
aecdb
2
2
abecb
4

Dacă aplicăm cele trei criterii pentru a fi centroid la fiecare dintre cele patru puncte din cluster, găsim:

Punct
Suma Suma Max Sq
abcd
11
5
43
aecdb
7
3
17
abecb
9
4
29
ecdab
11
5
45

Putem vedea din aceste măsurători că oricare dintre cele trei criterii este alegeți, aecdb va fi selectat ca clustroid. În general, criterii diferite ar putea produce clustroizi diferiți. ✷

Opțiunile pentru măsurarea distanței dintre clustere care au fost conturate în secțiunea 7.2.3 poate fi aplicat într-un cadru non-euclidian, cu condiția să folosim clustroid în locul centroidului. De exemplu, putem îmbina cele două clustere ale căror clustroizi sunt cei mai apropiați. Am putea folosi și distanța medie sau minimă între toate perechile de puncte din grupuri.

Alte criterii sugerate implicau măsurarea densității unui cluster, pe bază pe raza sau diametrul. Ambele noțiuni au sens în non-euclidian mediu inconjurator. Diametrul este încă distanța maximă dintre oricare două puncte din cluster. Raza poate fi definită folosind clustroidul în locul lui centroidul. Mai mult, este logic să folosiți același tip de evaluare pentru raza așa cum am folosit-o pentru a selecta clustroidul în primul rând. De exemplu, dacă luăm clustroidul drept punctul cu cea mai mică sumă de pătrate de distanțe față de celelalte noduri, apoi definiți raza pentru a fi acea sumă de pătrate (sau rădăcina ei pătrată).

În cele din urmă, Secțiunea 7.2.3 a discutat, de asemenea, criteriile pentru oprirea fuzionării clustere. Niciunul dintre aceste criterii nu a folosit în mod direct centroidul, decât prin noțiunea de rază și am observat deja că „raza” este bună sens în spațiile neeuclidiene. Astfel, nu există nicio schimbare substanțială în opțiuni pentru oprirea criteriilor atunci când trecem de la euclidian la non-euclidian

spații.

7.2.5 Exerciții pentru secțiunea 7.2

Exercițiul 7.2.1: Efectuați o grupare ierarhică a setului unidimensional
din punctele 1, 4, 9, 16, 25, 36, 49, 64, 81, presupunând că grupurile sunt reprezentate de

---

Pagina 286

centroul lor (mediu), iar la fiecare pas grupurile cu cei mai apropiați centroizi
sunt fuzionate.
Exercițiul 7.2.2: Cum s-ar schimba gruparea Exemplului 7.2 dacă am folosi
pentru distanța dintre două clustere:
(a) Minima distanțelor dintre oricare două puncte, una de la fiecare
grup.
(b) Media distanțelor dintre perechile de puncte, câte una de la fiecare dintre
două clustere.
Exercițiul 7.2.3: Repetați gruparea exemplului 7.2 dacă alegem să fuzionăm
cele două clustere al căror cluster rezultat are:
(a) Cea mai mică rază.
(b) Cel mai mic diametru.
Exercițiul 7.2.4: Calculați densitatea fiecăruia dintre cele trei clustere din Fig. 7.2,
dacă „densitatea” este definită ca fiind numărul de puncte împărțit la
(a) Pătratul razei.
(b) Diametrul (nu pătrat).
Care sunt densitățile, conform (a) și (b), a clusterelor care rezultă din
fuziunea oricăror două dintre aceste trei clustere. Face diferența de densitate
sugerează că grupurile ar trebui sau nu să fie combinate?
Exercițiul 7.2.5: Putem selecta clustere pentru clustere, chiar dacă spațiul este
Euclidian. Luați în considerare cele trei clustere naturale din Fig. 7.2 și calculați
clustroidele fiecăruia, presupunând că criteriul pentru selectarea clustroidului este punctul
cu suma minimă a distanțelor până la celălalt punct din cluster.
! Exercițiul 7.2.6: Luați în considerare spațiul șirurilor cu distanță de editare ca distanță
măsura. Dați un exemplu de set de șiruri astfel încât, dacă alegem clustroidul
prin minimizarea sumei distanțelor față de celelalte puncte obținem un punct
ca clustroid, dar dacă alegem clustroidul minimizând maximul
la distanță de celelalte puncte, un alt punct devine clustroid.

7.3 K-înseamnă algoritmi

În această secțiune începem studiul algoritmilor de atribuire a punctelor. Cel mai bun
familia cunoscută de algoritmi de grupare de acest tip se numește k-means. ei
presupune un spațiu euclidian și își asumă și numărul de clustere, k,
este cunoscut din timp. Cu toate acestea, este posibil să se deducă k prin încercare și eroare.
După o introducere în familia algoritmilor k-means, ne vom concentra pe a
un anumit algoritm, numit BFR după autorii săi, care ne permite să executăm
k-înseamnă date care sunt prea mari pentru a se potrivi în memoria principală.

Pagina 287

7.3.1 Bazele K-Means

Un algoritm k-înseamnă este prezentat în Fig. 7.7. Există mai multe moduri de selectare
k punctele inițiale care reprezintă clusterele și le vom discuta în
Secțiunea 7.3.2. Inima algoritmului este for-loop, în care luăm în considerare
fiecare punct diferit de cele k puncte selectate și atribuiți-l celui mai apropiat cluster,
unde „cel mai apropiat” înseamnă cel mai apropiat de centroidul clusterului. Rețineți că
Centroidul unui cluster poate migra pe măsură ce i se atribuie puncte. Cu toate acestea, din moment ce
este posibil ca numai punctele din apropierea grupului să fie alocate, centroul tinde să nu
mișcă-te prea mult.
Alegeți inițial k puncte care ar putea fi incluse
diferite clustere;
Faceți din aceste puncte centrele grupurilor lor;
PENTRU fiecare punct rămas p FĂ
găsiți centroidul la care p este cel mai apropiat;
Adăugați p la clusterul acelui centru;
Reglați centroidul acelui cluster pentru a ține cont de p;
SFÂRȘIT;

Figura 7.7: Schița algoritmilor k-means
Un pas opțional la sfârșit este de a fixa centroizii grupurilor și la
reatribuiți fiecare punct, inclusiv k punctele inițiale, la k grupuri. Obișnuit,
un punct p va fi atribuit aceluiași cluster în care a fost plasat pe
prima trecere. Cu toate acestea, există cazuri în care centroidul grupului original al lui p
s-a deplasat destul de departe de p după ce p a fost plasat acolo și p este atribuit unui alt
grup pe a doua trecere. De fapt, chiar și unele dintre k punctele originale ar putea
termină fiind reatribuit. Deoarece aceste exemple sunt neobișnuite, nu vom locui
la subiect.

7.3.2 Inițializarea clusterelor pentru K-Means

Vrem să alegem puncte care au șanse mari să se întindă în diferite clustere.
Există două abordări.
1. Alegeți puncte care sunt cât mai departe unul de celălalt posibil.
2. Clusterizați un eșantion de date, poate ierarhic, deci există k
ters. Alegeți un punct din fiecare cluster, poate cel mai apropiat de
centroid al clusterului.
A doua abordare necesită puține elaborări. Pentru prima abordare,
există variații. O alegere bună este:
Alegeți primul punct la întâmplare;

Pagina 288

CÂND sunt mai puține decât k puncte DO
Adăugați punctul a cărui distanță minimă de cel selectat
punctele este cât mai mare posibil;
SFÂRȘIT;
Exemplul 7.8: Să luăm în considerare cele douăsprezece puncte din Fig. 7.2, pe care le repro-
duce aici ca Fig. 7.8. În cel mai rău caz, alegerea noastră inițială a unui punct este aproape de
centru, să zicem (6,8). Cel mai îndepărtat punct de la (6,8) este (12,3), astfel încât punctul este ales
Următorul.
(4,10)
(7,10)
(9,3)
(11,4)
(12,6)
(5,2)
(2,2)
(3,4)
(6,8)
(4,8)
(10,5)
(12,3)
Figura 7.8: Repetați figura 7.2
Printre cele zece puncte rămase, cel a cărui distanță minimă față de oricare
(6,8) sau (12,3) este un maxim este (2,2). Acest punct are distanță $\sqrt{52} = 7{,}21$ de
(6,8) și distanța $\sqrt{101} = 10{,}05$ până la (12,3); astfel „scorul” său este de 7,21. Poti
verificați cu ușurință dacă orice alt punct este mai mic decât distanța 7.21 de cel puțin unul dintre
(6,8) și (12,3). Selecția noastră de trei puncte de pornire este astfel (6,8), (12,3) și
(2,2). Observați că aceste trei aparțin unor clustere diferite.
Dacă am fi început cu un punct diferit, să zicem (10,5), am primi altul
set de trei puncte inițiale. În acest caz, punctele de plecare ar fi (10,5),
(2,2) și (4,10). Din nou, aceste puncte aparțin celor trei clustere diferite. ✷

7.3.3 Alegerea valorii corecte a lui k

Este posibil să nu cunoaștem valoarea corectă a lui k de utilizat într-o grupare k-înseamnă. Cum-
oricând, dacă putem măsura calitatea grupării pentru diferite valori ale lui k, noi
poate ghici de obicei care este valoarea corectă a lui k. Amintiți-vă discuția din secțiunea
tiunea 7.2.3, în special Exemplul 7.5, unde am observat că dacă luăm o măsură
de adecvare pentru clustere, cum ar fi raza sau diametrul mediu, acea valoare
va crește încet, atâta timp cât numărul de clustere presupunem că rămâne la sau
peste adevăratul număr de clustere. Cu toate acestea, de îndată ce încercăm să formăm mai puține

---

Pagina 289

mai mult decât există în realitate, măsura va crește precipitat. Ideea este
exprimată prin diagrama din Fig. 7.9.
In medie
Diametru
Număr de clustere

Valoarea corectă a lui *k*
Figura 7.9: Diametrul mediu sau o altă măsură a difuzității crește repede
de îndată ce numărul de clustere scade sub numărul real prezent în date
Dacă nu avem nicio idee care este valoarea corectă a lui k, putem găsi o valoare bună
într-o serie de operațiuni de clusterizare care crește doar logaritmic cu
număr adevărat. Începeți prin rularea algoritmului k-means pentru k = 1, 2, 4, 8, ...
În cele din urmă, veți găsi două valori v și 2v între care există foarte puțin
scăderea diametrului mediu sau a oricărei măsuri de coeziune a clusterului
folosesc. Putem concluziona că valoarea lui k justificată de date se află
între v / 2 și v. Dacă utilizați o căutare binară (discutată mai jos) în intervalul respectiv,
puteți găsi cea mai bună valoare pentru k într-o altă operațiune de grupare jurnal $_2$ v, pentru a
total de 2log $_2$ v grupări. Deoarece adevărata valoare a lui k este cel puțin v / 2, avem
a folosit un număr de grupări care este logaritmic în k.
Deoarece noțiunea „nu prea schimbă” este imprecisă, nu putem spune cu exactitate
câtă schimbare este prea multă. Cu toate acestea, căutarea binară poate fi efectuată
după cum urmează, presupunând că noțiunea de „nu prea multă schimbare” este precisă de unii
formulă. Știm că există prea multe schimbări între v / 2 și v, sau altfel
nu am fi continuat să rulăm un cluster pentru clustere 2v. Să presupunem că la unii
punct, am restrâns intervalul k între x și y. Fie z = (x + y) / 2.
Rulați un cluster cu z ca număr țintă de clustere. Dacă nu există prea
multă schimbare între z și y, atunci adevărata valoare a lui k se află între x și z.
Deci restrângeți recursiv acest interval pentru a găsi valoarea corectă a lui k. Pe de altă parte
mână, dacă există prea multe schimbări între z și y, atunci utilizați căutarea binară în
intervalul dintre z și y în schimb.

7.3.4 Algoritmul lui Bradley, Fayyad și Reina

Acest algoritm, la care vom numi BFR după autorii săi, este o variantă a
k-înseamnă că este conceput pentru a grupa date într-un spațiu euclidian de înaltă dimensiune.
Face o presupunere foarte puternică cu privire la forma grupurilor: trebuie
să fie distribuit în mod normal în jurul unui centroid. Media și deviația standard
pentru un cluster poate diferi pentru diferite dimensiuni, dar dimensiunile trebuie să fie

---

independent. De exemplu, în două dimensiuni, un grup poate avea formă de trabuc,
dar trabucul nu trebuie rotit de pe axe. Figura 7.10 indică punctul.
Bine
Bine
Nu e bine
Figura 7.10: Clustere în date pentru care algoritmul BFR poate fi utilizat
poate avea abateri standard care diferă de-a lungul axelor diferite, dar axele
clusterul trebuie să se alinieze cu axele spațiului
Algoritmul BFR începe prin selectarea k puncte, folosind una dintre metode
discutate în secțiunea 7.3.2. Apoi, punctele fișierului de date sunt citite în bucăți.
Acestea ar putea fi bucăți dintr-un sistem de fișiere distribuit sau dintr-un fișier convențional
poate fi partiționat în bucăți de dimensiunea adecvată. Fiecare bucată trebuie
constau din puține puncte suficiente pentru a putea fi procesate în memoria principală. De asemenea
stocate în memoria principală sunt rezumate ale clusterelor k și a altor date,
deci întreaga memorie nu este disponibilă pentru a stoca o bucată. Datele din memoria principală
altul decât partea din intrare constă din trei tipuri de obiecte:
1. Setul de aruncare: Acestea sunt rezumate simple ale clusterelor.
Vom aborda forma de rezumare a clusterelor în scurt timp. Rețineți că
rezumatele cluster nu sunt „aruncate”; ele sunt de fapt esențiale.
Cu toate acestea, punctele pe care le prezintă rezumatul sunt eliminate și au
nicio reprezentare în memoria principală decât prin acest rezumat.
2. Setul comprimat: Acestea sunt rezumate, similare cu sumele cluster
maries, dar pentru seturi de puncte care au fost găsite aproape unul de altul,
dar nu aproape de niciun cluster. Punctele reprezentate de comprimat
seturile sunt, de asemenea, eliminate, în sensul că nu apar în mod explicit în
memoria principala. Numim seturile de puncte reprezentate miniclustere.
3. Setul reținut: anumite puncte nu pot fi atribuite nici unui cluster, nici
sunt suficient de aproape de orice alte puncte pe care le putem reprezenta
de un set comprimat. Aceste puncte sunt păstrate în memoria principală exact așa cum
ele apar în fișierul de intrare.
Imaginea din Fig. 7.11 sugerează modul în care sunt reprezentate punctele procesate până acum.
Seturile descărcate și comprimate sunt reprezentate de valori 2d + 1, dacă
datele sunt d-dimensionale. Aceste numere sunt:
(a) Numărul de puncte reprezentate, N.

Un cluster. Punctele sale
sunt în setul de aruncare.
Centroidul său
Seturi comprimate
Puncte în
set retinut

Figura 7.11: Puncte în seturile aruncate, comprimate și reținute

(b) Suma componentelor tuturor punctelor din fiecare dimensiune. Aceste date este o sumă vectorială de lungime d, iar componenta din dimensiunea a-i-a este $SUMA_i$.

(c) Suma pătratelor componentelor tuturor punctelor din fiecare dimension. Aceste date sunt un vector SUMSQ de lungime d și componenta sa în cea de-a doua dimensiune este $SUMSQ_i$.

Scopul nostru real este de a reprezenta un set de puncte după numărul lor, centroidul și abaterea standard în fiecare dimensiune. Cu toate acestea, aceste valori 2d + 1 ne oferă acele statistici. N este numărul. Coordonata centroului în dimensiunea i este SUM_i / N, adică suma din acea dimensiune împărțită la numărul de puncte. Varianța în dimensiunea i este $SUMSQ_i$ / N - (SUM_i / N) ^{2} . Putem calculați abaterea standard în fiecare dimensiune, deoarece este rădăcina pătrată a varianței.

Exemplul 7.9: Să presupunem că un cluster este format din punctele (5, 1), (6, −2) și (7, 0). Apoi N = 3, SUM = [18, −1] și SUMSQ = [110, 5]. Centroidul este SUM / N sau [6, -1 / 3]. Varianța în prima dimensiune este 110/3 - (18/3) ^{2} = 0,667, deci abaterea standard este $\sqrt{0,667}$ = 0,816. În a doua dimensiune, varianța este 5/3 - (−1/3) ^{2} = 1,56, deci abaterea standard este 1,25. ✷

7.3.5 Prelucrarea datelor în algoritmul BFR

Vom prezenta acum ce se întâmplă atunci când procesăm o parte din puncte.

Beneficiile N, USM , SUMSQ Reprezentanța

Există un avantaj semnificativ în reprezentarea seturilor de puncte pe măsură ce se face în algoritmul BFR, mai degrabă decât prin stocarea N, a centrului și a abaterea standard în fiecare dimensiune. Luați în considerare ce trebuie să facem când adăugăm un punct nou unui cluster. N este mărit cu 1, desigur. Dar noi poate adăuga, de asemenea, vectorul care reprezintă locația punctului la SUM la obține noul SUM și putem adăuga pătratele componentelor vector la SUMSQ pentru a obține noul SUMSQ . Dacă am fi folosit centroidul la locul său din SUM , atunci nu am putut regla centroidul pentru a explica noul punct fără a face unele calcule care implică N și recomputarea deviațiile standard ar fi, de asemenea, mult mai complexe. În mod similar, dacă noi dorim să combinăm două seturi, adăugăm doar valorile corespunzătoare ale lui N, SUM , și SUMSQ , în timp ce dacă am folosi abaterile standard și centroid ca a reprezentare, calculul ar fi mult mai complex.

1. În primul rând, toate punctele care sunt suficient de aproape de centroidul unui cluster sunt adăugat la acel cluster. Așa cum este descris în caseta privind beneficiile, este simplu pentru a adăuga informații despre punct la N, SUM și SUMSQ care reprezintă clusterul. Atunci aruncăm ideea. Întrebarea de ce Mijloacele „suficient de apropiate” vor fi abordate în scurt timp.

2. Pentru punctele care nu sunt suficient de apropiate de orice centroid, am încheiat terminați-le, împreună cu punctele din setul reținut. Orice memorie principală poate fi utilizat algoritmul de clusterizare, cum ar fi metodele ierarhice discutate în secțiunea 7.2. Trebuie să folosim un criteriu pentru a decide când este rezonabil să combinați două puncte într-un cluster sau două clustere într-unul. Secțiunea 7.2.3 a acoperit modalitățile prin care am putea lua această decizie. Clustere de mai multe puncte sunt rezumate și adăugate la setul comprimat.
Clusterele Singleton devin setul de puncte reținut.

3. Acum avem miniclusturi derivate din încercarea noastră de a grupa puncte noi și vechiul set păstrat și avem miniclusterele din vechile com-set presat. Deși niciunul dintre aceste miniclustere nu poate fi îmbinat cu unul din grupurile k, acestea ar putea fuziona între ele. Criteriul pentru

fuziunea poate fi din nou aleasă în conformitate cu discuția din secțiunea 7.2.3.
Rețineți că forma de reprezentare pentru seturile comprimate (N, SUM și SUMSQ) facilitează calcularea statisticilor, cum ar fi varianța pentru combinație de două miniclusturi pe care le considerăm fuzionate.
4. Puncte care sunt atribuite unui cluster sau unui minicluster, adică acelea care nu se află în setul reținut, sunt scrise, cu atribuirea lor, către memorie secundară.

Pagina 293

În cele din urmă, dacă aceasta este ultima bucată de date de intrare, trebuie să facem ceva cu mulțimile comprimate și reținute. Îi putem trata ca valori aberante și nu le grupa niciodată. Sau, putem atribui fiecare punct din setul reținut la grupul celui mai apropiat centroid. Putem combina fiecare minicluster cu cluster al cărui centroid este cel mai aproape de centroidul miniclusterului.
O decizie importantă care trebuie examinată este modul în care decidem dacă a noul punct p este suficient de apropiat de unul dintre grupurile k pe care are sens să le adăugăm p la cluster. Au fost sugerate două abordări.
(a) Adăugați p la un cluster dacă nu numai că are cel mai apropiat centroid de p, dar este foarte puțin probabil ca, după ce toate punctele să fi fost prelucrate, să existe altele se va constata că centroidul cluster este mai aproape de p. Această decizie este complexă calcul statistic. Trebuie să presupunem că punctele sunt ordonate aleatoriu, și că știm câte puncte vor fi procesate în viitor. Este avantajul este că, dacă găsim un centroid care este semnificativ mai aproape de p decât oricare altul, putem adăuga p la acel cluster și putem termina cu el, chiar dacă p este foarte departe de toți centroizii.
(b) Putem măsura probabilitatea ca, dacă p aparține unui cluster, ar fi să fie găsit cât de departe este de centroul acelui cluster. Acest calcul folosește faptul că credem că fiecare cluster constă în mod normal puncte distribuite cu axele distribuției aliniate cu axele a spațiului. Ne permite să facem calculul prin Mahalanobis distanța punctului, pe care o vom descrie în continuare.
Distanța Mahalanobis este în esență distanța dintre un punct și Centroidul unui cluster, normalizat prin deviația standard a clusterului în fiecare dimensiune. Deoarece algoritmul BFR își asumă axele clusterului aliniați cu axele spațiului, calculul distanței Mahalanobis este mai ales simplu. Fie $p = [p_1, p_2, ..., p_d]$ un punct și $c = [c_1, c_2, ..., c_d]$ centroid al unui cluster. Fie σ_i abaterea standard a punctelor din cluster în a zecea dimensiune. Apoi distanța Mahalanobis între p și c este

$$\sqrt{\sqrt{\sqrt{\sqrt{\sum_{i=1}^{d} \left(\frac{p_i - c_i}{\sigma_i}\right)^2}}}}$$

Adică, normalizăm diferența dintre p și c în dimensiunea ith cu împărțind la abaterea standard a clusterului în acea dimensiune. Restul formula combină distanțele normalizate în fiecare dimensiune în normal cale pentru un spațiu euclidian.
Pentru a atribui punctul p unui cluster, calculăm distanța dintre Mahalanobis p și fiecare dintre centroizii cluster. Alegem acel cluster al cărui centru are cea mai mică distanță Mahalanobis și adăugăm p la acel cluster cu condiția Distanța Mahalanobis este mai mică decât un prag. De exemplu, să presupunem că alegem patru ca prag. Dacă datele sunt distribuite în mod normal, atunci probabilitatea

---

Pagina 294

o valoare în măsura în care patru abateri standard de la medie este mai mică decât una la a milion. Astfel, dacă punctele din cluster sunt într-adevăr distribuite în mod normal, atunci

probabilitatea ca nu vom include un punct care aparține cu adevărat este mai mică
mai mult de 10^{-6}. Și un astfel de punct este probabil să fie atribuit acelui cluster în cele din urmă
oricum, atâta timp cât nu se termină mai aproape de un alt centru ca centroidii
migrează ca răspuns la punctele adăugate la clusterul lor.

7.3.6 Exerciții pentru secțiunea 7.3

Exercițiul 7.3.1: Pentru punctele din Fig. 7.8, dacă selectăm trei puncte de plecare
folosind metoda Secțiunii 7.3.2, iar primul punct pe care îl alegem este (3,4), care
sunt selectate alte puncte.

!! Exercițiul 7.3.2: Dovediți că indiferent de ce punct pornim în Fig. 7.8, dacă
selectăm trei puncte de plecare prin metoda secțiunii 7.3.2 obținem puncte în
fiecare dintre cele trei clustere. Sugestie: ați putea rezolva acest lucru exhaustiv începând
cu fiecare dintre cele douăsprezece puncte la rând. Cu toate acestea, o aplicație mai generală
soluția este de a lua în considerare diametrele celor trei clustere și de a lua în considerare
distanța minimă intercluster, adică distanța minimă între două
puncte alese din două clustere diferite. Poți dovedi o teoremă generală
pe baza acestor doi parametri ai unui set de puncte?

! Exercițiul 7.3.3: Dați un exemplu de set de date și o selecție de k inițiale
centroizi astfel încât atunci când punctele sunt realocate la cel mai apropiat centroid la
la final, cel puțin unul dintre k punctele inițiale este reatribuit la un cluster diferit.

Exercițiul 7.3.4: Pentru cele trei grupuri din Fig. 7.8:
(a) Calculați reprezentarea clusterului ca în algoritmul BFR. Acea
este, calculați N, SUM și SUMSQ .
(b) Calculați varianța și deviația standard a fiecărui cluster din fiecare
cele două dimensiuni.

Exercițiul 7.3.5: Să presupunem că un grup de puncte tridimensionale are standard
abateri de 2, 3 și 5, în cele trei dimensiuni, în această ordine. Calculați
Distanța Mahalanobis între origine (0, 0, 0) și punct (1, −3, 4).

7.4 Algoritmul CURE

Trecem acum la un alt algoritm de grupare pe scară largă în atribuirea punctelor
clasă. Acest algoritm, numit CURE (Clustering Using REpresentatives), ca-
somează un spațiu euclidian. Cu toate acestea, nu presupune nimic despre
forma clusterelor; nu trebuie să fie distribuite în mod normal și chiar pot avea
coturi ciudate, forme S sau chiar inele. În loc să reprezinte clustere după
centroid, folosește o colecție de puncte reprezentative, așa cum sugerează și numele.

Figura 7.12: Două clustere, una înconjurând-o pe cealaltă

Exemplul 7.10: Figura 7.12 este o ilustrare a două clustere. Clusul interior-
ter este un cerc obișnuit, în timp ce al doilea este un inel în jurul cercului. Acest
aranjamentul nu este complet patologic. O creatură dintr-o altă galaxie
s-ar putea să privească sistemul nostru solar și să observe că obiectele se adună într-un interior
cerc (planetele) și un inel exterior (centura Kuyper), cu puțin între ele.

✷

7.4.1 Inițializare în CURE

Începem algoritmul CURE prin:

1. Luați un mic eșantion de date și grupați-le în memoria principală. În principiu
ciple, ar putea fi utilizată orice metodă de clusterizare, dar așa cum este conceput pentru CURE
se ocupă de clustere în formă ciudată, este de multe ori recomandabil să utilizați un ierarhic
metoda prin care clusterele sunt fuzionate atunci când au o pereche apropiată de puncte.
Această problemă este discutată mai detaliat în Exemplul 7.11 de mai jos.
2. Selectați un set mic de puncte din fiecare cluster pentru a fi puncte reprezentative.
Aceste puncte trebuie alese pentru a fi cât mai îndepărtate unele de altele,
folosind metoda descrisă în secțiunea 7.3.2.
3. Mutați fiecare dintre punctele reprezentative o fracție fixă din distanță
între locația sa și centroidul clusterului său. Poate că 20% este un
fracție bună de ales. Rețineți că acest pas necesită un spațiu euclidian,
deoarece altfel, s-ar putea să nu existe o noțiune de linie între două
puncte.

Exemplul 7.11: Am putea folosi un algoritm ierarhic de grupare pe un eșantion
a datelor din Fig. 7.12. Dacă am lua ca distanță între clustere
cea mai mică distanță între orice pereche de puncte, câte unul de la fiecare grup, apoi noi
ar găsi corect cele două clustere. Adică, bucăți din inel s-ar lipi

---

Pagina 296

împreună și bucăți din cercul interior se lipeau împreună, dar bucăți de inel
ar fi întotdeauna departe de bucățile cercului. Rețineți că, dacă am folosit
regula că distanța dintre clustere a fost distanța dintre centroidele lor,
atunci s-ar putea să nu obținem rezultatul intuitiv corect. Motivul este că
centroizii ambelor clustere se află în centrul diagramei.
Figura 7.13: Selectați puncte reprezentative din fiecare cluster, departe de unul
un altul posibil
Pentru al doilea pas, alegem punctele reprezentative. Dacă proba din
pe care sunt construite clusterele este suficient de mare, putem conta pe un cluster
puncte de probă la cea mai mare distanță una de cealaltă situată la limita
grupul. Figura 7.13 sugerează care este selecția noastră inițială de puncte de eșantionare
ar putea arăta ca.
În cele din urmă, mutăm punctele reprezentative o fracție fixă a distanței
de la locația lor adevărată spre centroul clusterului. Rețineți că în Fig. 7.13
ambele grupuri au centroul lor în același loc: centrul cercului interior.
Astfel, punctele reprezentative din cerc se mișcă în interiorul clusterului, așa cum a fost
intenționat. Punctele de pe marginea exterioară a inelului se deplasează, de asemenea, în grupul lor, dar
punctele de pe marginea interioară a inelului se mișcă în afara clusterului. Locațiile finale ale
punctele reprezentative din Fig. 7.13 sunt sugerate de Fig. 7.14. ✷

7.4.2 Finalizarea algoritmului CURE

Următoarea fază a CURE este de a fuziona două clustere dacă au o pereche de repere
puncte resentative, câte unul din fiecare grup, care sunt suficient de apropiate. Utilizatorul
poate alege distanța care definește „aproape”. Acest pas de fuziune se poate repeta până la
nu mai există clustere suficient de apropiate.
Exemplul 7.12: Situația din Fig. 7.14 servește ca o ilustrare utilă. Acolo
este un argument potrivit căruia inelul și cercul ar trebui să fie combinate într-adevăr, deoarece lor
centroizii sunt la fel. De exemplu, dacă decalajul dintre inel și cerc ar fi

---

Pagina 297

Figura 7.14: Mutarea punctelor reprezentative 20% din distanța până la clus-
Centroidul ter
mult mai mic, s-ar putea argumenta că combinarea punctelor inelului și
cercul într-un singur grup reflecta adevărata stare de lucruri. De exemplu,
inelele lui Saturn au goluri înguste între ele, dar este rezonabil să vizualizăm
inelele ca un singur obiect, mai degrabă decât mai multe obiecte concentrice. În cazul
din Fig. 7.14 alegerea
1. Fracția distanței până la centroid pe care o mutăm reprezentantul
puncte tive și
2. Alegerea distanței dintre punctele reprezentative ale celor două clustere
fie pentru a evita fuziunea
împreună stabilim dacă considerăm Fig. 7.12 ca un cluster sau două. ✷
Ultimul pas al CURE este atribuirea punctelor. Fiecare punct p este adus de la
stocare secundară și comparată cu punctele reprezentative. Atribuim p
la grupul punctului reprezentativ cel mai apropiat de p.
Exemplul 7.13: În exemplul nostru de rulare, punctele din ring vor fi cu siguranță
să fie mai aproape de unul dintre punctele reprezentative ale inelului decât de orice reprezentant
punctul cercului. La fel, punctele din cerc vor fi cu siguranță cele mai apropiate de a
punctul reprezentativ al cercului. Un outlier - un punct care nu se află în ring sau
cercul - va fi atribuit inelului dacă acesta se află în afara inelului. Dacă valoarea anterioară este
între inel și cerc, acesta va fi atribuit uneia sau alteia, oarecum
favorizând inelul, deoarece punctele sale reprezentative au fost mutate spre
cerc. ✷

7.4.3 Exerciții pentru secțiunea 7.4

Exercițiul 7.4.1: Luați în considerare două grupuri care sunt un cerc și un inel înconjurător,
ca în exemplul de rulare al acestei secțiuni. Presupune:

Pagina 298

eu. Raza cercului este c.

ii. Cercurile interioare și exterioare care formează inelul au raze i și o, respectiv
tiv.
iii. Toate punctele reprezentative pentru cele două clustere se află la granițele
clustere.
iv. Punctele reprezentative sunt mutate cu 20% din distanța față de valoarea inițială
poziția spre centroidul grupului lor.
v. Clusterele sunt combinate dacă, după repoziționare, există puncte reprezentative
din cele două clustere la distanța d sau mai mică.
În ceea ce privește d, c, i și o, în ce circumstanțe vor fi inelul și cercul
fuzionat într-un singur cluster?

7.5 Clustering în spații non-euclidiene

În continuare vom lua în considerare un algoritm care tratează date care nu sunt de memorie principală, dar nu necesită un spațiu euclidian. Algoritmul, la care ne vom referi ca
GRGPF pentru autorii săi (V. Ganti, R. Ramakrishnan, J. Gehrke, A. Powell și
J. French), ia idei atât din abordările ierarhice, cât și din cele de atribuire a punctelor.
La fel ca CURE, reprezintă clustere după puncte de eșantionare în memoria principală. In orice caz, de asemenea, încearcă să organizeze clusterele ierarhic, într-un copac, astfel încât un nou punct să poată să fie atribuit clusterului corespunzător, trecându-l în jos în copac. Frunze de arborele conține rezumate ale unor clustere, iar nodurile interioare dețin subseturi ale informații care descriu clusterele accesibile prin acel nod. O incercare
este făcută pentru a grupa clusterele după distanța lor una de cealaltă, deci clusterele de la o frunză este apropiată, iar grupurile accesibile de la un nod interior sunt relativ
aproape de asemenea.

7.5.1 Reprezentarea clusterelor în algoritmul GRGPF

Pe măsură ce atribuim puncte grupurilor, acestea pot crește. Majoritatea punctelor
într-un cluster sunt stocate pe disc și nu sunt utilizate în ghidarea atribuirii
puncte, deși pot fi recuperate. Reprezentarea unui cluster în principal
memoria constă din mai multe caracteristici. Înainte de a enumera aceste caracteristici, dacă p există indicați într-un cluster, fie ROWSUM (p) să fie suma pătratelor distanțelor
de la p la fiecare dintre celelalte puncte din cluster. Rețineți că, deși nu suntem
într-un spațiu euclidian, există o anumită măsură a distanței d care se aplică punctelor,
altfel nu este posibil să se grupeze puncte deloc. Următoarele caracteristici formează
reprezentarea unui cluster.
1. N, numărul de puncte din cluster.

---

2. Clustroidul clusterului, care este definit în mod specific pentru a fi punctul
în cluster care minimizează suma pătratelor distanțelor până la
celelalte puncte; adică clustroidul este punctul din grupul cu
cea mai mică ROWSUM .
3. Suma de rând a clustroidului clusterului.
4. Pentru unele constante alese k, punctele k ale clusterului care sunt cele mai apropiate
clustroidul și sumele lor. Aceste puncte fac parte din reprezentanți
tarea în cazul în care adăugarea de puncte la cluster determină clustroidul să
Schimbare. Se presupune că noul clustroid ar fi unul dintre
aceste k puncte lângă vechiul clustroid.
5. Punctele k ale clusterului care sunt cele mai îndepărtate de clustroid și ale lor
sume de rânduri. Aceste puncte fac parte din reprezentare, astfel încât să putem
ia în considerare dacă două clustere sunt suficient de apropiate pentru a fuziona. Presupunerea
se face că dacă două clustere sunt apropiate, atunci o pereche de puncte distanță de
clustroizii lor respectivi ar fi aproape.

7.5.2 Inițializarea Arborelui Clusterului

Clusterele sunt organizate într-un copac, iar nodurile arborelui pot fi foarte
mari, probabil blocuri de disc sau pagini, așa cum ar fi cazul pentru un copac B sau un copac R,
cu care arborele care reprezintă clusterul seamănă. Fiecare frunză a copacului deține așa
multe reprezentări de grupuri care se pot potrivi. Rețineți că o reprezentare cluster are
o dimensiune care nu depinde de numărul de puncte din cluster.
Un nod interior al arborelui cluster conține un eșantion de clustroizi din
grupuri reprezentate de fiecare dintre subarborii săi, împreună cu indicatori către rădăcinile
acele subarburi. Probele sunt de dimensiuni fixe, deci numărul de copii care
un nod interior poate avea este independent de nivelul său. Observați că pe măsură ce urcăm
arborele, probabilitatea ca clustroidul unui cluster dat să facă parte din eșantion
diminuează.
Inițializăm arborele clusterului luând un eșantion de memorie principală a setului de date
și grupându-l ierarhic. Rezultatul acestei grupări este un copac T, dar T

nu este exact arborele folosit de algoritmul GRGPF. Mai degrabă, selectăm din
T anumite dintre nodurile sale care reprezintă clustere de aproximativ o dimensiune dorită
n. Acestea sunt clusterele inițiale pentru algoritmul GRGPF și le plasăm
reprezentări la frunza arborelui care reprezintă cluster. Apoi grupăm clus-
tere cu un strămoș comun în T în noduri interioare ale reprezentării cluster
arborele, deci, într-un anumit sens, grupurile descendente dintr-un nod interior sunt la fel de apropiate
pe cat posibil. În unele cazuri, reechilibrarea arborelui care reprezintă clusterul va fi
necesar. Acest proces este similar cu reorganizarea unui copac B, și vom face
nu examinați această problemă în detaliu.

Pagina 300

7.5.3 Adăugarea de puncte în algoritmul GRGPF

Acum citim puncte din stocarea secundară și le introducem pe fiecare în cel mai apropiat
grup. Începem de la rădăcină și ne uităm la probele de clustroizi pentru fiecare dintre ele
copiii rădăcinii. Orice copil are cel mai apropiat clustru de nou
punctul p este nodul pe care îl examinăm în continuare. Când ajungem la orice nod din copac, noi
uitați-vă la eșantionul de clustroizi pentru copiii săi și mergeți lângă copilul cu
clustroid cel mai apropiat de p. Rețineți că unii dintre eșantionul de clustroizi de la un nod pot
au fost văzute la un nivel superior, dar fiecare nivel oferă mai multe detalii despre
clustere situate dedesubt, așa că vedem multe eșantioane noi de clustere de fiecare dată când mergem a
nivelează în jos copacul.
În cele din urmă, ajungem la o frunză. Această frunză are caracteristicile clusterului pentru fiecare cluster
reprezentat de acea frunză și alegem clusterul al cărui clustroid este cel mai apropiat de p.
Reglăm reprezentarea acestui cluster pentru a ține cont de noul nod p. În
în special, noi:
1. Adăugați 1 la N.
2. Adăugați pătratul distanței dintre p și fiecare dintre nodurile q men-
cionat în reprezentarea la ROWSUM (q). Aceste puncte q includ
clustroid, cele mai apropiate puncte k și cele mai îndepărtate puncte k.
De asemenea, estimăm suma rândurilor lui p, în cazul în care p trebuie să facă parte din reprezentant
tarea (de exemplu, se dovedește a fi unul dintre k punctele cele mai apropiate de clustroid). Notă
nu putem calcula ROWSUM (p) exact, fără a merge pe disc și a recupera
toate punctele clusterului. Estimarea pe care o folosim este
ROWSUM (p) = ROWSUM (c) + Nd^2 (p, c)
unde d (p, c) este distanța dintre p și clustroidul c. Rețineți că N și
ROWSUM (c) din această formulă sunt valorile acestor caracteristici înainte de a fi
ajustat pentru a contura adăugarea p.
Ne-am putea întreba de ce funcționează această estimare. În secțiunea 7.1.3 am discutat
„blestemul dimensionalității”, în special observația că într-un
spațiul euclidian dimensional, aproape toate unghiurile sunt unghiuri drepte. Desigur
presupunerea algoritmului GRGPF este că spațiul ar putea să nu fie euclidian,
dar de obicei un spațiu non-euclidian suferă și de blestemul dimensionalității,
prin aceea că se comportă în multe feluri ca un spațiu euclidian de înaltă dimensiune. Dacă noi
presupunem că unghiul dintre p, c și un alt punct q din cluster este un drept
unghi, atunci teorema lui Pitagora ne spune că
d^2 (p, q) = d^2 (p, c) + d^2 (c, q)
Dacă însumăm peste toate q, altele decât c, și apoi adăugăm d^2 (p, c) la ROWSUM (p) la
cont de faptul că clustroidul este unul dintre punctele din cluster, noi
derivă ROWSUM (p) = ROWSUM (c) + Nd^2 (p, c).
Acum, trebuie să vedem dacă noul punct p este unul dintre cele mai apropiate sau mai îndepărtate k
punctele de la clustroid și, dacă da, p și suma sa de rânduri devin o caracteristică de cluster,

---

Pagina 301

înlocuind una dintre celelalte caracteristici - oricare dintre ele nu mai este una dintre cele mai apropiate
sau cel mai îndepărtat. De asemenea, trebuie să ne gândim dacă suma rândurilor pentru unul dintre k
cele mai apropiate puncte q sunt acum mai mici decât ROWSUM (c). Această situație s-ar putea întâmpla dacă p
erau mai aproape de unul dintre aceste puncte decât de actualul clustroid. Dacă da, schimbăm
rolurile lui c și q. În cele din urmă, este posibil ca adevăratul clustroid să nu
să mai fie unul dintre cele mai apropiate puncte k originale. De atunci nu avem de unde să știm
nu vedem celelalte puncte ale clusterului în memoria principală. Cu toate acestea, ei
sunt stocate pe disc și pot fi aduse periodic în memoria principală pentru un
recomputarea caracteristicilor clusterului.

7.5.4 Divizarea și fuzionarea clusterelor

Algoritmul GRGPF presupune că există o limită pe raza unui cluster ar putea avea. Definiția particulară utilizată pentru rază este $\sqrt{\text{ROWSUM}(c) / N}$, unde c este clustroidul clusterului și N numărul de puncte din cluster. Adică, raza este rădăcina pătrată a pătratului mediu al distanței de la clustroidul punctelor din cluster. Dacă raza unui cluster crește prea mare, este împărțit în două. Punctele acelui cluster sunt aduse în memoria principală, și împărțit în două clustere pentru a minimiza sumele de rânduri. Caracteristicile clusterului pentru ambele clustere sunt calculate.

Ca rezultat, frunza clusterului divizat are acum încă un cluster de reprezentat. Ar trebui să gestionăm arborele cluster ca un copac B, deci, de obicei, va fi loc într-o frunză pentru a adăuga încă un cluster. Cu toate acestea, dacă nu, atunci frunza trebuie împărțită în două frunze. Pentru a implementa divizarea, trebuie să adăugăm un alt indicator și altele proba de clustroizi la nodul părinte. Din nou, poate exista spațiu suplimentar, dar dacă nu, atunci și acest nod trebuie să fie împărțit și facem acest lucru pentru a minimiza pătratele de distanțele dintre eșantionul de clustroizi alocați diferitelor noduri. Ca în un copac B, această împărțire poate ondula până la rădăcină, care poate fi apoi împărțiți dacă este necesar.

Cel mai rău lucru care se poate întâmpla este că arborele care reprezintă clusterul este acum prea mare pentru a se potrivi în memoria principală. Există doar un singur lucru de făcut: îl realizăm mai mică prin creșterea limitei cât de mare poate fi raza unui cluster și noi ia în considerare fuzionarea perechilor de clustere. În mod normal, este suficient să se ia în considerare clusterele care sunt „în apropiere”, în sensul că reprezentanții lor sunt la aceeași frunză sau la frunze cu un părinte comun. Cu toate acestea, în principiu, putem lua în considerare fuzionând orice două clustere C_1 și C_2 într-un singur cluster C.

Pentru a fuziona clustere, presupunem că clustroidul lui C va fi unul dintre puncte care sunt cât mai departe de clustroidul lui C_1 sau clustroidul de C_2. Să presupunem că vrem să calculăm suma de rânduri în C pentru punctul p, care este unul din punctele k din C_1 care sunt cât mai departe posibil de centroidul lui C_1. Noi folosiți argumentul blestemului dimensionalității care spune că toate unghiurile sunt aproximativ unghiuri drepte, pentru a justifica următoarea formulă.

$$\text{ROWSUM}_C(p) = \text{ROWSUM}_{C_1}(p) + N_{C_2}(d_2(p, c_1) + d_2(c_1, c_2)) + \text{ROWSUM}_{C_2}(c_2)$$

În cele de mai sus, subscriem N și ROWSUM de la clusterul la care funcția respectivă

---

se referă. Folosim c_1 și c_2 pentru clustroizii C_1 și respectiv C_2. În detaliu, calculăm suma pătratelor distanțelor de la p la toate nodurile din clusterul combinat C începând cu $\text{ROWSUM}_{C_1}(p)$ pentru a obține termenii pentru punctele din același cluster ca p. Pentru N_{C_2} puncte q în C_2, considerăm calea de la p la clustroidul lui C_1, apoi la clustroidul lui C_2, și în cele din urmă la q. Presupunem că există un unghi drept între picioare de la p la c_1 și c_1 până la c_2 și un alt unghi drept între cea mai scurtă cale de la p la c_2 și piciorul de la c_2 la q. Apoi folosim teorema lui Pitagora pentru a justifica calculând pătratul lungimii căii la fiecare q ca suma a pătrate ale celor trei picioare.

Apoi trebuie să terminăm de calculat caracteristicile clusterului fuzionat. Noi trebuie să luăm în considerare toate punctele din clusterul fuzionat pentru care știm rânduri. Acestea sunt centroidele celor două grupuri, punctele k cele mai apropiate de clustroizi pentru fiecare grup și punctele k cele mai îndepărtate de clustroizi pentru fiecare cluster, cu excepția punctului care a fost ales ca nou clustroid.

Putem calcula distanțele față de noul clustroid pentru fiecare dintre aceste 4k + 1 puncte. Selectăm k cu cele mai mici distanțe ca puncte „apropiate” și k cu cele mai mari distanțe ca punctele „îndepărtate”. Putem apoi calcula sume de rânduri pentru punctele alese, folosind aceleași formule de mai sus cu care ne obișnuiam calculați sumele de rânduri pentru clustroizii candidați.

7.5.5 Exerciții pentru secțiunea 7.5

Exercițiul 7.5.1: Folosind reprezentarea cluster din secțiunea 7.5.1, reprezentați cele douăsprezece puncte din Fig. 7.8 ca un singur cluster. Utilizați parametrul k = 2 ca numărul de puncte apropiate și îndepărtate care trebuie incluse în reprezentare. Aluzie: Deoarece distanța este euclidiană, putem obține pătratul distanței dintre două puncte luând suma pătratelor diferențelor de-a lungul x- și axe-y.

Exercițiul 7.5.2: Calculați raza, în sensul folosit de GRGPF Algoritm (rădăcină pătrată a pătratului mediu al distanței de la clustroid) pentru grupul care reprezintă cele cinci puncte din dreapta jos din Fig. 7.8. Rețineți că (11,4) este clustroidul.

7.6 Clustering pentru fluxuri și paralelism

În această secțiune, vom analiza pe scurt modul în care s-ar putea grupa un flux. The modelul pe care îl avem în minte este unul în care există o fereastră glisantă (4.1.3) din N puncte și putem cere centroizii sau clustroizii din cele mai bune clustere formate din ultimii m ai acestor puncte, pentru orice m ≤ N. De asemenea studiați o abordare similară grupării unui set mare de puncte fixe folosind MapReduce pe un cluster de calcul (fără joc de cuvinte). Această secțiune oferă numai o schiță aproximativă pentru a sugera posibilitățile, care depind de presupunerile noastre despre cum evoluează clusterele într-un flux.

Pagina 303

7.6.1 Modelul de calcul al fluxului

Presupunem că fiecare element de flux este un punct într-un anumit spațiu. Alunecarea fereastra constă din cele mai recente N puncte. Scopul nostru este de a precluziona subseturi punctelor din flux, astfel încât să putem răspunde rapid la întrebările din formular

,,Care sunt grupurile din ultimele m puncte?” pentru orice m ≤ N. Există multe variante ale acestei interogări, în funcție de ceea ce presupunem despre ceea ce constituie un cluster. De exemplu, putem folosi o abordare k-means, unde ne aflăm cu adevărat cerând ca ultimele m puncte să fie partiționate în exact k clustere. Sau, putem permiteți numărul de clustere să varieze, dar utilizați unul dintre criteriile din secțiunea 7.2.3 sau 7.2.4 pentru a determina când se va opri combinarea clusterelor în clustere mai mari.
Nu facem nicio restricție în ceea ce privește spațiul în care punctele fluxului Trăi. Poate fi un spațiu euclidian, caz în care răspunsul la interogare este centroizii grupurilor selectate. Spațiul poate fi neeuclidian, în care caz, răspunsul este clusteroidele grupurilor selectate, unde oricare dintre pot fi utilizate definiții pentru „clustroid” (a se vedea secțiunea 7.2.4).
Problema este mult mai ușoară dacă presupunem că toate elementele fluxului sunt ales cu statistici care nu variază de-a lungul fluxului. Apoi, un eșantion de fluxul este suficient de bun pentru a estima clusterele și, de fapt, putem ignora flux după un timp. Cu toate acestea, modelul fluxului presupune în mod normal că statisticile elementelor fluxului variază în funcție de timp. De exemplu, centroizii grupurile pot migra încet odată cu trecerea timpului sau clusterele se pot extinde, contractează, divizează sau fuzionează.

7.6.2 Un algoritm de grupare a fluxului

În această secțiune, vom prezenta o versiune mult simplificată a unui algoritm denumit BDMO (pentru autori, B. Babcock, M. Datar, R. Motwani, și L. O'Callaghan). Versiunea adevărată a algoritmului implică mult mai mult structuri complexe, care sunt concepute pentru a oferi garanții de performanță în cel mai rău caz.
Algoritmul BDMO se bazează pe metodologia de numărare a celor dintr-un flux care a fost descris în secțiunea 4.6. Iată asemănările cheie și diferențe:

- La fel ca acel algoritm, punctele fluxului sunt partiționate în și rezumat de, găleți ale căror dimensiuni sunt o putere de două. Aici, dimensiunea o găleată este numărul de puncte pe care le reprezintă, mai degrabă decât numărul de flux elemente care sunt 1.
- Ca și până acum, dimensiunile găleților respectă restricția că există una sau două din fiecare dimensiune, până la o anumită limită. Cu toate acestea, nu presupunem că secvența de dimensiuni admisibile a cupei începe cu 1. Mai degrabă, sunt necesare numai pentru a forma o secvență în care fiecare dimensiune este de două ori mai mare decât dimensiunea anterioară, de exemplu, 3, 6, 12, 24, ...

Pagina 304

- Dimensiunile găleatelor sunt reduse din nou pentru a nu scădea pe măsură ce ne întoarcem timp. Ca și în secțiunea 4.6, putem concluziona că va exista O (log N) găleți.
- Conținutul unei găleți constă din:

1. Dimensiunea cupei.
2. Marcajul de timp al cupei, adică cel mai recent punct pe care contribuie la găleată. La fel ca în Secțiunea 4.6, pot fi marcaje de timp

înregistrat modulo N.
3. O colecție de înregistrări care reprezintă clusterele în care
punctele acelei găleți au fost partiționate. Aceste înregistrări conțin:
(a) Numărul de puncte din cluster.
(b) Centroidul sau clustroidul clusterului.
(c) Orice alți parametri necesari pentru a ne permite să fuzionăm clustere
și să mențină aproximări la setul complet de parametri pentru
cluster combinat. Vom da câteva exemple când vom discuta
procesul de fuziune din secțiunea 7.6.4.

7.6.3 Inițializarea găleților

Cea mai mică dimensiune a cupei noastre va fi p, o putere de 2. Astfel, fiecare p element de flux,
creăm o nouă găleată, cu cele mai recente p puncte. Marca de timp pentru aceasta
cupa este marca de timp a celui mai recent punct din cupă. S-ar putea să plecăm
fiecare punct dintr-un cluster de la sine, sau putem efectua o grupare a acestor puncte
conform oricărei strategii de clusterizare pe care am ales-o. De exemplu, dacă noi
alegeți un algoritm k-înseamnă, apoi (presupunând k <p) grupăm punctele în
k clustere de un algoritm.
Orice metodă folosim pentru a grupa inițial, presupunem că este posibil
calculați centroizii sau clustroizii pentru grupuri și numărați punctele în
fiecare cluster. Aceste informații devin parte a înregistrării pentru fiecare cluster. Noi
calculați, de asemenea, orice alți parametri pentru clustere vor fi necesari în
proces de fuziune.

7.6.4 Fuzionarea găleților

Urmând strategia din secțiunea 4.6, ori de câte ori creăm o nouă găleată, noi
trebuie să revizuiți secvența găleților. În primul rând, dacă o găleată are o marcă de timp
adică mai mult de N unități de timp înainte de ora curentă, atunci nimic din asta
cupa este în fereastră și este posibil să o scoatem din listă. În al doilea rând, s-ar putea să avem
am creat trei găleți de dimensiunea p, caz în care trebuie să fuzionăm cele mai vechi două dintre
cei trei. Fuziunea poate crea două găleți de dimensiunea 2p, caz în care este posibil
trebuie să fuzioneze găleți de dimensiuni crescânde, recursiv, la fel ca în secțiunea 4.6.
Pentru a îmbina două găleți consecutive, trebuie să facem mai multe lucruri:

---

1. Dimensiunea găleții este de două ori mai mare decât cea a celor două găleți care sunt îmbinate.
2. Marcajul de timp pentru cupa îmbinată este marcajul de timp al celui mai recent
dintre cele două găleți consecutive.
3. Trebuie să analizăm dacă să fuzionăm clustere și, dacă da, trebuie să calculăm
parametrii clusterelor fuzionate. Vom detalii despre această parte a
algoritmul luând în considerare mai multe exemple de criterii de fuziune și
modalități de estimare a parametrilor necesari.
Exemplul 7.14: Poate că cel mai simplu caz este în care folosim un k-înseamnă
abordare într-un spațiu euclidian. Reprezentăm clustere după numărul lor
punctele și centroidele lor. Fiecare găleată are exact k grupuri, așa că putem alege
p = k, sau putem alege p mai mare decât k și grupăm punctele p în k grupuri atunci când
creăm o găleată inițial ca în secțiunea 7.6.3. Trebuie să găsim cea mai bună potrivire
între grupurile k ale primei găleți și grupurile k ale celui de-al doilea. Aici,

„Cel mai bun” înseamnă potrivirea care minimizează suma distanțelor dintre
centroizii grupurilor potrivite.
Rețineți că nu avem în vedere fuzionarea a două clustere din același compartiment,
deoarece presupunerea noastră este că clusterele nu evoluează prea mult între
găleți secutive. Astfel, ne-am aștepta să găsim în fiecare dintre cele două găleți adiacente
o reprezentare a fiecăruia dintre k-clustere „adevărate” care există în flux.
Când decidem să îmbinăm două clustere, câte unul din fiecare cupă, numărul
de puncte din clusterul fuzionat este cu siguranță suma numărului de puncte din
două clustere. Centroidul clusterului fuzionat este media ponderată a
centroizi din cele două grupuri, unde ponderarea este după numărul de puncte
în clustere. Adică, dacă cele două clustere au n_1 și respectiv $_2$ puncte,
și au centroizi c_1 și c_2 (aceștia din urmă sunt vectori d-dimensionali pentru unii d),
atunci clusterul combinat are $n = n_1 + n_2$ puncte și are centroid

$$c = \frac{n_1 c_1 + n_2 c_2}{n_1 + n_2}$$

✷

Exemplul 7.15: Metoda din Exemplul 7.14 este suficientă atunci când grupurile sunt
schimbându-se foarte încet. Să presupunem că ne-am putea aștepta ca centroidii clusterului să

grătați suficient de repede încât, atunci când se potrivesc centroizii de la doi consecutivi
ar putea fi confruntate cu o situație ambiguă, în care nu este
clar care dintre cele două clustere se potrivește cel mai bine cu un cluster dat din cealaltă cupă.
O modalitate de a vă proteja împotriva unei astfel de situații este de a crea mai mult de k clustere în
fiecare găleată, chiar dacă știm că, atunci când interogăm (a se vedea secțiunea 7.6.5), vom face acest lucru
trebuie să fuzioneze în exact k clustere. De exemplu, am putea alege p a fi
mult mai mare decât k și, atunci când îmbinăm, unim clustere numai atunci când rezultatul
este suficient de coerent în conformitate cu unul dintre criteriile prezentate în secțiunea 7.2.3.
Sau am putea folosi o strategie ierarhică și să facem cele mai bune fuziuni, astfel încât
mențineți p> k grupe în fiecare găleată.

Pagina 306

Să presupunem, pentru a fi specific, că vrem să punem o limită asupra sumei
distanțele dintre toate punctele unui cluster și centroul său. Apoi în plus
la numărul de puncte și centroidul unui cluster, putem include o estimare
din această sumă din înregistrarea unui cluster. Când inițializăm o găleată, putem
calculați suma exact. Dar pe măsură ce îmbinăm clustere, acest parametru devine un
numai estimare. Să presupunem că îmbinăm două clustere și dorim să calculăm suma lui
distanțele pentru clusterul fuzionat. Utilizați notația pentru centroizi și numărări în
Exemplul 7.14 și, în plus, fie s_1 și s_2 să fie sumele pentru cele două clustere.
Apoi, putem estima raza clusterului fuzionat
$n_1 | c_1 - c | + n_2 | c_2 - c | + s_1 + s_2$
Adică, estimăm distanța dintre orice punct x și noul centru
c să fie distanța dintre acel punct și centrul său vechi (aceste distanțe însumează
$s_1 + s_2$, ultimii doi termeni din expresia de mai sus) plus distanța de la
Centroidul vechi la cel nou (aceste distanțe se însumează la primii doi termeni ai
expresia de mai sus). Rețineți că această estimare este o margine superioară, de triunghi
inegalitate.
O alternativă este înlocuirea sumei distanțelor cu suma pătratelor lui
distanțele de la puncte la centroid. Dacă aceste sume pentru cele două clustere
sunt t_1 și respectiv t_2, atunci putem produce o estimare pentru aceeași sumă
în noul cluster ca
$n_1 | c_1 - c |_2 + n_2 | c_2 - c |_2 + t_1 + t_2$
Această estimare este aproape corectă dacă spațiul este de înaltă dimensiune, prin „blestem
de dimensionalitate. ” ✷
Exemplul 7.16: Al treilea exemplu al nostru va presupune un spațiu neeuclidian și nr
constrângere asupra numărului de clustere. Vom împrumuta mai multe dintre tehnici
din algoritmul GRGPF din secțiunea 7.5. Mai exact, reprezentăm clustere
prin clustroidul și rândul lor (suma pătratelor distanțelor de la fiecare
nodul clusterului la clustroidul său). Includem în înregistrarea unui cluster
informații despre un set de puncte la distanță maximă de clustroid,
inclusiv distanțele lor față de clustroid și sumele lor rând. Reamintim că
scopul lor este de a sugera un clustroid atunci când acest cluster este fuzionat cu altul.
Când îmbinăm găleți, putem alege una dintre multele modalități de a decide care
clustere de fuzionat. De exemplu, putem considera perechi de clustere în ordine de
distanța dintre clustroizii lor. Putem alege, de asemenea, să fuzionăm clustere
când le considerăm, cu condiția ca suma sumelor lor rând să fie sub o anumită valoare
limită. Alternativ, putem efectua îmbinarea dacă suma de sume de rânduri este împărțită
după numărul de puncte din clustere este sub o limită. Oricare dintre celelalte
strategii discutate pentru a decide când să fuzioneze clustere pot fi folosite, de asemenea,
cu condiția să aranjăm să menținem datele (de exemplu, diametrul clusterului) necesare pentru
a lua decizii.
Atunci trebuie să alegem un nou clustroid, dintre punctele cele mai îndepărtate
din clustroidele celor două clustere fuzionate. Putem calcula sume de rânduri pentru

---

Pagina 307

fiecare dintre acești clustroizi candidați utilizând formulele date în secțiunea 7.5.4. Noi
urmează, de asemenea, strategia dată în acea secțiune pentru a alege un subset al îndepărtatului
punctele din fiecare cluster să fie setul de puncte îndepărtate pentru clusterul fuzionat,
și să calculeze noile sume de rânduri și distanța până la clustroid pentru fiecare. ✷

7.6.5 Răspunsul la întrebări

Amintiți-vă că presupunem că o interogare este o cerere pentru grupurile din cele mai recente m

puncte din flux, unde m ≤ N. Din cauza strategiei pe care am adoptat-o
de a combina găleți pe măsură ce ne întoarcem în timp, este posibil să nu putem găsi un set
de găleți care acoperă exact ultimele m puncte. Cu toate acestea, dacă alegem
cel mai mic set de găleți care acoperă ultimele m puncte, vom include în acestea
găleți nu mai mult de ultimele 2m puncte. Vom produce, ca răspuns la
interogare, centroizii sau clustroizii tuturor punctelor din gălețile selectate. În
pentru ca rezultatul să fie o bună aproximare la clustere pentru exact
ultimele m puncte, trebuie să presupunem că punctele cuprinse între 2m și m + 1 nu vor fi
au statistici radical diferite de cele mai recente m puncte. Cu toate acestea, dacă
statisticile variază prea rapid, amintim din secțiunea 4.6.6 că este mai complex
schema de cupare poate garanta că putem găsi găleți pentru a acoperi cel mult
ultimul m (1 + ϱ) puncte, pentru orice ϱ> 0.
După ce am selectat gălețile dorite, le punem în comun toate grupurile. Apoi folosim
o metodologie pentru a decide ce clustere să fuzioneze. Exemplele 7.14 și
7.16 sunt ilustrații ale două abordări ale acestei fuziuni. De exemplu, dacă suntem
necesare pentru a produce exact k clustere, atunci putem îmbina clustere cu
cei mai apropiați centroizi până când rămânem doar cu k clustere, ca în Exemplul 7.14. Sau
putem lua o decizie dacă fuzionăm sau nu clustere în diferite moduri, așa cum noi
eșantionat în exemplul 7.16.

7.6.6 Clusterizarea într-un mediu paralel

Acum, să luăm în considerare pe scurt utilizarea paralelismului disponibil într-un computer
grup. [3] Presupunem că ni se oferă o colecție foarte mare de puncte și ne dorim
să exploateze paralelismul pentru a calcula centroizii grupurilor lor. Cel mai simplu
abordarea este de a utiliza o strategie MapReduce, dar în majoritatea cazurilor suntem constrânși
pentru a utiliza o singură sarcină de reducere.
Începeți prin crearea mai multor sarcini de hartă. Fiecărei sarcini i se atribuie un subset al fișierului
puncte. Sarcina funcției Hărți este de a grupa punctele pe care i se acordă. Rezultatul său este
un set de perechi cheie-valoare cu o cheie fixă 1 și o valoare care este descrierea
dintr-un cluster. Această descriere poate fi oricare dintre posibilitățile sugerate în
Secțiunea 7.6.2, cum ar fi centroidul, numărul și diametrul clusterului.
Deoarece toate perechile cheie-valoare au aceeași cheie, poate exista o singură reducere
sarcină. Această sarcină primește descrieri ale clusterelor produse de fiecare hartă

[3] Nu uitați că termenul „cluster” are două semnificații complet diferite în acest sens
secțiune.

---

sarcini și trebuie să le îmbine în mod corespunzător. Putem folosi discuția din Sec-
tiunea 7.6.4 ca reprezentativă a diferitelor strategii pe care le-am putea folosi pentru a produce
gruparea finală, care este rezultatul sarcinii Reduce.

7.6.7 Exerciții pentru secțiunea 7.6

Exercițiul 7.6.1: Executați algoritmul BDMO cu p = 3 în cele ce urmează
Date euclidiene unidimensionale:
1, 45, 80, 24, 56, 71, 17, 40, 66, 32,48,96, 9, 41, 75, 11, 58,93, 28, 39, 77
Algoritmii de grupare sunt k-înseamnă cu k = 3. Numai centroidul unui cluster,
împreună cu numărul său, este necesar pentru a reprezenta un cluster.
Exercițiul 7.6.2: Folosind clusterele din exercițiul 7.6.1, produceți cele mai bune
centroizi ca răspuns la o interogare care solicită gruparea ultimelor 10 puncte.

7.7 Rezumatul capitolului 7

♦ Clusterizare: Clusterele sunt adesea un rezumat util al datelor care sunt în formă
de puncte într-un anumit spațiu. Pentru a grupa punctele, avem nevoie de o măsură a distanței
pe acel spațiu. În mod ideal, punctele din același cluster au distanțe mici între
între ele, în timp ce punctele din diferite grupuri au distanțe mari între ele
lor.

♦ Algoritmi de clusterizare: algoritmii de clusterizare au în general unul din doi
forme. Algoritmii ierarhici de clusterizare încep cu toate punctele dintr-un cluster
proprii, iar grupurile din apropiere sunt combinate iterativ. Alocarea punctelor
algoritmii de grupare iau în considerare punctele la rândul lor și le atribuie
ter în care se potrivesc cel mai bine.

♦ Blestemul dimensionalității: puncte în spa euclidian de înaltă dimensiune-
ces, precum și punctele din spațiile neeuclidiene se comportă adesea neintuitiv.
Două proprietăți neașteptate ale acestor spații sunt că punctele aleatorii sunt
aproape întotdeauna la aproximativ aceeași distanță, iar vectorii aleatori sunt aproape
întotdeauna ortogonale.

♦ Centroizi și clustroizi: într-un spațiu euclidian, membrii unui grup

poate fi mediat, iar această medie se numește centroid. În non-euclidian spații, nu există nicio garanție că punctele au o „medie”, așa că noi suntem obligat să folosească unul dintre membrii clusterului ca reprezentant sau element tipic al clusterului. Acest reprezentant se numește clustroid.

♦ Alegerea Clustroidului: Există multe modalități prin care putem defini un punct tipic a unui cluster într-un spațiu non-euclidian. De exemplu, am putea alege punct cu cea mai mică sumă de distanțe față de celelalte puncte, cea mai mică suma pătratelor acelor distanțe sau cea mai mică distanță maximă la orice alt punct din cluster.

Pagina 309

♦ Raza și diametrul: indiferent dacă spațiul este sau nu euclidian, putem de-amendează raza unui cluster pentru a fi distanța maximă de centroid sau clustroid în orice punct din acel cluster. Putem defini diametrul clusterul să fie distanța maximă dintre oricare două puncte din grup. Sunt cunoscute și definiții alternative, în special ale razei, de exemplu, distanța medie de la centroid la celelalte puncte.

♦ Clusterizare ierarhică: această familie de algoritmi are multe variații, care diferă în primul rând în două domenii. În primul rând, putem alege în diferite moduri care două clustere să fuzioneze în continuare. În al doilea rând, putem decide când să ne oprim procesul de îmbinare în diferite moduri.

♦ Alegerea grupurilor de îmbinare: o strategie pentru a decide cea mai bună pereche de clustere pentru a fuziona într-un cluster ierarhic este de a alege clustere cu cei mai apropiați centroizi sau clustroizi. O altă abordare este de a alege perechea de grupuri cu cele mai apropiate puncte, câte unul din fiecare grup. O a treia abordare este de a utiliza distanța medie între puncte de la cele două clustere.

♦ Oprirea procesului de fuziune: Un cluster ierarhic poate continua până la a rămas un număr fix de clustere. Alternativ, am putea fuziona până când este imposibil să găsești o pereche de clustere a căror fuziune este suficientă compact, de exemplu, clusterul fuzionat are o rază sau un diametru sub unele prag. O altă abordare implică fuziunea atâta timp cât rezultă clusterul are o „densitate” suficient de mare, care poate fi definită în diverse moduri, dar este numărul de puncte împărțit la o anumită măsură a dimensiunii clusterul, de exemplu, raza.

♦ Algoritmi K-Means: Această familie de algoritmi face parte din atribuirea punctelor tastează și își asumă un spațiu euclidian. Se presupune că există exact k clustere pentru unele k cunoscute. După alegerea k centrozelor inițiale ale clusterului, punctele sunt considerate unul câte unul și atribuite celui mai apropiat centroid. Centroidul unui cluster poate migra în timpul atribuirii punctelor și un ultimul pas opțional este să reatribuiți toate punctele, ținând în același timp centrele fixat la valorile lor finale obținute în timpul primei treceri.

♦ Inițializarea algoritmilor K-Means: o modalitate de a găsi k centroizi inițiali este pentru a alege un punct aleatoriu, apoi alegeți k - 1 puncte suplimentare, fiecare ca cât mai departe posibil de punctele alese anterior. O alternativă este pentru a începe cu un mic eșantion de puncte și a utiliza un cluster ierarhic pentru îmbinați-le în k clustere.

♦ Alegerea K într-un algoritm K-Means: Dacă numărul de clustere este necunoscut, putem folosi o tehnică de căutare binară, încercând un cluster k-înseamnă cu diferite valori ale k. Căutăm cea mai mare valoare a lui k pentru care a scăderea sub k grupuri are ca rezultat un diametru mediu radical mai mare a grupurilor. Această căutare poate fi efectuată într-o serie de grupări operații care este logaritmică în adevărata valoare a lui k.

Pagina 310

♦ Algoritmul BFR: Acest algoritm este o versiune a k-mijloacelor concepute pentru gestionați datele care sunt prea mari pentru a se potrivi în memoria principală. Presupune clustere sunt distribuite în mod normal în jurul axelor.

♦ Reprezentarea clusterelor în BFR: punctele sunt citite de pe un disc pe bucăți timp. Clusterele sunt reprezentate în memoria principală prin numărul de numere

ber de puncte, suma vectorială a tuturor punctelor și vectorul format din
însumând pătratele componentelor punctelor din fiecare dimensiune.
O altă colecție de puncte, prea departe de un centru de cluster pentru a fi inclusă
într-un cluster, sunt reprezentate ca „minicluster” în același mod ca k
clustere, în timp ce încă alte puncte, care nu sunt aproape de nici un alt punct, vor
să fie reprezentați ca ei înșiși și numiți puncte „reținute”.

♦ Puncte de procesare în BFR: majoritatea punctelor dintr-o încărcare de memorie principală
va fi atribuit unui cluster din apropiere și parametrilor pentru acel cluster
va fi ajustat pentru a ține cont de noile puncte. Punctele neatribuite pot
pot fi formate în miniclustere noi, iar aceste miniclustere pot fi combinate
cu miniclusteruri descoperite anterior sau puncte reținute. După ultima
încărcarea memoriei, miniclusterele și punctele reținute pot fi combinate cu cele ale acestora
cel mai apropiat grup sau păstrat ca valori aberante.

♦ Algoritmul CURE: Acest algoritm este de tipul atribuirii punctelor.
Este proiectat pentru un spațiu euclidian, dar clusterele pot avea orice formă. Aceasta
gestionează date prea mari pentru a se potrivi în memoria principală.

♦ Reprezentarea clusterelor în CURE: Algoritmul începe prin gruparea a
mic eșantion de puncte. Apoi selectează puncte reprezentative pentru fiecare
cluster, prin alegerea punctelor din cluster care sunt la fel de departe de fiecare
altele posibil. Scopul este de a găsi puncte reprezentative la marginea
grupul. Cu toate acestea, punctele reprezentative sunt apoi mutate cu o fracțiune
de drum spre centroul clusterului, așa că se află oarecum în
interiorul clusterului.

♦ Procesarea punctelor în CURE: După crearea punctelor reprezentative pentru fiecare
cluster, întregul set de puncte poate fi citit de pe disc și atribuit unui
grup. Atribuim un punct dat clusterului punctului reprezentativ
care este cel mai aproape de punctul dat.

♦ Algoritmul GRGPF: Acest algoritm este de tipul atribuirii punctelor.
Se ocupă de date care sunt prea mari pentru a se potrivi în memoria principală și nu
presupuneți un spațiu euclidian.

♦ Reprezentarea clusterelor în GRGPF: Un cluster este reprezentat de numărare
de puncte din cluster, clustroidul, un set de puncte cel mai apropiat de clustroid
și un set de puncte cel mai îndepărtat de clustroid. Punctele din apropiere permit
să schimbăm clustroidul dacă grupul evoluează și punctele îndepărtate
permite combinarea eficientă a clusterelor în circumstanțe adecvate. Pentru
fiecare dintre aceste puncte, înregistrăm și suma de rânduri, adică rădăcina pătrată

Pagina 311

a sumei pătratelor distanțelor de la acel punct la toate celelalte
punctele clusterului.

♦ Organizarea arborelui de clustere în GRGPF: reprezentările de clustere sunt sau-
organizat într-o structură de copac ca un copac B, unde sunt nodurile arborelui
de obicei blocuri de disc și conțin informații despre multe clustere. The
frunzele dețin reprezentarea a cât mai multe clustere posibil, în timp ce inte-
nodurile rior dețin un eșantion de clustroizi ai grupurilor la coborârea lor
frunze dant. Organizăm arborele astfel încât grupurile ale căror reprezentanți
tives sunt în orice subarbore sunt cât mai aproape posibil.

♦ Puncte de procesare în GRGPF: După inițializarea clusterelor dintr-un eșantion de
puncte, introducem fiecare punct în grupul cu cel mai apropiat clustroid.
Datorită structurii copacului, putem începe de la rădăcină și putem alege să vizităm
copilul cu eșantionul clustroid cel mai apropiat de punctul dat. Ca urmare a
această regulă pe o cale din copac ne conduce la o frunză, unde introducem
indicați în grupul cu cel mai apropiat clustroid de pe acea frunză.

♦ Fluxuri de cluster: o generalizare a algoritmului DGIM (pentru numărare)
1 în fereastra glisantă a unui flux) poate fi folosit pentru a grupa puncte
care fac parte dintr-un flux care evoluează lent. Algoritmul BDMO folosește
găleți similare cu cele ale DGIM, cu dimensiuni de găleți permise formând un
secvență în care fiecare dimensiune este de două ori dimensiunea precedentă.

♦ Reprezentarea găleților în BDMO: dimensiunea unei găleți este numărul
de puncte pe care le reprezintă. Găleată în sine deține doar o reprezentare a
grupuri de aceste puncte, nu punctele în sine. Un cluster reprezintă
includ un număr al punctelor, centroidul sau clustroidul,
și alte informații necesare pentru fuzionarea clusterelor conform

unele strategii selectate.

♦ Fuzionarea găleților în BDMO: Atunci când gălețile trebuie îmbinate, găsim cea mai bună potrivire a grupurilor, câte una din fiecare găleți, și combinați-le in perechi. Dacă fluxul evoluează lent, atunci ne așteptăm la găleți consecutive pentru a avea aproape aceleași centroizi de cluster, deci această potrivire are sens.

♦ Răspuns la interogări în BDMO: o interogare este lungimea unui sufix al glisării fereastră. Luăm toate grupurile din toate gălețile care sunt cel puțin parțial în acel sufix și combinați-le folosind o strategie. The clusterele rezultate sunt răspunsul la interogare.

♦ Clustering folosind MapReduce: Putem împărți datele în bucăți și grupați fiecare bucată în paralel, utilizând o sarcină Map. Clustere din fiecare Sarcina de hartă poate fi grupată în continuare într-o singură sarcină de reducere.

Pagina 312

7.8 Referințe pentru capitolul 7

Studiul ancestral al grupării pentru date la scară largă este algoritmul BIRCH din [6]. Algoritmul BFR este de la [2]. Algoritmul CURE se găsește în [5]. Lucrarea despre algoritmul GRGPF este [3]. Contextul necesar în ceea ce privește copacii B și copacii R, puteți găsi în [4]. Studiul grupării pe fluxurile sunt preluate din [1].

1. B. Babcock, M. Datar, R. Motwani și L. O'Callaghan, „Maintaining variantă și k-mediane peste ferestrele fluxului de date, ”Proc. ACM Symp. pe Principiile sistemelor de baze de date, pp. 234-243, 2003.
2. PS Bradley, UM Fayyad și C. Reina, „Scaling clustering algorithms la baze de date mari, ”Proc. Discovery Knowledge and Data Mining, pp. 9–15, 1998.
3. V. Ganti, R. Ramakrishnan, J. Gehrke, AL Powell și JC French :, ,,Clusterizarea seturilor de date mari în spații metrice arbitrare”, Proc. Intl. Conf. pe Ingineria datelor, pp. 502-511, 1999.
4. H. Garcia-Molina, JD Ullman și J. Widom, Sisteme de baze de date: The Ediția a doua carte completă, Prentice-Hall, Upper Saddle River, NJ, 2009.
5. S. Guha, R. Rastogi și K. Shim, „CURE: Un clustering eficient ritm pentru baze de date mari, ”Proc. ACM SIGMOD Intl. Conf. pe Gestionare-mentul datelor, pp. 73-84, 1998.
6. T. Zhang, R. Ramakrishnan și M. Livny, „BIRCH: o date eficientă metoda de clustering pentru baze de date foarte mari, ”Proc. ACM SIGMOD Intl. Conf. on Management of Data, pp. 103-114, 1996.

Pagina 313

Capitolul 8
Publicitate pe web

Una dintre marile surprize ale secolului 21 a fost abilitatea tuturor felurilor de aplicații web interesante pentru a se sprijini prin publicitate, mai degrabă decât abonamentul. În timp ce radioul și televiziunea au reușit să folosească publicitatea ca sursă principală de venit, majoritatea mass-media - ziare și reviste, de exemplu - a trebuit să utilizeze o abordare hibridă, combinând veniturile din publicitate și abonamente.

De departe, cel mai profitabil loc de publicitate online a fost căutarea și o mare parte din eficiența publicității în căutare provine din modelul „AdWords” de potrivire a interogărilor de căutare cu reclamele. Prin urmare, vom dedica mult din acest capitol la algoritmi pentru optimizarea modului în care se realizează această atribuire.

Algoritmii utilizați sunt de un tip neobișnuit; sunt lacomi și sunt „on-linie ”într-un anumit sens tehnic de discutat. Prin urmare, vom devia pentru a discuta aceste două probleme algoritmice - lăcomie și algoritmi on-line - în în general, înainte de a aborda problema AdWords.

O a doua problemă interesantă de publicitate online implică selectarea articolelor către faceți publicitate la un magazin online. Această problemă implică „filtrare colaborativă” unde încercăm să găsim clienți cu un comportament similar pentru a le sugera

cumpărați lucruri cumpărate de clienți similari. Acest subiect va fi tratat în
Secțiunea 9.3.

8.1 Probleme în publicitatea online

În această secțiune, rezumăm problemele tehnice prezentate de
oportunități pentru publicitate online. Începem prin a analiza tipurile de reclame
găsit pe web.

8.1.1 Oportunități de publicitate

Webul oferă mai multe moduri pentru un agent de publicitate de a-și afișa anunțurile potențialului
Clienți. Iată principalele locuri.

---

1. Unele site-uri, cum ar fi eBay, Craig's List sau site-urile de tranzacționare automată permit publicitatea
cumpărătorii își postează anunțurile direct, fie gratuit, contra cost, fie un comision.
2. Anunțurile grafice sunt plasate pe multe site-uri web. Agenții de publicitate plătesc pentru afișare
la o rată fixă pe afișare (un afișaj al anunțului cu descărcarea de
pagina de către un utilizator). În mod normal, o a doua descărcare a paginii, chiar
de către același utilizator, va avea ca rezultat afișarea unui alt anunț și este un al doilea
impresie.
3. Magazinele online, cum ar fi Amazon, afișează reclame în multe contexte. Reclamele
nu sunt plătite de producătorii produsului promovat, dar sunt
selectat de magazin pentru a maximiza probabilitatea ca clientul să o facă
fi interesat de produs. Considerăm acest tip de publicitate în
Capitolul 9.
4. Anunțurile de căutare sunt plasate printre rezultatele unei interogări de căutare. Agenți de publicitate
licitați pentru dreptul de a afișa anunțul lor ca răspuns la anumite întrebări, dar
plătesc numai dacă se face clic pe anunț. Anunțurile speciale care trebuie afișate sunt
selectat de un proces complex, care urmează să fie discutat în acest capitol, implicând
termenii de căutare pentru care a ofertat agentul de publicitate, valoarea ofertei acestora,
probabilitatea observată ca să se facă clic pe anunț și suma totală
bugetul oferit de agentul de publicitate pentru serviciu.

8.1.2 Plasarea directă a anunțurilor

Când agenții de publicitate pot plasa anunțuri direct, cum ar fi un anunț gratuit pe lista lui Craig sau
caracteristica „cumpără-o acum” de pe eBay, există mai multe probleme pe care trebuie să le facă site-ul
a avea de-a face cu. Anunțurile sunt afișate ca răspuns la termenii de interogare, de exemplu, „apartament Palo
Alto." Site-ul Web poate utiliza un index inversat de cuvinte, la fel ca un motor de căutare
face (vezi secțiunea 5.1.1) și returnează acele anunțuri care conțin toate cuvintele din
interogare. Alternativ, se poate cere agentului de publicitate să specifice parametrii anunțului,
care sunt stocate într-o bază de date. De exemplu, un anunț pentru o mașină uzată ar putea specifica
producătorul, modelul, culoarea și anul din meniurile derulante, deci numai clar
pot fi folosiți termeni înțelese. Interogații pot folosi aceleași meniuri de termeni în
interogările lor.
Clasarea anunțurilor este puțin mai problematică, deoarece nu există nimic asemănător linkurilor
pe Web pentru a ne spune ce reclame sunt mai „importante”. O strategie utilizată este
"Cel mai recent primul." Această strategie, deși echitabilă, este supusă abuzului, unde
agenții de publicitate postează mici variații ale anunțurilor lor la intervale frecvente. Tehnologia
tehnologia pentru descoperirea de reclame prea asemănătoare a fost deja acoperită, în
Secțiunea 3.4.
O abordare alternativă este să încercați să măsurați atractivitatea unui anunț.
De fiecare dată când este afișat, înregistrați dacă interogatorul a făcut clic sau nu pe acesta.
Se presupune că se va face clic mai frecvent pe anunțurile atractive decât pe cele care
nu sunt. Cu toate acestea, există mai mulți factori care trebuie luați în considerare la evaluare
reclame:

---

1. Poziția anunțului într-o listă are o influență mare asupra faptului că este sau nu
se face clic. Primul de pe listă are de departe cea mai mare probabilitate, iar
probabilitatea scade exponențial pe măsură ce poziția crește.
2. Anunțul poate avea atractivitate care depinde de termenii de interogare. Pentru
de exemplu, un anunț pentru o decapotabilă uzată ar fi mai atractiv dacă
interogarea de căutare include termenul „convertibil”, chiar dacă ar putea fi un

răspuns valid la întrebările care caută marca respectivă, fără a specifica
indiferent dacă se dorește sau nu un decapotabil.
3. Toate reclamele merită să fie afișate până la probabilitatea lor de clic
poate fi aproximat îndeaproape. Dacă începem toate anunțurile cu o probabilitate de clic
din 0, nu le vom arăta niciodată și astfel nu vom învăța niciodată dacă sunt sau nu
sunt reclame atractive.

8.1.3 Probleme pentru anunțurile grafice

Această formă de publicitate pe web seamănă cel mai mult cu cea tradițională
mass-media. Un anunț pentru un Chevrolet difuzat în paginile New York Times este un
anunțul grafic, iar eficiența acestuia este limitată. Poate fi văzut de mulți oameni, dar
majoritatea nu sunt interesați să cumpere o mașină, doar au cumpărat o mașină, nu conduc,
sau aveți un alt motiv bun pentru a ignora anunțul. Totuși, costul tipăririi anunțului
a fost încă suportat de ziar și, prin urmare, de agentul de publicitate. O impresie
a unui anunț similar pe Yahoo! pagina de pornire va fi relativ ineficientă
din esență același motiv. Taxa pentru plasarea unui astfel de anunț este de obicei o
fracțiune de cent pe impresie.
Răspunsul mass-media tradiționale la această lipsă de concentrare a fost crearea de ziare
pers sau reviste pentru interese speciale. Dacă sunteți producător de cluburi de golf,
difuzarea anunțului dvs. în Golf Digest vă va oferi o creștere a ordinii de mărime
în probabilitatea ca persoana care vă vede anunțul să fie interesată de acesta. Acest
fenomen explică existența multor maga-
zine. Ei pot încărca mult mai mult pe afișare pentru un anunț decât este
priză de uz general, cum ar fi un ziar zilnic. Același fenomen se
pere pe Web. Un anunț pentru cluburi de golf pe sports.yahoo.com/golf are multe
mai multă valoare pe afișare decât același anunț pe Yahoo! pagina de start sau
un anunț pentru Chevrolets pe Yahoo! pagina de golf.
Cu toate acestea, Web-ul oferă o oportunitate de a adapta anunțurile grafice într-un mod care
suportul copiat nu poate: este posibil să utilizați informații despre utilizator
stabiliți ce anunț ar trebui să fie afișat, indiferent de ce pagină sunt
uitandu-ma la. Dacă se știe că lui Sally îi place golful, atunci are sens să arăți
este un anunț pentru cluburi de golf, indiferent de ce pagină se uită. Am putea
determină dragostea lui Sally pentru golf în diferite moduri:
1. Poate aparține unui grup legat de golf pe Facebook.
2. Ea poate menționa frecvent „golf” în e-mailurile postate pe contul ei Gmail.

---

3. Poate petrece mult timp pe Yahoo! pagina de golf.
4. Ea poate emite frecvent interogări de căutare cu termeni legați de golf.
5. Poate marca site-urile Web ale unuia sau mai multor terenuri de golf.
Fiecare dintre aceste metode, precum și multe altele ca acestea, creează o intimitate enormă
probleme. Scopul acestei cărți nu este să încercăm să rezolvăm aceste probleme, care
în practică, probabil, nu au nicio soluție care să satisfacă toate preocupările. Pe
pe de o parte, oamenilor le plac serviciile gratuite care au devenit recent reclame-
acceptate, iar aceste servicii depind de faptul că publicitatea este mult mai eficientă
decât anunțurile convenționale. Există un acord general că, dacă trebuie să existe reclame,
este mai bine să vezi lucruri pe care s-ar putea să le folosești de fapt decât să ai ce pagini
vedere aglomerată de irelevante. Pe de altă parte, există un mare potențial
pentru utilizare abuzivă dacă informațiile părăsesc domeniul mașinilor care execută
algoritmi publicitari și ajunge în mâinile oamenilor reali.

8.2 Algoritmi on-line

Înainte de a aborda problema potrivirii reclamelor cu interogările de căutare, noi
va devia ușor examinând clasa generală la care acești algoritmi
aparține. Această clasă este denumită „on-line” și implică în general o
proach numit „lacom”. De asemenea, oferim, în secțiunea următoare, un exemplu preliminar
a unui algoritm lacom on-line pentru o problemă mai simplă: potrivire maximă.

8.2.1 Algoritmi on-line și off-line

Algoritmii tipici funcționează după cum urmează. Toate datele necesare algoritmului sunt
prezentat inițial. Algoritmul poate accesa datele în orice ordine. La sfarsit,
algoritmul își produce răspunsul. Un astfel de algoritm este numit off-line.
Cu toate acestea, există momente în care nu putem vedea toate datele înainte de
goritmul trebuie să ia unele decizii. Capitolul 4 a acoperit mineritul în flux, unde
am putut stoca doar o cantitate limitată din flux și a trebuit să răspundem la întrebări
despre întregul flux atunci când este chemat să facă acest lucru. Există o formă extremă de
procesarea fluxului, unde trebuie să răspundem cu o ieșire după fiecare element de flux
sosește ment. Astfel, trebuie să decidem despre fiecare element al fluxului, neștiind nimic

la tot viitorul. Algoritmii acestei clase sunt numiți algoritmi on-line. [1]
Așa cum ar fi cazul, ar fi selectarea anunțurilor pentru a fi afișate cu interogări de căutare relativ simplu dacă am putea face acest lucru off-line. Am vedea o lună în valoare de interogări de căutare și analizați și ofertele făcute de agenții de publicitate pentru termenii de căutare ca bugete publicitare pentru luna respectivă, iar apoi am putea atribui reclame

[1] Din păcate, ne confruntăm cu un alt caz de semnificații duale, cum ar fi coincidența implicând termenul „cluster” pe care l-am menționat în secțiunea 7.6.6, unde trebuia să interpretăm expresii corespunzătoare, cum ar fi „algoritmi pentru calcularea clusterelor pe clusterele de computere”. Aici termenul „on-line” se referă la natura algoritmului și nu trebuie confundat cu „on-line” înseamnă „pe internet” în fraze precum „algoritmi on-line pentru publicitate on-line”.

---

interogările într-un mod care maximizează atât veniturile pentru motorul de căutare, cât și numărul de afișări obținute de fiecare agent de publicitate. Problema cu off-line algoritmi este că majoritatea interogatorilor nu vor să aștepte o lună pentru a-și căuta rezultate.

Astfel, trebuie să folosim un algoritm on-line pentru a atribui reclame interogărilor de căutare. Adică, atunci când sosește o interogare de căutare, trebuie să selectăm anunțurile pentru a le afișa interogați imediat. Putem folosi informații despre trecut, de exemplu, nu trebuie să afișăm un anunț dacă bugetul agentului de publicitate a fost deja cheltuit, iar noi poate examina rata de clic (fracțiune din timpul pe care se face clic pe anunț când este afișat) pe care un anunț l-a obținut până acum. Cu toate acestea, nu putem folosi orice despre interogări de căutare viitoare. De exemplu, nu putem ști dacă vor exista o mulțime de interogări care vor ajunge mai târziu și vor folosi termeni de căutare pe care se bazează acest lucru agentul de publicitate a făcut oferte mai mari.

Exemplul 8.1: Să luăm un exemplu foarte simplu de ce să cunoaștem viitorul ar putea ajuta. Un producător A de replici de mobilier antic a licitat 10 cenți pe termenul de căutare „chesterfield”. [2] Un producător mai convențional B are licitați 20 de cenți atât pentru termenii „chesterfield”, cât și pentru „canapea”. Ambele au lunar bugete de 100 USD și nu există alți ofertanți în niciunul dintre acești termeni. Este începutul lunii și tocmai a sosit o interogare de căutare „chesterfield”.

Ne este permis să afișăm un singur anunț cu interogarea.

Lucrul evident de făcut este să afișați anunțul lui B, deoarece acestea licitează mai mult. Cum-vreodată, să presupunem că vor exista o mulțime de interogări de căutare luna aceasta pentru „canapea”, dar foarte puțini pentru „chesterfield”. Apoi A nu își va cheltui niciodată bugetul de 100 USD, în timp ce B. cheltuiți bugetul întreg, chiar dacă îi oferim interogarea lui A. Mai exact, dacă există faceți cel puțin încă 500 de interogări pentru „canapea” sau „chesterfield”, atunci există niciun rău și potențial un beneficiu în a da întrebarea lui A. Va fi în continuare este posibil ca B să-și cheltuiască întreg bugetul, în timp ce creștem suma de Bugetul lui A care va fi cheltuit. Rețineți că acest argument are sens atât din punctul de vedere al motorului de căutare, care dorește să maximizeze veniturile totale, și din punctul de vedere atât al lui A, cât și al lui B, care probabil vor să obțină totul impresiile pe care bugetele le permit.

Dacă am putea cunoaște viitorul, atunci am ști câte alte „canapele” interogări și câte alte întrebări „chesterfield” urmau să ajungă la aceasta lună. Dacă numărul respectiv este sub 500, atunci vrem să îi adresăm interogarea lui B către maximizați veniturile, dar dacă este de 500 sau mai mult, atunci vrem să-i dăm lui A. De când nu cunoaștem viitorul, un algoritm on-line nu poate merge întotdeauna la fel de bine ca un algoritm off-line. ✷

8.2.2 Algoritmi lacomi

Mulți algoritmi on-line sunt de tip lacom. Acești algoritmi ia decizia lor ca răspuns la fiecare element de intrare prin maximizarea unora funcția elementului de intrare și a trecutului.

[2] Un chesterfield este un tip de canapea. A se vedea, de exemplu, www.chesterfields.info.

Exemplul 8.2: Algoritmul lacom evident pentru situația descrisă în Exemplul 8.1 este de a atribui o interogare celui mai mare ofertant care mai are încă buget. Pentru datele din acest exemplu, ceea ce se va întâmpla este că primele 500 de „canapele” sau Interogările „chesterfield” vor fi atribuite lui B. În acel moment, B rămâne fără buget și nu i se mai atribuie interogări. După aceea, următoarele 1000 de interogări „chesterfield” sunt atribuite la A, iar interogările „canapea” nu primesc anunț și, prin urmare, câștigă căutarea motor fara bani.

Cel mai rău lucru care se poate întâmpla este că sosesc 500 de interogări „chesterfield”,

urmat de 500 de interogări „canapea”. Un algoritm off-line ar putea atribui în mod optim primele 500 până la A, câștigând 50 USD, iar următoarele 500 până la B, câștigând 100 USD sau un total de 150 USD. Cu toate acestea, algoritmul lacom va atribui primii 500 lui B, câștigând 100 USD, apoi nu are anunț pentru următoarele 500, fără a câștiga nimic. ✷

8.2.3 Raportul competitiv

După cum vedem din Exemplul 8.2, un algoritm on-line nu trebuie să dea un rezultat atât de bun ca cel mai bun algoritm off-line pentru aceeași problemă. Cel mai mult ne putem aștepta este că va exista o constantă c mai mică de 1, astfel încât pe orice intrare, rezultatul unui anumit algoritm on-line este de cel puțin c ori rezultatul optimului algoritm off-line. Constanta c, dacă există, se numește raportul competitiv pentru algoritmul on-line.

Exemplul 8.3: Algoritmul lacom, pe datele particulare din Exemplul 8.2, oferă un rezultat care este de 2/3 la fel de bun ca cel al algoritmului optim: 100 $ față de 150 USD. Acest lucru demonstrează că raportul competitiv nu este mai mare de 2/3. Dar ar putea fi mai puțin. Raportul competitiv pentru un algoritm poate depinde de ce fel de date poate fi introdus în algoritm. Chiar dacă restrângem intrările la situația descrisă în Exemplul 8.2, dar cu ofertele permise să varieze, atunci putem arăta că algoritmul lacom are un raport competitiv nu mai mare de 1/2. Doar ridicați oferta cu A la Q mai puțin de 20 de cenți. Pe măsură ce Q se apropie de 0, lacomul algoritmul produce în continuare doar 100 $, dar randamentul din algoritmul optim se apropie de 200 de dolari. Putem arăta că este imposibil să faci mai rău decât jumătate din optim în acest caz simplu, deci raportul competitiv este într-adevăr 1/2. In orice caz, vom lăsa acest tip de dovadă pentru secțiunile ulterioare. ✷

8.2.4 Exerciții pentru secțiunea 8.2

! Exercițiul 8.2.1: Un exemplu popular de proiectare a unui algoritm on-line pentru minimizarea raportului competitiv este problema cumpărării schiurilor. [3] Să presupunem că puteți cumpăra schiuri pentru 100 USD sau puteți închiria schiuri pentru 10 USD pe zi. Decizi să începi schiul, dar nu știi dacă îți va plăcea. Puteți încerca să schiați pentru orice număr de zile și apoi renunță la ea. Meritul unui algoritm este costul pe zi al schiurilor, și trebuie să încercăm să reducem la minimum acest cost.

[3] Mulțumesc Anna Karlin pentru acest exemplu.

Un algoritm online pentru luarea deciziei de închiriere / cumpărare este „cumpărați schiuri imediat. ” Dacă încercați să schiați o dată, cădeați în jos și renunțați la acesta, atunci acest lucru on-line algoritmul vă costă 100 USD pe zi, în timp ce algoritmul optim off-line ar costa ai închiriat schiuri cu 10 USD pentru ziua în care le-ai folosit. Astfel, competitivul Raportul algoritmului „cumpărați schiuri imediat” este cel mult 1/10, și anume de fapt, raportul concurențial exact, deoarece utilizarea schiurilor într-o zi este cea mai proastă posibil rezultat pentru acest algoritm. Pe de altă parte, algoritmul on-line

,,Întotdeauna închiriază schiuri” are un raport competitiv arbitrar mic. Dacă se dovedește a îți place foarte mult să schiezi și mergi regulat, apoi după n zile, vei fi plătit 10 milioane USD sau 10 USD / zi, în timp ce algoritmul optim off-line ar fi cumpărat schiuri dintr-o dată, și a plătit doar 100 USD, sau 100 USD / n pe zi.

Întrebarea dvs.: proiectați un algoritm on-line pentru problema cumpărării schiurilor care are cel mai bun raport competitiv posibil. Care este acel raport competitiv? Aluzie: Din moment ce ai putea, în orice moment, să faci o cădere și să te hotărăști să renunți la schi, singurul un lucru pe care îl poate folosi algoritmul online pentru a lua decizia este de câte ori anterior ai plecat la schi.

8.3 Problema potrivirii

Vom aborda acum o problemă care este o versiune simplificată a problemei potrivirea anunțurilor la interogările de căutare. Această problemă, numită „potrivire maximă”, este o problemă abstractă care implică grafice bipartite (grafice cu două seturi de noduri - stânga și dreapta - cu toate marginile conectând un nod din stânga setat la un nod din set corect. Figura 8.1 este un exemplu de grafic bipartit. Nodurile 1, 2, 3 și 4 formează setul din stânga, în timp ce nodurile a, b, c și d formează setul din dreapta.

8.3.1 Meciuri și Meciuri Perfecte

Să presupunem că ni se dă un grafic bipartit. O potrivire este un subset de margini astfel încât niciun nod să nu fie capătul a două sau mai multe muchii. O potrivire se spune că este perfect dacă fiecare nod apare în potrivire. Rețineți că o potrivire poate numai fii perfect dacă seturile din stânga și din dreapta sunt de aceeași dimensiune. O potrivire care este la fel mare ca orice altă potrivire pentru graficul în cauză se spune că este maximă.

Exemplul 8.4: Setul de margini {(1, a), (2, b), (3, d)} este o potrivire pentru grafic bipartit din Fig. 8.1. Fiecare membru al setului este o margine a bipartitului grafic și niciun nod nu apare de mai multe ori. Setul de margini

{(1, c), (2, b), (3, d), (4, a)}
este o potrivire perfectă, reprezentată de linii grele în Fig. 8.2. Fiecare nod apare
exact o dată. Este, de fapt, singura potrivire perfectă pentru acest grafic, deși
unele grafice bipartite au mai multe potriviri perfecte. Potrivirea lui
Fig. 8.2 este, de asemenea, maximă, deoarece fiecare potrivire perfectă este maximă. ✷

4
1
A
b
c
d
2
3
Figura 8.1: Un grafic bipartit

8.3.2 Algoritmul lacom pentru potrivirea maximă

Algoritmii off-line pentru găsirea unei potriviri maxime au fost studiați pentru
ades, și se poate ajunge foarte aproape de O (n 2) pentru un grafic n-nod. Algoritm online
au fost studiate, de asemenea, ritmurile problemei și este această clasă de algoritmi
vom lua în considerare aici. În special, algoritmul lacom pentru potrivirea maximă
funcționează după cum urmează. Considerăm marginile în orice ordine sunt date.
Când luăm în considerare (x, y), adăugați această margine la potrivire dacă nu sunt nici x, nici y
capetele oricărei margini selectate pentru potrivire până acum. În caz contrar, săriți (x, y).
Exemplul 8.5: Să considerăm o potrivire lacomă pentru graficul din Fig. 8.1. Cina-
poziționăm ordonăm nodurile lexicografic, adică după ordinea nodului lor stâng,
rupând legăturile de nodul drept. Apoi considerăm marginile în ordinea (1, a),
(1, c), (2, b), (3, b), (3, d), (4, a). Prima margine, (1, a), devine cu siguranță o parte din
potrivire. A doua muchie, (1, c), nu poate fi aleasă, deoarece nodul 1 deja
apare în potrivire. A treia margine, (2, b), este selectată, deoarece niciuna dintre ele
nodul 2 și nici nodul b apar în potrivire până acum. Edge (3, b) este respins pentru
meciul deoarece b este deja egalat, dar apoi (3, d) se adaugă la meci
deoarece nici 3 și nici d nu au fost potrivite până acum. În cele din urmă, (4, a) este respins
deoarece apare un meci a. Astfel, asortarea produsă de lacomi
algoritmul pentru această ordonare a muchiilor este {(1, a), (2, b), (3, d)}. După cum am văzut,
această potrivire nu este maximă. ✷
Exemplul 8.6: Un meci lacom poate fi chiar mai rău decât cel din Exemplul 8.5.
În graficul din Fig. 8.1, orice ordonare care începe cu cele două margini (1, a)
și (3, b), în ambele ordine, se vor potrivi cu aceste două perechi, dar apoi nu vor putea
pentru a se potrivi cu nodurile 2 sau 4. Astfel, dimensiunea potrivirii rezultate este de numai 2. ✷

4
1
A
b
c
d
2
3
Figura 8.2: Singura potrivire perfectă pentru graficul din Fig. 8.1

8.3.3 Raportul competitiv pentru potrivirea lacomă

Putem arăta un raport competitiv de 1/2 pentru algoritmul de potrivire lacom de
Secțiunea 8.3.2. În primul rând, raportul nu poate fi mai mare de 1/2. Am văzut deja asta
pentru graficul din Fig. 8.1, există o potrivire perfectă a mărimii 4. Cu toate acestea, dacă
marginile sunt prezentate în oricare dintre comenzile discutate în Exemplul 8.6, dimensiunea
din meci este doar 2, sau jumătate din optim. Întrucât raportul competitiv pentru
un algoritm este minimul peste toate intrările posibile ale raportului dintre ceea ce
algoritmul atinge rezultatul optim, vedem că 1/2 este o limită superioară
pe raportul competitiv.
Să presupunem că M_o este o potrivire maximă, iar M_g este potrivirea cu cea lacomă
algoritmul produce. Fie L setul de noduri din stânga care se potrivesc în M_o dar
nu în M_g . Fie R setul de noduri dreapta care sunt conectate prin margini la oricare

nod în L. Noi susținem că fiecare nod din R este potrivit în M_g. Să presupunem că nu;
în special, să presupunem că nodul r din R nu se potrivește în M_g. Apoi lacomii
algoritmul va lua în considerare în cele din urmă unele margini (ℓ, r), unde ℓ este în L. În acest caz
timp, niciunul dintre capetele acestei margini nu se potrivește, pentru că am presupus că niciuna dintre ele
ℓ și nici r nu se potrivesc vreodată cu algoritmul lacom. Această observație contrazice
definirea modului în care funcționează algoritmul lacom; adică algoritmul lacom
s-ar potrivi cu adevărat (ℓ, r). Concluzionăm că fiecare nod din R este potrivit în
M_g.
Acum, știm mai multe lucruri despre dimensiunile seturilor și ale potrivirilor.
1. $|M_o| \leq |M_g| + |L|$, deoarece printre nodurile din stânga, numai nodurile din L pot
se potrivește în M_o dar nu M_g.
2. $|L| \leq |R|$, deoarece în M_o, toate nodurile din L au fost potrivite.

Pagina 322

3. $|R| \leq |M_g|$, deoarece fiecare nod din R este potrivit în M_g.
Acum, (2) și (3) ne dă $|L| \leq |M_g|$. Asta, împreună cu (1), ne oferă
$|M_o| \leq 2|M_g|$ sau $|M_g| \geq \frac{1}{2}|M_o|$. Ultima inegalitate spune că competitivul
raportul este de cel puțin 1/2. Întrucât am observat deja că raportul competitiv este nr
mai mult de 1/2, ajungem la concluzia că raportul este exact 1/2.

8.3.4 Exerciții pentru secțiunea 8.3

Exercițiul 8.3.1: Definiți graficul G_n pentru a avea 2n noduri
$a_0, a_1, ..., a_{n-1}, b_0, b_1, ..., b_{n-1}$
și următoarele margini. Fiecare nod a_i, pentru i = 0, 1, ..., n - 1, este conectat la
nodurile b_j și b_k, unde
j = 2i mod n și k = (2i + 1) mod n
De exemplu, graficul G_4 are următoarele margini: (a_0, b_0), (a_0, b_1), (a_1, b_2),
(a_1, b_3), (a_2, b_0), (a_2, b_1), (a_3, b_2) și (a_3, b_3).
(a) Găsiți o potrivire perfectă pentru G_4.
(b) Găsiți o potrivire perfectă pentru G_5.
!! (c) Dovediți că pentru fiecare n, G_n are o potrivire perfectă.
! Exercițiul 8.3.2: Câte potriviri perfecte fac graficele G_4 și G_5 ale
Exercițiul 8.3.1 are?
! Exercițiul 8.3.3: Dacă algoritmul lacom ne oferă sau nu o potrivire perfectă -
pentru graficul din Fig. 8.1 depinde de ordinea în care considerăm marginile.
Dintre cele 6! ordinele posibile ale celor șase margini, câte ne oferă o potrivire perfectă-
ing? Dați un test simplu pentru a distinge acele ordine care dau perfectul
potrivire dintre cele care nu.

8.4 Problema AdWords

Acum luăm în considerare problema fundamentală a publicității în căutare, pe care o numim
„problema AdWords”, deoarece a fost întâmpinată pentru prima dată în Google Adwords
sistem. Discutăm apoi despre un algoritm lacom numit „Echilibru” care oferă un bun
raport competitiv. Analizăm acest algoritm pentru un caz simplificat al AdWords
problemă.

Pagina 323

8.4.1 Istoricul publicității în căutare

În jurul anului 2000, o companie numită Overture (cumpărată ulterior de Yahoo!)
a introdus un nou tip de căutare. Agenții de publicitate licitează pentru cuvinte cheie (cuvinte într-un
interogare de căutare), iar atunci când un utilizator a căutat acel cuvânt cheie, linkurile către toate fișierele
agenții de publicitate care licitează pentru acel cuvânt cheie sunt afișați în ordinea cu cea mai mare ofertă-primul.
Dacă s-a făcut clic pe linkul agentului de publicitate, acesta a plătit suma licitată.
Acest tip de căutare a fost foarte util pentru cazul în care interogatorul de căutare
într-adevăr căuta reclame, dar a fost destul de inutil pentru interogator
care doar căuta informații. Reamintim discuția noastră din secțiunea 5.1.1
despre punctul în care, cu excepția cazului în care un motor de căutare poate oferi răspunsuri fiabile
interogări care sunt pentru informații generale, nimeni nu va dori să folosească căutarea
atunci când caută să cumpere ceva.
Câțiva ani mai târziu, Google a adaptat ideea într-un sistem numit Adwords. De
în acel moment, fiabilitatea Google a fost bine stabilită, astfel încât oamenii au fost dispuși
să aibă încredere în anunțurile în care au fost afișate. Google a păstrat lista de răspunsuri pe baza

PageRank și alte criterii obiective separate de lista de anunțuri, deci la fel
sistemul a fost util pentru interogatorul care dorea doar informații, precum și
interogator care caută să cumpere ceva.
Sistemul Adwords a depășit sistemul anterior în mai multe moduri
a făcut selecția anunțurilor mai complexă.
1. Google ar afișa doar un număr limitat de anunțuri la fiecare interogare. Prin urmare,
în timp ce Overture pur și simplu a comandat toate anunțurile pentru un anumit cuvânt cheie, Google a trebuit să o facă
decideți ce anunțuri să afișați, precum și ordinea în care să le afișați.
2. Utilizatorii sistemului Adwords au specificat un buget: suma în care erau
dispuși să plătească pentru toate clicurile pe anunțurile lor într-o lună. Aceste constrângeri
face ca problema atribuirii anunțurilor pentru interogările de căutare să devină mai complexă, ca și noi
indicat în Exemplul 8.1.
3. Google nu a comandat pur și simplu reclame după valoarea ofertei, ci după
suma pe care se așteptau să o primească pentru afișarea fiecărui anunț. Adică clicul
rata de trecere a fost observată pentru fiecare anunț, pe baza istoricului afișărilor de
acel anunț. Valoarea unui anunț a fost considerată a fi produsul ofertei și
rata de clic.

8.4.2 Definiția problemei AdWords

Desigur, decizia cu privire la ce reclame să fie afișate trebuie luată on-line.
Astfel, vom lua în considerare doar algoritmi online pentru rezolvarea AdWords
problemă, care este după cum urmează.

- Dat:

1. Un set de oferte ale agenților de publicitate pentru interogări de căutare.
2. O rată de clic pentru fiecare pereche de interogare-agent de publicitate.

---

3. Un buget pentru fiecare agent de publicitate. Vom presupune că bugetele sunt pentru un
lună, deși ar putea fi utilizată orice unitate de timp.
4. O limită a numărului de anunțuri care trebuie afișate la fiecare interogare de căutare.

- Răspundeți la fiecare interogare de căutare cu un set de agenți de publicitate astfel încât:

1. Dimensiunea setului nu este mai mare decât limita numărului de reclame
pe interogare.
2. Fiecare agent de publicitate a licitat pentru interogarea de căutare.
3. Fiecărui agent de publicitate îi rămâne suficient buget pentru a plăti anunțul, dacă este
făcut clic pe.

Venitul unei selecții de anunțuri reprezintă valoarea totală a anunțurilor selectate, unde
valoarea unui anunț este produsul ofertei și rata de clic pentru
anunț și interogare. Meritul unui algoritm online este venitul total obținut
peste o lună (unitatea de timp peste care se presupune că se aplică bugetele). Noi
va încerca să măsoare raportul competitiv pentru algoritmi, adică minimul
venituri totale pentru algoritmul respectiv, pentru orice succesiune de interogări de căutare, împărțite la
veniturile algoritmului off-line optim pentru aceeași secvență de căutare
întrebări.

8.4.3 Abordarea lacomă a problemei AdWords

Deoarece doar un algoritm on-line este potrivit pentru problema AdWords, ar trebui
examinați mai întâi performanța algoritmului evident lacom. Vom face
o serie de simplificări pentru mediu; scopul nostru este să arătăm
de fapt, există un algoritm mai bun decât algoritmul lacom evident. The
simplificări:
(a) Există un anunț afișat pentru fiecare interogare.
(b) Toți agenții de publicitate au același buget.
(c) Toate ratele de clic sunt aceleași.
(d) Toate ofertele sunt fie 0, fie 1. Alternativ, putem presupune că valoarea lui
fiecare anunț (produsul ofertei și rata de clic) este același.
Algoritmul lacom alege, pentru fiecare interogare de căutare, orice agent de publicitate care are
licitați 1 pentru acea interogare. Raportul competitiv pentru acest algoritm este 1/2, ca și
următorul exemplu arată.
Exemplul 8.7: Să presupunem că există doi agenți de publicitate A și B și doar doi
interogări posibile, x și y. Agentul de publicitate A licită numai pentru x, în timp ce B licită pentru ambele
x și y. Bugetul pentru fiecare agent de publicitate este 2. Observați similaritatea cu
situația din Exemplul 8.1; singurele diferențe sunt faptul că ofertele de fiecare
agentul de publicitate sunt la fel și bugetele sunt mai mici aici.

Aspecte AdWords care nu sunt în modelul nostru

Există mai multe moduri în care sistemul AdWords real diferă de
model simplificat al acestei secțiuni.
Sumele licitate și interogările de căutare: în modelul nostru simplificat,
Licitează pentru seturi de cuvinte, iar oferta unui agent de publicitate este eligibilă pentru a fi afișată
interogări de căutare care au exact același set de cuvinte ca și agentul de publicitate
licitație. În realitate, Google, Yahoo !, și Microsoft oferă tuturor agenților de publicitate o funcție
cunoscut sub numele de potrivire amplă, în cazul în care un anunț este eligibil pentru a fi afișat pentru căutare
interogări care sunt potriviri inexacte ale cuvintelor cheie licitate. Exemplele includ
interogări care includ un subset sau superset de cuvinte cheie, precum și interogări
care utilizează cuvinte cu semnificații foarte asemănătoare cu cuvintele oferite de agentul de publicitate
pe. Pentru astfel de potriviri ample, motoarele de căutare taxează agentul de publicitate pe baza
formule complicate luând în considerare cât de strâns legată de căutare
interogarea se referă la suma licitată a agentului de publicitate. Aceste formule variază în funcție de motoarele de căutare
și nu sunt făcute publice.
Încărcarea agenților de publicitate pentru clicuri: în modelul nostru simplificat, când este un utilizator
face clic pe anunțul unui agent de publicitate, acesta este taxat cu suma pe care o are
licitație. Această politică este cunoscută sub numele de licitație la primul preț.
În realitate, căutați
motoarele folosesc un sistem mai complicat cunoscut sub numele de licitație la al doilea preț,
unde fiecare agent de publicitate plătește aproximativ oferta agentului de publicitate care
plasate imediat în spatele lor în licitație. De exemplu, primul
loc agentul de publicitate pentru o căutare ar putea plăti oferta agentului de publicitate în al doilea rând
loc, plus un cent. S-a demonstrat că licitațiile la al doilea preț sunt mai mici
susceptibile de a fi jucate de agenții de publicitate decât de licitațiile la primul preț și
duce la venituri mai mari pentru motorul de căutare.
Fie succesiunea interogărilor xxyy. Algoritmul lacom este capabil să aloce
primele două x-uri către B, după care nu există nimeni cu un buget necheltuit
plătiți pentru cei doi y. În acest caz, venitul pentru algoritmul lacom este de 2.
Cu toate acestea, algoritmul optim off-line va aloca x-urile la A și y-urile
la B, realizând un venit de 4. Raportul competitiv pentru algoritmul lacom
nu este deci mai mult de 1/2. Putem argumenta că, în orice secvență de interogări,
raportul dintre veniturile pentru algoritmii lacomi și optimi este de cel puțin 1/2, folosind
în esență, aceeași idee ca în secțiunea 8.3.3. ✷

8.4.4 Algoritmul de echilibru

Există o simplă îmbunătățire a algoritmului lacom care oferă o competitivitate
raport de 3/4 pentru cazul simplu al secțiunii 8.4.3. Acest algoritm, numit
Algoritmul de echilibru, atribuie o interogare agentului de publicitate care licită pentru interogare și
are cel mai mare buget rămas. Legăturile pot fi rupte în mod arbitrar.

Exemplul 8.8: Luați în considerare aceeași situație ca în Exemplul 8.7. Balanta
Algoritmul poate atribui prima interogare x fie A, fie B, deoarece ambele licitează
pe x și bugetele lor rămase sunt aceleași. Cu toate acestea, al doilea x trebuie să fie
atribuit celuilalt dintre A și B, deoarece acestea au apoi cel mai mare rămas
buget. Primul y este atribuit lui B, deoarece are buget rămas și este
singur ofertant pe y. Ultimul y nu poate fi atribuit, deoarece B este în afara bugetului și
A nu a licitat. Astfel, veniturile totale pentru algoritmul de echilibru pe aceste date
este 3. În comparație, venitul total pentru algoritmul off-line optim este
4, deoarece poate atribui x-urile lui A și y-urile lui B. Concluzia noastră este că,
pentru problema AdWords simplificată din secțiunea 8.4.3, raportul competitiv al
Algoritmul de echilibru nu depășește 3/4. Vom vedea asta doar cu
doi agenți de publicitate, 3/4 este exact raportul competitiv, deși este numărul
de agenți de publicitate crește, raportul competitiv scade la 0,63 (de fapt 1 - 1 / e), dar
nu mai jos. ✷

8.4.5 O limită inferioară a raportului competitiv pentru echilibru

În această secțiune vom demonstra că, în cazul simplu al algoritmului de echilibru
pe care o luăm în considerare, raportul competitiv este de 3/4. Având în vedere exemplul 8.8, noi
trebuie doar să demonstreze că venitul total obținut de algoritmul de echilibru este
cel puțin 3/4 din venituri pentru algoritmul off-line optim. Astfel, ia în considerare un
situație în care există doi agenți de publicitate, A_1 și A_2, fiecare cu un buget de
B. Vom presupune că fiecare interogare este atribuită unui agent de publicitate de către optim

algoritm. Dacă nu, putem șterge acele interogări fără a afecta veniturile din
algoritmul optim și posibil reducerea veniturilor soldului. Astfel,
cel mai mic raport competitiv posibil este atins atunci când secvența de interogare constă
numai din reclame alocate de algoritmul optim.
De asemenea, vom presupune că bugetele ambilor agenți de publicitate sunt consumate de
algoritm optim. Dacă nu, putem reduce bugetele și vom argumenta din nou că
venitul algoritmului optim nu este redus, în timp ce cel al Balance poate
doar micșorează. Această modificare ne poate obliga să folosim bugete diferite pentru cei doi
agenții de publicitate, dar vom continua să presupunem că bugetele sunt ambele B. Plecăm
ca exercițiu extinderea probei la cazul în care bugetele din
doi agenți de publicitate sunt diferiți.
Figura 8.3 sugerează modul în care interogările 2B sunt atribuite agenților de publicitate de către
doi algoritmi. În (a) vedem că interogările B sunt atribuite fiecăruia dintre A_1 și A_2
prin algoritmul optim. Acum, luați în considerare modul în care sunt atribuite aceleași interogări
de Balance. În primul rând, observați că Soldul trebuie să epuizeze bugetul a cel puțin unul
dintre agenții de publicitate, să zicem A_2 . Dacă nu, atunci ar exista o anumită interogare atribuită
niciun agent de publicitate, chiar dacă ambii aveau buget. Cunoaștem cel puțin unul dintre
agenții de publicitate licită la fiecare interogare, deoarece acea interogare este atribuită în mod optim
algoritm. Această situație contrazice modul în care este definit să funcționeze echilibrul; aceasta
atribuie întotdeauna o interogare dacă poate.
Astfel, vedem în Fig. 8.3 (b) că lui A_2 i se atribuie B interogări. Aceste întrebări
ar fi putut fi atribuit fie A_1, fie A_2 de către algoritmul optim. Noi

---

Pagina 327

8.4. PROBLEMA ADWORDS
307
A_1
A_2
A_1
A_2
B
B
X
X
y
Nu
folosit
(a) Optim
(b) Sold
Figura 8.3: Ilustrația atribuirilor de interogări către agenții de publicitate din opțiune
algoritmi mama și Balance
vezi și în Fig. 8.3 (b) că folosim y ca număr de interogări atribuite lui A_1 și
x ca B - y. Scopul nostru este să arătăm $y \geq x$. Această inegalitate va arăta veniturile
de echilibru este cel puțin 3B / 2 sau 3 / 4th veniturile algoritmului optim.
Observăm că x este, de asemenea, numărul de interogări neatribuite pentru Sold
Algoritm și că toate interogările neatribuite trebuie să fi fost atribuite lui A_2
prin algoritmul optim. Motivul este că A_1 nu rămâne niciodată fără buget,
deci orice interogare atribuită de algoritmul optim lui A_1 este cu siguranță licitată de către A_1 .
Deoarece A_1 are întotdeauna buget în timpul rulării algoritmului de echilibru, asta
algoritmul va atribui cu siguranță această interogare, fie A_1, fie A_2 .
Există două cazuri, în funcție de mai multe interogări
atribuite lui A_1 de algoritmul optim sunt atribuite lui A_1 sau A_2 de către Balanță.
1. Să presupunem că cel puțin jumătate dintre aceste interogări sunt atribuite de Sold către A_1 . Apoi
$y \geq B / 2$, deci sigur $y \geq x$.
2. Să presupunem că mai mult de jumătate din aceste interogări sunt atribuite de Sold către A_2 .
Luați în considerare ultima dintre aceste interogări q care este atribuită lui A_2 de către Sold
Algoritm. În acel moment, A_2 trebuie să fi avut cel puțin un buget la fel de mare

---

Pagina 328

308
CAPITOLUL 8. PUBLICITATE PE WEB
disponibil ca A_1 sau altfel Balance ar fi atribuit interogarea q către A_1 , doar
așa cum a făcut algoritmul optim. Din moment ce mai mult de jumătate din B interogează acest lucru
algoritmul optim atribuie lui A_1 este atribuit lui A_2 de către Balance, noi
să știți că chiar înainte de a fi atribuit q, bugetul rămas al lui A_2 era la
majoritatea B / 2. Prin urmare, la acel moment, bugetul rămas al A_1 era, de asemenea
cel mult B / 2. Deoarece bugetele scad doar, știm că $x \leq B / 2$. Aceasta

urmează că $y \geq x$, deoarece $x + y = B$.
Concluzionăm că $y \geq x$ în ambele cazuri, deci raportul competitiv al soldului
Algoritmul este 3/4.

8.4.6 Algoritmul echilibrului cu mulți ofertanți

Atunci când sunt mulți agenți de publicitate, raportul competitiv pentru Balance Algoritmul poate fi sub 3/4, dar nu prea sub această fracțiune. Cel mai rău caz
pentru Balance este după cum urmează.

1. Există N agenți de publicitate, A_1, A_2, ..., A_N.
2. Fiecare agent de publicitate are un buget B = N !.
3. Există N interogări q_1, q_2, ..., q_N.
4. Agentul de publicitate A_i licită la interogările q_1, q_2, ..., q_i și la alte interogări.
5. Secvența de interogare constă din N runde. A doua rundă este formată din B
apariții ale interogării q_i și nimic altceva.

Algoritmul off-line optim atribuie interogările B q_i din runda ith la
A_i pentru toti i. Astfel, toate interogările sunt atribuite unui ofertant și veniturile totale ale
algoritmul optim este NB.
Cu toate acestea, algoritmul de echilibru atribuie fiecare dintre interogările din runda 1 către
N agenții de publicitate în mod egal, deoarece toți licitează pentru q_1 și pentru algoritmul de echilibru
preferă ofertantul cu bugetul rămas cel mai mare. Astfel, fiecare agent de publicitate
obține B / N al interogărilor q_1. Acum luați în considerare interogările q_2 din runda 2. Toate, cu excepția A_1
licitați pentru aceste interogări, astfel încât acestea să fie împărțite în mod egal între A_2 până la A_N, cu
fiecare dintre acești N-1 ofertanți primind interogări B / (N - 1). Modelul, sugerat
după Fig. 8.4, se repetă pentru fiecare rundă i = 3, 4, ..., cu A_i prin A_N obținând
Interogări B / (N - i + 1).
Cu toate acestea, în cele din urmă, bugetele agenților de publicitate cu număr mai mare vor fi
epuizat. Asta se va întâmpla la cea mai joasă rundă j astfel încât

$$B\left(\frac{1}{N} + \frac{1}{N-1} + \cdots + \frac{1}{N-j+1}\right) \geq B$$

acesta este,

$$\frac{1}{N} + \frac{1}{N-1} + \cdots + \frac{1}{N-j+1} \geq 1$$

A_1
A_2
A_3
A_{n-1}
A_n
B/n
$B/(n-1)$
$B/(n-2)$
...

Figura 8.4: Distribuirea interogărilor către N agenți de publicitate în cel mai rău caz

Euler a arătat că pe măsură ce k devine mare, $\sum_{i=1}^{k} 1/i$ se apropie de $\log_e k$. Folosind aceasta
observație, putem aproxima suma de mai sus ca $\log_e N - \log_e (N - j)$.
Căutăm astfel j astfel încât $\log_e N - \log_e (N - j) = 1$, aproximativ
mately. Dacă înlocuim $\log_e N - \log_e (N - j)$ cu logul echivalent $_e (N / (N - j))$
și exponențiați ambele părți ale ecuației $\log_e (N / (N - j)) = 1$, obținem
$N / (N - j) = e$. Rezolvând această ecuație pentru j, obținem

$j = N (1 - \frac{1}{e})$
ca valoare aproximativă a lui j pentru care toți agenții de publicitate sunt fie în afara bugetului sau nu licitați pentru oricare dintre interogările rămase. Astfel, venitul aproximativ obținut prin algoritmul de echilibru este $BN (1 - \frac{1}{e})$, adică întrebările din primele j runde. Prin urmare, raportul competitiv este de $1 - \frac{1}{e}$, sau aproximativ 0,63.

8.4.7 Algoritmul de echilibru generalizat

Algoritmul de echilibru funcționează bine atunci când toate sumele licitate sunt 0 sau 1. Cu toate acestea, în practică, sumele licitate pot fi arbitrare, iar cu ofertele și bugetele arbitrare echilibrul nu reușește cântăriți corect dimensiunile ofertelor. Următorul exemplu ilustrează punct.

Exemplul 8.9: Să presupunem că există doi agenți de publicitate A_1 și A_2 și o interogare q. Ofertele pentru q și bugete sunt:

Ofertant
Buget de licitare
A_1
1
110
A_2
10
100

Dacă există 10 apariții de q, algoritmul off-line optim va fi atribuit pe toate la A_2 și câștigă venituri 100. Cu toate acestea, deoarece bugetul lui A_1 este mai mare, Soldul va atribui toate cele zece interogări către A_1 pentru un venit de 10. De fapt, se poate extindeți ușor această idee pentru a arăta că pentru situații ca aceasta, nu există raport competitiv mai mare de 0, care este valabil pentru algoritmul de echilibru. ✷

Pagina 330

Pentru a face echilibrul să funcționeze în situații mai generale, trebuie să facem două modificări. În primul rând, trebuie să influențăm alegerea anunțului în favoarea unor sume licitate mai mari. În al doilea rând, trebuie să fim mai puțin absoluți cu privire la bugetul rămas. Mai degrabă, noi ia în considerare fracția din bugetele rămase, așa că suntem părtinitori spre utilizare o parte din bugetul fiecărui agent de publicitate. Ultima schimbare va face echilibrul Algoritm mai „avers de risc”; nu va lăsa prea mult din niciun agent de publicitate buget nefolosit. Se poate arăta (vezi referințele capitolului) că următoarele generalizarea algoritmului de echilibru are un raport competitiv de $1 - 1 / e =$ 0,63.

- Să presupunem că sosește o interogare q, agentul de publicitate A_i a licitat x_i pentru această interogare (rețineți că x_i ar putea fi 0). De asemenea, să presupunem că acea fracțiune f_i din buget din A_i nu este cheltuită în prezent. Fie $\Psi_i = x_i (1 - e^{-f_i})$. Apoi atribuiți q la agent de publicitate A_i astfel încât Ψ_i este un maxim. Rupeți legăturile în mod arbitrar.

Exemplul 8.10: Luați în considerare modul în care ar funcționa algoritmul de echilibru generalizat pe datele din Exemplul 8.9. Pentru prima apariție a interogării q,
$\Psi_1 = 1 \times (1 - e^{-1})$
deoarece A_1 a licitat 1 și fracția 1 din bugetul lui A_1 rămâne. Acesta este,
$\Psi_1 = 1 - 1 / e = 0,63$
Pe de altă parte, $\Psi_2 = 10 \times (1 - e^{-1}) = 6.3$. Astfel, primul q este acordat lui A_2.
Același lucru se întâmplă pentru fiecare dintre q-uri. Adică, Ψ_1 rămâne la 0,63, în timp ce Ψ_2 scade. Cu toate acestea, nu coboară niciodată sub 0,63. Chiar și pentru a 10-a q, când 90% din bugetul lui A_2 a fost deja utilizat, $\Psi_2 = 10 \times (1 - e^{-1/10})$. Reamintim (Secțiunea 1.3.5) extinderea Taylor $e^x = 1 + x$ cu $^2/2! + x$ cu $^3/$ cu $3! + \cdots$.Prin urmare,
$e^{-1/10} = 1 - \frac{1}{10} + \frac{1}{200} - \frac{1}{6000} \cdots$
sau aproximativ, $e^{-1-10} = 0,905$. Astfel, $\Psi_2 = 10 \times 0,095 = 0,95$. ✷

Lăsăm nedovedită afirmația că raportul competitiv pentru acest algoritm
este 1 - 1 / e. De asemenea, lăsăm nedovedit un fapt surprinzător suplimentar: fără on-line
algoritmul pentru problema AdWords descris în această secțiune poate avea un
raport competitiv peste 1 - 1 / e.

8.4.8 Observații finale despre problema AdWords

Algoritmul de echilibru, așa cum este descris, nu ia în considerare posibilitatea
că rata de clic diferă pentru diferite anunțuri. Este simplu să se înmulțească
suma licitată în funcție de rata de clic atunci când calculați Ψ_i și, dacă faceți acest lucru, o veți face
maximizați veniturile așteptate. Putem chiar să încorporăm informații despre

---

rata de clic pentru fiecare anunț din fiecare interogare pentru care o valoare diferită de zero
a fost licitat. Când ne confruntăm cu problema alocării unei anumite interogări q,
încorporăm un factor care este rata de clic pentru anunțul respectiv la interogarea q,
atunci când calculăm fiecare dintre Ψ-uri.
O altă problemă pe care trebuie să o luăm în considerare în practică este frecvența istorică a
întrebări. Dacă, de exemplu, știm că agentul de publicitate A_i are un buget suficient
mici, pentru că sunt sigur că vor fi suficiente întrebări care vor veni mai târziu în lună
satisfacerea cererii lui A_i, atunci nu are rost să crești Ψ_i dacă o parte din bugetul lui A_i
a fost deja cheltuit. Adică, menținem $\Psi_i = x_i (1 - e^{-1})$ atâta timp cât noi
se poate aștepta ca în lună să rămână suficiente întrebări pentru a da A_i
bugetul său complet de reclame. Această modificare poate face ca soldul să se comporte mai rău dacă
secvența de interogări este guvernată de un adversar care poate controla secvența
de interogări. Un astfel de adversar poate provoca interogările A_i licitate brusc la
dispărea. Cu toate acestea, motoarele de căutare primesc atât de multe întrebări și generația lor
este atât de aleatoriu, încât nu este necesar în practică să ne imaginăm abateri semnificative
din normă.

8.4.9 Exerciții pentru secțiunea 8.4

Exercițiul 8.4.1: Să presupunem că folosind ipotezele simplificatoare din exemplul 8.7
că există trei agenți de publicitate, A, B și C. Există trei interogări, x, y și
z. Fiecare agent de publicitate are un buget de 2. Agentul de publicitate A licită numai pentru x; B licită pe x
și y, în timp ce C licitează pentru x, y și z. Rețineți că în secvența de interogare xxyyzz,
algoritmul optim off-line ar genera un venit de 6, deoarece toate interogările pot
să fie atribuit.
! (a) Arătați că algoritmul lacom va atribui cel puțin 4 dintre aceste 6 interogări.
!! (b) Găsiți o altă secvență de interogări, astfel încât algoritmul lacom să poată as-
semnați cât mai puțin de jumătate din interogările pe care algoritmul off-line optim le atribuie
pe acea secvență.
!! Exercițiul 8.4.2: Extindeți dovada secțiunii 8.4.5 la cazul în care cele două
agenții de publicitate au bugete inegale.
! Exercițiul 8.4.3: Arătați cum să modificați Exemplul 8.9 schimbând ofertele și / sau
bugetele pentru a face ca raportul competitiv să iasă cât de aproape de 0 doriți.

8.5 Implementarea AdWords

Deși acum ar trebui să avem o idee despre cum sunt selectate anunțurile pentru a merge cu răspunsul
la o interogare de căutare, nu am abordat problema găsirii ofertelor care
au fost făcute la o anumită interogare. Atâta timp cât ofertele sunt pentru setul exact de
cuvinte într-o interogare, soluția este relativ ușoară. Cu toate acestea, există o serie de
extensii la procesul de potrivire a interogării / licitării care nu sunt la fel de simple. Vom
explicați detaliile din această secțiune.

8.5.1 Sumele potrivite și interogările de căutare

După cum am descris problema AdWords și așa cum apare în mod normal
practică, agenții de publicitate licită pe seturi de cuvinte. Dacă apare o interogare de căutare având
exact acel set de cuvinte într-o anumită ordine, atunci se spune că oferta se potrivește cu
interogare și devine un candidat pentru selecție. Putem evita să ne ocupăm
cu ordinea cuvintelor prin stocarea tuturor seturilor de cuvinte reprezentând o ofertă în lexicografic
ordinea (alfabetică). Lista cuvintelor în ordine sortată formează cheia hash pentru
licitate, iar aceste sume licitate pot fi stocate într-un tabel hash utilizat ca index, după cum sa discutat
în secțiunea 1.3.2.
De asemenea, interogările de căutare au cuvintele lor sortate înainte de căutare. Când hash

lista sortată, găsim în tabelul hash toate ofertele pentru exact acel set de cuvinte.
Acestea pot fi recuperate rapid, deoarece nu trebuie decât să ne uităm la conținutul acestuia
găleată.
Mai mult, există șanse mari să putem păstra întregul tabel hash
memoria principala. Dacă există un milion de agenți de publicitate, fiecare licitează pentru 100 de interogări,
iar înregistrarea ofertei necesită 100 de octeți, apoi avem nevoie de zece gigaocteți de
memoria principală, care se încadrează în limitele a ceea ce este fezabil pentru un singur
mașinărie. Dacă este necesar mai mult spațiu, putem împărți gălețile mesei hash
printre câte mașini avem nevoie. Interogările de căutare pot fi hash și trimise
la aparatul adecvat.
În practică, interogările de căutare pot ajunge mult prea rapid pentru o singură ma-
chine, sau grup de mașini care colaborează la o singură interogare la un moment dat, la
gestionează-le pe toate. În acest caz, fluxul de interogări este împărțit în cât mai multe bucăți
după cum este necesar, și fiecare piesă este manipulată de un grup de mașini. De fapt,
modificarea interogării de căutare, independent de reclame, va necesita un grup de mașini
lucrând în paralel oricum, pentru ca întreaga procesare a unei interogări să poată
să se facă în memoria principală.

8.5.2 Probleme de potrivire mai complexe

Cu toate acestea, potențialul de potrivire a ofertelor la obiecte nu este limitat la caz
unde obiectele sunt interogări de căutare și criteriul de potrivire este același-set-of-
cuvinte. De exemplu, Google corelează și ofertele AdWords cu e-mailurile. Acolo,
criteriul de potrivire nu se bazează pe egalitatea seturilor. Mai degrabă, o ofertă pentru un set de
cuvintele S corespund unui e-mail dacă toate cuvintele din S apar oriunde în e-mail.
Această problemă de potrivire este mult mai grea. Putem menține în continuare un tabel hash
index pentru sume licitate, dar numărul de subseturi de cuvinte dintr-un e-mail cu o sută de cuvinte
este mult prea mare pentru a căuta toate seturile sau chiar toate seturile mici de (să zicem)
trei sau mai puține cuvinte. Există o serie de alte aplicații potențiale ale
acest tip de potrivire care, în momentul redactării acestui articol, nu este implementat
dar ar putea fi. Toate implică interogări permanente - interogări pe care utilizatorii le postează într-un
site-ul, așteptând ca site-ul să îi notifice ori de câte ori se potrivește ceva cu interogarea
devine disponibil pe site. De exemplu:
1. Twitter vă permite să urmăriți toate „tweet-urile” unei anumite persoane. In orice caz,

---

este fezabil să permiteți utilizatorilor să specifice un set de cuvinte, cum ar fi
muzică gratuită pentru iPod
și vedeți toate tweet-urile în care apar toate aceste cuvinte, nu neapărat în
ordine, și nu neapărat adiacente.
2. Site-urile de știri online permit adesea utilizatorilor să selecteze dintre anumite chei-
cuvinte sau fraze, de exemplu, „asistență medicală” sau „Barack Obama” și primiți
alerte ori de câte ori un nou articol de știri conține acel cuvânt sau consecutiv
succesiune de cuvinte. Această problemă este mai simplă decât problema e-mail / AdWords
din mai multe motive. Potrivirea unor cuvinte simple sau secvențe consecutive
de cuvinte, chiar și într-un articol lung, nu consumă timp ca potrivirea
mici seturi de cuvinte. Mai mult, seturile de termeni pe care îi puteți căuta
este limitată, deci nu există prea multe „sume licitate”. Chiar dacă mulți oameni doresc
alerte despre același termen, o singură intrare index, cu lista tuturor celor
persoane asociate, este obligatoriu. Cu toate acestea, un sistem mai avansat ar putea
permite utilizatorilor să specifice alerte pentru seturi de cuvinte într-un articol de știri, la fel ca
Sistemul AdWords permite oricui să liciteze pentru un set de cuvinte dintr-un e-mail.

8.5.3 Un algoritm de potrivire pentru documente și oferte

Vom oferi un algoritm care se va potrivi cu multe „sume licitate” cu multe „documente”.
mențiuni. " Ca și înainte, o ofertă este un set (de obicei mic) de cuvinte. Un document este un
un set mai mare de cuvinte, cum ar fi un e-mail, un tweet sau un articol de știri. Presupunem acolo
sute de documente pe secundă care ajung, deși dacă există
multe, fluxul de documente poate fi împărțit între mai multe mașini sau grupuri de
mașini. Presupunem că există multe oferte, poate de ordinul a o sută
milioane sau un miliard. Ca întotdeauna, vrem să facem la fel de mult în memoria principală ca și noi
poate sa.
Vom reprezenta, ca și până acum, o ofertă prin cuvintele sale enumerate într-o anumită ordine. Acolo
sunt două elemente noi în reprezentare. În primul rând, vom include un statut cu
fiecare listă de cuvinte. Starea este un număr întreg care indică câte dintre primele
cuvintele de pe listă au fost potrivite cu documentul curent. Când este o ofertă
stocat în index, starea acestuia este întotdeauna 0.
În al doilea rând, în timp ce ordinea cuvintelor ar putea fi lexicografică, putem reduce
cantitate de muncă prin ordonarea cuvintelor cel mai rar. Cu toate acestea, din moment ce numărul

de cuvinte diferite care pot apărea în e-mailuri este în esență nelimitat, nu este
fezabil să ordonați toate cuvintele în acest mod. Ca un compromis, am putea identifica
cele mai frecvente cuvinte de pe web sau dintr-un eșantion al fluxului de documente
procesăm. Aici, n ar putea fi o sută de mii sau un milion. Aceste
n cuvintele sunt sortate după frecvență și ocupă sfârșitul listei, cu
cuvintele cele mai frecvente chiar la final. Toate cuvintele nu se numără printre cele mai frecvente
poate fi presupus la fel de rar și ordonat lexicografic. Apoi,
cuvintele din orice document pot fi comandate. Dacă un cuvânt nu apare pe listă
de n cuvinte frecvente, plasează-l în fața ordinii, lexicografic. Acestea

---

Pagina 334

apar cuvintele din document care apar pe lista celor mai frecvente cuvinte
după cuvintele rare, în ordinea inversă a frecvenței (adică cu cele mai multe
cuvinte frecvente ale documentelor comandate ultima).
Exemplul 8.11: Să presupunem că documentul nostru este
- A fost strălucit, iar tocul tăios

,,The" este cel mai frecvent cuvânt în engleză, iar „și" este doar puțin mai puțin
frecvent. Să presupunem că „twas" face lista cuvintelor frecvente, deși
frecvența sa este cu siguranță mai mică decât cea a „celor" sau „și". Celelalte cuvinte da
nu face lista de cuvinte frecvente.
Apoi sfârșitul listei constă din „două", „și" și „cel", în această ordine,
întrucât aceasta este ordinea inversă a frecvenței. Celelalte trei cuvinte sunt plasate
în fruntea listei în ordine lexicografică. Prin urmare,
brillig slithy toves twas and the
este secvența de cuvinte din document, ordonată corespunzător. ✷
Sumele licitate sunt stocate într-un tabel hash, a cărui cheie hash este primul cuvânt al
oferta, în ordinea explicată mai sus. Înregistrarea ofertei va include, de asemenea
informații despre ce trebuie făcut atunci când oferta este potrivită. Starea este 0 și
nu trebuie să fie stocate în mod explicit. Există o altă masă de hash, a cărei sarcină este să
să conțină copii ale acelor oferte care au fost parțial potrivite. Aceste oferte au
o stare care este cel puțin 1, dar mai mică decât numărul de cuvinte din set. Dacă
starea este i, apoi tasta hash pentru acest tabel hash este (i + 1) st cuvânt. The
aranjarea tabelelor de hash este sugerată de Fig. 8.5. Pentru a procesa un document,
urmează următoarele instrucțiuni.
1. Sortați cuvintele documentului în ordinea discutată mai sus. Înlătura
cuvinte duplicat.
2. Pentru fiecare cuvânt w, în ordinea sortată:
(i) Folosind w ca cheie hash pentru tabelul ofertelor parțial potrivite, găsiți
acele oferte având ca cheie w.
(ii) Pentru fiecare astfel de ofertă b, dacă w este ultimul cuvânt al lui b, mutați b la tabelul lui
sume licitate potrivite.
(iii) Dacă w nu este ultimul cuvânt al lui b, adăugați 1 la starea b și refaceți b folosind
cuvântul a cărui poziție este una mai mult decât noul statut, ca
tasta Diez.
(iv) Folosind w ca cheie hash pentru tabelul tuturor ofertelor, găsiți acele oferte pentru
care w este primul lor cuvânt în ordinea sortată.
(v) Pentru fiecare astfel de ofertă b, dacă există un singur cuvânt pe lista sa, copiați-l în
tabelul ofertelor potrivite.

Pagina 335

Hash − table
Index de
ofertele
ofertele
asortat
Parțial
ofertele
asortat
Lista de
Cuvântul *w*
cheie hash
(a) Uită-te în sus
w

(b) Treceți la
ieșire dacă
este ultimul cuvânt
w
(c) Reheat
cu una mai mare
w
w
stare dacă
nu este ultimul
hash − key *w*
licitează cu
(d) Uită-te în sus
licitează cu
este singurul cuvânt
ieșire dacă *w*
(e) Copiați în
Hash pe al doilea cuvânt
decât un cuvânt
starea 1 dacă mai mult
(f) Copiați cu
Figura 8.5: Gestionarea unui număr mare de oferte și a unui număr mare de documente
(vi) Dacă b constă din mai multe cuvinte, adăugați-l, cu starea 1, la
tabel cu sume licitate parțial potrivite, folosind al doilea cuvânt al lui b ca
tasta Diez.
3. Produceți lista ofertelor potrivite ca rezultat.
Avantajul celei mai rare-prima comenzi ar trebui să fie acum vizibil. O ofertă este numai copiat în al doilea tabel hash dacă cel mai rar cuvânt al acestuia apare în document. În comparație, dacă s-ar folosi ordinea lexicografică, s-ar copia mai multe oferte în a doua masă hash. Minimizând dimensiunea acelui tabel, nu numai că reducem cantitatea de muncă în pașii 2 (i) –2 (iii), dar facem mai probabil ca acest lucru să fie întreg tabelul poate fi păstrat în memoria principală.

8.6 Rezumatul capitolului 8

♦ Publicitate direcționată: marele avantaj pe care îl are publicitatea bazată pe web publicitatea în mass-media convențională, cum ar fi ziarele, este Web publicitatea poate fi selectată în funcție de interesele fiecărei persoane utilizator. Acest avantaj a permis sprijinirea multor servicii Web în totalitate prin venituri din publicitate.

♦ Algoritmi on-line și off-line: algoritmi convenționali care sunt permiși pentru a vedea toate datele lor înainte de a produce un răspuns sunt numiți off-line. Un on-

---

algoritmul de linie este necesar pentru a răspunde la fiecare element dintr-un flux imediat, cu cunoașterea doar a trecutului, nu a elementelor viitoare din curentul.

♦ Algoritmi lacomi: Mulți algoritmi on-line sunt lacomi, în sensul că își selectează acțiunea la fiecare pas prin minimizarea unor funcții obiective ziune.

♦ Raport competitiv: Putem măsura calitatea unui algoritm on-line prin minimizarea, peste toate intrările posibile, a valorii rezultatului algoritm de linie comparat cu valoarea rezultatului celui mai bun posibil algoritm off-line.

♦ Potrivirea bipartită: această problemă implică două seturi de noduri și un set de muchii între membrii celor două seturi. Scopul este de a găsi un maxim potrivire - un set cât mai mare de margini posibil, care să nu includă niciun nod mai mult decât o dată.

♦ Soluție on-line la problema de potrivire: Un algoritm lacom pentru găsirea unei potriviri într-un grafic bipartit (sau orice alt grafic, de altfel) este pentru a ordona marginile într-un fel și pentru fiecare margine la rândul său, adăugați-le la se potrivește dacă niciunul dintre capetele sale nu face încă parte dintr-o margine selectată anterior meci. Se poate dovedi că acest algoritm are un raport competitiv de 1/2; adică nu reușește niciodată să se potrivească cu cel puțin jumătate din numărul de noduri ca cel mai bun meciuri de algoritmi off-line.

♦ Gestionarea anunțurilor în căutare: un motor de căutare primește sume licitate de la agenții de publicitate pentru anumite interogări de căutare. Unele reclame sunt afișate cu fiecare interogare de căutare,

iar motorului de căutare i se plătește suma licitată numai dacă interogatorul
face clic pe anunț. Fiecare agent de publicitate poate oferi un buget, suma totală
sunt dispuși să plătească pentru clicuri într-o lună.

♦ Problema AdWords: datele pentru problema AdWords sunt un set de sume licitate
de către agenții de publicitate cu privire la anumite interogări de căutare, împreună cu un buget total pentru
fiecare agent de publicitate și informații despre rata de clic istorică pentru
fiecare anunț pentru fiecare interogare. O altă parte a datelor este fluxul de căutare
interogări primite de motorul de căutare. Obiectivul este de a selecta on-line
un set de anunțuri de dimensiuni fixe ca răspuns la fiecare interogare care va maximiza
venituri pentru motorul de căutare.

♦ Problemă simplificată AdWords: pentru a vedea unele dintre nuanțele selectării anunțurilor,
am considerat o versiune simplificată în care toate sumele licitate sunt fie 0, fie 1,
este afișat un singur anunț la fiecare interogare și toți agenții de publicitate au același lucru
buget. Conform acestui model, algoritmul lacom evident al difuzării reclamei
destinație de plasare pentru oricine a licitat pentru interogare și îi rămâne buget
se poate demonstra că are un raport competitiv de 1/2.

Pagina 337

♦ Algoritmul de echilibru: Acest algoritm îmbunătățește simplul lacom
algoritm. Un anunț de interogare este dat agentului de publicitate care a licitat pentru interogare
și are cel mai mare buget rămas. Legăturile pot fi rupte în mod arbitrar.

♦ Raportul competitiv al algoritmului de echilibru: pentru AdWords simplificat
model, raportul competitiv al algoritmului de echilibru este de 3/4 pentru caz
din doi agenți de publicitate și $1-1 / e$, sau aproximativ 63% pentru orice număr de agenți de publicitate.

♦ Algoritmul de echilibru pentru problema generalizată AdWords: Când licitați
Ders pot face oferte diferite, au bugete diferite și au diferite
ratele de clic pentru diferite interogări, algoritmul de echilibru acordă un
anunț către agentul de publicitate cu cea mai mare valoare a funcției $\Psi = x (1 - e^{-f})$.
Aici, x este produsul ofertei și rata de clic pentru anunțul respectiv
vertiser și interogare, iar f este fracția din bugetul agentului de publicitate care
rămâne necheltuită.

♦ Implementarea unui algoritm AdWords: cea mai simplă versiune a implementării
menționarea servește în situații în care ofertele sunt exact pe setul de
cuvinte din interogarea de căutare. Putem reprezenta o interogare prin lista acesteia
cuvinte, în ordine sortată. Ofertele sunt stocate într-un tabel hash sau în structuri similare
tura, cu o cheie hash egală cu lista sortată de cuvinte. O interogare de căutare poate
apoi să fie asortate cu ofertele printr-o căutare simplă în tabel.

♦ Potrivirea seturilor de cuvinte împotriva documentelor: o versiune mai dificilă a anunțului
problema de implementare a cuvintelor permite sume licitate, care sunt încă seturi mici de
cuvinte ca într-o interogare de căutare, care trebuie asortate cu documente mai mari, de exemplu
ca e-mailuri sau tweets. Un set de sume licitate se potrivește cu documentul dacă toate cuvintele
apar în document, în orice ordine și nu neapărat adiacente.

♦ Stocare Hash a seturilor de cuvinte: O structură de date utilă stochează cuvintele
fiecare ofertă stabilită în ordinea cea mai rară-prima. Documentele au cuvintele lor sortate
în aceeași ordine. Seturile de cuvinte sunt stocate într-un tabel hash cu primul
cuvânt, în ordinea cea mai rară-primă, ca cheie.

♦ Prelucrarea documentelor pentru meciurile de licitație: Procesăm cuvintele
document rarest-first. Seturi de cuvinte al căror prim cuvânt este cuvântul curent
sunt copiate într-un tabel hash temporar, cu al doilea cuvânt ca cheie.
Seturile deja în tabelul de hash temporar sunt examinate pentru a vedea dacă cuvântul
aceasta este cheia lor care se potrivește cu cuvântul curent și, dacă da, sunt refăcute
folosind următorul lor cuvânt. Seturile al căror ultim cuvânt se potrivește sunt copiate în
ieșire.

8.7 Referințe pentru capitolul 8

[1] este o investigație a modului în care poziția anunțului influențează rata de clic.
Algoritmul de echilibru a fost dezvoltat în [2] și aplicarea sa la publicitate
problemele cuvintelor provin din [3].

Pagina 338

1. N. Craswell, O. Zoeter, M. Taylor și W. Ramsey, „Un experiment
compararea modelelor de polarizare a poziției de clic, ”Proc. Intl. Conf. pe Căutare web
și Web Data Mining pp. 87-94, 2008.
2. B. Kalyanasundaram și KR Pruhs, „Un algoritm determinist optim
ritm pentru potrivirea b, ”Theoretical Computer Science 233: 1-2, pp. 319–
325, 2000.
3. A Mehta, A. Saberi, U. Vazirani și V. Vazirani, „Adwords și general-
potrivire on-line, ”IEEE Symp. pe fundamentele informaticii,
pp. 264-273, 2005.

Capitolul 9
Sisteme de recomandare

Există o clasă extinsă de aplicații web care implică prezicerea utilizatorului
răspunsuri la opțiuni. O astfel de facilitate se numește sistem de recomandare. Noi
va începe acest capitol cu o analiză a celor mai importante exemple ale acestora
sisteme. Cu toate acestea, pentru a pune problema în atenție, două exemple bune de
sistemele de recomandare sunt:
1. Oferirea de articole de știri cititorilor de ziare online, pe baza unei previziuni
a intereselor cititorului.
2. Oferind clienților unui retailer online sugestii despre ceea ce ei
ar putea dori să cumpere, pe baza istoricului lor trecut de achiziții și / sau produse
căutări.
Sistemele de recomandare utilizează o serie de tehnologii diferite. Putem
clasifica aceste sisteme în două mari grupuri.

- Sistemele bazate pe conținut examinează proprietățile articolelor recomandate. Pentru
de exemplu, dacă un utilizator Netflix a urmărit multe filme despre cowboy, atunci recomandă
remediați un film clasificat în baza de date ca având genul „cowboy”.
- Sistemele de filtrare colaborativă recomandă elemente bazate pe măsuri de similaritate
surse între utilizatori și / sau articole. Elementele recomandate unui utilizator sunt
cele preferate de utilizatori similari. Acest tip de sistem de recomandare poate
folosiți bazele stabilite în capitolul 3 pentru căutarea similarității și în capitolul 7
pe grupare. Cu toate acestea, aceste tehnologii de la sine nu sunt suficiente
și există câțiva algoritmi noi care s-au dovedit eficienți pentru
sisteme de recomandare.

9.1 Un model pentru sistemele de recomandare

În această secțiune introducem un model pentru sistemele de recomandare, bazat pe
o matrice de utilități a preferințelor. Introducem conceptul de „coadă lungă”

ceea ce explică avantajul vânzătorilor online față de produsele convenționale, cărămidă și
vânzători de mortar. Apoi analizăm pe scurt tipurile de aplicații în care
sistemele de recomandare s-au dovedit utile.

9.1.1 Matricea utilitară

Într-o aplicație-sistem de recomandare există două clase de entități, care
ne vom referi ca utilizatori și articole. Utilizatorii au preferințe pentru anumite articole,
iar aceste preferințe trebuie scoase din date. Datele în sine sunt reprezentate
trimis ca o matrice de utilități, oferind pentru fiecare pereche utilizator-articol, o valoare care reprezintă
ceea ce se știe despre gradul de preferință al utilizatorului respectiv pentru acel articol. Valori
provin dintr-un set ordonat, de exemplu, numere întregi 1-5 care reprezintă numărul de stele
pe care utilizatorul a acordat-o drept evaluare pentru acel articol. Presupunem că matricea este
rare, ceea ce înseamnă că majoritatea intrărilor sunt „necunoscute”. O evaluare necunoscută implică
că nu avem informații explicite despre preferința utilizatorului pentru articol.
Exemplul 9.1: În Fig. 9.1 vedem un exemplu de matrice utilitară, reprezentând
evaluări ale filmelor pe o scară de la 1 la 5, cu 5 cea mai mare evaluare. Blankurile reprezintă
situația în care utilizatorul nu a evaluat filmul. Numele filmelor sunt
HP1, HP2 și HP3 pentru Harry Potter I, II și III, TW pentru Twilight și SW1,
SW2 și SW3 pentru Star Wars episoadele 1, 2 și 3. Utilizatorii sunt reprezentați
cu majuscule de la A la D.

HP1 HP2 HP3 TW SW1 SW2 SW3
A
4
5
1
B
5
5
4
C
2
4
5
D
3
3
Figura 9.1: O matrice de utilități care reprezintă evaluări ale filmelor pe o scară 1-5
Observați că majoritatea perechilor utilizator-film au spații libere, ceea ce înseamnă că utilizatorul nu are
a apreciat filmul. În practică, matricea ar fi și mai rară, cu cea tipică
utilizatorii evaluează doar o mică parte din toate filmele disponibile. ✷
Scopul unui sistem de recomandare este de a prezice golurile din utilitar
matrice. De exemplu, utilizatorul A ar dori SW2? Există puține dovezi din
matricea minusculă din Fig. 9.1. S-ar putea să proiectăm sistemul nostru de recomandare pentru
să ia în considerare proprietățile filmelor, cum ar fi producătorul, regizorul, vedetele,
sau chiar asemănarea numelor lor. Dacă da, s-ar putea să observăm similaritatea
între SW1 și SW2 și apoi concluzionează că, deoarece lui A nu i-a plăcut SW1, ei
Era puțin probabil să se bucure nici de SW2. Alternativ, cu mult mai multe date, noi
s-ar putea observa că oamenii care au evaluat atât SW1, cât și SW2 au avut tendința de a da
le evaluări similare. Astfel, am putea concluziona că A ar da și SW2 a
rating scăzut, similar cu ratingul lui SW1.

De asemenea, ar trebui să fim conștienți de un obiectiv ușor diferit, care are sens în mulți
aplicații. Nu este necesar să se prevadă fiecare intrare goală dintr-o matrice de utilități.
Mai degrabă, este necesar doar să descoperiți câteva intrări în fiecare rând care sunt probabile
a fi înalt. În majoritatea aplicațiilor, sistemul de recomandare nu oferă
utilizatorilor o clasare a tuturor articolelor, ci mai degrabă sugerează câteva pe care utilizatorul ar trebui să le aprecieze
extrem de. Este posibil să nu fie chiar necesar să găsiți toate articolele cu cel mai mare nivel de așteptare
evaluări, dar numai pentru a găsi un subgrup mare de persoane cu cele mai mari evaluări.

9.1.2 Coada lungă

Înainte de a discuta despre principalele aplicații ale sistemelor de recomandare, permiteți-ne
medita la fenomenul cozii lungi care face ca sistemele de recomandare să fie
sary. Sistemele de livrare fizică sunt caracterizate de o penurie de resurse.
Magazinele de cărămidă și mortar au spațiu limitat pe raft și pot arăta clientului
doar o mică parte din toate alegerile care există. Pe de altă parte, on-line
magazinele pot pune la dispoziția clientului orice există. Astfel, un fizic
librăria poate avea câteva mii de cărți pe rafturile sale, dar Amazon oferă
milioane de cărți. Un ziar fizic poate imprima câteva zeci de articole pe
zi, în timp ce serviciile de știri online oferă mii pe zi.
Recomandarea în lumea fizică este destul de simplă. În primul rând, nu este
este posibil să adaptați magazinul la fiecare client în parte. Astfel, alegerea
ceea ce este disponibil este guvernat numai de numerele agregate. De obicei, a
librăria va afișa doar cărțile cele mai populare și un ziar
va imprima doar articolele pe care crede că le vor interesa cei mai mulți oameni.
În primul caz, cifrele de vânzări guvernează alegerile, în al doilea caz, editorial
judecata servește.
A fost numită distincția dintre lumile fizice și cele online
fenomenul cozii lungi și este sugerat în Fig. 9.2. Axa verticală
reprezintă popularitate (de câte ori este ales un articol). Elementele sunt
ordonate pe axa orizontală în funcție de popularitatea lor. Instituții fizice
acțiunile oferă doar cele mai populare articole din stânga liniei verticale, în timp ce
instituțiile on-line corespunzătoare furnizează întreaga gamă de articole: coada
precum și articolele populare.
Fenomenul de coadă lungă obligă instituțiile on-line să recomande articole
utilizatorilor individuali. Nu este posibil să prezentați utilizatorului toate articolele disponibile,
modul în care instituțiile fizice pot. Nici nu ne putem aștepta ca utilizatorii să fi auzit

din fiecare dintre elementele pe care le-ar putea plăcea.

9.1.3 Aplicații ale sistemelor de recomandare

Am menționat câteva aplicații importante ale sistemelor de recomandare,
dar aici vom consolida lista într-un singur loc.

1. Recomandări despre produse: Poate că este cea mai importantă utilizare a recomandărilor
sistemele de reparații se desfășoară la comercianții cu amănuntul online. Am observat cum Amazon
sau furnizori similari on-line se străduiesc să le ofere fiecărui utilizator care revine

---

Figura 9.2: Coada lungă: instituțiile fizice pot furniza doar ceea ce este popular,
în timp ce instituțiile on-line pot pune la dispoziție totul
sugestii de produse pe care ar putea să le cumpere. Aceste sugestii sunt
nu aleatoriu, ci se bazează pe deciziile de cumpărare luate de similare
clienților sau despre alte tehnici pe care le vom discuta în acest capitol.

2. Recomandări de filme: Netflix oferă clienților săi recomandări
de filme care ar putea să le placă. Aceste recomandări se bazează pe evaluări
furnizate de utilizatori, la fel ca evaluările sugerate în exemplul de utilitate
matricea din Fig. 9.1. Importanța prezicerii exacte a evaluărilor este
atât de mare, încât Netflix a oferit un premiu de un milion de dolari pentru primul
algoritm care ar putea bate propriul sistem de recomandare cu 10%. [1]
The
premiul a fost în cele din urmă câștigat în 2009, de către o echipă de cercetători numită „Bellkor
Haos pragmatic ”, după peste trei ani de concurență.

3. Articole de știri: Serviciile de știri au încercat să identifice articole de informație
interes pentru cititori, pe baza articolelor pe care le-au citit în trecut.
Similitudinea s-ar putea baza pe similaritatea cuvintelor importante din
documente sau pe articolele care sunt citite de persoane cu o lectură similară
gusturi. Aceleași principii se aplică recomandării blogurilor din rândul acestora
milioane de bloguri disponibile, videoclipuri pe YouTube sau alte site-uri unde
conținutul este furnizat în mod regulat.

[1] Pentru a fi exact, algoritmul trebuia să aibă o eroare pătrată-medie-rădăcină (RMSE) de 10%
mai puțin decât RMSE al algoritmului Netflix pe un set de test preluat din evaluările reale ale Netflix
utilizatori. Pentru a dezvolta un algoritm, concurenților li s-a oferit un set de date de instruire, preluat și din
date reale Netflix.

În aer subțire și atingerea vidului

Un exemplu extrem al modului în care coada lungă, împreună cu un design bine conceput
sistemul de recomandare poate influența evenimentele este povestea spusă de Chris An-
derson despre o carte numită Atingerea vidului. Acest alpinism
cartea nu a fost un mare vânzător la vremea ei, dar la mulți ani după ce a fost publicată
a fost publicată o altă carte despre același subiect, numită Into Thin Air.
dorit. Sistemul de recomandare Amazon a remarcat câteva persoane care
a cumpărat ambele cărți și a început să recomande Atingerea vidului să
cei care au cumpărat sau au luat în considerare Into Thin Air. Dacă nu ar fi existat
vânzător de cărți online, Touching the Void nu ar fi putut fi văzut niciodată de potențial
cumpărători, dar în lumea online, atingerea Vidului a devenit în cele din urmă
foarte popular în sine, de fapt, mai mult decât în Into Thin Air.

9.1.4 Popularea matricei de utilități

Fără o matrice de utilități, este aproape imposibil să recomandăm articole. In orice caz,
dobândirea datelor din care să construiți o matrice de utilități este adesea dificilă. Sunt
două abordări generale pentru a descoperi valoarea pe care utilizatorii o acordă articolelor.

1. Putem cere utilizatorilor să evalueze articolele. În general, ratingurile filmelor sunt obținute
și unele magazine online încearcă să obțină evaluări de la cumpărătorii lor.
Site-urile care furnizează conținut, cum ar fi unele site-uri de știri sau YouTube, solicită de asemenea utilizatorilor
pentru a evalua articolele. Această abordare este limitată în ceea ce privește eficacitatea sa, deoarece în general
utilizatorii nu sunt dispuși să ofere răspunsuri și informațiile din acestea
cine o face poate fi părtinitor de chiar faptul că provine de la oameni dornici
pentru a oferi evaluări.

2. Putem face inferențe din comportamentul utilizatorilor. Cel mai evident, dacă este un utilizator
cumpără un produs de la Amazon, urmărește un film pe YouTube sau citește o știre
articol, atunci se poate spune că utilizatorului îi place „acest articol”. Rețineți că acest tip

a sistemului de rating are într-adevăr o singură valoare: 1 înseamnă că îi place utilizatorului elementul. Adesea, găsim o matrice de utilități cu acest tip de date afișate cu 0, mai degrabă decât spații libere, în cazul în care utilizatorul nu a cumpărat sau vizualizat elementul. Cu toate acestea, în acest caz, 0 nu este mai mic decât 1; este nu evaluare deloc. Mai general, se poate deduce interesul din comportamentul celuilalt decât cumpărarea. De exemplu, dacă un client Amazon vizualizează informații despre un articol, putem deduce că sunt interesați de articol, chiar dacă nu o cumpără.

Pagina 344

9.2 Recomandări bazate pe conținut

După cum am menționat la începutul capitolului, există două arhitecturi de bază pentru un sistem de recomandare:

1. Sistemele bazate pe conținut se concentrează pe proprietățile articolelor. Asemănarea articolelor se determină prin măsurarea similarității în proprietățile lor.
2. Sistemele de filtrare colaborativă se concentrează pe relația dintre utilizatori și articole. Similitudinea articolelor este determinată de similaritatea evaluările acestor articole de către utilizatorii care au evaluat ambele articole.

În această secțiune, ne concentrăm pe sistemele de recomandare bazate pe conținut. Urmatorul secțiunea va acoperi filtrarea colaborativă.

9.2.1 Profiluri de articole

Într-un sistem bazat pe conținut, trebuie să construim pentru fiecare articol un profil, care este o înregistrare sau o colecție de înregistrări care reprezintă caracteristici importante ale acesteia articol. În cazuri simple, profilul constă din câteva caracteristici ale articolului care sunt ușor de descoperit. De exemplu, ia în considerare caracteristicile unui film care poate fi relevant pentru un sistem de recomandare.

1. Setul de actori ai filmului. Unii telespectatori preferă filmele cu actori preferați.
2. Directorul. Unii spectatori au o preferință pentru munca anumitor regizori.
3. Anul în care a fost realizat filmul. Unii telespectatori preferă filmele vechi; alții urmăresc doar cele mai recente versiuni.
4. Genul sau tipul general de film. Unora dintre spectatori le plac doar comediile, alții drame sau romanțe.

Există multe alte caracteristici ale filmelor care ar putea fi folosite și ele. Cu exceptia pentru ultimul gen, informațiile sunt disponibile din descrierile filme. Genul este un concept mai vag. Cu toate acestea, recenziile de filme atribuie în general un gen dintr-un set de termeni frecvent folosiți. De exemplu, filmul pe internet Baza de date (IMDB) atribuie un gen sau genuri fiecărui film. Vom discuta construcția mecanică a genurilor din secțiunea 9.3.3.

Multe alte clase de articole ne permit, de asemenea, să obținem caracteristici disponibile date, chiar dacă aceste date trebuie introduse la un moment dat manual. De exemplu, produsele au adesea descrieri scrise de producător, care oferă caracteristici relevante pentru acea clasă de produs (de exemplu, dimensiunea ecranului și culoarea dulapului pentru un TELEVIZOR). Cărțile au descrieri similare cu cele pentru filme, așa că putem obține funcții precum autorul, anul publicării și genul. Produse muzicale precum deoarece descărcările de CD-uri și MP3 au funcții disponibile precum artist, compozitor, și gen.

Pagina 345

9.2.2 Descoperirea caracteristicilor documentelor

Există alte clase de articole în care nu este imediat evident ce valorile caracteristicilor ar trebui să fie. Vom lua în considerare două dintre ele: colectarea documentelor acțiuni și imagini. Documentele prezintă probleme speciale și vom discuta tehnologia pentru extragerea caracteristicilor din documente din această secțiune. Imagini va fi discutat în secțiunea 9.2.3 ca un exemplu important în cazul în care furnizează utilizator caracteristicile au o oarecare speranță de succes.

Există multe tipuri de documente pentru care poate face un sistem de recomandare fii util. De exemplu, există multe articole de știri publicate în fiecare zi și nu le putem citi pe toate. Un sistem de recomandare poate sugera articole despre subiectele de care este interesat un utilizator, dar cum putem distinge între subiecte? Web

paginile sunt, de asemenea, o colecție de documente. Putem sugera pagini pe care le poate folosi un utilizator vreau sa vad? La fel, blogurile ar putea fi recomandate utilizatorilor interesați, dacă noi ar putea clasifica blogurile după subiecte.
Din păcate, aceste clase de documente nu tind să fie disponibile informații capabile care oferă caracteristici. Un substitut care a fost util în practică este identificarea cuvintelor care caracterizează tema unui document. Cum facem identificarea a fost prezentată în secțiunea 1.3.1. Mai întâi, eliminați cuvintele de oprire - cele câteva sute de cuvinte cele mai frecvente, care tind să spună puțin despre subiectul unui document. Pentru cuvintele rămase, calculați scorul TF.IDF pentru fiecare cuvânt din document. Cele cu cele mai mari scoruri sunt cuvintele care caracterizează documentul.
Putem lua apoi ca caracteristici ale unui document cele n cuvinte cu cel mai mare Scoruri TF.IDF. Este posibil să alegeți n pentru a fi același pentru toate documentele sau pentru fie n un procent fix din cuvintele din document. Am putea alege și noi pentru a face ca toate cuvintele ale căror scoruri TF.IDF sunt peste un prag dat să fie a parte a setului de caracteristici.
Acum, documentele sunt reprezentate de seturi de cuvinte. Intuitiv, ne așteptăm aceste cuvinte pentru a exprima subiectele sau ideile principale ale documentului. De exemplu, într-un articol de știri, ne-am aștepta ca cuvintele cu cel mai mare scor TF.IDF să fie include numele persoanelor discutate în articol, proprietăți neobișnuite ale evenimentul descris și locația evenimentului. Pentru a măsura asemănarea a două documente, există mai multe măsuri naturale de distanță pe care le putem folosi:
1. Am putea folosi distanța Jaccard între seturile de cuvinte (amintim Sec. ziunea 3.5.3).
2. Am putea folosi distanța cosinusului (reamintim secțiunea 3.5.4) dintre seturi, tratate ca vectori.
Pentru a calcula distanța cosinusului în opțiunea (2), gândiți-vă la seturile de Cuvintele TF.IDF ca vector, cu o componentă pentru fiecare cuvânt posibil. The vectorul are 1 dacă cuvântul este în set și 0 dacă nu. Deoarece între două documente mențiuni există doar un număr finit de cuvinte între cele două seturi ale lor, infinitul dimensionalitatea vectorilor este lipsită de importanță. Aproape toate componentele au 0 in

---

Două tipuri de similitudini ale documentelor

Reamintim că în secțiunea 3.4 am dat o metodă pentru găsirea documentelor care au fost „similare”, folosind șindrilă, minhashing și LSH. Acolo, noțiunea de similaritate a fost lexical - documentele sunt similare dacă conțin mari, secvențe identice de personaje. Pentru sistemele de recomandare, noțiunea de asemănare este diferit. Ne interesează doar aparițiile multora cuvinte importante în ambele documente, chiar dacă există puțină asemănare lexicală între documente. Cu toate acestea, metodologia pentru a găsi similare documentele rămân aproape la fel. Odată ce avem o măsură de distanță, fie Jaccard, fie cosinus, putem folosi minhashing (pentru Jaccard) sau aleatoriu hiperplanuri (pentru distanța cosinusului; vezi Secțiunea 3.7.2) care alimentează date către un LSH algoritm pentru a găsi perechile de documente similare în sensul partajând multe cuvinte cheie comune.
atât, cât și 0 nu au impact asupra valorii produsului punct. Pentru a fi precis, punctul produs este mărimea intersecției celor două seturi de cuvinte și a lungimilor dintre vectori sunt rădăcinile pătrate ale numărului de cuvinte din fiecare set. Acea calculul ne permite să calculăm cosinusul unghiului dintre vectori ca produs punct împărțit la produsul lungimilor vectorului.

9.2.3 Obținerea caracteristicilor articolelor din etichete

Să luăm în considerare o bază de date cu imagini ca un exemplu al modului în care caracteristicile au obținute pentru obiecte. Problema cu imaginile este că datele lor, de obicei o serie de pixeli, nu ne spune nimic util despre caracteristicile lor. Putem calculați proprietățile simple ale pixelilor, cum ar fi cantitatea medie de roșu din poză, dar puțini utilizatori caută imagini roșii sau mai ales ca imagini roșii.
Au existat mai multe încercări de a obține informații despre caracteristici de articole, invitând utilizatorii să eticheteze articolele introducând cuvinte sau fraze care descrie articolul. Astfel, o imagine cu mult roșu ar putea fi etichetată „Tianan-men Square ”, în timp ce altul este etichetat„ apus la Malibu ”. Distincția este nu ceva ce ar putea fi descoperit de programele existente de analiză a imaginii.
Aproape orice fel de date pot avea caracteristicile sale descrise de etichete. Una dintre primele încercări de a eticheta cantități masive de date au fost site-ul del.icio.us, cumpărat ulterior de Yahoo !, care a invitat utilizatorii să eticheteze pagini Web. Scopul acestui lucru etichetarea trebuia să facă disponibilă o nouă metodă de căutare, unde utilizatorii au introdus un

set de etichete ca interogare de căutare și sistemul a recuperat paginile web care
fusese etichetat astfel. Cu toate acestea, este posibilă și utilizarea etichetelor ca fișier
sistemul de recomandare. Dacă se observă că un utilizator recuperează sau marcează
multe pagini cu un anumit set de etichete, apoi putem recomanda alte pagini cu
aceleași etichete.
Problema cu etichetarea ca abordare a descoperirii caracteristicilor este că

Etichete din jocuri pe computer

O direcție interesantă pentru încurajarea etichetării este abordarea „jocurilor”
pionierat de Luis von Ahn. A permis doi jucători să colaboreze la
etichetă pentru o imagine. În runde, ei ar sugera o etichetă, iar etichetele ar face-o
să fie schimbat. Dacă au fost de acord, atunci „au câștigat”, iar dacă nu, ar face-o
joacă o altă rundă cu aceeași imagine, încercând să fii de acord simultan
pe o etichetă. Deși este o direcție inovatoare de încercat, este îndoielnic dacă
se poate genera suficient interes public pentru a produce suficientă muncă gratuită
satisface nevoile de date etichetate.
procesul funcționează numai dacă utilizatorii sunt dispuși să-și facă probleme cu crearea etichetelor și
există destule etichete pentru ca cele eronate ocazionale să nu influențeze sistemul
prea mult.

9.2.4 Reprezentarea profilurilor articolelor

Scopul nostru final pentru recomandarea bazată pe conținut este să creăm atât un element
profil format din perechi caracteristică-valoare și un profil de utilizator care rezumă
erențele utilizatorului, pe baza rândului lor din matricea utilitară. În secțiunea 9.2.2
am sugerat cum ar putea fi construit un profil de articol. Ne-am imaginat un vector
din 0 și 1, unde un 1 a reprezentat apariția unui cuvânt cu TF.IDF mare
în document. Deoarece caracteristicile pentru documente erau toate cuvinte, a fost ușor de făcut
reprezintă profilurile în acest fel.
Vom încerca să generalizăm această abordare vectorială la tot felul de caracteristici. Este
ușor de făcut pentru caracteristicile care sunt seturi de valori discrete. De exemplu, dacă unul
caracteristica filmelor este setul de actori, apoi imaginați-vă că există o componentă
pentru fiecare actor, cu 1 dacă actorul este în film și 0 dacă nu. La fel, noi
poate avea o componentă pentru fiecare regizor posibil și pentru fiecare gen posibil. Toate
aceste caracteristici pot fi reprezentate folosind doar 0 și 1.
Există o altă clasă de caracteristici care nu este ușor reprezentată de Boolean
vectori: acele caracteristici care sunt numerice. De exemplu, am putea lua
ratingul mediu pentru filme ca o caracteristică, [2] și această medie este un număr real.
Nu are sens să ai o componentă pentru fiecare dintre media posibilă
evaluări, iar acest lucru ne-ar determina să pierdem structura implicită în cifre.
Adică, două evaluări apropiate, dar nu identice, ar trebui considerate mai mult
asemănătoare decât evaluările foarte diferite. De asemenea, caracteristicile numerice ale produselor,
precum dimensiunea ecranului sau capacitatea discului pentru PC-uri, ar trebui considerate similare dacă
valorile lor nu diferă foarte mult.
Caracteristicile numerice ar trebui să fie reprezentate de componente individuale ale vectorilor
reprezentând obiecte. Aceste componente dețin valoarea exactă a acelei caracteristici.

[2] Evaluarea nu este o caracteristică foarte fiabilă, dar va servi drept exemplu.

Nu există niciun rău dacă unele componente ale vectorilor sunt booleeni și altele sunt
cu valoare reală sau cu valori întregi. Încă putem calcula distanța cosinusului dintre
vectori, deși dacă facem acest lucru, ar trebui să ne gândim în mod adecvat
a scalat componentele non-booleene, astfel încât nici ele să nu domine
calcul și nici nu sunt irelevante.
Exemplul 9.2: Să presupunem că singurele caracteristici ale filmelor sunt setul de actori și
ratingul mediu. Luați în considerare două filme cu cinci actori fiecare. Două dintre
actorii sunt în ambele filme. De asemenea, un film are un rating mediu de 3 și
altele o medie de 4. Vectorii arată ceva de genul
0 1 1 0 1 1 0 1 3α
1 1 0 1 0 1 1 0 4α
Cu toate acestea, există, în principiu, un număr infinit de componente suplimentare,
fiecare cu 0 pentru ambii vectori, reprezentând toți actorii posibili pe care nici unul dintre ei
filmul are. Deoarece distanța cosinusului vectorilor nu este afectată de componentele din

pe care ambii vectori au 0, nu trebuie să ne facem griji cu privire la efectul actorilor care nu sunt în niciun film.
Ultima componentă afișată reprezintă ratingul mediu. Am arătat ca având un factor de scalare necunoscut α.În termeni de α,putem calcula cosinusul unghiului dintre vectori. Produsul punct este $2 + 12\alpha^2$, iar lungimile vectorilor sunt $\sqrt{5+ 9\alpha^2}$ și $\sqrt{5+ 16\alpha^2}$. Astfel, cosinusul unghiul dintre vectori este

$$\frac{2 + 12\alpha^2}{\sqrt{25 + 125\alpha^2 + 144\alpha^4}}$$

Dacă alegem $\alpha= 1$, adică luăm evaluările medii așa cum sunt, atunci valoarea expresiei de mai sus este 0,816. Dacă folosim $\alpha= 2$, adică ne dublăm evaluările, atunci cosinusul este 0,940. Adică vectorii par mult mai apropiați în direcție decât dacă folosim $\alpha= 1$. La fel, dacă folosim $\alpha= 1/2$, atunci cosinusul este 0,619, ceea ce face ca vectorii să arate destul de diferit. Nu putem spune care este valoarea αeste „corect”, dar vedem că alegerea factorului de scalare pentru caracteristicile numerice afectează decizia noastră cu privire la cât de similare sunt articolele. ✷

9.2.5 Profiluri de utilizator

Nu trebuie doar să creăm vectori care descriu elemente; trebuie să creăm vectori cu aceleași componente care descriu preferințele utilizatorului. Avem matrice utilitară reprezentând conexiunea dintre utilizatori și articole. Reamintim intrările de matrice nonblank ar putea fi doar 1 reprezentând achiziții ale utilizatorilor sau a conexiune similară, sau ar putea fi numere arbitrare reprezentând o evaluare sau gradul de afecțiune pe care utilizatorul îl are pentru articol.
Cu aceste informații, cea mai bună estimare pe care o putem face cu privire la ce articole utilizatorului îi place o agregare a profilurilor acelor articole. Dacă utilitatea matricea are doar 1, atunci agregatul natural este media componentelor

---

a vectorilor care reprezintă profilurile articolelor pentru articolele în care utilitarul matrix are 1 pentru acel utilizator.
Exemplul 9.3: Să presupunem că articolele sunt filme, reprezentate de profiluri booleene cu componente corespunzătoare actorilor. De asemenea, matricea utilitară are 1 dacă utilizatorul a văzut filmul și este necompletat în caz contrar. Dacă 20% din filmele acelui utilizator U like-urile o au pe Julia Roberts ca unul dintre actori, apoi profilul utilizatorului pentru U va fi au 0,2 în componenta pentru Julia Roberts. ✷
Dacă matricea utilitară nu este booleană, de exemplu, evaluările 1-5, atunci putem pondera vectorii reprezentând profilurile articolelor după valoarea utilității. Face sensul de a normaliza utilitățile prin scăderea valorii medii pentru un utilizator.
În acest fel, obținem greutăți negative pentru articolele cu o evaluare sub medie și ponderi pozitive pentru articolele cu evaluări peste medie. Acest efect se va dovedi util atunci când discutăm în secțiunea 9.2.6 cum să găsim elemente pe care ar trebui să le utilizeze ca.
Exemplul 9.4: Luați în considerare aceleași informații despre film ca în Exemplul 9.3, dar acum să presupunem că matricea utilitară are intrări non-blank care sunt cotate în 1-5 gamă. Să presupunem că utilizatorul U acordă un rating mediu de 3. Există trei filme cu Julia Roberts ca actor, iar acele filme au primit rating de 3, 4 și 5.
Apoi, în profilul de utilizator al U, componenta pentru Julia Roberts va avea valoare adică media de 3 - 3, 4 - 3 și 5 - 3, adică o valoare de 1.
Pe de altă parte, utilizatorul V acordă o evaluare medie de 4 și, de asemenea, a evaluat trei filme cu Julia Roberts (nu contează dacă sunt sau nu aceleași trei filme apreciate de U). Utilizatorul V acordă acestor trei filme evaluări de 2, 3, și 5. Profilul de utilizator pentru V are, în componenta pentru Julia Roberts, media de 2 - 4, 3 - 4 și 5 - 4, adică valoarea $-2/3$. ✷

9.2.6 Recomandarea de articole utilizatorilor pe baza conținutului

Cu vectori de profil atât pentru utilizatori, cât și pentru articole, putem estima gradul până la între care un utilizator ar prefera un articol calculând distanța dintre cosinus între vectorii utilizatorului și articolului. La fel ca în Exemplul 9.2, s-ar putea să dorim să scalăm variații Ious componente ale căror valori nu sunt booleene. Hiperplanul aleator și tehnicile de hashing sensibile la localitate pot fi folosite pentru a plasa (doar) profiluri de articole în găleți. În acest fel, dat unui utilizator căruia dorim să-l recomandăm itemi, putem aplica aceleași două tehnici - hiperplane aleatorii și LSH - pentru a determina în ce găleți trebuie să căutăm obiecte care ar putea avea un mic distanța cosinus de utilizator.
Exemplul 9.5: Luați în considerare mai întâi datele din Exemplul 9.3. Profilul utilizatorului va fi au componente pentru actori proporționale cu probabilitatea ca actorul să aibă

apare într-un film care îi place utilizatorului. Astfel, cele mai înalte recomandări (cele mai mici distanța cosinusului) aparțin filmelor cu o mulțime de actori care apar în mulți

---

Pagina 350

din filmele care îi plac utilizatorului. Atâta timp cât actorii sunt singurele informații pe care le avem despre caracteristicile filmelor, acesta este probabil cel mai bun lucru pe care îl putem face. [3]
Acum, ia în considerare Exemplul 9.4. Acolo, am observat că vectorul pentru un utilizator va avea numere pozitive pentru actorii care tind să apară în filme de către utilizator aprecieri și numere negative pentru actorii care tind să apară în filme de utilizator nu-i place. Luați în considerare un film cu mulți actori care îi plac utilizatorului și doar câțiva sau niciuna care să nu-i placă utilizatorului. Cosinusul unghiului dintre cel al utilizatorului iar vectorii filmului vor fi o fracțiune pozitivă mare. Asta implică un unghi închis la 0 și, prin urmare, o mică distanță de cosinus între vectori.
Apoi, ia în considerare un film cu aproximativ la fel de mulți actori care îi plac utilizatorului ca aceia utilizatorului nu-i place. În această situație, cosinusul unghiului dintre utilizator și filmul este în jur de 0 și, prin urmare, unghiul dintre cei doi vectori este în jur 90 de grade. În cele din urmă, luați în considerare un film cu majoritatea actori pe care utilizatorul nu îi place. În acest caz, cosinusul va fi o fracție negativă mare, iar unghiul dintre cei doi vectori vor fi aproape de 180 de grade - cosinusul maxim posibil distanță. ✷

9.2.7 Algoritmi de clasificare

O abordare complet diferită a unui sistem de recomandare care utilizează profiluri de articole iar matricile utilitare sunt de a trata problema ca fiind una de învățare automată. Privire datele date ca set de instruire și, pentru fiecare utilizator, construiți un clasificator care prezice evaluarea tuturor articolelor. Există un număr mare de clasificatori diferiți și asta nu este scopul nostru de a preda acest subiect aici. Cu toate acestea, ar trebui să fii conștient a opțiunii de a dezvolta un clasificator pentru recomandare, așa că vom discuta un clasificator comun - arbori de decizie - pe scurt.
Un arbore de decizie este o colecție de noduri, aranjate ca un arbore binar. The lasă să ia decizii; în cazul nostru, decizia ar fi „aprecieri” sau „nu” ca." Fiecare nod interior este o condiție pentru obiectele clasificate; în a noastră în cazul în care condiția ar fi un predicat care implică una sau mai multe caracteristici ale unui articol.
Pentru a clasifica un element, începem de la rădăcină și aplicăm predicatul la rădăcină la articol. Dacă predicatul este adevărat, mergeți la copilul din stânga și, dacă este fals, mergeți la copilul potrivit. Apoi repetați același proces la nodul vizitat, până la o frunză este atins. Această frunză clasifică articolul ca apreciat sau nu.
Construirea unui arbore de decizie necesită selectarea unui predicat pentru fiecare nod interior. Există multe modalități de a alege cel mai bun predicat, dar toate încercați să aranjați ca unul dintre copii să primească toate sau majoritatea exemplelor pozitive în setul de instruire (adică elementele care îi plac utilizatorului dat, în cazul nostru) și alt copil primește toate sau majoritatea exemplelor negative (articolele pe care le face acest utilizator nu ca).

[3] Rețineți că faptul că toate componentele utilizator-vector vor fi fracții mici nu afectează recomandare, deoarece calculul cosinusului implică împărțirea la lungimea fiecărui vector. Adică, vectorii de utilizator vor avea tendința de a fi mult mai scurți decât vectorii de film, ci doar direcția de vectori contează.

Pagina 351

Odată ce am selectat un predicat pentru un nod N, împărțim articolele în cele două grupuri: cele care satisfac predicatul și cele care nu. Pentru fiecare grup, găsim din nou predicatul care separă cel mai bine pozitivul și exemple negative în acel grup. Aceste predicate sunt atribuite copiilor din N. Acest proces de împărțire a exemplelor și construirea copiilor poate continua la orice număr de niveluri. Putem opri și crea o frunză, dacă grupul de articole pentru un nod este omogen; adică toate sunt exemple pozitive sau negative.
Cu toate acestea, este posibil să dorim să ne oprim și să creăm o frunză cu decizia majorității pentru un grup, chiar dacă grupul conține atât exemple pozitive, cât și negative.
Motivul este că semnificația statistică a unui grup mic poate să nu fie ridicată suficient pentru a te baza. Din acest motiv, o variantă de strategie este crearea unui ansamblu de arborii de decizie, fiecare folosind diferite predicate, dar permit copacilor să fie mai adânci decât ceea ce justifică datele disponibile. Astfel de copaci sunt numiți supra-echipați. La clasificați un articol, aplicați toți copacii din ansamblu și lăsați-i să voteze

rezultat. Nu vom lua în considerare această opțiune aici, ci vom da o ipotetică simplă
exemplu de arbore de decizie.
Exemplul 9.6: Să presupunem că articolele noastre sunt articole de știri, iar caracteristicile sunt cele mai importante
TF.IDF cuvinte (cuvinte cheie) în acele documente. Mai mult, să presupunem că există un utilizator
U căruia îi plac articolele despre baseball, cu excepția articolelor despre New York Yankees.
Rândul matricei utilitare pentru U are 1 dacă U a citit articolul și este necompletat dacă
nu. Vom lua 1-urile ca „like” și spațiile libere ca „nu-mi place”. Predicati
vor fi expresii booleene ale cuvintelor cheie.
Întrucât lui U îi place în general baseball-ul, am putea găsi că este cel mai bun predicat pentru
rădăcina este „homerun” SAU („aluat” ȘI „ulcior”). Articole care satisfac
predicat va tinde să fie exemple pozitive (articole cu 1 în rând pentru U în
matricea utilitară), iar articolele care nu reușesc să satisfacă predicatul vor tinde să fie
exemple negative (spații libere în rândul matricei utilitare pentru U). Figura 9.3 prezintă
rădăcina, precum și restul arborelui decizional.
Să presupunem că grupul de articole care nu satisfac predicatul include
suficient de puține exemple pozitive în care putem concluziona că toate aceste elemente se află
clasa „nu-mi place”. Putem pune apoi o frunză cu decizia „nu-mi place” ca
copil drept al rădăcinii. Cu toate acestea, articolele care satisfac predicatul includ
o serie de articole care nu-i plac utilizatorului U; acestea sunt articolele care menționează
Yankees. Astfel, la copilul stâng al rădăcinii, construim un alt predicat.
Am putea descoperi că predicatul „Yankees” SAU „Jeter” SAU „Teixeira” este
cel mai bun indicator posibil al unui articol despre baseball și despre Yankees.
Astfel, vedem în Fig. 9.3 copilul stâng al rădăcinii, care aplică acest predicat.
Ambii copii ai acestui nod sunt frunze, deoarece putem presupune că articolele
satisfacerea acestui predicat sunt predominant negative și cei care nu-l satisfac
sunt predominant pozitive. ✷
Din păcate, clasificatorii de toate tipurile tind să construiască mult timp.
De exemplu, dacă dorim să folosim arbori de decizie, avem nevoie de un arbore per utilizator. Con-
structurarea unui arbore nu necesită doar să ne uităm la toate profilurile articolelor, ci și noi

---

Pagina 352

(„aluat” ȘI „ulcior”)

,,homerun” SAU

,,Yankees” SAU

,,Jeter” SAU „Teixeira”
Nu
Ca
Nu
Ca
Îi place
Figura 9.3: Un arbore de decizie
trebuie să ia în considerare multe predicate diferite, care ar putea implica com-
bинări de caracteristici. Astfel, această abordare tinde să fie utilizată doar pentru relativ
dimensiuni mici ale problemei.

9.2.8 Exerciții pentru secțiunea 9.2

Exercițiul 9.2.1: Trei calculatoare, A, B și C, au caracteristicile numerice
enumerate mai jos:
Caracteristică
A
B
C
Viteza procesorului
3,06 2,68 2,92
Dimensiunea discului
500
320
640
Dimensiunea memoriei principale
6
4
6
Ne putem imagina aceste valori ca definind un vector pentru fiecare computer; pentru in-
poziția, vectorul lui A este [3.06, 500, 6]. Putem calcula distanța cosinusului dintre
oricare dintre doi vectori, dar dacă nu scalăm componentele, atunci discul
dimensiunea va domina și va face diferențe în celelalte componente în esență în

vizibil. Să folosim 1 ca factor de scară pentru viteza procesorului, αpentru dimensiunea discului, și βpentru dimensiunea memoriei principale.
(a) În ceea ce privește αși β,calculați cosinusurile unghiurilor dintre vectori pentru fiecare pereche din cele trei calculatoare.
(b) Care sunt unghiurile dintre vectori dacă $\alpha = \beta = 1$?
(c) Care sunt unghiurile dintre vectori dacă $\alpha = 0{,}01$ și $\beta = 0{,}5$?

! (d) Un mod corect de selectare a factorilor de scală este de a face ca fiecare să fie proporțional invers proporțional cu valoarea medie a componentei sale. Care ar fi valorile dintre αși β,și care ar fi unghiurile dintre vectori?

Exercițiul 9.2.2: O modalitate alternativă de scalare a componentelor unui vector este de a începe prin normalizarea vectorilor. Adică, calculați media pentru fiecare companie ponent și scade-l din valoarea componentei respective în fiecare vector.
(a) Normalizați vectorii pentru cele trei calculatoare descrise în exercițiul 9.2.1.
!! (b) Această întrebare nu necesită calcul dificil, dar necesită unele gândire serioasă despre ce înseamnă unghiurile dintre vectori. Când toate ponenții nu sunt negativi, așa cum sunt în datele din Exercițiul 9.2.1, nr vectorii pot avea un unghi mai mare de 90 de grade. Cu toate acestea, atunci când noi normaliza vectorii, putem (și trebuie) să obținem câteva componente negative, deci unghiurile pot fi acum orice, adică de la 0 la 180 de grade. În plus, mediile sunt acum 0 în fiecare componentă, deci sugestia din partea (d) din Exercițiul 9.2.1 pe care ar trebui să-l scalăm în proporție inversă cu media nu are niciun sens. Sugerează o modalitate de a găsi o scară adecvată pentru fiecare componentă a vectorilor normalizați. Cum ați interpreta un mare sau mic unghi între vectori normalizați? Pentru ce ar fi unghiurile vectorii normalizați derivați din datele din exercițiul 9.2.1?

Exercițiul 9.2.3: Un anumit utilizator a evaluat cele trei calculatoare din Exercițiul 9.2.1 după cum urmează: A: 4 stele, B: 2 stele, C: 5 stele.
(a) Normalizați evaluările pentru acest utilizator.
(b) Calculați un profil de utilizator pentru utilizator, cu componente pentru viteza procesorului, dimensiunea discului și dimensiunea memoriei principale, pe baza datelor din exercițiul 9.2.1.

9.3 Filtrare colaborativă

Vom adopta acum o abordare semnificativ diferită a recomandării.
În loc să folosim caracteristicile elementelor pentru a determina similitudinea lor, ne concentrăm pe similaritatea evaluărilor utilizatorilor pentru două articole. Adică, în locul articolului-profil vector pentru un articol, folosim coloana sa în matricea utilitară. Mai departe, în schimb de a crea un vector de profil pentru utilizatori, îi reprezentăm după rândurile lor din matrice utilitară. Utilizatorii sunt similari dacă vectorii lor sunt apropiați în funcție de unii măsurarea distanței, cum ar fi Jaccard sau distanța cosinusului. Recomandare pentru a utilizatorul U se face apoi uitându-se la utilizatorii care sunt cei mai asemănători cu U în acest și recomandă elemente care le plac acestor utilizatori. Procesul de identificare utilizatori similari și recomandând ceea ce le place utilizatorilor similari se numește colaborativ filtrare.

9.3.1 Măsurarea similitudinii

Prima întrebare cu care trebuie să ne ocupăm este cum să măsurăm similitudinea utilizatorilor sau elemente din rândurile sau coloanele lor din matricea utilitară. Am reprodus Fig. 9.1 aici ca Fig. 9.4. Aceste date sunt prea mici pentru a trage concluzii fiabile, dar dimensiunea sa redusă va clarifica unele dintre capcanele alegerii unei distanțe măsura. Observați în mod specific utilizatorii A și C. Au evaluat două filme în comune, dar par a avea opinii aproape diametral opuse despre aceste filme. Ne-am aștepta să se facă o măsură bună a distanței ei destul de departe unul de altul. Iată câteva măsuri alternative de luat în considerare.

HP1 HP2 HP3 TW SW1 SW2 SW3
A
4
5
1
B
5

5
4
C
2
4
5
D
3
3
Figura 9.4: Matricea utilitară introdusă în Fig. 9.1
Distanța Jaccard
Am putea ignora valorile din matrice și ne putem concentra numai asupra seturilor de elemente evaluate. Dacă matricea de utilități reflectă doar achiziţiile, această măsură ar fi bună unul de ales. Cu toate acestea, atunci când utilitățile sunt evaluări mai detaliate, Jaccard distanța pierde informații importante.
Exemplul 9.7: A și B au o intersecție de dimensiunea 1 și o uniune de dimensiunea 5. Astfel, asemănarea lor Jaccard este 1/5, iar distanța lor Jaccard este 4/5; adică sunt foarte departate. În comparație, A și C au o similaritate Jaccard de 2/4, deci distanța lor Jaccard este aceeași, 1/2. Astfel, A apare mai aproape de C decât pentru B. Cu toate acestea, concluzia pare intuitiv greșită. A și C nu sunt de acord cele două filme pe care le-au vizionat amândoi, în timp ce A și B par să le fi plăcut amândouă un film pe care l-au vizionat în comun. ✷
Distanța cosinusului
Putem trata golurile ca o valoare 0. Această alegere este discutabilă, deoarece are efectul de a trata lipsa unei evaluări ca fiind mai asemănător cu antipatia filmului decât plăcându-i.
Exemplul 9.8: Cosinusul unghiului dintre A și B este
4×5
$\sqrt{4^2 + 5^2 + 1^2}\sqrt{5^2 + 5^2 + 4^2}$
$= 0{,}380$

Cosinusul unghiului dintre A și C este
$5 \times 2 + 1 \times 4$
$\sqrt{4^2 + 5^2 + 1^2}\sqrt{2^2 + 4^2 + 5^2}$
$= 0{,}322$
Deoarece un cosinus mai mare (pozitiv) implică un unghi mai mic și, prin urmare, un mai mic distanță, această măsură ne spune că A este puțin mai aproape de B decât de C. ✷
Rotunjirea datelor
Am putea încerca să eliminăm similitudinea aparentă dintre filmele pe care un utilizator le apreciază extrem și celor cu scoruri mici prin rotunjirea evaluărilor. De exemplu, am putea consideră ratingurile 3, 4 și 5 ca „1” și consideră ratingurile 1 și 2 ca nevalorificate. Matricea utilitară ar arăta apoi ca în Fig. 9.5. Acum, distanța Jaccard între A și B este 3/4, în timp ce între A și C este 1; adică, C apare mai departe din A decât B, ceea ce este intuitiv corect. Aplicarea distanței cosinusului la Fig. 9.5 ne permite să tragem aceeași concluzie.
HP1 HP2 HP3 TW SW1 SW2 SW3
A
1
1
B
1
1
1
C
1
1
D
1
1
Figura 9.5: Utilitățile de 3, 4 și 5 au fost înlocuite cu 1, în timp ce evaluările de 1 și 2 sunt omise
Normalizarea evaluărilor
Dacă normalizăm ratingurile, scăzând din fiecare rating ratingul mediu din acel utilizator, transformăm evaluările scăzute în cifre negative și cele mari în

numere pozitive. Dacă luăm distanța cosinusului, vom găsi că utilizatorii cu vizualizările opuse ale filmelor pe care le-au vizionat în comun vor avea aproape aproape vectori direcții opuse și pot fi considerate cât mai îndepărtate posibil. In orice caz, utilizatorii cu opinii similare despre filmele clasificate în comun vor avea un unghi relativ mic între ele.

Exemplul 9.9: Figura 9.6 prezintă matricea din Fig. 9.4 cu toate evaluările normalizat. Un efect interesant este că ratingurile lui D au dispărut efectiv, deoarece un 0 este același cu un gol când se calculează distanța cosinusului. Rețineți că D a dat doar 3 și nu s-a diferențiat între filme, deci este foarte posibil că opiniile lui D nu merită luate în serios.

Să calculăm cosinusul unghiului dintre A și B:

$$\frac{(2/3) \times (1/3)}{\sqrt{(2/3)^2 + (5/3)^2 + (-7/3)^2}\sqrt{(1/3)^2 + (1/3)^2 + (-2/3)^2}} = 0{,}092$$

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	2/3	5/3	−7/3				
B	1/3	1/3	−2/3				
C	−5/3	1/3	4/3				
D	0	0					

Figura 9.6: Matricea utilitară introdusă în Fig. 9.1

Cosinusul unghiului dintre A și C este

$$\frac{(5/3) \times (-5/3) + (-7/3) \times (1/3)}{\sqrt{(2/3)^2 + (5/3)^2 + (-7/3)^2}\sqrt{(-5/3)^2 + (1/3)^2 + (4/3)^2}} = -0.559$$

Observați că în conformitate cu această măsură, A și C sunt mult mai departe decât A și B, și niciuna dintre perechi nu este foarte apropiată. Ambele observații au un sens intuitiv, având în vedere că A și C nu sunt de acord cu cele două filme pe care le-au evaluat în comun, în timp ce A și B dau scoruri similare cu filmul pe care l-au evaluat în comun. ✷

9.3.2 Dualitatea asemănării

Matricea utilitară poate fi privită ca ne spune despre utilizatori sau despre articole sau ambii. Este important să ne dăm seama că oricare dintre tehnicile pe care le-am sugerat Secțiunea 9.3.1 pentru găsirea utilizatorilor similari poate fi utilizată pe coloanele utilitarului matrice pentru a găsi elemente similare. Există două moduri în care este simetria rupt în practică.

1. Putem folosi informații despre utilizatori pentru a recomanda articole. Adică dat un utilizator, putem găsi un număr dintre cei mai similari utilizatori, poate utilizând tehnicile din Capitolul 3. Ne putem baza recomandarea pe deciziile luate de acești utilizatori similari, de exemplu, recomandă articolele pe care cel mai mare număr dintre aceștia au achiziționat sau au evaluat cu mare valoare. Cu toate acestea, acolo nu este simetrie. Chiar dacă găsim perechi de articole similare, trebuie să luăm un pas suplimentar pentru a recomanda articole utilizatorilor. Acest punct este explorat în continuare la sfârșitul acestei subsecțiuni.

2. Există o diferență în comportamentul tipic al utilizatorilor și al articolelor, ca și el se referă la similitudine. Intuitiv, articolele tind să fie clasificabile în simple termeni. De exemplu, muzica tinde să aparțină unui singur gen. Este imposibil ble, de exemplu, pentru ca o piesă muzicală să fie atât rockul anilor 60, cât și barocul anilor 1700. Pe pe de altă parte, există persoane cărora le place atât rockul anilor 60, cât și anii 1700 baroc și care cumpără exemple de ambele tipuri de muzică. Consecința este că este mai ușor să descoperiți articole similare, deoarece aparțin la același gen, decât este de a detecta că doi utilizatori sunt similari deoarece preferă un gen în comun, în timp ce fiecăruia îi plac și unele genuri care celuilalt nu-i pasă.

După cum am sugerat în (1) de mai sus, o modalitate de a prezice valoarea utilității-intrarea matricei pentru utilizatorul U și elementul I este de a găsi n utilizatori (pentru unii predeterminați n) cele mai asemănătoare cu U și media evaluărilor lor pentru articolul I, numărându-le doar pe cele printre cei n utilizatori similari care au evaluat I. Este, în general, mai bine să se normalizeze matricea mai întâi. Adică, pentru fiecare dintre cei n utilizatori scădeți ratingul mediu pentru articolele din ratingul lor pentru i. Media diferenței pentru acei utilizatori care am evaluat I și apoi adăugați această medie la nota medie pe care o acordă U toate obiectele. Această normalizare ajustează estimarea în cazul în care U are tendința acordă evaluări foarte mari sau foarte mici, sau o fracțiune mare din utilizatorii similari care evaluat I (dintre care poate exista doar câțiva) sunt utilizatori care tind să evalueze foarte mult sau foarte scăzut.

În mod regulat, putem folosi similaritatea articolelor pentru a estima intrarea pentru utilizatorul U și articolul I. Găsiți m elemente cele mai asemănătoare cu I, pentru unii m, și luați ratingul mediu, dintre m articole, din ratingurile pe care le-a acordat U. În ceea ce privește similitudinea utilizator-utilizator, luăm în considerare numai acele elemente dintre m pe care U le-a evaluat și probabil că este Înțelept să normalizăm mai întâi evaluările articolelor.

Rețineți că oricare dintre abordările de estimare a intrărilor în matricea de utilități noi utilizarea, nu este suficient să găsiți o singură intrare. Pentru a recomanda articole către un utilizator U, trebuie să estimăm fiecare intrare din rândul matricei utilitare pentru U sau cel puțin găsiți toate sau majoritatea intrărilor din acel rând care sunt goale, dar care au o valoare estimată ridicată. Există un compromis în ceea ce privește dacă ar trebui să lucrăm de la utilizatori similari sau articole similare.

- Dacă găsim utilizatori similari, atunci trebuie să facem procesul o singură dată pentru utilizator U. Din setul de utilizatori similari putem estima toate golurile din matrice utilitară pentru U. Dacă lucrăm din elemente similare, trebuie să calculăm articole similare pentru aproape toate articolele, înainte de a putea estima rândul pentru U.
- Pe de altă parte, similitudinea element-articol oferă adesea informații mai fiabile formare, datorită fenomenului observat mai sus, și anume că este este mai ușor să găsești articole de același gen decât să găsești utilizatori cărora le place doar elemente dintr-un singur gen.

Indiferent de metoda pe care o alegem, ar trebui să calculăm în prealabil articolele preferate pentru fiecare utilizator, mai degrabă decât să așteptăm până când trebuie să luăm o decizie. Din moment ce utilitatea matricea evoluează lent, în general este suficient să o calculăm rar și presupunem că rămâne fix între recomputații.

9.3.3 Clusterizarea utilizatorilor și a elementelor

Este greu să detectăm similitudinea dintre articole sau utilizatori, deoarece avem puține informații despre perechile utilizator-articol din matricea de utilități rare. În perspectiva Secțiunii 9.3.2, chiar dacă două elemente aparțin aceluiași gen, acolo este probabil să fie foarte puțini utilizatori care au cumpărat sau evaluat ambii. La fel, chiar dacă ambilor utilizatori le place amândouă un gen sau genuri, este posibil să nu fi cumpărat niciun articol din uzual.

O modalitate de a face față acestei capcane este de a grupa articole și / sau utilizatori. Selectați oricare dintre măsurile de distanță sugerate în secțiunea 9.3.1 sau orice altă distanță măsurați și utilizați-o pentru a efectua o grupare de, să zicem, elemente. Oricare dintre metode sugerat în capitolul 7 poate fi folosit. Cu toate acestea, vom vedea că poate fii un motiv mic pentru a încerca să te grupezi într-un număr mic de clustere imediat.

Mai degrabă, o abordare ierarhică, în care lăsăm mai multe clustere nemiscate poate este suficient ca un prim pas. De exemplu, s-ar putea să lăsăm jumătate din numărul de clustere există articole.

HP
TW SW
A
4
5
1
B
4,67

C
2
4.5
D
3
3
Figura 9.7: Matrice de utilități pentru utilizatori și grupuri de articole
Exemplul 9.10: Figura 9.7 arată ce se întâmplă cu matricea utilitară din Fig. 9.4
dacă reușim să grupăm cele trei filme Harry-Potter într-un singur cluster, notat
HP și, de asemenea, grupează cele trei filme Star-Wars într-un singur cluster SW. ✷
Având elemente grupate într-o anumită măsură, putem revizui matricea utilitară astfel încât
coloanele reprezintă grupuri de articole, iar intrarea pentru utilizatorul U și clusterul este
ratingul mediu pe care U l-a acordat acelor membri ai grupului C pe care l-a evaluat U.
Rețineți că U poate nu a evaluat niciunul dintre membrii clusterului, caz în care
intrarea pentru U și C este încă necompletată.
Putem utiliza această matrice de utilități revizuită pentru a grupa utilizatorii, din nou folosind
măsură de timp considerată cea mai potrivită. Folosiți un algoritm de grupare care
lasă din nou multe clustere, de exemplu, jumătate din numărul de clustere din numărul de utilizatori. Re-
vise matricea utilitară, astfel încât rândurile să corespundă clusterelor de utilizatori, la fel ca
coloanele corespund grupurilor de articole. În ceea ce privește grupurile de elemente, calculați
intrare pentru un cluster de utilizatori prin media evaluărilor utilizatorilor din cluster.
Acum, acest proces poate fi repetat de mai multe ori, dacă ne place. Adică putem
grupați clusterele de elemente și combinați din nou coloanele matricei utilitare care
aparțin unui singur grup. Putem apoi să apelăm din nou la utilizatori și să grupăm fișierul
clustere de utilizatori. Procesul se poate repeta până când avem un intuitiv rezonabil
numărul de clustere de fiecare fel.
Odată ce am grupat utilizatorii și / sau articolele în măsura dorită și
a calculat matricea utilitară cluster-cluster, putem estima intrările din
matricea utilitară inală după cum urmează. Să presupunem că vrem să prezicem intrarea pentru utilizatorul U
și articolul I:
(a) Găsiți grupurile la care aparțin U și eu, spunem grupurile C și D, respectiv
tiv.

(b) Dacă intrarea în matricea de utilitate cluster-cluster pentru C și D este ceva
în afară de gol, utilizați această valoare ca valoare estimată pentru intrarea U – I
în matricea utilitară originală.
(c) Dacă intrarea pentru C-D este necompletată, atunci utilizați metoda descrisă în secțiunea 9.3.2
pentru a estima acea intrare luând în considerare clustere similare cu C sau D. Folosiți
estimare rezultată ca estimare pentru intrarea UI.

9.3.4 Exerciții pentru secțiunea 9.3

abcdef
gh
A
4 5
5 1
3 2
B
3 4 3 1 2
1
C
2
1 3
4
5 3
Figura 9.8: O matrice utilitară pentru exerciții
Exercițiul 9.3.1: Figura 9.8 este o matrice de utilități, reprezentând evaluările, pe a
Scala de 1–5 stele, de opt elemente, de la a la h, de către trei utilizatori A, B și C. Calculați
următoarele din datele acestei matrice.
(a) Tratând matricea utilitară ca booleană, calculați distanța Jaccard
între fiecare pereche de utilizatori.
(b) Repetați partea (a), dar utilizați distanța cosinusului.
(c) Tratați evaluările de 3, 4 și 5 ca 1 și 1, 2 și necompletate ca 0. Calculați
Distanța Jaccard între fiecare pereche de utilizatori.
(d) Repetați partea (c), dar utilizați distanța cosinusului.
(e) Normalizați matricea scăzând din fiecare intrare non-blank media

valoare pentru utilizatorul său.
(f) Folosind matricea normalizată din partea (e), calculați distanța cosinusului
între fiecare pereche de utilizatori.
Exercițiul 9.3.2: În acest exercițiu, grupăm elemente în matricea din Fig. 9.8.
Faceți pașii următori.
(a) Clusterizați cele opt elemente ierarhic în patru clustere. Următoarele
metoda ar trebui utilizată pentru a grupa. Înlocuiți toate cele 3, 4 și 5 cu 1 și
înlocuiți 1, 2 și goluri cu 0. folosiți distanța Jaccard pentru a măsura
distanța dintre vectorii de coloană rezultați. Pentru clustere de mai mult
decât un element, luați distanța dintre clustere pentru a fi minimă
distanță între perechi de elemente, câte una de la fiecare grup.

(b) Apoi, construiți din matricea originală din Fig. 9.8 o nouă matrice a cărei
rândurile corespund utilizatorilor, ca și până acum, și ale căror coloane corespund
clustere. Calculați intrarea pentru un utilizator și un grup de articole prin medie
intrările nonblank pentru acel utilizator și toate elementele din cluster.
(c) Calculați distanța cosinusului între fiecare pereche de utilizatori, în funcție de
matrice din partea (b).

9.4 Reducerea dimensionalității

O abordare complet diferită pentru estimarea intrărilor goale din utilitar
matricea este de a conjectura că matricea utilitară este de fapt produsul a două
matrice lungi și subțiri. Această viziune are sens dacă există un set relativ mic
a caracteristicilor articolelor și utilizatorilor care determină reacția majorității utilizatorilor la
majoritatea articolelor. În această secțiune, schițăm o abordare a descoperirii a două astfel
matrici; abordarea se numește „descompunere UV” și este un exemplu de
o teorie mai generală numită SVD (descompunere cu valoare singulară).

9.4.1 Descompunerea UV

Luați în considerare filmele ca un exemplu. Majoritatea utilizatorilor răspund la un număr mic
de caracteristici; le plac anumite genuri, pot avea anumiți actori celebri sau
actrițe care le plac și poate că sunt câțiva regizori cu un rol semnificativ
ca urmare a. Dacă începem cu matricea utilitară M, cu n rânduri și m coloane
(de exemplu, există n utilizatori și m elemente), atunci am putea găsi o matrice
U cu n rânduri și d coloane și o matrice V cu d rânduri și m coloane,
astfel încât UV să se apropie îndeaproape de M în acele intrări în care M este nonblank.
Dacă da, atunci am stabilit că există d dimensiuni care ne permit acest lucru
caracterizează atât utilizatorii, cât și articolele îndeaproape. Putem folosi apoi intrarea în
produs UV pentru a estima intrarea necompletată corespunzătoare în matricea de utilitate M.
Acest proces se numește descompunerea UV a lui M.

□
□
□
□
□
□
5 2 4 4 3
3 1 2 4 1
2
3 1 4
2 5 4 3 5
4 4 5 4
□
□
□
□
□
□
=
□
□

□
□
□
□
u 11
u 12
u 21
u 22
u 31
u 32
u 41
u 42
u 51
u 52
□
□
□
□
□
□
× [
v 11
v 12
v 13
v 14
v 15
v 21
v 22
v 23
v 24
v 25]
Figura 9.9: Descompunerea UV a matricei M
Exemplul 9.11: Vom folosi ca exemplu de rulare o matrice 5-cu-5 M cu
toate, cu excepția a două dintre intrările sale cunoscute. Dorim să descompunem M într-un 5-pe-2 și
Matricea 2-la-5, respectiv U și V. Sunt prezentate matricile M, U și V
în Fig. 9.9 cu intrările cunoscute ale lui M indicate și matricile U și

---

Pagina 361

V afișat cu intrările lor ca variabile de determinat. Acest exemplu este
în esență cel mai mic caz netrivial în care există mai multe intrări cunoscute decât
există intrări în U și V combinate și, prin urmare, ne putem aștepta ca
cea mai bună descompunere nu va produce un produs care este de acord exact în nonblank
intrările lui M. ✷

9.4.2 Eroare Root-Mean-Square

În timp ce putem alege dintre mai multe măsuri cu privire la cât de aproape de produs este UV
M, alegerea tipică este eroarea rădăcină-medie-pătrat (RMSE), unde noi
1. Suma, peste toate intrările neblane din M, pătratul diferenței dintre
acea intrare și intrarea corespunzătoare din produsul UV.
2. Luați media (media) acestor pătrate împărțind la numărul de
termeni în sumă (adică, numărul de intrări neblane în M).
3. Luați rădăcina pătrată a mediei.
Minimizarea sumei pătratelor este aceeași cu reducerea rădăcinii pătrate
din pătratul mediu, deci omitem în general ultimii doi pași din alergarea noastră
exemplu.
□
□
□
□
□
□
1 1

1 1
1 1
1 1
1 1
□
□
□
□
□
□
× [
1 1 1 1 1
1 1 1 1 1]
=
□
□
□
□
□
□
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
2 2 2 2 2
□
□
□
□
□
□
Figura 9.10: Matrici U și V cu toate intrările 1

Exemplul 9.12: Să presupunem că U și V ar trebui să aibă fiecare intrări care sunt toate 1, așa cum se arată în Fig. 9.10. Aceasta este o presupunere slabă, deoarece produsul, format din toate cele 2, are intrări care sunt mult sub media înregistrărilor în M. Cu toate acestea, putem calcula RMSE pentru acest U și V; de fapt regularitatea în intrări face calculul mai ușor de urmărit.

Luați în considerare primele rânduri de M și UV. Scădem 2 (fiecare intrare în UV) din intrările din primul rând al lui M, pentru a obține 3, 0, 2, 2, 1. Pătrăm și însumăm acestea pentru a obține 18. În al doilea rând, facem același lucru pentru a obține 1, −1, 0, 2, −1, pătrat și suma pentru a obține 7. În al treilea rând, a doua coloană este goală, astfel încât intrarea este ignorat la calculul RMSE. Diferențele sunt 0, 1, −1, 2 și suma de pătrate este 6. Pentru al patrulea rând, diferențele sunt 0, 3, 2, 1, 3 și suma de pătrate este 23. Al cincilea rând are o intrare goală în ultima coloană, deci diferențele sunt 2, 2, 3, 2 și suma pătratelor este 21. Când însumăm sumele din fiecare dintre cele cinci rânduri, obținem 18 + 7 + 6 + 23 + 21 = 75. În general, vom

---

Pagina 362

opriți-vă în acest moment, dar dacă vrem să calculăm adevăratul RMSE, îl împărțim la 23 (numărul de intrări nonblank în M) și luați rădăcina pătrată. În acest caz

$\sqrt{75 / 23} = 1{,}806$ este RMSE. ✷

9.4.3 Calculul incremental al unei descompuneri UV

Găsirea descompunerii UV cu cel mai mic RMSE implică începând cu unii au ales în mod arbitrar U și V și au ajustat în mod repetat U și V pentru a face RMSE mai mic. Vom lua în considerare doar ajustările aduse unui singur element de U sau V, deși, în principiu, s-ar putea face ajustări mai complexe.

Indiferent de ajustările pe care le permitem, într-un exemplu tipic vor exista multe minime cal - matrici U și V astfel încât nu se reduce nicio ajustare admisibilă RMSE. Din păcate, doar unul dintre aceste minime locale va fi global minim - matricile U și V care produc cel mai mic RMSE posibil. La

crește șansele de a găsi minimul global, trebuie să alegem multe
puncte de plecare ferente, adică diferite alegeri ale matricilor inițiale U și V.
Cu toate acestea, nu există niciodată o garanție că cel mai bun minim local al nostru va fi
minim global.
Vom începe cu U și V din Fig. 9.10, unde toate intrările sunt 1 și vom face
câteva ajustări la unele dintre intrări, găsind valorile acelor intrări care
oferi cea mai mare îmbunătățire posibilă RMSE. Din aceste exemple,
calculul general ar trebui să devină evident, dar vom urma exemplele
prin formula pentru minimizarea RMSE prin schimbarea unei singure intrări. In ce
urmează, ne vom referi la intrările lui U și V după numele lor variabile u 11 și
așa mai departe, așa cum este dat în Fig. 9.9.
Exemplul 9.13: Să presupunem că începem cu U și V ca în Fig. 9.10 și că decidem
a modifica u 11 pentru a reduce RMSE cât mai mult posibil. Lăsați valoarea u 11 BE
X. Apoi, noile U și V pot fi exprimate ca în Fig. 9.11.
□
□
□
□
□
□
x 1
1 1
1 1
1 1
1 1
□
□
□
□
□
□
× [
1 1 1 1 1
1 1 1 1 1]
=
□
□
□
□
□
□
x + 1 x + 1 x + 1 x + 1 x + 1
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
□

□
□
□
□
□
Figura 9.11: Făcând u 11 o variabilă
Observați că singurele intrări ale produsului care s-au modificat sunt cele din primul rând. Astfel, atunci când comparăm UV cu M, singura schimbare la RMSE vine din primul rând. Contribuția la suma pătratelor din primul rând este
(5 - (x + 1))
2
+ (2 - (x + 1))
2
+ (4 - (x + 1))
2
+ (4 - (x + 1))
2
+ (3 - (x + 1))
2

Această sumă simplifică la
$(4 - x)^2 + (1 - x)^2 + (3 - x)^2 + (3 - x)^2 + (2 - x)^2$
Vrem valoarea lui x care minimizează suma, deci luăm derivata și setați-l egal cu 0, ca:
$-2 \times ((4 - x) + (1 - x) + (3 - x) + (3 - x) + (2 - x)) = 0$
sau $-2 \times (13 - 5x) = 0$, din care rezultă că $x = 2.6$.
□
□
□
□
□
□
2.6 1
1
1
1
1
1
1
1
1
□
□
□
□
□
□
× [
1 1 1 1 1
1 1 1 1 1]
=
□
□
□
□
□
□
3,6 3,6 3,6 3,6 3,6

2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
□
□
□
□
□
□
Figura 9.12: Cea mai bună valoare pentru u 11 este 2.6
Figura 9.12 arată U și V după ce u 11 a fost setat la 2.6. Rețineți că suma din pătratele erorilor din primul rând a fost redus de la 18 la 5,2, deci RMSE total (ignorând rădăcina medie și pătrată) a fost redus de la 75 la 62,2.
□
□
□
□
□
□
2.6 1
1
1
1
1
1
1
1
□
□
□
□
□
□
× [
y 1 1 1 1
1 1 1 1 1]
=
□
□
□
□
□
□

2.6y + 1 3.6 3.6 3.6 3.6
y + 1
2
2
2
2
y + 1
2
2
2
2
y + 1
2
2
2
2
y + 1
2
2
2
2
□
□
□
□
□
□
Figura 9.13: v 11 devine o variabilă y
Să presupunem că următoarea noastră intrare care va varia este v 11 . Fie ca valoarea acestei intrări să fie y, ca sugerat în Fig. 9.13. Doar prima coloană a produsului este afectată de y, deci
trebuie doar să calculăm suma pătratelor diferențelor dintre
intrări în primele coloane M și UV. Această sumă este
(5− (2,6y + 1))
2
+ (3− (y + 1))
2
+ (2− (y + 1))
2
+ (2− (y + 1))
2
+ (4− (y + 1))
2
Această expresie simplifică la
$(4 - 2.6y)^2 + (2 - y)^2 + (1 - y)^2 + (1 - y)^2 + (3 - y)^2$
Ca și înainte, găsim valoarea minimă a acestei expresii prin diferențierea și
echivalent cu 0, ca:
$-2 \times (2,6\,(4 - 2,6y) + (2 - y) + (1 - y) + (1 - y) + (3 - y)) = 0$

Pagina 364

□
□
□
□
□
□
2.6 1
1
1
1
1
1
1
1

1
□
□
□
□
□
□
× [
1.617 1 1 1 1
1
1 1 1 1]
=
□
□
□
□
□
□
5.204 3.6 3.6 3.6 3.6
2.617 2
2
2
2
2.617 2
2
2
2
2.617 2
2
2
2
2.617 2
2
2
2
□
□
□
□
□
□
Figura 9.14: Înlocuiți y cu 1.617
Soluția pentru y este y = 17,4 / 10,76 = 1,617. Estimările îmbunătățite ale U și V sunt prezentate în Fig. 9.14.
Vom mai face o schimbare, pentru a ilustra ce se întâmplă când intrările lui M sunt goale. Vom varia u 31 , numindu-l z temporar. Noile U și V sunt prezentat în Fig. 9.15. Valoarea z afectează numai intrările din al treilea rând.
□
□
□
□
□
□
2.6 1
1
1
z
1
1
1
1
1

□
□
□
□
□
□
× [
1.617 1 1 1 1
1
1 1 1 1]
=
□
□
□
□
□
□
5.204
3.6
3.6
3.6
3.6
2.617
2
2
2
2
1.617z + 1 z + 1 z + 1 z + 1 z + 1
2.617
2
2
2
2
2.617
2
2
2
2
□
□
□
□
□
□
Figura 9.15: u 31 devine o variabilă z
Putem exprima suma pătratelor erorilor ca
(2 - (1.617z + 1))
2
+ (3 - (z + 1))
2
+ (1 - (z + 1))
2
+ (4 - (z + 1))
2
Reținați că nu există nicio contribuție din elementul din a doua coloană a
al treilea rând, deoarece acest element este gol în M. Expresia se simplifică la
(1 - 1.617z) 2 + (2 - z) 2 + (−z) 2 + (3 - z) 2
Procesul obișnuit de setare a derivatei la 0 ne oferă
−2 × (1.617 (1 - 1.617z) + (2 - z) + (- z) + (3 - z)) = 0
a cărei soluție este z = 6.617 / 5.615 = 1.178. Următoarea estimare a descompunerii
secțiunea UV este prezentată în Fig. 9.16. ✷

9.4.4 Optimizarea unui element arbitrar

După ce am văzut câteva exemple de alegere a valorii optime pentru un singur element în
matricea U sau V, să dezvoltăm acum formula generală. Ca și înainte, presupuneți

□
□
□
□
□
□
2.6
1
1
1
1.178 1
1
1
1
1
□
□
□
□
□
□
× [
1.617 1 1 1 1
1
1 1 1 1]
=
□
□
□
□
□
□
5.204 3.6
3.6
3.6
3.6
2.617 2
2
2
2
2.905 2.178 2.178 2.178 2.178
2.617 2
2
2
2
2.617 2
2
2
2
□
□
□
□
□
□
Figura 9.16: Înlocuiți z cu 1.178
că M este o matrice de utilitate n-by-m cu unele intrări necompletate, în timp ce U și V sunt matrici de dimensiuni n-cu-d și d-cu-m, pentru unele d. Vom folosi m_{ij}, u_{ij} și v_{ij} pentru intrările din rândul i și respectiv coloana j din M, U și respectiv V. De asemenea, lăsați P = UV și utilizați p_{ij} pentru elementul din rândul i și coloana j din matricea produsului P.

Să presupunem că vrem să variați u_{rs} și să găsim valoarea acestui element care mini-
mărește RMSE între M și UV. Rețineți că u_{rs} afectează numai elementele
în rândul r al produsului P = UV. Astfel, trebuie doar să ne preocupăm
elementele

$$p_{rj} = \sum_{k=1}^{d} u_{rk} v_{kj} = \sum_{k=s} u_{rk} v_{kj} + xv_{sj}$$

pentru toate valorile lui j astfel încât m_{rj} este nonblank. În expresia de mai sus, avem
a înlocuit u_{rs} , elementul pe care dorim să îl modificăm, cu o variabilă x, și folosim
convenție

• $\sum_{k=s}$ este o prescurtare pentru suma pentru k = 1, 2, ..., d, cu excepția lui k = s.

Dacă m_{rj} este o intrare non-albă a matricei M, atunci contribuția acesteia
element la suma pătratelor erorilor este

$$(m_{rj} - p_{rj})^2 = (m_{rj} - \sum_{k=s} u_{rk} v_{kj} - xv_{sj})^2$$

Vom folosi o altă convenție:

• $\sum_j$ este prescurtat pentru suma peste toate j astfel încât m_{rj} este nonblank.

Apoi putem scrie suma pătratelor erorilor care sunt afectate de
valoarea lui $x = u_{rs}$ ca

$$\sum_j (m_{rj} - \sum_{k=s} u_{rk} v_{kj} - xv_{sj})^2$$

Luați derivata celor de mai sus cu privire la x și setați-o egală cu 0, în
pentru a găsi valoarea lui x care minimizează RMSE. Acesta este,

$$\sum_j -2v_{sj} (m_{rj} - \sum_{k=s} u_{rk} v_{kj} - xv_{sj}) = 0$$

Ca și în exemplele anterioare, factorul comun -2 poate fi eliminat. Rezolvăm
ecuația de mai sus pentru x și obțineți

$$x = \frac{\sum_j v_{sj} (m_{rj} - \sum_{k=s} u_{rk} v_{kj})}{\sum_j v_{sj}^2}$$

Există o formulă analogă pentru valoarea optimă a unui element de V. Dacă
vrem să variem $v_{rs} = y$, atunci valoarea lui y care minimizează RMSE este

$$y = \frac{\sum_i u_{ir} (m_{este} - \sum_{k=r} u_{ik} v_{ks})}{\sum_i u_{ir}^2}$$

Aici, $\sum_i$ este o prescurtare pentru suma peste tot i astfel încât m_{este} este neblan și
$\sum_{k=r}$ este suma peste toate valorile lui k între 1 și d, cu excepția lui k = r.

9.4.5 Construirea unui algoritm complet de descompunere UV

Acum, avem instrumentele pentru a căuta descompunerea optimă globală a
matrice de utilitate M. Există patru domenii în care vom discuta despre opțiuni.

1. Preprocesarea matricei M.
2. Inițializarea U și V.
3. Ordonarea optimizării elementelor lui U și V.
4. Încheierea încercării de optimizare.

Preprocesare

Deoarece diferențele în ceea ce privește calitatea articolelor și scalele de evaluare ale utilizatorilor sunt
factori atât de importanți în determinarea elementelor lipsă ale matricei M, ea
este adesea util pentru a elimina aceste influențe înainte de a face orice altceva. Ideea
a fost introdus în secțiunea 9.3.1. Putem scădea din fiecare element non-alb
m_{ij} nota medie a utilizatorului i. Apoi, matricea rezultată poate fi modificată
prin scăderea ratingului mediu (în matricea modificată) a articolului j. Este, de asemenea
este posibil să se scadă mai întâi ratingul mediu al articolului j și apoi să se scadă

evaluarea medie a utilizatorului i în matricea modificată. Rezultatele obținute de la
a face lucruri în aceste două ordine diferite nu trebuie să fie la fel, ci va avea tendința
a fi aproape. O a treia opțiune este de a normaliza scăzând din m_{ij} media
din ratingul mediu al utilizatorului i și al articolului j, adică scăzând jumătate din sumă
din media utilizatorului și media articolului.
Dacă alegem să normalizăm M, atunci când facem predicții, trebuie
anulați normalizarea. Adică, dacă orice metodă de predicție folosim rezultă
în estimarea e pentru un element m_{ij} al matricei normalizate, atunci valoarea
prezicem pentru m_{ij} în adevărata matrice de utilitate este e plus orice sumă a fost
scăzută din rândul i și din coloana j în timpul procesului de normalizare.

Pagina 367

Inițializare
Așa cum am menționat, este esențial să existe o anumită întâmplare în felul în care noi
căutați o soluție optimă, deoarece existența multor minime locale justifică
rulăm multe optimizări diferite în speranța de a ajunge la global
minim pe cel puțin o alergare. Putem varia valorile inițiale ale lui U și V, sau noi
poate varia modul în care căutăm optimul (care va fi discutat în continuare) sau ambele.
Un punct de plecare simplu pentru U și V este de a da fiecărui element aceeași valoare,
iar o alegere bună pentru această valoare este cea care oferă elementele produsului
UV media elementelor neblane ale lui M. Rețineți că dacă ne-am normalizat
M, atunci această valoare va fi neapărat 0. Dacă am ales d ca lungimi ale lui
laturile scurte ale lui U și V, iar a este elementul mediu neblan al lui M, atunci
elementele lui U și V ar trebui să fie $\sqrt{a/d}$.
Dacă vrem multe puncte de pornire pentru U și V, atunci putem perturba
valoare $\sqrt{a/d}$ aleatoriu și independent pentru fiecare dintre elemente. Sunt
multe opțiuni pentru modul în care facem perturbarea. Avem de ales cu privire la
distribuția diferenței. De exemplu, am putea adăuga la fiecare element a
valoare distribuită în mod normal cu media 0 și o anumită abatere standard aleasă.
Sau am putea adăuga o valoare aleasă în mod uniform din intervalul −c până la + c pentru unele c.
Efectuarea optimizării
Pentru a atinge un minim local dintr-o valoare inițială dată de U și V, noi
trebuie să alegem o comandă în care vizităm elementele lui U și V. Cel mai simplu
lucru de făcut este să alegeți o comandă, de exemplu, rând cu rând, pentru elementele lui U și V,
și vizitați-le în mod rotund. Rețineți că doar pentru că am optimizat un
element o dată nu înseamnă că nu putem găsi o valoare mai bună pentru acel element după
alte elemente au fost ajustate. Astfel, trebuie să vizităm elemente în mod repetat,
până când nu avem motive să credem că nu sunt posibile alte îmbunătățiri.
Alternativ, putem urmări multe căi de optimizare diferite dintr-o singură
valoarea de pornire prin alegerea aleatorie a elementului pentru optimizare. Pentru a vă asigura că
fiecare element este luat în considerare în fiecare rundă, am putea alege în schimb o permuta-
elementelor și urmați acea ordine pentru fiecare rundă.
Convergerea la un minim
În mod ideal, la un moment dat RMSE devine 0 și știm că nu putem face mai bine.
În practică, din moment ce există în mod normal mai multe elemente neblane în M decât
există elemente în U și V împreună, nu avem dreptul să ne așteptăm ca noi
poate reduce RMSE la 0. Astfel, trebuie să detectăm când există un beneficiu mic
de revăzut în elementele lui U și / sau V. Putem urmări cantitatea de
îmbunătățirea RMSE obținută într-o rundă de optimizare și oprire
atunci când acea îmbunătățire scade sub un prag. O mică variație este de observat
îmbunătățirile rezultate din optimizarea elementelor individuale și
opriți atunci când îmbunătățirea maximă în timpul unei runde este sub un prag.

Pagina 368

Coborâre în gradient
Tehnica pentru găsirea unei descompuneri UV discutată în secțiunea 9.4
este un exemplu de coborâre în gradient. Ni se oferă câteva puncte de date -
elemente neblane ale matricei M - și pentru fiecare punct de date găsim
direcția de schimbare care scade cel mai mult funcția de eroare: RMSE
între produsul UV actual și M. Vom avea mult mai mult de făcut
spuneți despre coborârea în gradient în secțiunea 12.3.4. De asemenea, merită menționat acest lucru
în timp ce am descris metoda ca vizitând fiecare punct nonblank al

M de mai multe ori până când abordăm o descompunere cu erori minime, adică
poate fi prea mult lucru pe o matrice mare M. Astfel, o alternativă
abordarea ne face să analizăm doar o fracțiune aleasă aleatoriu a datelor atunci când
căutând să minimalizeze eroarea. Această abordare, numită gradient stochastic
descendența este discutată în secțiunea 12.3.5.
Evitarea supra-dotării
O problemă care apare adesea atunci când efectuăm o descompunere UV este că noi
ajunge la una dintre numeroasele minime locale care se conformează bine datelor date, dar
preia valori din date care nu reflectă bine procesul de bază care
dă naștere la date. Asta este, deși RMSE poate fi mic în ceea ce privește datele
date, nu face bine prezicerea datelor viitoare. Există mai multe lucruri care pot
pentru a face față acestei probleme, care este numită suprasolicitare de către statistici.
1. Evitați să favorizați primele componente care urmează să fie optimizate doar mutând
valoarea unei componente la o fracțiune din drum, să spunem la jumătatea drumului, față de curentul său
valoare spre valoarea sa optimizată.
2. Nu mai revizitați elementele lui U și V cu mult înainte de a converge procesul.
3. Luați mai multe descompuneri UV diferite, și atunci când prezicați un nou
intrare în matricea M, luați media rezultatelor utilizării fiecăruia
descompunere.

9.4.6 Exerciții pentru secțiunea 9.4

Exercițiul 9.4.1: Începând cu descompunerea din Fig. 9.10, putem alege
oricare dintre cele 20 de intrări în U sau V pentru a optimiza mai întâi. Efectuați această primă optimizare
pas presupunând că alegem: (a) u_{32} (b) v_{41} .
Exercițiul 9.4.2: Dacă dorim să începem, ca în Fig. 9.10, cu toate U și V
intrări setate la aceeași valoare, ce valoare minimizează RMSE pentru matrice
M din exemplul nostru de alergare?

Exercițiul 9.4.3: Începând cu matricile U și V din Fig. 9.16, efectuați
urmând în ordine:
(a) Reconsiderați valoarea lui u_{11} . Găsiți noua sa cea mai bună valoare, având în vedere schimbările
care s-au făcut până acum.
(b) Apoi alegeți cea mai bună valoare pentru u_{52} .
(c) Apoi alegeți cea mai bună valoare pentru v_{22} .
Exercițiul 9.4.4: Derivați formula pentru y (valoarea optimă a elementului v_{rs}
prezentat la sfârșitul secțiunii 9.4.4.
Exercițiul 9.4.5: Normalizați matricea M a exemplului nostru de execuție prin:
(a) Mai întâi scădem din fiecare element media rândului său și apoi
scăzând din fiecare element media coloanei sale (modificate).
(b) Mai întâi scădem din fiecare element media coloanei sale și apoi
scăzând din fiecare element media rândului său (modificat).
Există diferențe în rezultatele (a) și (b)?

9.5 Provocarea Netflix

Un impuls semnificativ pentru cercetarea sistemelor de recomandare a fost dat când
Netflix a oferit un premiu de 1.000.000 de dolari primei persoane sau echipei care le-a învins
algoritm de recomandare propriu, numit CineMatch, cu 10%. După peste trei
ani de muncă, premiul a fost acordat în septembrie 2009.
Provocarea Netflix a constat dintr-un set de date publicat, oferind evaluările de către
aproximativ jumătate de milion de utilizatori pe (de obicei mici subseturi de) aproximativ
17.000 de filme. Aceste date au fost selectate dintr-un set de date mai mare și au fost propuse
goritmii au fost testați cu privire la capacitatea lor de a prezice evaluările într-un rest secret
din setul de date mai mare. Informațiile pentru fiecare pereche (utilizator, film) din publicitate
Setul de date dorit a inclus o evaluare (1-5 stele) și data la care a fost evaluat
a fost facut.
RMSE a fost utilizat pentru a măsura performanța algoritmilor. CineMatch
are un RMSE de aproximativ 0,95; adică, evaluarea tipică ar fi dezactivată de
aproape o stea plină. Pentru a câștiga premiul, a fost necesar ca algoritmul dvs.
au un RMSE care a reprezentat cel mult 90% din RMSE al CineMatch.
Notele bibliografice pentru acest capitol includ referințe la descrieri
a algoritmilor câștigători. Aici, menționăm unele interesante și poate
fapte neintuitive despre provocare.

- CineMatch nu a fost un algoritm foarte bun. De fapt, a fost descoperit devreme
că algoritmul evident al prezicerii, pentru evaluarea de către utilizator u pe film
m, media de:

---

1. Evaluarea medie acordată de u pentru toate filmele clasificate și
2. Media evaluărilor pentru filmul m de către toți utilizatorii care au evaluat acest lucru film.

a fost cu doar 3% mai rău decât CineMatch.

• Algoritmul de descompunere UV descris în secțiunea 9.4 a fost găsit de trei studenți (Michael Harris, Jeffrey Wang și David Kamm) să dea o îmbunătățire cu 7% față de CineMatch, atunci când este asociat cu normalizarea și alte câteva trucuri.

• Intrarea câștigătoare a fost de fapt o combinație a mai multor ritmuri care fuseseră dezvoltate independent. O a doua echipă, care a trimis o înregistrare care ar fi câștigat, dacă ar fi fost trimisă câteva minute mai devreme, a existat, de asemenea, un amestec de algoritmi independenți. Această strategie egy - combinând diferiți algoritmi - a fost folosit anterior într-un număr de probleme grele și este ceva demn de reținut.

• Au fost făcute mai multe încercări de a utiliza datele conținute în IMDB, Baza de date de filme pe internet, pentru a se potrivi cu numele filmelor de pe Netflix provocare cu numele lor în IMDB și, astfel, extrageți informații utile nu sunt conținute în datele Netflix în sine. IMDB are informații despre actori și regizori și clasifică filmele într-unul sau mai multe din cele 28 de genuri. S-a constatat că genul și alte informații nu erau utile. O poziție motivul considerabil este că algoritmii de învățare automată au reușit să descopere informațiile relevante oricum și o a doua este rezoluția entității problema potrivirii numelor de filme, așa cum este dat în datele Netflix și IMDB este nu atât de ușor de rezolvat exact.

• Timpul de evaluare sa dovedit a fi util. Se pare că există filme care sunt mai susceptibile de a fi apreciate de persoanele care o evaluează imediat după vizionare decât de către cei care așteaptă un timp și apoi o evaluează. „Patch Adams” a fost dat ca exemplu al unui astfel de film. În schimb, există și alte filme care nu au fost apreciate de cei care l-au evaluat imediat, dar au fost mai bine apreciat după un timp; „Memento” a fost citat ca exemplu. Deși nu se poate scoate din date informații despre cât a fost întârzierea dintre vizionare și evaluare, este în general sigur să presupunem că majoritatea oamenilor văd un film la scurt timp după ce a ieșit. Astfel, se poate examina evaluările oricărui film pentru a vedea dacă evaluările sale au o creștere sau o scădere panta cu timpul.

9.6 Rezumatul capitolului 9

♦ Matrici de utilități: sistemele de recomandare se ocupă de utilizatori și articole. O matrice utilitară oferă informații cunoscute despre gradul în care a utilizatorului îi place un articol. În mod normal, cele mai multe intrări sunt necunoscute și esențiale

---

problema recomandării utilizatorilor de articole este prezicerea valorilor intrări necunoscute pe baza valorilor intrărilor cunoscute.

♦ Două clase de sisteme de recomandare: Aceste sisteme încearcă să predictează răspunsul unui utilizator la un articol descoperind articole similare și răspunsul utilizatorului la acestea. O clasă de sistem de recomandare este bazat pe conținut; măsoară similitudinea căutând trăsături comune ale obiectele. O a doua clasă de sistem de recomandare folosește filtre colaborative tering; acestea măsoară similitudinea utilizatorilor în funcție de preferințele articolului și / sau măsurați similitudinea articolelor de către utilizatorii cărora le plac.

♦ Profiluri de articole: Acestea constau în caracteristici ale articolelor. Diferite tipuri de articole au caracteristici diferite pe care se poate baza similitudinea bazată pe conținut. Caracteristicile documentelor sunt de obicei cuvinte importante sau neobișnuite. Prod-produsele au atribute precum dimensiunea ecranului pentru un televizor. Medii precum filmele au un gen și detalii precum actor sau interpret. Etichetele pot, de asemenea să fie utilizate ca funcții dacă pot fi achiziționate de la utilizatorii interesați.

♦ Profiluri de utilizator: un sistem de filtrare colaborativă bazat pe conținut poate con-profiluri struct pentru utilizatori prin măsurarea frecvenței cu care caracteristicile apar în elementele care îi plac utilizatorului. Putem apoi estima gradul la

pe care un utilizator îi va plăcea un articol prin apropierea profilului articolului de
profilul utilizatorului.

♦ Clasificarea articolelor: o alternativă la construirea unui profil de utilizator este să
construiți un clasificator pentru fiecare utilizator, de exemplu, un arbore de decizie. Rândul utilitarului
matricea pentru acel utilizator devine datele de instruire, iar clasificatorul trebuie
prezice răspunsul utilizatorului la toate articolele, indiferent dacă rândul avea sau nu
o intrare pentru acel element.

♦ Similitudinea rândurilor și coloanelor matricei de utilități: filtru colaborativ
algoritmii de verificare trebuie să măsoare similaritatea rândurilor și / sau coloanelor
a matricei utilitare. Distanța Jaccard este adecvată atunci când matricea
constă doar din 1 și spații goale (pentru „fără rating”). Cosinusul la distanță funcționează
pentru valori mai generale în matricea utilitară. De multe ori este util să
ize matricea utilitară scăzând valoarea medie (fie prin rând, cu
coloană sau ambele) înainte de a măsura distanța cosinusului.

♦ Clusterizarea utilizatorilor și a elementelor: întrucât matricea utilitară tinde să fie în cea mai mare parte
spațiile libere, măsurile de distanță, cum ar fi Jaccard sau cosinusul au adesea prea puțin
date cu care să comparați două rânduri sau două coloane. Un preliminar
pas sau pași, în care similitudinea este utilizată pentru a grupa utilizatorii și / sau elementele
în grupuri mici cu asemănare puternică, poate ajuta la furnizarea mai comună
componente cu care să compare rânduri sau coloane.

♦ Descompunere UV: o modalitate de a prezice valorile necompletate într-un utilitar
matricea este de a găsi două matrice lungi și subțiri U și V, al căror produs este un
aproximare la matricea de utilitate dată. Deoarece produsul matrice UV

---

oferă valori pentru toate perechile utilizator-articol, această valoare poate fi utilizată pentru a prezice
valoarea unui gol în matricea utilitară. Motivul intuitiv al acestei metode
are sens este că adesea există un număr relativ mic de probleme (care
numărul este dimensiunea „subțire” a lui U și V) care determină dacă sau
unui utilizator nu îi place un articol.

♦ Eroare Root-Mean-Square: O măsură bună a cât de aproape de produs este UV
este la matricea de utilitate dată este RMSE (eroare rădăcină-medie-pătrat). The
RMSE se calculează prin media pătratului diferențelor dintre
UV și matricea utilitară, în acele elemente în care se află matricea utilitară
nonblank. Rădăcina pătrată a acestei medii este RMSE.

♦ Calculul U și V: o modalitate de a găsi o alegere bună pentru U și V într-un
Descompunerea UV trebuie să înceapă cu matrici arbitrare U și V. Repeta-
reglați unul dintre elementele lui U sau V pentru a minimiza RMSE între
produsul UV și matricea de utilitate dată. Procesul converge către
un optim local, deși să ai șanse mari să obții un global
optim trebuie fie să repetăm procesul din mai multe matrice de pornire,
sau căutați din punctul de plecare în multe moduri diferite.

♦ Provocarea Netflix: un factor important de cercetare a recomandărilor-
sisteme de acțiune a fost provocarea Netflix. A fost oferit un premiu de 1.000.000 de dolari
pentru un concurent care ar putea produce un algoritm cu 10% mai bun
decât algoritmul propriu al Netflix de a prezice evaluările filmelor de către utilizatori. The
premiul a fost acordat în septembrie 2009.

9.7 Referințe pentru capitolul 9

[1] este un sondaj al sistemelor de recomandare începând cu 2005. Argumentul referitor la
importanța cozii lungi în sistemele on-line este de la [2], care a fost
extins într-o carte [3].
[8] discută despre utilizarea jocurilor pe computer pentru a extrage etichete pentru articole.
Consultați [5] pentru o discuție despre similaritatea articol-articol și despre modul în care Amazon a proiectat
algoritmul său de filtrare colaborativă pentru recomandări de produse.
Există trei lucrări care descriu cei trei algoritmi care, în combinație,
a câștigat provocarea NetFlix. Ele sunt [4], [6] și [7].
1. G. Adomavicius și A. Tuzhilin, „Către următoarea generație de
sisteme de comendatori: o cercetare a stadiului tehnicii și a posibilelor extinderi

,,IEEE Trans. privind ingineria datelor și cunoașterii 17: 6, pp. 734–
749, 2005.
2. C. Anderson,
http://www.wired.com/wired/archive/12.10/tail.html

9.7. REFERINȚE PENTRU CAPITOLUL 9
353
2004.
3. C. Anderson, Coada lungă: De ce viitorul afacerilor se vinde mai puțin
of More, Hyperion Books, New York, 2006.
4. Y. Koren, „Soluția BellKor la marele premiu Netflix”
www.netflixprize.com/assets/GrandPrize2009 BPC BellKor.pdf
2009.
5. G. Linden, B. Smith și J. York, „Recomandări Amazon.com: item-
filtrare colaborativă la articol, ”Internet Computing 7: 1, pp. 76-80, 2003.
6. M. Piotte și M. Chabbert, „The Pragmatic Theory solution to the Net-
flix marele premiu "
www.netflixprize.com/assets/
GrandPrize2009 BPC PragmaticTheory.pdf
2009.
7. A. Toscher, M. Jahrer și R. Bell, „Soluția BigChaos pentru Netflix
marele Premiu,"
www.netflixprize.com/assets/GrandPrize2009 BPC BigChaos.pdf
2009.
8. L. von Ahn, „Jocuri cu un scop”, IEEE Computer Magazine, pp. 96–
98, iunie 2006.

Capitolul 10
Rețea socială minieră
Grafice

Există multe informații de obținut prin analiza datelor pe scară largă care
este derivat din rețelele sociale. Cel mai cunoscut exemplu de rețea socială
este relația „prieteni” găsită pe site-uri precum Facebook. Totuși, așa cum vom vedea
există multe alte surse de date care conectează persoane sau alte entități.
În acest capitol, vom studia tehnicile de analiză a acestor rețele. Un
întrebarea importantă despre o rețea socială este cum să identificăm „comunitățile”,
adică subseturi de noduri (persoane sau alte entități care formează rețeaua)
cu conexiuni neobișnuit de puternice. Câteva dintre tehnicile utilizate pentru identificare
comunitățile sunt similare cu algoritmii de grupare pe care i-am discutat în capitolul 7.
Cu toate acestea, comunitățile aproape niciodată nu partiționează setul de noduri dintr-o rețea.
Mai degrabă, comunitățile se suprapun de obicei. De exemplu, puteți aparține mai multor persoane
comunități de prieteni sau colegi de clasă. Oamenii dintr-o comunitate tind să
se cunosc, dar oamenii din două comunități diferite rareori se cunosc
alte. Nu ați dori să fiți desemnat doar unei comunități și nici
ar avea sens să grupați toți oamenii din toate comunitățile dvs. în
un cluster.
De asemenea, în acest capitol explorăm algoritmi eficienți pentru a descoperi altele
proprietățile graficelor. Ne uităm la „simrank”, o modalitate de a descoperi similitudini
printre nodurile unui grafic. Una dintre aplicațiile interesante ale simrank este
că ne oferă o modalitate de a identifica comunitățile ca seturi de noduri „similare”. Noi
apoi explorează numărarea triunghiului ca o modalitate de a măsura conectivitatea unui
comunitate. În plus, oferim algoritmi eficienți pentru exact și aproximativ
măsurarea dimensiunilor vecinătății nodurilor într-un grafic și ne uităm la ele
algoritmi eficienți pentru calculul închiderii tranzitive.
355

CAPITOLUL 10. GRAFII SOCIALE-REȚELE MINIERE

10.1 Rețelele sociale ca grafice

Începem discuția noastră despre rețelele sociale prin introducerea unui model grafic. Nu fiecare grafic este o reprezentare adecvată a ceea ce considerăm intuitiv ca social rețea. Prin urmare, discutăm ideea de „localitate”, proprietatea socială rețelele care spun că nodurile și marginile graficului tind să se grupeze în comunități. Această secțiune analizează, de asemenea, unele dintre tipurile de rețele sociale care apar în practică.

10.1.1 Ce este o rețea socială?

Când ne gândim la o rețea socială, ne gândim la Facebook, Twitter, Google+, sau un alt site web numit „rețea socială” și, într-adevăr, acest gen de rețeaua este reprezentativă pentru clasa mai largă de rețele numite „sociale”. The caracteristicile esențiale ale unei rețele sociale sunt:

1. Există o colecție de entități care participă la rețea. De obicei, aceste entități sunt oameni, dar ar putea fi cu totul altceva. Noi va discuta câteva alte exemple în secțiunea 10.1.3.
2. Există cel puțin o relație între entitățile din rețea. Pe Facebook sau așa ceva, această relație se numește prieteni. Uneori relația este totul sau nimic; doi oameni sunt fie prieteni, fie sunt nu. Cu toate acestea, în alte exemple de rețele sociale, relația are un grad. Acest grad ar putea fi discret; de exemplu, prieteni, familie, cunoscuți, sau niciunul ca în Google+. Ar putea fi un număr real; un exemplu ar fi fie fracțiunea din ziua medie pe care doi oameni o petrec vorbind cu fiecare alte.
3. Există o presupunere de non-aleatoriu sau de localitate. Această condiție este cel mai greu de formalizat, dar intuiția este că relațiile tind grup. Adică, dacă entitatea A este legată atât de B, cât și de C, atunci există o probabilitate mai mare decât media ca B și C să fie legate.

10.1.2 Rețelele sociale ca grafice

Rețelele sociale sunt modelate în mod natural ca grafice, la care uneori ne referim ca grafic social. Entitățile sunt nodurile, iar o margine conectează două noduri dacă nodurile sunt legate de relația care caracterizează rețeaua. Dacă există un grad asociat cu relația, acest grad este reprezentat de etichetarea marginilor. Adesea, graficele sociale nu sunt direcționate, ca și pentru Facebook graficul prietenilor. Dar pot fi grafice direcționate, ca de exemplu graficele lui urmăritori pe Twitter sau Google+.

Exemplul 10.1: Figura 10.1 este un exemplu de mică rețea socială. The entitățile sunt nodurile de la A la G. Relația, la care ne-am putea gândi

---

A

B

D

E

G

F

C

Figura 10.1: Exemplu de mică rețea socială

ca „prieteni”, este reprezentat de margini. De exemplu, B este prieten cu A, C, și D.

Este acest grafic cu adevărat tipic unei rețele sociale, în sensul că prezintă localitatea relațiilor? În primul rând, rețineți că graficul are nouă margini din $\binom{7}{2} = 21$ de perechi de noduri care ar fi putut avea o margine între ele. Presupune X, Y și Z sunt noduri din Fig. 10.1, cu margini între X și Y și, de asemenea între X și Z. La ce ne-am aștepta probabilitatea unei margini între Y și Z să fie? Dacă graficul ar fi mare, această probabilitate ar fi foarte apropiată la fracțiunea perechilor de noduri care au margini între ele, adică 9/21 = .429 în acest caz. Cu toate acestea, deoarece graficul este mic, există un aspect vizibil diferența dintre probabilitatea reală și raportul dintre numărul de muchii și numărul de perechi de noduri. Deoarece știm deja că există margini (X, Y) și (X, Z), mai sunt doar șapte margini. Cele șapte margini ar putea alerga între oricare dintre cele 19 perechi de noduri rămase. Astfel, probabilitatea unui muchia (Y, Z) este 7/19 = .368.

Acum, trebuie să calculăm probabilitatea ca muchia (Y, Z) să existe în Fig. 10.1, dat fiind faptul că muchiile (X, Y) și (X, Z) există. Ceea ce vom conta de fapt

este perechi de noduri care ar putea fi Y și Z, fără a vă face griji cu privire la ce nod
este Y și care este Z. Dacă X este A, atunci Y și Z trebuie să fie B și C, în unele
Ordin. Deoarece marginea (B, C) există, A contribuie cu un exemplu pozitiv (unde
marginea există) și nu există exemple negative (unde marginea este absentă). The
cazurile în care X este C, E sau G sunt în esență aceleași. În fiecare caz, X are numai
doi vecini, iar marginea dintre vecini există. Astfel, am văzut
patru exemple pozitive și zero exemple negative până acum.
Acum, ia în considerare X = F. F are trei vecini, D, E și G. Există margini
între două din cele trei perechi de vecini, dar nici o margine între G și E.
Astfel, vedem încă două exemple pozitive și vedem primul nostru exemplu negativ.
Dacă X = B, există din nou trei vecini, dar o singură pereche de vecini,
A și C, au o margine. Astfel, avem încă două exemple negative și unul
exemplu pozitiv, pentru un total de șapte pozitive și trei negative. În cele din urmă, când
X = D, sunt patru vecini. Din cele șase perechi de vecini, doar două au
marginile dintre ele.
Astfel, numărul total de exemple pozitive este nouă și numărul total
de exemple negative este șapte. Vedem că în Fig. 10.1, fracțiunea de ori

---

Pagina 378

a treia margine există este astfel 9/16 = .563. Această fracțiune este considerabil mai mare
decât valoarea așteptată .368 pentru acea fracțiune. Concluzionăm că Fig. 10.1
expune într-adevăr localitatea așteptată într-o rețea socială. ✷

10.1.3 Soiuri de rețele sociale

Există multe exemple de rețele sociale, altele decât rețelele de „prieteni”.
Aici, să enumerăm câteva dintre celelalte exemple de rețele care prezintă, de asemenea
localitatea relațiilor.

Rețele telefonice

Aici nodurile reprezintă numere de telefon, care sunt cu adevărat persoane fizice. Acolo
este o margine între două noduri dacă un apel a fost plasat între aceste telefoane
într-o perioadă determinată de timp, cum ar fi luna trecută sau „vreodată”. Marginile ar putea
să fie ponderat de numărul de apeluri efectuate între aceste telefoane în timpul
perioadă. Comunitățile dintr-o rețea telefonică se vor forma din grupuri de oameni
care comunică frecvent: grupuri de prieteni, membri ai unui club sau oameni
lucrează la aceeași companie, de exemplu.

Rețele de e-mail

Nodurile reprezintă adrese de e-mail, care sunt din nou individuale. O margine
reprezintă faptul că a existat cel puțin un e-mail în cel puțin o direcție
între cele două adrese. Alternativ, putem plasa o margine numai dacă există
erau e-mailuri în ambele direcții. În acest fel, evităm să privim spam-ul ca fiind
,,Prieteni” cu toate victimele lor. O altă abordare este de a eticheta marginile ca fiind slabe sau
puternic. Marginile puternice reprezintă comunicarea în ambele direcții, în timp ce slabe
marginile indică faptul că comunicarea a fost într-o singură direcție. Com-
Comunitățile văzute în rețelele de e-mail provin din aceleași tipuri de grupări
menționată în legătură cu rețelele telefonice. Un tip similar de rețea
implică oameni care trimit mesaje text altor persoane prin intermediul telefoanelor lor mobile.

Rețele de colaborare

Nodurile reprezintă persoane care au publicat lucrări de cercetare. Este o
marginea dintre două persoane care au publicat una sau mai multe lucrări în comun. Opțiune-
aliat, putem eticheta marginile după numărul de publicații comune. Comunitățile
în această rețea sunt autori care lucrează la un anumit subiect.
O vedere alternativă a aceloraași date este ca un grafic în care se află nodurile
hârtii. Două lucrări sunt conectate printr-o margine dacă au cel puțin un autor
in comun. Acum, formăm comunități care sunt colecții de lucrări pe
aceleași subiect.
Există mai multe alte tipuri de date care formează două rețele într-o similară
cale. De exemplu, ne putem uita la persoanele care editează articole Wikipedia și
articolele pe care le editează. Doi editori sunt conectați dacă au editat un

---

Pagina 379

articol în comun. Comunitățile sunt grupuri de editori care sunt interesați
în aceleași subiect. Dual, putem construi o rețea de articole și ne putem conecta
articole dacă au fost editate de aceeași persoană. Aici, obținem comunități

de articole pe subiecte similare sau conexe.
De fapt, datele implicate în filtrarea colaborativă, după cum sa discutat în
Capitolul 9, deseori poate fi privit ca formând o pereche de rețele, una pentru
clienți și unul pentru produse. Clienții care cumpără aceleași tipuri de
produse, de exemplu, cărți de știință-ficțiune, vor forma comunități și, în mod dual, vor produce
produsele cumpărate de aceiași clienți vor forma comunități, de exemplu, toate
cărți de știință-ficțiune.
Alte exemple de grafice sociale
Multe alte fenomene dau naștere la grafice care arată ca ceva social
grafice, în special localitatea care prezintă. Exemplele includ: rețelele de informații
(documente, grafice web, brevete), rețele de infrastructură (drumuri, avioane, apă
conducte, rețele electrice), rețele biologice (gene, proteine, rețele alimentare ale animalelor
mâncându-se reciproc), precum și alte tipuri, cum ar fi rețelele de co-cumpărare a produselor
(de exemplu, Groupon).

10.1.4 Grafice cu mai multe tipuri de noduri

Există și alte fenomene sociale care implică entități de diferite tipuri. Noi
tocmai discutat sub titlul „rețele de colaborare”, mai multe tipuri de
grafice care sunt într-adevăr formate din două tipuri de noduri. Rețele de autor
se poate vedea că are noduri de autor și noduri de hârtie. În discuția de mai sus, noi
a construit două rețele sociale prin eliminarea nodurilor unuia dintre cele două tipuri, dar
nu trebuie să facem asta. Ne putem gândi mai degrabă la structură ca un întreg.
Pentru un exemplu mai complex, utilizatorii de pe un site precum del.icio.us plasează etichete pe
Pagini web. Există astfel trei tipuri diferite de entități: utilizatori, etichete și
pagini. S-ar putea să credem că utilizatorii erau cumva conectați dacă aveau tendința
utilizați frecvent aceleași etichete sau dacă au avut tendința de a eticheta aceleași pagini. În mod similar,
etichetele ar putea fi considerate înrudite dacă au apărut pe aceleași pagini sau au fost
utilizate de aceiași utilizatori și paginile ar putea fi considerate similare dacă ar avea multe
din aceleași etichete sau au fost etichetate de mulți dintre aceiași utilizatori.
Modul natural de a reprezenta astfel de informații este ca un grafic k-partit pentru
unele k> 1. Am întâlnit grafice bipartite, cazul k = 2, în secțiunea 8.3. În
în general, un grafic k-partit este format din k seturi de noduri disjuncte, fără margini
între nodurile aceluiași set.
Exemplul 10.2: Figura 10.2 este un exemplu de grafic tripartit (cazul k = 3
a unui grafic k-partit). Există trei seturi de noduri, la care ne-am putea gândi
ca utilizatori $\{U_1, U_2\}$, etichete $\{T_1, T_2, T_3, T_4\}$ și pagini web $\{W_1, W_2, W_3\}$. Înștiințare
că toate marginile conectează noduri din două seturi diferite. Putem presupune acest grafic
reprezintă informații despre cele trei tipuri de entități. De exemplu, marginea
(U_1, T_2) înseamnă că utilizatorul U_1 a plasat eticheta T_2 pe cel puțin o pagină. Notă

U
U
T
T
T
T
W
W
W
1
1
1
2
2
2
3
3
4

Figura 10.2: Un grafic tripartit care reprezintă utilizatori, etichete și pagini web
că graficul nu ne spune un detaliu care ar putea fi important: cine a plasat
ce etichetă pe ce pagină? Pentru a reprezenta astfel de informații ternare ar necesita
o reprezentare mai complexă, cum ar fi o relație de bază de date cu trei coloane
corespunzător utilizatorilor, etichetelor și paginilor. ✷

10.1.5 Exerciții pentru secțiunea 10.1

Exercițiul 10.1.1: Este posibil să ne gândim la marginile unui grafic G ca la
nodurile unui alt grafic G'. Construim G' din G prin construcția duală:
1. Dacă (X, Y) este o margine a lui G, atunci XY, reprezentând setul neordonat al lui X
iar Y este un nod al lui G'. Rețineți că XY și YX reprezintă același nod
din G', nu două noduri diferite.

2. Dacă (X, Y) și (X, Z) sunt muchii ale lui G, atunci în G ′ există o margine între XY și XZ. Adică, nodurile lui G ′ au o margine între ele dacă muchiile lui G pe care le reprezintă aceste noduri au un nod (al lui G) în comun.
(a) Dacă aplicăm construcția duală unei rețele de prieteni, care este interpretarea marginilor graficului rezultat?
(b) Aplicați construcția duală la graficul din Fig. 10.1.
! (c) Cum este gradul unui nod XY în G ′ legat de gradele lui X și Y în G?
!! (d) Numărul muchiilor lui G ′ este legat de gradele nodurilor lui G de o anumită formulă. Descoperă acea formulă.

! (e) Ceea ce am numit dual nu este un dual adevărat, deoarece aplicarea construcția la G ′ nu dă neapărat un grafic izomorf lui G. Give un exemplu de grafic G unde dualul lui G ′ este izomorf pentru G și altul exemplu în care dualul lui G ′ nu este izomorf pentru G.

10.2 Clusterizarea graficelor din rețelele sociale

Un aspect important al rețelelor sociale este că acestea conțin comunități de entități care sunt conectate prin mai multe margini. Acestea corespund de obicei grupurilor de prieteni la școală sau grupuri de cercetători interesați de același subiect, pentru exemplu. În această secțiune, vom considera gruparea graficului ca o modalitate de a identificați comunitățile. Se pare că tehnicile pe care le-am învățat în capitolul 7 sunt, în general, nepotrivite pentru problema grupării graficelor rețelelor sociale.

10.2.1 Măsuri la distanță pentru graficele de rețea socială

Dacă ar fi să aplicăm tehnici standard de clusterizare unui grafic de rețea socială, primul nostru pas ar fi să definim o măsură de distanță. Când marginile grafice au etichete, aceste etichete ar putea fi utilizabile ca măsurătoare a distanței, în funcție de pe ceea ce au reprezentat. Dar când marginile sunt nemarcate, ca într-un „prieteni” grafic, nu putem face multe pentru a defini o distanță adecvată.

Primul nostru instinct este să presupunem că nodurile sunt apropiate dacă au o margine între ele și îndepărtate dacă nu. Astfel, am putea spune că distanța d (x, y) este 0 dacă există o margine (x, y) și 1 dacă nu există o astfel de margine. Am putea folosi orice alte două valori, cum ar fi 1 și ∞, atâta timp cât distanța este mai mică atunci când este acolo este o margine.

Niciuna dintre aceste „măsuri de distanță” cu două valori - 0 și 1 sau 1 și ∞- nu este o adevărată măsură a distanței. Motivul este că încalcă inegalitatea triunghiului când sunt trei noduri, cu două margini între ele. Adică, dacă există (A, B) și (B, C), dar fără margine (A, C), apoi distanța de la A la C depășește suma distanțelor de la A la B până la C. Am putea remedia această problemă folosind, să zicem, distanța 1 pentru o margine și distanța 1,5 pentru o margine lipsă. Dar problema cu funcțiile de distanță cu două valori nu se limitează la triunghi inegalitate, așa cum vom vedea în secțiunea următoare.

10.2.2 Aplicarea metodelor standard de clusterizare

Reamintim din secțiunea 7.1.2 că există două abordări generale ale clusterizării: ierarhic (aglomerativ) și de atribuire a punctelor. Să ne gândim cum fiecare dintre acestea ar funcționa pe un grafic de rețea socială. În primul rând, ia în considerare ierarhizarea metodele acoperite în secțiunea 7.2. În special, să presupunem că folosim ca intercluster distanța distanța minimă între nodurile celor două clustere.

Clusterul ierarhic al unui grafic de rețea socială începe prin combinarea unora două noduri care sunt conectate printr-o margine. Succesiv, margini care nu sunt între două noduri ale aceluiași cluster ar fi aleatoriu combinate

---

grupurile la care aparțin cele două noduri ale acestora. Alegerile ar fi aleatorii, deoarece toate distanțele reprezentate de o margine sunt aceleași.

Exemplul 10.3: Luați în considerare din nou graficul din Fig. 10.1, repetat aici ca Fig. 10.3. În primul rând, să ne punem de acord asupra a ceea ce sunt comunitățile. La cel mai înalt nivel, se pare că există două comunități {A, B, C} și {D, E, F, G}. Cum-oricând, am putea vedea și {D, E, F} și {D, F, G} ca două subcomunități ale {D, E, F, G}; aceste două subcomunități se suprapun în doi dintre membrii lor și astfel nu ar putea fi niciodată identificat printr-un algoritm de clusterizare pur. În cele din urmă, am putea

ia în considerare fiecare pereche de indivizi care sunt conectați de o margine ca o comunitate
de dimensiunea 2, deși astfel de comunități sunt neinteresante.
A
B
D
E
G
F
C
Figura 10.3: Repetați figura 10.1
Problema cu gruparea ierarhică a unui grafic precum cea din Fig. 10.3 este
că la un moment dat este probabil să alegem să combinăm B și D, chiar dacă
cu siguranță aparțin unor clustere diferite. Motivul pentru care este probabil să combinăm B
și D este acel D și orice cluster care îl conține, este la fel de aproape de B și de orice cluster
conținând-o, deoarece A și C sunt la B. Există chiar și o probabilitate de 1/9 ca
primul lucru pe care îl facem este să combinăm B și D într-un singur cluster.
Există lucruri pe care le putem face pentru a reduce probabilitatea de eroare. Putem
rulați clusterizarea ierarhică de mai multe ori și alegeți rularea care oferă cel mai mult
clustere coerente. Putem folosi o metodă mai sofisticată pentru măsurarea
distanța dintre grupurile de mai multe noduri, așa cum sa discutat în secțiunea 7.2.3.
Dar, indiferent de ceea ce facem, într-un grafic mare cu multe comunități există un
șansă semnificativă ca în fazele inițiale să folosim câteva margini care se conectează
două noduri care nu aparțin împreună în nicio comunitate mare. ✷
Acum, luați în considerare o abordare de atribuire a punctelor pentru gruparea rețelelor sociale.
Din nou, faptul că toate muchiile sunt la aceeași distanță va introduce un număr
a factorilor aleatori care vor duce la atribuirea unor greșeli unor noduri
grup. Un exemplu ar trebui să ilustreze punctul.
Exemplul 10.4: Să presupunem că încercăm o abordare k-means a grupării Fig. 10.3.
Deoarece dorim două clustere, alegem k = 2. Dacă alegem două noduri de pornire la întâmplare,
s-ar putea să fie amândoi în același grup. Dacă, așa cum este sugerat în secțiunea 7.3.2, noi
începeți cu un nod ales aleatoriu și apoi alegeți altul cât mai departe

---

posibil, nu ne descurcăm mult mai bine; am putea alege astfel orice pereche de noduri nu
conectate printr-o margine, de exemplu, E și G în Fig. 10.3.
Cu toate acestea, să presupunem că avem două noduri de pornire adecvate, cum ar fi B și F.
Vom atribui apoi A și C grupului de B și atribuim E și G către
grup de F. Dar D este la fel de aproape de B ca de F, deci ar putea merge în orice sens, chiar
deși este „evident” că D aparține lui F.
Dacă decizia cu privire la locul unde se plasează D este amânată până când ne-am atribuit
alte noduri către clustere, atunci probabil că vom lua decizia corectă.
De exemplu, dacă atribuim un nod clusterului cu cea mai mică distanță medie
tuturor nodurilor clusterului, atunci D ar trebui să fie atribuit clusterului F, așa cum
atâta timp cât nu încercăm să plasăm D înainte de atribuirea altor noduri. În orice caz,
în grafice mari, cu siguranță vom face greșeli pe unele dintre primele noduri pe care le avem
loc. ✷

10.2.3 Interval

Deoarece există probleme cu metodele standard de clusterizare, mai multe specializate
tehnici de clusterizare au fost dezvoltate pentru a găsi comunități în rețelele sociale
lucrări. În această secțiune vom considera una dintre cele mai simple, bazate pe constatare
marginile care sunt cel mai puțin probabil să se afle în interiorul unei comunități.
Definiți intervalul unei muchii (a, b) pentru a fi numărul de perechi de noduri
x și y astfel încât muchia (a, b) să se afle pe cea mai scurtă cale dintre x și y.
Pentru a fi mai precis, deoarece pot exista mai multe căi scurte între x și y,
muchia (a, b) este creditată cu fracția dintre cele mai scurte căi care includ
margine (a, b). La fel ca în golf, un scor mare este rău. Aceasta sugerează că marginea (a, b) rulează
între două comunități diferite; adică a și b nu aparțin aceluiași
comunitate.
Exemplul 10.5: În Fig. 10.3 muchia (B, D) are cea mai mare distanță, ca
nu ar trebui să surprindă pe nimeni. De fapt, această margine este pe fiecare cale cea mai scurtă dintre
oricare dintre A, B și C la oricare dintre D, E, F și G. Între acestea, deci este
3 × 4 = 12. În schimb, muchia (D, F) este pe doar cele mai scurte căi: cele
de la A, B, C și D la F. ✷

10.2.4 Algoritmul Girvan-Newman

Pentru a exploata între margini, trebuie să calculăm numărul de
cele mai scurte căi care trec prin fiecare margine. Vom descrie o metodă numită

Algoritmul Girvan-Newman (GN), care vizitează fiecare nod X o dată și calculează
numărul celor mai scurte căi de la X la fiecare dintre celelalte noduri care trec
fiecare dintre margini. Algoritmul începe prin efectuarea unei căutări întinse
(BFS) al graficului, începând de la nodul X. Rețineți că nivelul fiecărui nod din
prezentarea BFS este lungimea celei mai scurte căi de la X la acel nod.
Astfel, marginile care merg între noduri la același nivel nu pot face parte niciodată
o cale mai scurtă de la X.

Pagina 384

Marginile dintre niveluri se numesc margini DAG („DAG” înseamnă direcționat,
grafic aciclic).
Fiecare margine DAG va face parte din cel puțin o cale mai scurtă
din rădăcina X. Dacă există o margine DAG (Y, Z), unde Y este la nivelul de deasupra Z
(adică mai aproape de rădăcină), atunci vom numi Y un părinte al lui Z și Z un copil al lui Y,
deși părinții nu sunt neapărat unici într-un DAG așa cum ar fi într-un
copac.
E
D
F
B
G
A
C
1
1
2
1
1
1
1
Nivelul 1
Nivelul 2
Nivelul 3
Figura 10.4: Pasul 1 al algoritmului Girvan-Newman
Exemplul 10.6: Figura 10.4 este o prezentare largă a graficului din Fig.
10.3, începând cu nodul E. Marginile solide sunt marginile DAG și marginile punctate se conectează
noduri la același nivel. ✷
Al doilea pas al algoritmului GN este de a eticheta fiecare nod cu numărul de
cele mai scurte căi care o ating de la rădăcină. Începeți prin etichetarea rădăcinii 1. Apoi,
de sus în jos, etichetați fiecare nod Y prin suma etichetelor părinților săi.
Exemplul 10.7: În Fig. 10.4 sunt etichetele pentru fiecare dintre noduri. Mai întâi, etichetați
rădăcina E cu 1. La nivelul 1 sunt nodurile D și F. Fiecare are doar E ca a
părinte, deci și ei sunt etichetați 1. Nodurile B și G sunt la nivelul 2. B are numai
D ca părinte, deci eticheta lui B este aceeași cu eticheta lui D, care este 1. Cu toate acestea,
G are părinții D și F, deci eticheta sa este suma etichetelor lor, sau 2. În cele din urmă, la
nivelul 3, A și C au fiecare părinte B, deci etichetele lor sunt eticheta lui B,
care este 1. ✷
Al treilea și ultimul pas este de a calcula pentru fiecare muchie e suma peste toate nodurile
Y din fracțiunea celor mai scurte căi de la rădăcina X la Y care trec prin e.
Acest calcul implică calcularea acestei sume atât pentru noduri cât și pentru margini, de la
fundul. Fiecare nod, altul decât rădăcina, primește un credit de 1, reprezentând
cea mai scurtă cale către acel nod. Acest credit poate fi împărțit între noduri și

---

Pagina 385

marginile de deasupra, deoarece ar putea exista mai multe căi scurte diferite către nod.
Regulile de calcul sunt după cum urmează:
1. Fiecare frunză din DAG (o frunză este un nod fără margini DAG la noduri la
nivelurile de mai jos) primește un credit de 1.
2. Fiecare nod care nu este o frunză primește un credit egal cu 1 plus suma lui
creditele marginilor DAG de la acel nod până la nivelul de mai jos.
3. O margine DAG e care introduce nodul Z de la nivelul de mai sus primește o parte din
credit Z proporțional cu fracțiunea celor mai scurte căi de la rădăcină la
Z care trec prin e. În mod formal, părinții lui Z să fie Y_1, Y_2, ..., Y_k. Lăsa
p_i fi numărul celor mai scurte căi de la rădăcină la Y_i; acest număr a fost

calculat în Pasul 2 și este ilustrat de etichetele din Fig. 10.4. Apoi
creditul pentru margine (Y_i, Z) este creditul de Z ori p_i împărțit la $\sum_{j=1}^{k} p_j$.
După efectuarea calculului creditului cu fiecare nod ca rădăcină, sumăm
creditele pentru fiecare margine. Apoi, deoarece fiecare cale cea mai scurtă va fi descoperită
modelat de două ori - o dată când fiecare dintre punctele sale finale este rădăcina - trebuie să împărțim
credit pentru fiecare margine cu 2.
Exemplul 10.8: Să efectuăm calculul creditului pentru prezentarea BFS
din Fig. 10.4. Vom începe de la nivelul 3 și vom continua în sus. În primul rând, A și C,
fiind frunze, primiți credit 1. Fiecare dintre aceste noduri are un singur părinte, deci lor
creditul se acordă muchiilor (B, A) și respectiv (B, C).
E
D
F
B
G
A
C
1
1
3
1
1
1
Figura 10.5: Pasul final al algoritmului Girvan-Newman - nivelurile 3 și 2
La nivelul 2, G este o frunză, deci capătă credit 1. B nu este o frunză, deci capătă credit
egal cu 1 plus creditele de pe marginile DAG intrându-l de jos. De cand
ambele aceste margini au credit 1, creditul lui B este 3. Intuitiv 3 reprezintă
faptul că toate cele mai scurte căi de la E la A, B și C trec prin B. Figura 10.5
arată creditele atribuite până acum.

Pagina 386

Acum, să trecem la nivelul 1. B are un singur părinte, D, deci marginea
(D, B) primește întregul credit al lui B, care este 3. Cu toate acestea, G are doi părinți, D
și F. Prin urmare, trebuie să împărțim creditul de 1 pe care îl are G între margini
(D, G) și (F, G). În ce proporție împărțim? Dacă examinați etichetele
din Fig. 10.4, vedeți că atât D cât și F au eticheta 1, reprezentând faptul că
există o cale mai scurtă de la E la fiecare dintre aceste noduri. Astfel, dăm jumătate
creditul lui G pentru fiecare dintre aceste margini; adică creditul lor este fiecare 1 / (1 + 1) = 0,5.
Dacă etichetele D și F din Fig. 10.4 ar fi fost 5 și 3, adică erau cinci
cele mai scurte căi spre D și doar trei spre F, atunci creditul de margine (D, G) ar fi
au fost 5/8 și creditul de margine (F, G) ar fi fost 3/8.
E
D
F
B
G
A
C
1
1
3
1
1
1
0,5
0,5
4.5
1.5
3
4.5
1.5
Figura 10.6: Pasul final al algoritmului Girvan-Newman - completarea
calculul creditului
Acum, putem atribui credite nodurilor de la nivelul 1. D primește 1 plus creditele
a marginilor care intră în ea de jos, care sunt 3 și 0,5. Adică creditul lui D
este 4,5. Creditul lui F este 1 plus creditul marginii (F, G) sau 1,5. În cele din urmă,
muchiile (E, D) și (E, F) primesc creditul lui D și, respectiv, F, deoarece fiecare

dintre aceste noduri are un singur părinte. Aceste credite sunt prezentate în Fig. 10.6.
Creditul pe fiecare dintre marginile din Fig. 10.6 este contribuția la obținerea
între această margine datorită celor mai scurte căi de la E. De exemplu, această contri-
butonul pentru margine (E, D) este 4,5. ✷
Pentru a finaliza calculul între diferențe, trebuie să repetăm acest calcul
pentru fiecare nod ca rădăcină și suma contribuțiilor. În cele din urmă, trebuie să ne împărțim
cu 2 pentru a obține adevărata întreținere, deoarece fiecare cale cea mai scurtă va fi descoperită
de două ori, o dată pentru fiecare dintre obiectivele sale finale.

10.2.5 Utilizarea Betweenness pentru a găsi comunități

Scorurile de interval pentru marginile unui grafic se comportă ca o distanță
măsurați pe nodurile graficului. Nu este tocmai o măsură de distanță, pentru că

---

nu este definit pentru perechile de noduri care nu sunt conectate de o margine și ar putea
să nu satisfacă inegalitatea triunghiului chiar și atunci când este definit. Cu toate acestea, putem grupa
prin luarea marginilor în ordinea creșterii între și adăugați-le la
grafic pe rând. La fiecare pas, componentele conectate ale graficului
formează niște clustere. Cu cât permițența este mai mare, cu atât avem mai multe margini
crește și cu cât grupurile devin mai mari.
Mai frecvent, această idee este exprimată ca un proces de îndepărtare a muchiilor. start
cu graficul și toate marginile acestuia; apoi îndepărtați marginile cu cea mai mare
până când graficul s-a rupt într-un număr adecvat de componente conectate
nents.
Exemplul 10.9: Să începem cu exemplul nostru de rulare, graficul din Fig. 10.1.
O vedem cu intervalul pentru fiecare margine în Fig. 10.7. Calculul
intervalul va fi lăsat în seama cititorului. Singura parte dificilă a numărării
este de a observa că între E și G există două căi scurte, una mergând
prin D și cealaltă prin F. Astfel, fiecare dintre margini (D, E), (E, F),
(D, G) și (G, F) sunt creditate cu o jumătate de cale mai scurtă.
A
B
D
E
G
F
C
5
12
4
1
5
4.5
1.5
1.5
4.5
Figura 10.7: Scorurile de egalitate pentru graficul din Fig. 10.1
În mod clar, muchia (B, D) are cea mai mare distanță, deci este îndepărtată mai întâi.
Asta ne lasă exact cu comunitățile pe care le-am observat care au cel mai mult sens,
și anume: {A, B, C} și {D, E, F, G}. Cu toate acestea, putem continua să eliminăm
margini. Urmează să pleci sunt (A, B) și (B, C) cu un scor de 5, urmat de (D, E)
și (D, G) cu un scor de 4,5. Apoi, (D, F), al cărui scor este 4, ar părăsi
grafic. Vedem în Fig. 10.8 graficul care rămâne.
,,Comunitățile” din Fig. 10.8 arată ciudat. O implicație este că A și
C sunt mai strâns legate între ele decât de B. Adică, într-un anumit sens, B este a
,,Trădător” pentru comunitate {A, B, C} pentru că are un prieten D în afara acestuia
comunitate. La fel, D poate fi văzut ca un „trădător” al grupului {D, E, F, G},
motiv pentru care în Fig. 10.8, numai E, F și G rămân conectate. ✷

10.2.6 Exerciții pentru secțiunea 10.2

Exercițiul 10.2.1: Figura 10.9 este un exemplu de grafic de rețea socială. Utilizare
abordarea Girvan-Newman pentru a găsi numărul celor mai scurte căi din fiecare

---

A

B
D
E
G
F
C
Figura 10.8: Toate marginile cu între 4 sau mai multe au fost eliminate

Accelerarea calculului Betweenness

Dacă aplicăm metoda secțiunii 10.2.4 unui grafic de n noduri și e margini,
este nevoie de O (ne) timp de rulare pentru a calcula între fiecare margine.
Adică, BFS dintr-un singur nod necesită timp O (e), la fel ca cele două etichetări
pași. Trebuie să începem de la fiecare nod, deci există n din calcule
descris în secțiunea 10.2.4.
Dacă graficul este mare - și chiar și un milion de noduri este mare atunci când
algoritmul necesită O (ne) timp - nu ne putem permite să-l executăm așa cum s-a sugerat.
Cu toate acestea, dacă alegem un subset de noduri la întâmplare și le folosim ca
rădăcinile primelor căutări de lățime, putem obține o aproximare la
între fiecare margine care va servi în majoritatea aplicațiilor.
dintre următoarele noduri care trec prin fiecare dintre margini. (a) A (b) B.
Exercițiul 10.2.2: Folosind simetria, calculele exercițiului 10.2.1 sunt toate
trebuie să calculați între fiecare margine. Faceți calculul.
Exercițiul 10.2.3: Folosind valorile de interval dintre Exercițiul 10.2.2, determinați
10. Candidați rezonabili pentru comunitățile din Fig. 10.9 prin eliminarea tuturor
margini cu o intervalitate peste un prag.

10.3 Descoperirea directă a comunităților

În secțiunea anterioară am căutat comunități prin partiționarea tuturor informațiilor
persoane fizice într-o rețea socială. Deși această abordare este relativ eficientă, da
au mai multe limitări. Nu este posibil să plasezi un individ în două
comunități și toată lumea este alocată unei comunități. În această secțiune, vom face
vedeți o tehnică pentru descoperirea comunităților direct, căutând subseturi de
nodurile care au un număr relativ mare de muchii între ele. Interes-
de fapt, tehnica de efectuare a acestei căutări pe un grafic mare implică găsirea mare
obiecte frecvente, după cum sa discutat în capitolul 6.

A
B
C
D
E
G
H
F
Eu
Figura 10.9: Grafic pentru exerciții

10.3.1 Găsirea clicurilor

Primul nostru gând despre cum am putea găsi seturi de noduri cu multe margini
între ele este să începeți prin a găsi o clică mare (un set de noduri cu margini
între oricare dintre ele). Cu toate acestea, această sarcină nu este ușoară. Nu numai că este găsirea
clici maxime NP-complete, dar este printre cele mai grele dintre NP-complete
probleme în sensul că până și apropierea clicei maxime este dificilă.
Mai mult, este posibil să aveți un set de noduri cu aproape toate muchiile între ele
ele și totuși au doar clici relativ mici.
Exemplul 10.10: Să presupunem că graficul nostru are noduri numerotate 1, 2, ..., n și acolo
este o margine între două noduri i și j, cu excepția cazului în care i și j au aceeași rămână-
der când este împărțit la k. Apoi fracția de margini posibile care sunt de fapt
prezentul este aproximativ (k - 1) / k. Există multe clici de mărimea k, dintre care
{1, 2, ..., k} este doar un exemplu.
Cu toate acestea, nu există clici mai mari decât k. Pentru a vedea de ce, observați că orice set de
k + 1 noduri are două care lasă același rest atunci când sunt împărțite la k. Acest
punctul este o aplicare a „principiului porumbarului”. Deoarece există doar k
diferite resturi posibile, nu putem avea resturi distincte pentru fiecare dintre ele
k + 1 noduri. Astfel, nici un set de k + 1 noduri nu poate fi o clică în acest grafic. ✷

10.3.2 Completarea graficelor bipartite

Reamintim discuția noastră despre graficele bipartite din secțiunea 8.3. Un bipartit complet
graficul constă din s noduri pe o parte și t noduri pe cealaltă parte, cu toate st

posibile margini între nodurile unei părți și ale celeilalte prezente. Denotăm acest grafic de $K_{s,t}$. Ar trebui să faceți o analogie între bipartitul complet grafice ca subgrafe ale graficelor generale bipartite și clici ca subgrafe ale grafice generale. De fapt, o clică de noduri este adesea denumită completă

---

grafic și notat K_s, în timp ce un subgraf bipartit complet este uneori numit o bi-clică.

Deși așa cum am văzut în Exemplul 10.10, nu este posibil să garantăm că a cu multe muchii are neapărat o clică mare, este posibil să garantăm înainte că un grafic bipartit cu multe muchii are un bipartit complet mare subgraf. [1]

Putem privi un subgraf bipartit complet (sau o clică dacă vom a descoperit unul mare) ca nucleu al unei comunități și îi adaugă noduri cu multe margini pentru membrii existenți ai comunității. Dacă graficul în sine este k-partit așa cum sa discutat în secțiunea 10.1.4, atunci putem lua noduri de două tipuri iar marginile dintre ele pentru a forma un grafic bipartit. În acest grafic bipartit, putem căuta subgrafe bipartite complete ca nuclee ale comunităților.

De exemplu, în Exemplul 10.2, ne-am putea concentra pe eticheta și nodurile de pagină ale unui grafic precum Fig. 10.2 și încercați să găsiți comunități de etichete și pagini web. O astfel de comunitatea ar consta din etichete și pagini conexe care meritau multe sau toate aceste etichete.

Cu toate acestea, putem folosi și subgrafe bipartite complete pentru găsirea comunității în grafice obișnuite în care nodurile au toate același tip. Împărțiți nodurile în două grupe egale la întâmplare. Dacă există o comunitate, atunci ne-am aștepta aproximativ jumătate din nodurile sale se încadrează în fiecare grup și ne-am aștepta la aproximativ jumătate din marginile sale ar merge între grupuri. Astfel, avem încă o șansă rezonabilă de identificare a unui mare subgraf bipartit complet în comunitate. La acest nucleu putem adăuga noduri din oricare dintre cele două grupuri, dacă au margini la multe dintre nodurile deja identificate ca aparținând comunității.

10.3.3 Găsirea subgrafelor bipartite complete

Să presupunem că ni se dă un grafic bipartit mare G și vrem să găsim cazuri de $K_{s,t}$ în interiorul său. Este posibil să vizualizați problema găsirii instanțelor lui $K_{s,t}$ în cadrul G ca fiind unul dintre găsirea unor obiecte frecvente. În acest scop, lăsați „articolele” fie nodurile de pe o parte a lui G, pe care o vom numi partea stângă. Presupunem că instanța lui $K_{s,t}$ pe care o căutăm are t noduri pe partea stângă și vom avea presupuneți, de asemenea, pentru eficiență că $t \leq s$. „Coșurile” corespund nodurilor pe cealaltă parte a lui G (partea dreaptă). Membrii coșului pentru nodul v sunt nodurile din partea stângă la care este conectat v. În cele din urmă, lăsați sprijinul pragul este s, numărul de noduri pe care instanța lui $K_{s,t}$ le are în dreapta latură.

Putem afirma acum problema găsirii instanțelor $K_{s,t}$ ca cea a găsirii obiecte frecvente F de mărime t. Adică, dacă un set de noduri t din partea stângă este frecvente, apoi toate apar împreună în cel puțin s coșuri. Dar coșurile sunt nodurile din partea dreaptă. Fiecare coș corespunde unui nod care este conectat la toate t de noduri în F. Astfel, elementele frecvente de mărime t și s

[1] Este important să înțelegem că nu ne referim la un subgraf generat - unul format prin selectarea unor noduri și includerea tuturor muchiilor. În acest context, avem nevoie doar de asta acolo să fie margini între orice pereche de noduri pe laturi diferite. De asemenea, este posibil ca unele noduri să fie activate aceeași parte este conectată și prin margini.

---

din coșurile în care apar toate acele articole formează o instanță de $K_{s,t}$.

4
1
A
b
c
d
2
3

Figura 10.10: Graficul bipartit din Fig. 8.1

Exemplul 10.11: Reamintim graficul bipartit din Fig. 8.1, pe care îl repetăm aici ca Fig. 10.10. Partea stângă este nodurile {1, 2, 3, 4}, iar partea dreaptă este {a, b, c, d}.

Acestea din urmă sunt coșurile, deci coșul a constă din „obiecte” 1 și 4; acesta este, a = {1, 4}. În mod similar, b = {2, 3}, c = {1} și d = {3}.
Dacă s = 2 și t = 1, trebuie să găsim elemente de mărime 1 care apar cel puțin în două coșuri. {1} este un astfel de set de articole, iar {3} este altul. Cu toate acestea, în acest mic de exemplu nu există seturi de articole pentru valori mai mari și mai interesante ale lui s și t, precum s = t = 2. ✷

10.3.4 De ce trebuie să existe grafice complete bipartite

Acum trebuie să ne referim la problema demonstrării faptului că orice grafic bipartit cu o fracție suficient de mare a muchiilor prezente va avea o instanță de $K_{s,t}$. În cele ce urmează, presupunem că graficul G are n noduri în stânga și alte n noduri din dreapta. Presupunând că cele două părți au același număr de noduri simplifică calculul, dar argumentul se generalizează la laturile oricărui mărimea. În cele din urmă, fie d gradul mediu al tuturor nodurilor.
Argumentul implică numărarea numărului de obiecte frecvente de mărime t la care contribuie un coș cu d articole. Când însumăm acest număr peste toate noduri din partea dreaptă, obținem frecvența totală a tuturor subseturilor de mărime t stanga. Când împărțim la $\binom{n}{t}$, obținem frecvența medie a tuturor seturilor de articole de mărime t. Cel puțin unul trebuie să aibă o frecvență care este cel puțin medie, deci dacă aceasta media este cel puțin s, știm că o instanță de $K_{s,t}$ există.
Acum, oferim calculul detaliat. Să presupunem gradul ith nodul din dreapta este d_i; adică d_i este mărimea coșului ith. Apoi asta

---

coșul contribuie la $\binom{d_i}{t}$ seturi de mărimi t. Contribuția totală a n nodurile din dreapta sunt $\sum_i \binom{d_i}{t}$. Valoarea acestei sume depinde de d_i's, de curs. Cu toate acestea, știm că valoarea medie a d_i este d. Se știe că această sumă este minimizată când fiecare d_i este d. Nu vom demonstra acest punct, ci a un exemplu simplu va sugera raționamentul: din moment ce $\binom{d_i}{t}$ crește aproximativ ca al t-lea puterea lui d_i, mutarea 1 de la un d_i mare la un d_j mai mic va reduce suma din $\binom{d_i}{t} + \binom{d_j}{t}$.
Exemplul 10.12: Să presupunem că există doar două noduri, t = 2 și media gradul nodurilor este 4. Atunci $d_1 + d_2 = 8$, iar suma dobânzii este $\binom{d_1}{2} + \binom{d_2}{2}$.
Dacă $d_1 = d_2 = 4$, atunci suma este $\binom{4}{2} + \binom{4}{2} = 6 + 6 = 12$. Cu toate acestea, dacă $d_1 = 5$ și $d_2 = 3$, suma este $\binom{5}{2} + \binom{3}{2} = 10 + 3 = 13$. Dacă $d_1 = 6$ și $d_2 = 2$, atunci suma este $\binom{6}{2} + \binom{2}{2} = 15 + 1 = 16$. ✷
Astfel, în cele ce urmează, vom presupune că toate nodurile au gradul mediu d. Așadar, acest lucru minimizează contribuția totală la numărul de elemente și astfel, este cel mai puțin probabil să existe un set de articole frecvente (itemset with cu suport s sau mai mult) de mărime t. Respectați următoarele:

- Contribuția totală a nodurilor n din dreapta la numărul de seturile de mărime t este $n\binom{d}{t}$.
- Numărul de articole de mărimea t este $\binom{n}{t}$.
- Astfel, numărul mediu al unui set de mărimi t este $n\binom{d}{t} / \binom{n}{t}$; această expresie

trebuie să fie cel puțin s dacă vrem să argumentăm că există o instanță a lui $K_{s,t}$.
Dacă extindem coeficienții binomiali din punct de vedere factorial, vom găsi
n (
d

t) / (
n
t) = nd! (n - t)! t! / ((d - t)! t! n!) =
n (d) (d - 1) ··· (d - t + 1) / (n (n - 1) ··· (n - t + 1))
Pentru a simplifica formula de mai sus, să presupunem că n este mult mai mare decât d și
d este mult mai mare decât t. Atunci d (d - 1) ··· (d - t + 1) este aproximativ d_t și
n (n - 1) ··· (n - t + 1) este aproximativ n_t . Astfel, avem nevoie de asta
$n (d / n)_t \geq s$
Adică, dacă există o comunitate cu n noduri pe fiecare parte, gradul mediu
dintre noduri este d și $n (d / n)_t \geq s$, atunci această comunitate este garantată să aibă
un subgraf bipartit complet $K_{s, t}$. Mai mult, putem găsi instanța lui $K_{s, t}$
eficient, folosind metodele din Capitolul 6, chiar dacă această comunitate mică este
încorporat într-un grafic mult mai mare. Adică putem trata toate nodurile din întreg
grafic ca coșuri și ca articole și rulați A-priori sau una dintre îmbunătățirile sale
întregul grafic, căutând seturi de elemente t cu suport s.

Pagina 393

Exemplul 10.13: Să presupunem că există o comunitate cu 100 de noduri pe fiecare parte,
iar gradul mediu al nodurilor este de 50; adică, există jumătate din marginile posibile. Acest
comunitatea va avea o instanță de $K_{s, t}$, cu condiția ca $100 (1/2)_t \geq s$. De exemplu,
dacă t = 2, atunci s poate fi la fel de mare ca 25. Dacă t = 3, s poate fi 11, iar dacă t = 4, s poate
fii 6.
Din păcate, aproximarea pe care am făcut-o ne oferă o legătură pe s care este a
puțin prea sus. Dacă revenim la formula originală $n \binom{d}{t} / \binom{n}{t} \geq s$, vedem că
pentru cazul t = 4 avem nevoie de $100 \binom{50}{4} / \binom{100}{4} \geq s$. Acesta este,
100 × 50 × 49 × 48 × 47
100 × 99 × 98 × 97
≥ s
Expresia din stânga nu este 6, ci doar 5,87. Cu toate acestea, dacă media
suportul pentru un set de mărimi 4 este 5,87, atunci este imposibil ca toate acestea
seturile de articole au suport 5 sau mai puțin. Astfel, putem fi siguri că cel puțin un set de articole
de dimensiunea 4 are suport pentru 6 sau mai multe și există o instanță de $K_{6.4}$ în această comunitate.
✷

10.3.5 Exerciții pentru secțiunea 10.3

Exercițiul 10.3.1: Pentru exemplul de rulare al unei rețele sociale din Fig. 10.1,
pentru câte cazuri de $K_{s, t}$ există:
(a) s = 1 și t = 3.
(b) s = 2 și t = 2.
(c) s = 2 și t = 3.
Exercițiul 10.3.2: Să presupunem că există o comunitate de 2n noduri. Împărțiți
comunitate în două grupuri de n membri, la întâmplare, și formează bipartitul
grafic între cele două grupuri. Să presupunem că gradul mediu al nodurilor de
graficul bipartit este d. Găsiți setul de perechi maxime (t, s), cu $t \leq s$, astfel
că există o instanță de $K_{s, t}$, pentru următoarele combinații
din n și d:
(a) n = 20 și d = 5.
(b) n = 200 și d = 150.
(c) n = 1000 și d = 400.
Prin „maxim”, înseamnă că nu există o pereche diferită (s', t') astfel încât ambele $s' \geq s$
și $t' \geq t$ menține.

Pagina 394

10.4 Partiționarea graficelor

În această secțiune, vom examina o altă abordare a organizării rețelelor sociale
grafice. Folosim câteva instrumente importante din teoria matricii („metoda spectrală
ods ”) pentru a formula problema partiționării unui grafic pentru a minimiza numărul
de margini care leagă diferite componente. Scopul minimizării „tăierii”

dimensiunea trebuie înțeleasă cu atenție înainte de a continua. De exemplu, dacă tu
tocmai v-ați înscris pe Facebook, nu sunteți încă conectat la niciun prieten. Noi nu facem
doriți să împărțiți graficul prietenilor cu dvs. într-un singur grup și restul
lumea din celălalt grup, chiar dacă acest lucru ar împărți graficul fără
existând orice margini care leagă membrii celor două grupuri. Această tăiere nu este
de dorit deoarece cele două componente sunt prea inegale ca dimensiune.

10.4.1 Ce face o partiție bună?

Având în vedere un grafic, am dori să împărțim nodurile în două seturi, astfel încât tăietura sau
setul de margini care conectează nodurile în diferite seturi este minimizat. Cu toate acestea, și noi
doriți să restricționați selecția tăieturii astfel încât cele două seturi să fie aproximativ
egală ca mărime. Următorul exemplu ilustrează acest punct.
Exemplul 10.14: Reamintim exemplul nostru de rulare al graficului din Fig. 10.1. Acolo,
este evident că cea mai bună partiție pune {A, B, C} într-un singur set și {D, E, F, G}
in cealalta. Tăierea constă numai din margine (B, D) și are dimensiunea 1. Nu
tăierea netivială poate fi mai mică.
A
B
D
E
G
F
C
H
Cel mai mic
Cea mai bună tăiere
a tăia
Figura 10.11: Cea mai mică tăietură ar putea să nu fie cea mai bună tăietură
În Fig. 10.11 este o variantă a exemplului nostru, în care am adăugat nodul
H și două margini suplimentare, (H, C) și (C, G). Dacă tot ce doream era să reducem la minimum
dimensiunea tăieturii, atunci cea mai bună alegere ar fi să puneți H într-un singur set și totul
celelalte noduri din celălalt set. Dar ar trebui să fie evident că dacă respingem

---

partiții în care un set este prea mic, atunci cel mai bun lucru pe care îl putem face este să folosim
tăietură formată din muchii (B, D) și (C, G), care împarte graficul în două
seturi de dimensiuni egale {A, B, C, H} și {D, E, F, G}. ✷

10.4.2 Tăieturi normalizate

O definiție adecvată a unei tăieturi „bune” trebuie să echilibreze dimensiunea tăieturii în sine
împotriva diferenței de dimensiuni a seturilor pe care le creează tăietura. O singură alegere
care servește bine este „tăierea normalizată”. Mai întâi, definiți volumul unui set S de
noduri, notate Vol (S), să fie numărul de muchii cu cel puțin un capăt în S.
Să presupunem că partiționăm nodurile unui grafic în două seturi disjuncte S și T.
Fie Cut (S, T) numărul de margini care conectează un nod din S la un nod din T.
Atunci valoarea normală de tăiere pentru S și T este
Tăiere (S, T)
Vol (S)
+
Tăiere (S, T)
Volt)
Exemplul 10.15: Luați în considerare din nou graficul din Fig. 10.11. Dacă alegem S = {H}
și T = {A, B, C, D, E, F, G}, apoi tăiați (S, T) = 1. Vol (S) = 1, deoarece există
este doar o margine conectată la H. Pe de altă parte, Vol (T) = 11, deoarece toate
marginile au cel puțin un capăt la un nod de T. Astfel, tăietura normalizată pentru
această partiție este 1/1 + 1/11 = 1,09.
Acum, ia în considerare tăierea preferată pentru acest grafic format din margini (B, D)
și (C, G). Apoi S = {A, B, C, H} și T = {D, E, F, G}. Tăiere (S, T) = 2,
Vol (S) = 6 și Vol (T) = 7. Decuparea normalizată pentru această partiție este deci numai
2/6 + 2/7 = 0,62. ✷

10.4.3 Câteva matrice care descriu grafice

Pentru a dezvolta teoria modului în care algebra matricială ne poate ajuta să găsim un grafic bun
partiții, trebuie mai întâi să aflăm despre trei matrici diferite care descriu
aspecte ale unui grafic. Primul ar trebui să fie familiar: matricea de adiacență care are
a 1 în rândul i și coloana j dacă există o margine între nodurile i și j și 0
in caz contrar.
A
B
D

E
G
F
C
Figura 10.12: Repetați graficul din Fig. 10.1

Pagina 396

Exemplul 10.16: Repetăm graficul nostru de execuție din Fig. 10.12. Este
matricea de adiacență apare în Fig. 10.13. Rețineți că rândurile și coloanele corespund
răspunde la nodurile A, B, ..., G în această ordine. De exemplu, muchia (B, D)
este reflectat de faptul că intrarea în rândul 2 și coloana 4 este 1 și la fel este
intrare în rândul 4 și coloana 2. ✷

□
□
□
□
□
□
□
□
□
□

0 1 1 0 0 0 0
1 0 1 1 0 0 0
1 1 0 0 0 0 0
0 1 0 0 1 1 1
0 0 0 1 0 1 0
0 0 0 1 1 0 1
0 0 0 1 0 1 0

□
□
□
□
□
□
□
□
□
□

Figura 10.13: Matricea de adiacență pentru Fig. 10.12
A doua matrice de care avem nevoie este matricea de grade pentru un grafic. Acest grafic are
intrări diferite de zero doar pe diagonală. Intrarea pentru rândul și coloana i este
gradul nodului ith.
Exemplul 10.17: Matricea de grade pentru graficul din Fig. 10.12 este prezentată în
Fig. 10.14. Folosim aceeași ordine a nodurilor ca în Exemplul 10.16. Pentru
de exemplu, intrarea în rândul 4 și coloana 4 este 4 deoarece nodul D are margini la
alte patru noduri. Intrarea din rândul 4 și coloana 5 este 0, deoarece intrarea respectivă este
nu pe diagonală. ✷

□
□
□
□
□
□
□
□
□

□
2 0 0 0 0 0 0
0 3 0 0 0 0 0
0 0 2 0 0 0 0
0 0 0 4 0 0 0
0 0 0 0 2 0 0
0 0 0 0 0 3 0
0 0 0 0 0 0 2
□
□
□
□
□
□
□
□
□
□
Figura 10.14: Matricea de grade pentru Fig. 10.12
Să presupunem că graficul nostru are matricea de adiacență A și matricea de grade D. A treia noastră matricea, numită matricea laplaciană, este L = D - A, diferența dintre
matricea de grade și matricea de adiacență. Adică matricea laplaciană L are
aceleași intrări ca D pe diagonală. În afara diagonalei, la rândul i și coloana j,
L are -1 dacă există o margine între nodurile i și j și 0 dacă nu.
Exemplul 10.18: este prezentată matricea laplaciană pentru graficul din Fig. 10.12
în Fig. 10.15. Observați că fiecare rând și fiecare coloană se ridică la zero, așa cum trebuie să fie
cazul pentru orice matrice laplaciană. ✷

Pagina 397

□
□
□
□
□
□
□
□
□
□
2 -1 -1
0
0
0
0
-1
3 -1 -1
0
0
0
-1 -1
2
0
0
0
0
0 -1
0
4 -1 -1 -1
0

0
0 -1
2 -1
0
0
0
0 -1 -1
3 -1
0
0
0 -1
0 -1
2
□
□
□
□
□
□
□
□
□
□
Figura 10.15: Matricea Laplaciană pentru Fig. 10.12

10.4.4 Valori proprii ale matricei laplaciene

Putem avea o idee bună despre cel mai bun mod de partiționare a unui grafic din valorile proprii și vectori proprii ai matricei sale laplaciene. În secțiunea 5.1.2 am observat cum vector propriu principal (vector propriu asociat cu cea mai mare valoare proprie) a matricea de tranziție a Web-ului ne-a spus ceva util despre importanța Pagini web. De fapt, în cazuri simple (fără impozitare), vectorul propriu principal este Vectorul PageRank. Cu toate acestea, atunci când avem de-a face cu matricea laplaciană, aceasta se transformă afirma că cele mai mici valori proprii și vectorii proprii ale acestora dezvăluie informațiile noi dorim.

Cea mai mică valoare proprie pentru fiecare matrice laplaciană este 0 și corespondența sa vectorul propriu este [1, 1, ..., 1]. Pentru a vedea de ce, să fie L matricea laplaciană pentru a graficul de n noduri și să fie 1 vectorul coloanei tuturor celor 1 cu lungimea n. Noi revendicarea L1 este un vector coloană al tuturor 0. Pentru a vedea de ce, luați în considerare rândul i din L. Its element diagonal are gradul d de nod i. De asemenea, rândul i va avea d apariții din −1 și toate celelalte elemente ale rândului i sunt 0. Înmulțind rândul i cu vectorul coloană 1 are efectul de a însuma rândul, iar această sumă este clar d + (-1) d = 0.

Astfel, putem concluziona L1 = 01, ceea ce demonstrează că 0 este o valoare proprie și 1 vectorul propriu corespunzător.

Există o modalitate simplă de a găsi a doua cea mai mică valoare proprie pentru orice matrice, cum ar fi matricea laplaciană, care este simetrică (intrarea în rândul i și coloana j este egal cu intrarea din rândul j și coloana i). Deși nu vom dovedi acest lucru de fapt, a doua cea mai mică valoare proprie a lui L este minimul lui $x^T Lx$, unde $x = [x_1, x_2, ..., x_n]$ este un vector coloană cu n componente, iar minimul este luate sub constrângeri:

1. Lungimea lui x este 1; adică $\sum_{i=1}^{n} x_i^2 = 1$.
2. x este ortogonală la vectorul propriu asociat cu cea mai mică valoare proprie.

Mai mult, valoarea lui x care atinge acest minim este al doilea vector propriu.

Când L este o matrice laplaciană pentru un grafic cu n-nod, știm ceva Mai Mult. Vectorul propriu asociat cu cea mai mică valoare proprie este 1. Astfel, dacă x

---

Pagina 398

este ortogonală la 1, trebuie să avem

$$x^T 1 = \sum_{i=1}^{n}$$

$x_i = 0$
În plus pentru matricea laplaciană, expresia $x^T Lx$ are un echivalent util
expresie alent. Reamintim că $L = D - A$, unde D și A sunt gradul și
matrici de adiacență ale aceluiași grafic. Astfel, $x^T Lx = x^T Dx - x^T Ax$. Permiteți-ne
evaluați termenul cu D și apoi termenul pentru A. Aici, Dx este coloana vec-
tor $[d_1 x_1, d_2 x_2, ..., d_n x_n]$, unde d_i este gradul nodului ith al graficului.
Astfel, $x^T Dx$ este $\sum_{i=1}^{n} d_i x_{eu}^2$.
Acum, treceți la $x^T Ax$. Componenta a-a a vectorului de coloană Ax este suma
de x_j peste toate j astfel încât să existe o margine (i, j) în grafic. Astfel, $-x^T Ax$ este
suma de $-2x_i x_j$ pe toate perechile de noduri {i, j} astfel încât să existe o margine între
lor. Rețineți că factorul 2 apare deoarece fiecare set {i, j} corespunde a două
termeni, $-x_i x_j$ și $-x_j x_i$.
Putem grupa termenii lui $x^T Lx$ într-un mod care distribuie termenii fiecăruia
pereche {i, j}. Din $-x^T Ax$, avem deja termenul $-2x_i x_j$. Din $x^T Dx$, noi
distribuie termenul $d_i x_i^2$ la perechile d_i care includ nodul i. Drept urmare, noi
se poate asocia cu fiecare pereche {i, j} care are o margine între nodurile i și j
termeni $x_i^2 - 2x_i x_j + x_j^2$. Această expresie este echivalentă cu $(x_i - x_j)^2$. Prin urmare, noi
au demonstrat că $x^T Lx$ este egală cu suma peste toate muchiile grafice (i, j) ale lui $(x_i - x_j)^2$.
Amintiți-vă că a doua cea mai mică valoare proprie este minimul acestei expresii
sub constrângerea că $\sum_{i=1}^{n} x_i^2 = 1$. Intuitiv, o reducem la minimum prin realizarea
x_i și x_j se închid ori de câte ori există o margine între nodurile i și j în grafic.
Ne-am putea imagina că am putea alege $x_i = 1 / \sqrt{n}$ pentru tot i și, astfel, să facem acest lucru
suma 0. Totuși, reamintim că suntem constrânși să alegem x pentru a fi ortogonali
1, ceea ce înseamnă că suma lui x_i este 0. De asemenea, suntem obligați să facem $\sum_{i=1}^{n} x_{voi}^2$ fi
1, deci toate componentele nu pot fi 0. În consecință, x trebuie să aibă unele pozitive
și unele componente negative.
Putem obține o partiție a graficului luând un set pentru a fi noduri
i a cărei componentă vectorială corespunzătoare x_i este pozitivă și cealaltă setată la
fie cei ale căror componente sunt negative. Această alegere nu garantează o
partiția în seturi de dimensiuni egale, dar este posibil ca dimensiunile să fie apropiate. Noi credem
că tăietura dintre cele două seturi va avea un număr mic de margini deoarece
$(x_i - x_j)^2$ este probabil să fie mai mic dacă ambele x_i și x_j au același semn decât dacă au
au semne diferite. Astfel, se va minimiza $x^T Lx$ sub constrângerile necesare
tind să dea x_i și x_j același semn dacă există o margine (i, j).
Exemplul 10.19: Să aplicăm tehnica de mai sus la graficul din Fig. 10.16.
Matricea laplaciană pentru acest grafic este prezentată în Fig. 10.17. Prin metoda standard
Ods sau pachete de matematică putem găsi toate valorile proprii și vectorii proprii ai acesteia
matrice. Le vom tabela pur și simplu în Fig. 10.18, de la cea mai mică valoare proprie la
cel mai inalt. Rețineți că nu am scalat vectorii proprii pentru a avea lungimea 1, dar
am putea face asta cu ușurință dacă am dori.

Pagina 399

2
3
6
5
4
1
Figura 10.16: Grafic pentru ilustrarea partiționării prin analiză spectrală
□
□
□
□
□

□
□
□
3 -1 -1 -1
0
0
-1
2 -1
0
0
0
-1 -1
3
0
0 -1
-1
0
0
3 -1 -1
0
0
0 -1
2 -1
0
0 -1 -1 -1
3
□
□
□
□
□
□
□
□
Figura 10.17: Matricea Laplaciană pentru Fig. 10.16
Al doilea vector propriu are trei componente pozitive și trei componente negative.
Face sugestia surprinzătoare că un grup ar trebui să fie {1, 2, 3},
noduri cu componente pozitive, iar celălalt grup ar trebui să fie {4, 5, 6}. ✷
Valoare proprie
0
1
3
3
4
5
Vector propriu 1
1 −5 −1 −1 −1
1
2
4 −2
1
0
1
1
1
3 −1
1
1
−1 −5 −1
1
1
1
−2
4 −2 −1
0
1

−1
1
3
1 −1
Figura 10.18: Valori proprii și vectori proprii pentru matricea din Fig. 10.17

10.4.5 Metode alternative de partiționare

Metoda secțiunii 10.4.4 ne oferă o bună partiție a graficului în două
bucăți care au o mică tăietură între ele. Există mai multe moduri în care putem folosi
aceiași vectori proprii pentru a sugera alte alegeri bune de partiție. În primul rând, suntem
nu este constrâns să pună toate nodurile cu componente pozitive în vectorul propriu
într-un grup și cele cu componente negative în celălalt. Am putea stabili
pragul la un alt punct decât zero.
De exemplu, să presupunem că am modificat Exemplul 10.19 astfel încât pragul să fie
nu zero, ci -1,5. Apoi cele două noduri 4 și 6, cu componentele -1 în
al doilea vector propriu din Fig. 10.18, ar uni 1, 2 și 3, lăsând cinci noduri într-unul

---

componentă și numai nodul 5 în cealaltă. Această partiție ar avea o dimensiune redusă
două, la fel ca alegerea bazată pe pragul zero, dar cele două componente
au dimensiuni radical diferite, așa că am tinde să preferăm alegerea noastră originală.
Cu toate acestea, există și alte cazuri în care pragul zero oferă dimensiuni inegale
componente, așa cum ar fi cazul dacă am folosi al treilea vector propriu din Fig. 10.18.
Este posibil să dorim, de asemenea, o partiție în mai mult de două componente. O abordare
este să folosiți metoda descrisă mai sus pentru a împărți graficul în două, apoi utilizați
în mod repetat pe componente pentru a le împărți cât de mult doriți. O secunda
abordarea este de a folosi mai mulți dintre vectorii proprii, nu doar al doilea, pentru partiționare
graficul. Dacă folosim vectori proprii și stabilim un prag pentru fiecare, putem obține un
partiție în grupuri de 2 m , fiecare grup constând din noduri care sunt deasupra sau
sub pragul pentru fiecare dintre vectorii proprii, într-un anumit model.
Este demn de remarcat faptul că fiecare vector propriu, cu excepția primului, este vectorul x care
minimizează x T Lx, sub rezerva constrângerii că este ortogonală față de toate anterioare
vectori proprii. Această constrângere generalizează constrângerile pe care le-am descris pentru
al doilea vector propriu într-un mod natural. Ca rezultat, în timp ce fiecare vector propriu încearcă
pentru a produce o tăietură de dimensiuni minime, faptul că vectorii proprii succesivi trebuie
satisfacerea din ce în ce mai multe constrângeri determină în general reducerile pe care le descriu
progresiv mai rău.
Exemplul 10.20: Să reconsiderăm graficul din Fig. 10.16, pentru care
vectorii proprii ai matricei sale laplaciene au fost tabelați în Fig. 10.18. Al treilea
vectorul propriu, cu un prag de 0, pune nodurile 1 și 4 într-un singur grup și
alte patru noduri în celălalt. Aceasta nu este o partiție proastă, dar dimensiunea sa tăiată este
patru, comparativ cu tăietura de mărimea doi pe care o obținem din al doilea vector propriu.
Dacă folosim atât al doilea cât și al treilea vector propriu, punem nodurile 2 și 3 în
un grup, deoarece componentele lor sunt pozitive în ambii vectori proprii. Noduri
5 și 6 se află într-un alt grup, deoarece componentele lor sunt negative în
al doilea vector propriu și pozitiv în al treilea. Nodul 1 se află într-un grup singur
deoarece este pozitiv în al doilea vector propriu și negativ în al treilea, în timp ce
nodul 4 este, de asemenea, într-un grup de la sine, deoarece componenta sa este negativă în ambele
vectori proprii. Această partiție a unui grafic cu șase noduri în patru grupuri este prea bună a
partția să fie semnificativă. Dar cel puțin grupurile de mărime două au fiecare câte un
marginea dintre noduri, deci este la fel de bun pe cât am putea obține vreodată pentru o partiție
în grupuri de aceste dimensiuni. ✷

10.4.6 Exerciții pentru secțiunea 10.4

Exercițiul 10.4.1: Pentru graficul din Fig. 10.9, construiți:
(a) Matricea de adiacență.
(b) Matricea gradelor.
(c) Matricea laplaciană.

! Exercițiul 10.4.2: Pentru matricea Laplaciană construită în Exercițiul 10.4.1 (c),
găsiți a doua cea mai mică valoare proprie și vectorul ei propriu. Ce partiție a
noduri sugerează?
!! Exercițiul 10.4.3: Pentru matricea Laplaciană construită în Exercițiul 10.4.1 (c),

construiți a treia și ulterior cele mai mici valori proprii și vectorii lor proprii.

10.5 Găsirea comunităților suprapuse

Până în prezent, ne-am concentrat pe gruparea unui grafic social pentru a găsi comunități. Dar comunitățile sunt rareori disjuncte. În această secțiune, vă explicăm a metoda de a lua un grafic social și de a-i potrivi un model care explică cel mai bine cum ar fi putut fi generat de un mecanism care presupune probabilitatea că doi indivizi sunt conectați printr-o margine (sunt „prieteni”) crește pe măsură ce să devină membri ai mai multor comunități în comun. Un instrument important în acest sens analiza este „estimarea maximă a probabilității”, pe care o vom explica mai înainte ajungând la problema găsirii unor comunități suprapuse.

10.5.1 Natura comunităților

Pentru început, să analizăm la ce ne-am aștepta două comunități suprapuse să arate ca. Datele noastre sunt un grafic social, în care nodurile sunt oameni și există un între două noduri dacă oamenii sunt „prieteni”. Să ne imaginăm că asta graficul reprezintă elevii de la o școală și există două cluburi în această școală: Clubul de șah și Clubul spaniol. Este rezonabil să presupunem că fiecare dintre aceste cluburi formează o comunitate. De asemenea, este rezonabil să presupunem că două persoane în Clubul de șah sunt mai predispuși să fie prieteni în grafic pentru că știu reciproc din club. La fel, dacă două persoane sunt în clubul spaniol, atunci există șanse mari să se cunoască și să fie prieteni.

Ce se întâmplă dacă două persoane sunt în ambele cluburi? Acum au două motive pentru care au s-ar putea cunoaște reciproc și așa ne-am aștepta la o probabilitate și mai mare că vor fi prieteni în graficul social. Concluzia noastră este că ne așteptăm marginile să fie dense în cadrul oricărei comunități, dar ne așteptăm ca marginile să fie și mai dense în intersecția a două comunități, mai densă decât cea din intersecția trei comunități și așa mai departe. Ideea este sugerată de Fig. 10.19.

10.5.2 Estimarea probabilității maxime

Înainte să vedem algoritmul pentru găsirea comunităților care se suprapun un fel sugerat în secțiunea 10.5.1, haideți să divagăm și să învățăm o modelare utilă instrument numit estimare maximă a probabilității sau MLE. Ideea din spatele MLE este că facem o presupunere despre procesul generativ (modelul) că creează instanțe ale unor artefacte, de exemplu, „graficele prietenilor”. Modelul are parametrii care determină probabilitatea de a genera o anumită instanță a artefactului; această probabilitate se numește probabilitatea acelor valori ale parametrilor.

Pagina 402

Club de sah

Clubul spaniol

Intersecție

Figura 10.19: Suprapunerea a două comunități este mai densă decât non-suprapunerea părți din aceste comunități

Presupunem că valoarea parametrilor care oferă cea mai mare valoare a probabilitatea este modelul corect pentru artefactul observat.

Un exemplu ar trebui să clarifice principiul MLE. De exemplu, am putea doresc să genereze grafice aleatorii. Presupunem că fiecare margine este prezentă cu probabilitatea p și neprezentarea cu probabilitatea 1 − p, cu prezența sau absența a fiecărei muchii alese independent. Singurul parametru pe care îl putem ajusta este p. Pentru fiecare valoare a lui p are o probabilitate mică dar diferită de zero ca graficul generat va fi exact cea pe care o vedem. Urmând principiul MLE, vom declara că adevărata valoare a lui p este cea pentru care probabilitatea de a genera graficul observat este cel mai mare.

Exemplul 10.21: Luați în considerare graficul din Fig. 10.19. Există 15 noduri și 23 de margini. Așa cum există $\binom{15}{2} = 105$ perechi de 15 noduri, vedem că dacă fiecare margine se alege cu probabilitatea p, apoi probabilitatea (probabilitatea) de a genera exact graficul din Fig. 10.19 este dat de funcția $p^{23}(1 - p)^{82}$. Nu conteaza ce valoare p are între 0 și 1, adică un număr incredibil de mic. Cu exceptia funcția are un maxim, pe care îl putem determina luând derivata sa și setarea la 0. Adică:

$$23p^{22}(1 - p)^{82} - 82p^{23}(1 - p)^{81} = 0$$

Putem grupa termenii pentru a rescrie cele de mai sus ca

$$p^{22}(1 - p)^{81}(23(1 - p) - 82p) = 0$$

Pagina 403

Probabilităţi anterioare

Când facem o analiză MLE, presupunem în general că parametrii pot lua orice valoare în domeniul lor şi nu există nicio părtinire în favoarea anumitor valori. Cu toate acestea, dacă nu este cazul, atunci putem înmulţi formula obţinem probabilitatea generării artefactului observat, aşa cum o funcţie a valorilor parametrilor, prin funcţia care reprezintă probabilitatea relativă ca acele valori ale parametrului să fie adevăratele valori. Exerciţiile oferă exemple de MLE cu presupuneri despre precedent distribuirea parametrilor.

Singurul mod în care partea dreaptă poate fi 0 este dacă p este 0 sau 1 sau ultimul factor,

(23 (1 - p) - 82p)

este 0. Când p este 0 sau 1, valoarea funcţiei de probabilitate $p^{23} (1 - p)^{82}$ este minimizat, nu maximizat, deci trebuie să fie ultimul factor care este 0. Adică probabilitatea de a genera graficul din Fig. 10.19 este maximizată atunci când

23 - 23p - 82p = 0

sau p = 23/105.

Rezultatul nu este o surpriză. Se spune că cea mai probabilă valoare pentru p este fracţia observată a marginilor posibile care sunt prezente în grafic. Cu toate acestea, când folosim un mecanism mai complicat pentru a genera grafice sau alte artefacte, valoarea parametrilor care produc cu maxim artefactul observat probabilitatea este departe de a fi evidentă. ✷

10.5.3 Modelul grafic de afiliere

Vom introduce acum un mecanism rezonabil, numit grafic de afiliere model, pentru a genera grafice sociale din comunităţi. Odată ce vedem cum parametrii modelului influenţează probabilitatea de a vedea un grafic dat, putem abordează modul în care s-ar rezolva valorile parametrilor care dau probabilitate maximă. Elementele mecanismului de generare a graficelor sunt:

1. Există un număr dat de comunităţi şi există un număr dat de indivizi (noduri ale graficului).
2. Fiecare comunitate poate avea ca membri orice grup de persoane. Acesta este, apartenenţa la comunităţi este parametrul modelului.
3. Fiecare comunitate C are o probabilitate p_C asociată, probabilitatea că doi membri ai comunităţii C sunt conectaţi printr-o margine, deoarece ei sunt ambii membri ai C. Aceste probabilităţi sunt, de asemenea, parametri ai model.

---

4. Dacă o pereche de noduri se află în două sau mai multe comunităţi, atunci există o margine între ele dacă induce oricare dintre comunităţile ale căror membri sunt ambii această margine conform regulii (3).
5. Decizia dacă o comunitate induce o margine între două dintre ele membrii sunt independenţi de acea decizie pentru orice altă comunitate din care aceşti doi indivizi sunt şi ei membri.

Trebuie să calculăm probabilitatea ca un grafic dat cu numărul corespunzător de noduri este generat de acest mecanism. Observaţia cheie este modul în care marginea probabilităţile sunt calculate, având în vedere o atribuire de indivizi către comunităţi şi valorile p_C ’. Luaţi în considerare o margine (u, v) între nodurile u şi v. Să presupunem u şi v sunt membri ai comunităţilor C şi D, dar nu în alte comunităţi.

Apoi, din cauza presupunerii independenţei, regula (5) de mai sus, probabilitatea că nu există margine între u şi v este produsul probabilităţilor că nu există nici o margine din cauza comunităţii C şi nici o margine din cauza comunităţii D. Că este, cu probabilitatea $(1 - p_C) (1 - p_D)$ nu există margine (u, v) în grafic şi desigur probabilitatea ca există o astfel de margine este de 1 minus încât.

Mai general, dacă u şi v sunt membri ai unui set de comunităţi care nu sunt goale M şi nu orice altul, atunci p_{uv} , probabilitatea unei margini între u şi v este dat de:

$$p_{uv} = 1 - \prod_{C \text{ în } M} (1 - p_C) \quad (10.1)$$

Ca un caz special important, dacă u şi v nu sunt în comunităţi împreună, apoi luăm p_{uv} ca fiind ϱ, un număr foarte mic. Trebuie să alegem acest lucru probabilitatea de a fi diferit de zero, altfel nu putem atribui niciodată o probabilitate diferită de zero oricărui set de comunităţi care nu au fiecare pereche de persoane în comun

o comunitate. Dar, luând probabilitatea de a fi foarte mici, ne părtinim
calcul pentru a favoriza soluții astfel încât fiecare margine observată să fie explicată prin
apartenență comună la unele comunități.
Dacă știm care sunt nodurile în ce comunități, atunci putem calcula
probabilitatea graficului dat pentru aceste probabilități de margine utilizând un simplu
generalizarea exemplului 10.21. Fie p_{uv} probabilitatea unei margini între
nodurile u și v, definite de ecuația (10.1), sau ϱ dacă u și v nu împărtășesc
o comunitate. Atunci probabilitatea ca E să fie exact setul de muchii din
graficul observat este

$$\prod_{(u,v) \text{ în } E} p_{uv} \prod_{(u,v) \text{ nu în } E} (1 - p_{uv})$$

Exemplul 10.22: Luați în considerare minusculul grafic social din Fig. 10.20. Să presupunem că acolo
sunt două comunități C și D, cu probabilitățile asociate p_C și p_D. De asemenea,
să presupunem că am stabilit (sau folosim ca ipoteză temporară) că
C = {w, x, y} și D = {w, y, z}. Pentru început, luați în considerare perechea de noduri w și
X. M_{wx} = {C}; adică această pereche se află în comunitatea C, dar nu în comunitatea D.
De aceea, $p_{wx} = 1 - (1 - p_C) = p_C$.

y
z
w
X

Figura 10.20: Un grafic social

În mod similar, x și y sunt împreună împreună în C, y și z sunt împreună împreună în D,
și, de asemenea, w și z sunt împreună doar în D. Astfel, găsim $p_{xy} = p_C$ și
$p_{yz} = p_{wz} = p_D$. Acum, perechea w și y sunt împreună în ambele comunități, deci
$p_{wy} = 1- (1-p_C) (1-p_D) = p_C + p_D -p_C p_D$. În cele din urmă, x și z nu sunt împreună
în oricare dintre comunități, deci p_{xz} = ϱ.
Acum, putem calcula probabilitatea graficului din Fig. 10.20, având în vedere valoarea noastră
ipoteze despre apartenența la cele două comunități. Această probabilitate este
produs al probabilităților asociate cu fiecare dintre cele patru perechi de noduri
ale cărui margini apar în grafic, de câte ori minus probabilitatea pentru fiecare dintre ele
cele două perechi ale căror margini nu sunt acolo. Adică vrem

$$p_{wx} p_{wy} p_{xy} p_{yz} (1 - p_{wz}) (1 - p_{xz})$$

Înlocuind expresiile pe care le-am dezvoltat mai sus pentru fiecare dintre aceste probabilități,
primim

$$(p_C)^2 p_D (p_C + p_D - p_C p_D) (1 - p_D) (1 - \varrho) \quad (10.2)$$

Rețineți că ϱ este foarte mic, deci ultimul factor este în esență 1 și poate fi abandonat.
Trebuie să găsim valorile lui p_C și p_D care maximizează Expresia (10.2).
În primul rând, observați că toți factorii sunt fie independente de p_C sau să crească cu p_C. The
singurul pas greu în acest argument este să ne amintim că $p_D \leq 1$, deci

$$p_C + p_D - p_C p_D$$

trebuie să crească în mod pozitiv cu p_C. Rezultă că probabilitatea este maximizată atunci când
p_C este cât mai mare posibil; adică $p_C = 1$.
Apoi, trebuie să găsim valoarea lui p_D care maximizează expresia dată
că $p_C = 1$. Expresia devine $p_D (1 - p_D)$ și este ușor de văzut asta
această expresie are maximul său la $p_D = 0{,}5$. Adică, având în vedere C = {w, x, y} și
D = {w, y, z}, probabilitatea maximă pentru graficul din Fig. 10.20 apare atunci când
membrii C sunt sigur că au o margine între ei și există un procent de 50%
șansa ca apartenența comună la D să provoace un avantaj între membri.

✷

10.5.4 Optimizarea discretă a sarcinilor comunitare

Exemplul 10.22 reflectă doar o parte a procesului prin care dezvoltăm o afiliere -
model grafic pentru o anumită rețea. De asemenea, trebuie să găsim o sarcină de memorie

se adresează comunităților astfel încât soluția de maximă probabilitate pentru această semnarea are cea mai mare probabilitate pentru orice misiune. Odată ce ne fixăm pe un misiune, putem găsi probabilitățile p c asociate fiecărei comunități C. Cea mai populară metodă pentru a face acest lucru este „coborârea în gradient”, o tehnică pe care l-am introdus în secțiunea 9.4.5 și care va fi discutat în continuare în secțiunea 12.3.4.

Din păcate, nu este evident modul în care se încorporează setul de membri ai fiecare comunitate în soluția gradient-coborâre, deoarece modificările aduse bershipul comunităților se face prin pași discreți, nu în conformitate cu unele continue funcționează așa cum este necesar pentru coborârea în gradient. Singurul mod fezabil de a căuta în spațiul posibilelor alocări ale membrilor către comunități este începând cu un misiune, poate aleasă la întâmplare. Pe măsură ce căutăm sarcini mai bune, avem întotdeauna o singură sarcină „curentă”.

Luăm în considerare mici modificări ale atribuirii, de exemplu prin inserție sau ștergere a unui membru pentru o comunitate. Pentru fiecare astfel de misiune, putem rezolva pentru cele mai bune probabilități ale comunității (p c ’) după gradient descendent. Dacă o schimbare la atribuirea curentă duce la o probabilitate mai mare, apoi faceți rezultatul misiunea să fie misiunea curentă. Dacă schimbarea nu îmbunătățește , apoi încercați o altă modificare față de misiunea curentă. În cele din urmă, tu va ajunge la o misiune curentă pentru care nu se modifică deloc sarcina duce la o probabilitate mai mare. Atribuirea rezultată, împreună cu probele pentru comunități care maximizează probabilitatea pentru această misiune, formează modelul de grafic de afiliere pentru acest grafic.

Rețineți că, deoarece permitem doar un anumit set de modificări simple, cum ar fi ca inserare sau ștergere a unui membru, atribuirea rezultată a nodurilor către comunitățile pot să nu fie optime la nivel global. Este posibil ca modelul cu cea mai mare probabilitate este radical diferită de cea mai bună sarcină pe care o putem a ajunge. Este posibil și adesea adecvat să repetați acest proces cu mulți diferite soluții de pornire aleatorii și pentru a lua cele mai bune dintre modelele rezultate.

În cele din urmă, în discuția de mai sus am presupus numărul de comunități a fost reparat. De asemenea, putem permite modificări „mici” care ajustează numărul de comunități, de exemplu, prin fuzionarea a două comunități sau prin adăugarea unei noi comunități cu un set aleatoriu de membri inițiali. O altă abordare este de a stabili numărul de comunități, dar după ce a găsit cel mai bun model pentru acel număr de comunități, adăugați încă o comunitate și rezolvați din nou, pentru a vedea dacă probabilitatea celor mai buni modelul se îmbunătățește. Dacă nu, scădeți una din numărul de comunități pentru a vedea dacă probabilitatea se îmbunătățește. Dacă oricare dintre aceste modificări îmbunătățește lucrurile, repetați procesul de deplasare în direcția sau mai multe sau mai puține comunități, oricare ar fi a dus la o îmbunătățire.

10.5.5 Evitarea utilizării modificărilor discrete de membru

Există o soluție la problema cauzată de modelul grafic de afiliere, unde apartenența la persoane în comunități este discretă: fie sunteți membru comunității sau nu. În loc de apartenența la noduri „totul sau nimic”

Probabilitatea jurnalului

De obicei, calculăm logaritmul funcției de probabilitate (jurnalul ca-lihood), mai degrabă decât funcția în sine. Acest lucru oferă mai multe avantaje. Produsele devin sume, ceea ce simplifică adesea expresia. De asemenea, amestecarea mai multor numere este mai puțin predispusă la erori de rotunjire numerică decât este luând produsul multor numere minuscule.

comunități, putem presupune că pentru fiecare nod și fiecare comunitate, acolo este o „forță a calității de membru” pentru acel nod și comunitate. Intuitiv, cu cât este mai puternică componența a două persoane din aceeași comunitate, cu atât este mai mare probabil că această comunitate va face să existe o margine între ele.

În acest model, putem ajusta puterea de apartenență pentru o persoană într-un comunitate continuu, la fel cum putem ajusta probabilitatea asociată o comunitate în modelul grafic de afiliere. Această îmbunătățire ne permite utilizați metode pentru optimizarea funcțiilor continue, cum ar fi coborârea în gradient, către maximizați expresia pentru probabilitate. În noul model, avem

1. S-au fixat seturi de comunități și indivizi, ca înainte.
2. Pentru fiecare comunitate C și x individuală, există o forță de membru parametrul F xC . Acești parametri pot lua orice valoare non-negativă și a valoarea 0 înseamnă că individul nu se află cu siguranță în comunitate.
3. Probabilitatea ca comunitatea C să determine o margine între nodurile u și v este

$p_C(u, v) = 1 - e^{-F_{uC} F_{vC}}$
Ca și înainte, probabilitatea ca o margine între u și v să fie de 1 minus
probabilitatea ca niciuna dintre comunități să nu provoace o margine între
lor. Adică, fiecare comunitate cauzează în mod independent margini și există o margine
între două noduri dacă o comunitate face ca aceasta să existe. Mai formal, p_{uv} , the
probabilitatea unei margini între nodurile u și v, este
$p_{uv} = 1 - \prod_C (1 - p_C(u, v))$
Dacă înlocuim formula cu $p_C(u, v)$ care este presupusă în model, obținem
$p_{uv} = 1 - e^{-\sum_C F_{uC} F_{vC}}$
În cele din urmă, fie E setul de muchii din graficul observat. Ca și înainte, putem
scrie formula pentru probabilitatea graficului observat ca produs al
expresie pentru p_{uv} pentru fiecare margine (u, v) care este în E, ori produsul de $1 - p_{uv}$

Optimizare continuă versus discretă

Trucul folosit în secțiunea 10.5.5 este surprinzător de comun. Ori de câte ori tu
încearcă să găsească o soluție optimă la o problemă în care unele dintre
elementele sunt continue (de exemplu, probabilitățile de a induce comunitățile
marginile) și unele elemente sunt discrete (de exemplu, apartenența la
comunități), ar putea avea sens să înlocuim fiecare element discret cu un con-
variabila tinuous. Această schimbare, deși nu reprezintă strict vorbind
realitate, ne permite să realizăm o singură optimizare, care va utiliza un
metoda de optimizare continuă, cum ar fi coborârea în gradient. Rețineți, totuși,
fie că realizăm o optimizare discretă, ca în secțiunea 10.5.4 sau
o optimizare continuă ca în secțiunea 10.5.5, riscăm înfășurarea
într-un optim local care nu este cea mai bună soluție la nivel global.
pentru fiecare margine (u, v) care nu este în E. Astfel, în noul model, formula pentru
probabilitatea graficului cu muchiile E este

$$\prod_{(u, v) \text{ în } E} (1 - e^{-\sum_C F_{uC} F_{vC}})$$

$$\prod_{(u, v) \text{ nu în } E} e^{-\sum_C F_{uC} F_{vC}}$$

(10.3)

Putem simplifica oarecum Expresia (10.3) luând logaritmul său. Re-
membru care maximizează o funcție maximizează și logaritmul acelei funcții
și invers. Deci putem lua logaritmul natural al Expresului
(10.3) pentru a înlocui produsele cu sume. De asemenea, primim simplificare de la
faptul că $\log(e^x) = x$.

$$\sum_{(u, v) \text{ în } E} \text{jurnal}\,(1 - e^{-\sum_C F_{uC} F_{vC}}) - \sum_{(u, v) \text{ nu în } E} \sum_C F_{uC} F_{vC}$$

(10.4)

Acum putem găsi valorile pentru F_{xC} care maximizează expresia
(10.4). O modalitate este de a folosi coborârea de gradient într-un mod similar cu ceea ce a fost
realizat în secțiunea 9.4.5. Adică, alegem un nod x și ajustăm toate valorile lui
F_{xC} este în direcția care îmbunătățește cel mai mult valoarea de (10.4). Observa asta
singurii factori ale căror valori se schimbă ca răspuns la modificările F_{xC} sunt
cele în care unul dintre u și v este x și celălalt dintre u și v este un nod adiacent
X. Deoarece gradul unui nod este de obicei mult mai mic decât numărul de muchii
în grafic, putem evita să privim majoritatea termenilor din (10.4) la fiecare pas.

10.5.6 Exerciții pentru secțiunea 10.5

Exercițiul 10.5.1: Să presupunem că graficele sunt generate prin alegerea unei probabilități p și
alegând fiecare margine independent cu probabilitatea p, ca în Exemplul 10.21. Pentru
graficul din Fig. 10.20, ce valoare a lui p oferă probabilitatea maximă de a vedea
acel grafic? Care este probabilitatea generării acestui grafic?

Exercițiul 10.5.2: Calculați MLE pentru graficul din Exemplul 10.22 pentru urmărind presupunerile apartenenței celor două comunități.
(a) C = {w, x}; C = {y, z}.
(b) C = {w, x, y, z}; C = {x, y, z}.
Exercițiul 10.5.3: Exemplul 10.22 a considerat atribuirea inițială a nodurilor către două comunități C = {w, x, y} și D = {w, y, z}. Să presupunem că păstrăm numărul de comunități fixate la 2, dar permit modificări incrementale ale comunităților care adaugă sau șterge un singur nod. Care ar fi atribuirea finală a nodurilor către comunitățile C și D care maximizează probabilitatea de a vedea graficul din Fig. 10.20? Care este această probabilitate?
! Exercițiul 10.5.4: Există alte atribuții de noduri către (orice număr de) comunități pentru Fig. 10.20 care ar duce la aceeași probabilitate ca și soluție obținută pentru exercițiul 10.5.3?
Exercițiul 10.5.5: Să presupunem că avem o monedă, care poate să nu fie o monedă corectă și o întoarcem de câteva ori, văzând h capete și t cozi.
(a) Dacă probabilitatea p de a obține un cap pe orice flip este p, care este MLE pentru p, în termeni de h și t?
! (b) Să presupunem că ni se spune că există o probabilitate de 90% ca moneda să fie corectă (adică, p = 0,5) și o șansă de 10% ca p = 0,1. Pentru ce valori ale h și Este mai probabil ca moneda să fie corectă?
!! (c) Să presupunem că probabilitatea a-priori că p are o anumită valoare este proporțională până la | p - 0,5 |. Adică, p este mai probabil să fie aproape de 0 sau 1, decât aproximativ 1/2. Dacă vedem h capete și t cozi, care este estimarea maximă a probabilității p?

10.6 Simrank

În această secțiune, vom aborda o altă abordare a analizei rețelelor sociale grafice. Această tehnică, numită „simrank”, se aplică cel mai bine graficelor cu mai multe tipuri de noduri, deși în principiu poate fi aplicat oricărui grafic. The Scopul simrank este de a măsura similitudinea dintre nodurile acestuia tastați și face acest lucru văzând unde se termină mersul aleatoriu pe grafic când începeți de la un anumit nod, nodul sursă. Pentru că calculul trebuie pentru fiecare nod sursă, există o limită a dimensiunii graficelor care poate fi analizat complet în acest mod. Cu toate acestea, vom oferi și noi un algoritm care se apropie de simrank, dar este mult mai eficient decât iterat multiplicare matrice-vector. În cele din urmă, arătăm cum poate fi folosit simrank pentru a găsi comunitățile.

Pagina 410

10.6.1 Random Walkers pe un grafic social

Reamintim punctul nostru de vedere al PageRank din secțiunea 5.1 ca reflectând ceea ce este un „surfer aleatoriu” ar face dacă ar merge pe graficul web. În mod similar ne putem gândi la o fiul „mergând” pe o rețea socială. Graficul unei rețele sociale este în general nedirecționat, în timp ce graficul web este direcționat. Cu toate acestea, diferența este portant. Un walker la un nod N al unui grafic nedirecționat se va deplasa cu egal probabilitate pentru oricare dintre vecinii lui N (acele noduri cu care N împarte un margine).
Să presupunem, de exemplu, că un astfel de walker începe la nodul T_1 din Fig. 10.2, pe care le reproducem aici ca Fig. 10.21. La primul pas, ar merge fie la U_1 sau W_1. Dacă la W_1, atunci ar reveni fie la T_1, fie la T_2. Dacă Walker-ul s-a mutat mai întâi la U_1, ar fi încheiat fie la T_1, T_2, fie la T_3 în continuare.
U
U
T
T
T
T
W
W
W
1
1
1
2
2
2
3
3

4
Figura 10.21: Repetați figura 10.2
Concluzionăm că, începând cu T_1, există șanse mari pe care ar merge-o vizitați T_2, cel puțin inițial, și acea șansă este mai bună decât șansa pe care ar avea-o vizitați T_3 sau T_4. Ar fi interesant dacă am putea deduce că etichetele T_1 și T_2 sunt deci legat sau similar într-un fel. Dovada este că le au pe amândouă au fost plasate pe o pagină Web comună, W_1, și au fost, de asemenea, utilizate de un etichetator comun, U_1.
Cu toate acestea, dacă permitem mersului să continue parcurgerea graficului la întâmplare, atunci probabilitatea ca mersul să se afle la un anumit nod nu există depinde de unde începe. Această concluzie vine din teoria lui Markov procesele menționate în secțiunea 5.1.2, deși independența față de punctul de plecare necesită condiții suplimentare pe lângă conexiunea pe care graficul din Fig. 10.21 satisface.

10.6.2 Plimbări aleatorii cu repornire

Din observațiile de mai sus vedem că nu este posibil să se măsoare similar-la un anumit nod uitându-se la distribuția limitativă a mersului.
Cu toate acestea, am văzut deja, în secțiunea 5.1.5, introducerea unui mic probabilitatea ca mersul să înceteze să meargă la întâmplare. Mai târziu, am văzut în Sec-5.3.2 că au existat motive pentru a selecta doar un subset de pagini Web ca fișier set de teleportare, paginile la care mergea mersul când se opreau din navigare Web-ul la întâmplare.
Aici, ducem această idee la extrem. Ne concentrăm pe un anume nodul sursă S al unei rețele sociale și vrem să vedem unde este mersul aleator sfârșește pe scurtele plimbări de la S. Pentru a afla unde ar tinde să fie mersul, modificăm matricea probabilităților de tranziție pentru a avea o probabilitate mică de trecând înapoi la S de la orice nod. Adică, vom calcula subiectul-calcul PageRank sensibil cu un set de teleportare format doar din nod S. În mod formal, fie M matricea de tranziție a graficului G. Adică intrarea în rândul i și coloana j a lui M este 1 / k dacă nodul j al lui G are gradul k și unul dintre nodurile adiacente este i. În caz contrar, această intrare este 0. Vom discuta despre teleportare în un moment, dar mai întâi, să ne uităm la un exemplu simplu de matrice de tranziție.
Exemplul 10.23: Figura 10.22 este un exemplu de rețea foarte simplă care implică trei imagini și două etichete, „Cer” și „Arborele” care au fost plasate pe unele dintre ei. Imaginile 1 și 3 au ambele etichete, în timp ce Imaginea 2 are doar eticheta "Cer." Intuitiv, ne așteptăm ca imaginea 3 să fie mai asemănătoare cu imaginea 1 decât Imaginea 2 este și o analiză folosind un walker aleatoriu cu repornire la Imaginea 1 va susține acea intuiție.
Imaginea 1
Imaginea 2
Poza 3
Copac
Cer
Figura 10.22: Un grafic social bipartit simplu
Să comandăm nodurile ca Imaginea 1, Imaginea 2, Imaginea 3, Cerul, Arborele. Apoi matricea de tranziție pentru graficul din Fig. 10.22 este
□
□
□
□
□
□
0
0
0
1/3 1/2
0
0
0
1/3
0
0
0

0
1/3 1/2
1/2 1 1/2
0
0
1/2 0 1/2
0
0
□
□
□
□
□
□
De exemplu, a patra coloană corespunde nodului „Sky” și acestui nod
se conectează la fiecare dintre nodurile imaginii arborelui. Prin urmare, are gradul trei, deci

---

Pagina 412

intrările diferite de zero în coloana sa trebuie să fie 1/3. Nodurile de imagine corespund
primele trei rânduri și coloane, astfel încât intrarea 1/3 apare în primele trei rânduri
din coloana 4. Deoarece nodul „Sky” nu are o margine nici pentru sine, nici pentru
Nodul „Tree”, intrările din ultimele două rânduri ale coloanei 4 sunt 0. ✷
Ca și înainte, să folosim βca probabilitate ca mersul să continue la
dom, deci 1 - βeste probabilitatea ca walker-ul să se teleporteze la nodul sursă S.
Fie e s vectorul coloanei care are 1 în rând pentru nodul S și 0 în altă parte.
Atunci dacă v este vectorul coloană care reflectă probabilitatea, walker-ul este la fiecare
a nodurilor la o anumită rundă și v ′ este probabilitatea la care se află mersul
fiecare dintre nodurile din runda următoare, atunci v ′ este legat de v prin:
v ′ = βMv + (1 - β)e s
Exemplul 10.24: Să presupunem că M este matricea din Exemplul 10.23 și β= 0.8. De asemenea,
presupunem că nodul S este imaginea 1; adică vrem să calculăm similaritatea
a altor imagini la Imaginea 1. Apoi ecuația pentru noua valoare v ′ a
distribuția pe care trebuie să o repetăm este
v ′ =
□
□
□
□
□
□
0
0
0
15.04.2 / 5
0
0
0
15.04
0
0
0
0
15.04.2 / 5
2/5 4/5 2/5
0
0
2/5
0
2/5
0
0
□

□
□
□
□
□
v +
□
□
□
□
□
□
1/5
0
0
0
0
□
□
□
□
□
□
Deoarece graficul din Fig. 10.22 este conectat, matricea originală M este stocazică tic și putem deduce că dacă vectorul inițial v are componente care însumează 1, atunci v ′ va avea și componente care însumează 1. Ca rezultat, putem simplifica ecuația de mai sus adăugând 1/5 la fiecare dintre intrările din primul rând al matrice. Adică putem itera multiplicarea matrice-vector
v ′ =
□
□
□
□
□
□
1/5 1/5 1/5 7/15 3/5
0
0
0
15.04
0
0
0
0
15.04.2 / 5
2/5 4/5 2/5
0
0
2/5
0
2/5
0
0
□
□
□
□
□
□
v

Dacă începem cu v = e s , atunci succesiunea estimărilor distribuției lui mersul pe care îl obținem este

□

□

□

□

□

□

1

0

0

0

0

□

□

□

□

□

□

,

□

□

□

□

□

□

1/5

0

0

2/5

2/5

□

□

□

□

□

□

,

□

□

□

□

□

□

35/75

8/75

20/75

6/75

6/75

□

□

□

□

□

□

,

□

□
□
□
□
□
95/375
8/375
20/375
142/375
110/375
□
□
□
□
□
□
,
□
□
□
□
□
□
2353/5625
568/5625
1228/5625
786/5625
690/5625
□
□
□
□
□
□
, ...,
□
□
□
□
□
□
.345
.066
.145
.249
.196
□
□
□
□
□
□
Observăm din cele de mai sus că în limită, mersul este mai mult decât dublu probabil să fie la imaginea 3 decât la imaginea 2. Această analiză confirmă intuiția că Imaginea 3 seamănă mai mult cu Imaginea 1 decât Imaginea 2. ✷

Există mai multe observații suplimentare pe care le-am putea lua de la Ex-
amplu 10.24. În primul rând, amintiți-vă că această analiză se aplică doar pentru imaginea 1. Dacă noi
am vrut să știm ce imagini seamănă cel mai mult cu o altă imagine, aș vrea
trebuie să încep analiza pentru imaginea respectivă. La fel, dacă am vrut să știm
despre ce etichete au fost cele mai strâns asociate cu eticheta „Sky” (un
în acest mic exemplu, deoarece există doar o altă etichetă), atunci
ar trebui să aranjăm ca walkerul să se teleporteze doar la nodul „Sky”.
În al doilea rând, observați că convergența durează timp, deoarece există o oscilare inițială
lation. Adică, inițial, toată greutatea este la imagini, iar la a doua
pasul cea mai mare parte a greutății este la etichete. La al treilea pas, cea mai mare greutate a revenit
la imagini, dar la al patrulea pas o mare parte din greutate se mută la etichete
din nou. Cu toate acestea, în limită există convergență, cu 5/9 din greutatea la
imagini și 4/9 din greutatea la etichete. În general, procesul converge
pentru orice grafic k-partit conectat.

10.6.3 Simrank aproximativ

Calculul simplu al simrank implică repetarea matrice-vector mul-
tiplications, la fel ca pentru orice calcul de tip PageRank. Cu toate acestea, atunci când există
un singur nod S din setul de teleportare, nodurile care sunt departe de S sunt sigur de obținut
un simrank foarte mic. Dacă suntem dispuși să acceptăm o aproximare la adevărat
simrank, putem folosi o altă abordare, în care niciodată nu ne uităm niciodată la noduri
care sunt departe de S.
Vom simula un alt tip de mers pe grafic. În loc de mers
mutarea către un vecin aleatoriu (dacă nu se teleportează către nodul sursă), acesta
se va muta sau va rămâne acolo unde este, cu probabilitate egală. Dacă decide
mutați, apoi se va muta la orice vecin cu probabilitate egală. Cu toate acestea, acolo
este un alt aspect al deciziei dacă se mută sau nu. Putem privi un
walker ca parte a PageRank-ului limitativ pentru nodul său sau parte a
PageRank rezidual, adică PageRank care nu a fost încă atribuit niciunei
nodul. Numai în ultimul caz luăm în considerare mutarea acelui mers.
Prin urmare, vom folosi doi vectori, r și q, fiecare cu o componentă pentru
fiecare nod. În cele ce urmează, r vor fi estimările noastre actuale pentru PageRank
a fiecărui nod și q va fi PageRank-ul rezidual curent pentru fiecare nod. Cum-
Rețineți că, pe măsură ce calculul continuă, este posibil să nu calculăm niciodată multe dintre
componentele acestor vectori. De asemenea, vom folosi notația q_U pentru
componentă a q pentru nodul U și utilizați r_U în mod analog. Initailly, presupunem că r este
vectorul all-0, în timp ce q este tot 0, cu excepția $q_S = 1$. Adică, q spune că noi
presupunem că toți plimbătorii încep de la S, în timp ce r spune că nu am rezolvat încă unde
oricare dintre ele se va termina asimptotic.
Calculul simrank aproximativ utilizează doi parametri. Continuăm să
utilizați βca fracție netaxată; adică, $1 - \beta$este probabilitatea ca un walker să o facă
teleportați la nodul sursă S. Al doilea parametru de care avem nevoie este ϱ, un mic
fracțiune care oferă cantitatea maximă de PageRank rezidual pe care o vom obține
fii dispus să nu treci de la un nod la un vecin. Astfel, smaller mai mic este,

mai aproape ne apropiem de adevăratul PageRank, dar și cu cât durează mai mult calculul.
Ar trebui să alegem ϱ pentru a fi foarte mic, să zicem 0,01 împărțit la numărul de margini
în grafic. În acest fel, nu încetăm decât până la cantitatea totală de reziduuri
PageRank este cel mult 0,01. Algoritmul aproximativ-simrank este de a repeta
urmând pașii până când nu mai putem găsi niciun nod U al cărui PageRank rezidual
împărțit la gradul său (de exemplu, cantitatea de PageRank pe care ar putea să o transmită oricăruia dintre
vecinii săi) este mai mare decât ϱ.
1. Fie U orice nod astfel încât q_U , componenta lui q pentru nodul U, să depășească ϱ
de ori gradul de U.
2. Add $(1 - \beta)q_U$ la valoarea curentă a r_U . Adică, credităm nodul U cu
o sumă de PageRank adevărat egală cu impozitul pe reziduul său curent.
3. Setați q_U la $\beta q_U / 2$. Adică, din porțiunea de reziduu U care nu este impozitată,
jumătate din acest reziduu rămâne la U.
4. Să presupunem că U are gradul d. Apoi pentru fiecare nod V adiacent lui U, adăugați la
q_V o cantitate egală cu $\beta q_U / (2d)$. Adică cealaltă jumătate a celor neimpozitați
rezidual pentru U este împărțit în mod egal între vecinii săi.
Exemplul 10.25: Să reconsiderăm exemplul 10.23 din punct de vedere

de PageRank aproximativ, folosind din nou Picture1 ca nod sursă S. We
vom folosi din nou β= 0,8, dar vom lăsa ϱ nespecificat, deoarece nu suntem
urmează să efectueze calculul până la finalizare pentru orice ably rezonabil de mic.
Figura 10.23 rezumă care ar putea fi primii cinci pași ai algoritmului.

U	r_{P1}	r_{P2}	r_{P3}	r_{Sk}	r_{Tr}	$'q_{P1}$	q_{P2}	q_{P3}	q_{Sk}	q_{Tr}
Inițială	0	0	0	0	0	1	0	0	0	0
Imaginea1	.2	0	0	0	0	.4	0	0	.2	.2
Copac	.2	0	0	0	.04	.44	0	.04	.2	.08
Imagine1	.288	0	0	0	.04	.176	0	.04	.288	.168
Cer	.288	0	0	.058	.04	.214	.038	.078	.115	.168
Poza3	.288	0	.016	.058	.04	.214	.038	.031	.131	.184

Figura 10.23: Primii cinci pași ai unui calcul PageRank aproximativ
Primul rând, etichetat „Initial”, arată valorile inițiale ale lui r și q. Noi arătăm
fiecare componentă pentru fiecare vector, în aceeași ordine ca în Exemplul 10.23: Imagini
1, 2 și 3, apoi Sky și Tree. Prima noastră alegere de U trebuie să fie Picture1,
nod sursă, deoarece este singurul nod cu PageRank rezidual. În pasul (1)

convertim (1 - β)q_{P1} din reziduul pentru Picture1 pentru a aproxima PageRank.
Astfel, r_{P1} este setat la 0,2. Din valoarea rămasă 0,8 pentru q_{P1}, rămâne 0,4 și
celălalt 0.4 este împărțit în mod egal între vecinii Sky și Tree of Picture1.
Aceste observații explică al doilea rând al tabelului.

Pagina 415

Să presupunem că vom alege apoi Arborele ca următoare valoare a nodului U. Valoarea lui
q_{Tr} este acum 0,2, deci mutăm o cincime din această valoare la r_{Tr}, ceea ce face asta
valoarea să fie 0,04. Din restul de 0,16, jumătate rămâne ca q_{Tr}, iar restul este
împărțit între q_{P1}, care devine 0,44 și q_{P3}, care devine 0,04. Aceste
pașii explică al treilea rând. Arătăm o succesiune de trei pași suplimentari, fiecare
cu o posibilă alegere a lui U, presupunând că Q este suficient de mare pentru a justifica continuarea
algoritmul. Sunteți invitați să lucrați singuri și următorii trei pași
continuați simularea, care este afișată doar cu trei zecimale. ✷

10.6.4 De ce funcționează aproximativ Simrank

Există mai multe întrebări pe care le puteți pune în mod natural despre algoritmul de-
descris în secțiunea anterioară.
1. De ce este acest algoritm mai eficient decât implementarea simplă
tarea PageRank cu un singur nod sursă ca set de teleportare?
2. De ce converge acest algoritm la o aproximare a adevăratului sim-
rang?
3. De ce distribuim doar jumătate din valoarea rezidială pentru nodul U, mai degrabă decât
întregul reziduu?
Răspunsul la ultima întrebare este cel mai simplu. Este acolo pentru a ajuta la distribuirea fișierului
valorile reziduale cât mai repede posibil. Pentru unele grafice, este bine să omiteți pasul (3)
și în etapa (4) pentru a da fiecărui nod V un reziduu suplimentar de βq_U / d. In orice caz,
în alte cazuri, cum ar fi un grafic bipartit, distribuind întregul reziduu al nodului
U va face ca reziduurile să sară înainte și înapoi între cele două părți ale
graficul, provocând astfel să existe cel puțin un reziduu mare pentru o perioadă mai lungă de timp,
decât dacă am distribui doar jumătate din reziduuri.
Pentru prima întrebare, observăm că fiecare nod care se califică ca posibil
alegerea lui U are $q_U > Q$; de obicei va fi considerabil mai mare decât Q deoarece
limita inferioară reală pe q_U este de Q ori gradul de U. Observați că dacă $q_U > Q$,
atunci o cantitate de cel puțin Q (1 - β)este mutată din vectorul rezidual q în
vectorul PageRank aproximativ r. Cu toate acestea, suma tuturor componentelor
din ambii vectori este 1, deci nu putem avea un număr de runde ale acestui algoritm
care depășește

$$\frac{1}{Q(1-\beta)}$$

. Acesta este un număr constant de runde, în timp ce obișnuitul
Algoritmul PageRank necesită timp care este cel puțin proporțional cu numărul de
noduri în grafic ori numărul de iterații ale matricei-vector multiplica-
ziune. Mai mult, dacă suntem atenți, putem organiza componentele q în a
coadă prioritară, astfel încât să putem selecta un candidat pentru U în timpul O (jurnal n) dacă există
n noduri în grafic.
În cele din urmă, trebuie să abordăm a doua întrebare: de ce rezultatul aprox.
imita adevăratul simrank? Iată un argument intuitiv. În primul rând, mișcarea reziduală
valorile din jurul graficului sunt o simulare a mersului aleator pe grafic. The
faptul că umblătorii se mișcă asincron (deoarece se mișcă cu 50%

Pagina 416

probabilitatea, mai degrabă decât deplasarea cu siguranță la fiecare pas) nu este importantă. the
plimbătorii se vor distribui în cele din urmă așa cum ar face dacă ar lua un
mers aleatoriu sincron. Dar distribuția reziduurilor nu este răspunsul
că algoritmul dă. Mai degrabă, răspunsul este vectorul r atunci când algoritmul
se termină. De fapt, valorile care sunt mutate de la q la r sunt fracția 1 - βa
valorile reziduurilor la momente aleatorii, în funcție de ordinea în care
selectăm noduri pentru a fi U. Cu toate acestea, adevăratul PageRank pentru orice nod V este
probabilitatea ca un walker să se afle la nodul V la un moment dat aleatoriu, având în vedere că a început
la sursa S. Alegând U la întâmplare (printre noduri cu suficient
mare rezidual), simulăm selectarea unui timp aleator la care să
vizualizați fiecare walker și, prin urmare, obțineți aproximativ distribuția corectă
a umblătorilor.

10.6.5 Aplicarea Simrank la găsirea comunităţilor

O aplicaţie interesantă a simrank este găsirea comunităţilor în care a
nodul dat (nodul sursă) aparţine. Am putea găsi apoi tot ceea ce este rezonabil
comunităţi dintr-o reţea selectând ca noduri sursă acele noduri care au
care nu au fost încredinţate niciunei comunităţi sau care au fost atribuite doar unei
număr mic de comunităţi.
Să presupunem că începem cu nodul sursă S şi că găsim simrank-ul pentru toate nodurile
pentru această sursă. Am putea folosi algoritmul simrank exact sau aproximativ
pentru a face acest lucru. Apoi, ordonaţi nodurile, altele decât S, prin simrankul lor şi luaţi în considerare
adăugând fiecare, cel mai înalt simrank mai întâi, la o comunitate care conţine S. Trebuie
ştim când să nu mai adăugăm noduri, deci avem nevoie de un criteriu de oprire. O simpla
abordarea este de a adăuga noduri până la densitatea muchiilor - fracţia de margini
care conectează noduri din comunitate împărţit la numărul de margini posibile
conectarea nodurilor în comunitate (adică ($\binom{n}{2}$) pentru o comunitate cu n-nod) - merge
sub un prag.
Exemplul 10.26: Să luăm în considerare reţeaua foarte simplă din Fig. 10.22. Noi
vom folosi Picture1 ca nod sursă, aşa că încercăm să găsim comunităţi bune
care conţine acel nod. Reţeaua este prea mică pentru a oferi un set rezonabil de
comunităţi, dar va servi pentru un exemplu de algoritm.
Am aflat din Exemplul 10.23 că, cu Picture1 ca nod sursă,
valorile simrank pentru cele cinci noduri sunt în ordinea Picture1, Sky, Tree, Picture3,
Imaginea 2. Comunitatea formată din primele două noduri din această listă are un
densitatea de 1, deoarece unica margine posibilă între Picture1 şi Sky este într-adevăr
prezent. Când adăugăm al treilea nod, Tree, densitatea scade la 2/3, doar din moment ce
sunt prezente două dintre cele trei margini posibile. Adăugarea celui de-al patrulea nod, Picture3,
menţine densitatea la 2/3; acum sunt prezente patru din cele şase margini posibile. Dacă noi
adăugaţi ultimul nod, densitatea este până la 1/2 - cinci din cele zece margini posibile sunt
prezent în întreaga reţea. Dacă pragul nostru de densitate este mai mare de 2/3,
atunci am alege doar primele două noduri ca comunitate, în timp ce dacă avem
un prag t, cu $2/3 \geq t > 1/2$, atunci am alege primele patru noduri ca
comunitatea. ✷

O altă măsură a bunătăţii unei comunităţi C este conductanţa. La
definim conductanţa, trebuie mai întâi să definim volumul unei comunităţi C, care
este cea mai mică dintre suma scăderii nodurilor din C şi suma lui
grade ale nodurilor care nu se află în C. Atunci conductanţa lui C este raportul dintre
numărul de muchii cu exact un capăt în C împărţit la volumul lui C.
ideea intuitivă din spatele conductanţei este că arată seturile C care nu sunt nici ele
mici sau prea mari şi totuşi pot fi deconectate de la reţea prin tăiere
un număr relativ mic de margini. Conductanţa mică sugerează o legătură strânsă
comunitate.
Când luăm în considerare secvenţa comunităţilor formate prin adăugarea unui nod
la un moment dat, în ordinea lor simplă, găsim adesea că există mai multe puncte
în secvenţa în care conductanţa este un minim local. Acestea sunt cele mai bune
alegeri pentru o comunitate care conţine nodul sursă. Dacă sunt mai multe
minim local, atunci există mai multe comunităţi, una cuibărită în alta,
de care se poate spune că aparţine nodul sursă.
Exemplul 10.27: Să repetăm exemplul 10.26, dar folosiţi conductanţa ca
criteriu pentru bunătatea unei comunităţi. Reamintim ordinea în care adăugăm noduri
este Picture1, Sky, Tree, Picture3, Picture2. Fie C_i comunitatea care conţine
primul i al acestor noduri, şi lăsaţi c_{am} fi conductibilitatea acestei comunităţi.
Prima posibilitate este C_1 care constă din numai nodul Picture1. Sunt
două muchii cu exact un capăt în C_1 . Volumul C_1 este cea mai mică dintre 2 (The
gradul unui singur nod) şi 8 (suma gradelor celorlalte patru noduri.
Astfel, $c_1 = 2 / \min(2, 8) = 1$.
Acum ia în considerare C_2 = {Imagine1, Cer}. Există trei margini care au exact
un capăt în C_2 . Volumul lui C_2 este cel mai mic dintre suma gradelor
două noduri în C_2 (i, e., 2 + 3 = 5) şi suma gradelor celorlalte trei
noduri, care este şi 5. Astfel, $c_2 = 3 / \min(5, 5) = 3/5$.
Acum, luaţi în considerare adăugarea Tree pentru a forma C_3 = {Picture1, Sky, T ree}. Numărul
ber de margini cu exact un capăt în C_3 este trei. Suma gradelor
trei noduri sunt 2 + 3 + 2 = 7, în timp ce suma gradelor celorlalte două noduri
este trei. Astfel, $c_3 = 3 / \min(7, 3) = 1$. Puteţi continua această secvenţă şi găsiţi
că $c_4 = 1 / \min(9, 1) = 1$. În acest caz, există un singur minim şi că

este c_2 = 3/5. Astfel, concluzionăm că „cea mai bună” comunitate care conține Picture1
este C_2 = {Imagine1, Cer}. ✷

10.6.6 Exerciții pentru secțiunea 10.6

Exercițiul 10.6.1: Dacă, în Fig. 10.22 porniți mersul din Imaginea 2, ce va fi
să fie asemănarea cu imaginea 2 a celorlalte două imagini? La ce te aștepți
pentru a fi mai asemănător cu imaginea 2?
Exercițiul 10.6.2: Dacă, în Fig. 10.22, începeți mersul din Imaginea 3, ce faceți?
te aștepți să fie asemănarea cu celelalte două imagini?

! Exercițiul 10.6.3: Repetați analiza exemplului 10.24 și calculați
asemănări ale imaginii 1 cu celelalte imagini, dacă sunt următoarele modificări
realizat în Fig. 10.22:
(a) Eticheta „Tree” este adăugată la imaginea 2.
(b) O a treia etichetă „Apă” este adăugată la Imaginea 3.
(c) O a treia etichetă „Apă” este adăugată atât la Imaginea 1, cât și la Imaginea 2.
Notă: modificările se fac independent pentru fiecare parte; nu sunt cumulati
tive.
Exercițiul 10.6.4: Continuați simularea începută în Fig. 10.23, presupunând că
următoarele trei valori pentru U sunt Picture1, Sky și Tree.
Exercițiul 10.6.5: Care este cea mai mică valoare a lui ϱ pentru care putem spune asta
Fig. 10.23 este executarea completă a algoritmului PageRank aproximativ?
!! Exercițiul 10.6.6: Să presupunem că rețeaua noastră constă doar din două noduri cu un
marginea dintre ele. Fie ca nodurile să fie S (sursa, la care apar teleporturile)
și T, celălalt nod.
(a) Să presupunem că rulăm o versiune modificată a altor-PageRank aproximativ
goritmul secțiunii 10.6.3, unde toate (mai degrabă decât jumătate) din cei neimpozitați
reziduul nodului selectat este distribuit vecinilor săi (în acest caz,
către un singur vecin). Care este secvența de valori pentru vectorii r și
q rezultă? Rețineți că, în acest caz, alegerea lui U este deterministă,
deoarece la orice pas doar unul dintre cele două noduri are PageRank rezidual. Pentru
foarte mic ϱ, care este limita vectorului PageRank aproximativ?
(b) Să presupunem că, pentru aceeași rețea, executăm algoritmul original,
unde doar jumătate din reziduuri sunt distribuite vecinului. Ce este
limitând PageRank în acest caz, presupunând că ϱ este mic?
Exercițiul 10.6.7: Pentru rețeaua din Fig. 10.22, calculați simrank atunci când
nodul sursă este Tree. Apoi găsiți cea mai bună alegere a comunităților care conțin
Arborele dacă cerința dvs. pentru o comunitate este
(a) Densitate cel puțin 2/3.
(b) Minimul local de conductivitate.
! Exercițiul 10.6.8: Să presupunem că rețeaua noastră este formată din noduri dintr-un singur lanț, adică
este, nodurile U_1, U_2, ..., U_n, cu o margine între U_i și U_{i+1}, pentru $1 \leq i < n$.
(a) În funcție de k, calculați densitatea „comunității” constând din
primele k noduri: {U_1, U_2, ..., U_k }.

(b) În funcție de k, calculați conductanța comunității constând
a primelor k noduri.
(c) Folosind densitatea ca măsură a bunătății pentru o comunitate, care este
cea mai bună valoare a lui k?
(d) Utilizarea conductanței ca măsură a bunătății pentru o comunitate, ce este
cea mai bună valoare a lui k?

10.7 Numărarea triunghiurilor

Una dintre cele mai utile proprietăți ale graficelor din rețelele sociale este numărul de tri-
unghiuri și alte subgrafe simple. În această secțiune vom prezenta metode pentru
estimarea sau obținerea unui număr exact de triunghiuri într-un grafic foarte mare. Noi suntem
gin cu o motivație pentru astfel de numere și apoi dați câteva metode de numărare
eficient.

10.7.1 De ce să numărăm triunghiurile?

Dacă începem cu n noduri și adăugăm m margini la un grafic la întâmplare, vor exista
un număr așteptat de triunghiuri în grafic. Putem calcula acest număr

fără prea mari dificultăţi. Există (n 3) seturi de trei noduri, sau aproximativ $n^3/6$ seturi de trei noduri care ar putea fi un triunghi. Probabilitatea unei margini între oricare două noduri date adăugate este $m / (n^2)$, sau aproximativ $2m / n^2$.
Probabilitatea ca orice set de trei noduri să aibă margini între fiecare pereche, dacă aceste margini sunt alese independent pentru a fi prezente sau absente este aproximativ $(2m / n^2)^3 = 8m^3 / n^6$. Astfel, numărul aşteptat de triunghiuri într-un grafic de n noduri şi m muchii selectate aleatoriu este de aproximativ $(8m^3 / n^6)(n^3/6) = \frac{4}{3}(m / n)^3$.
Dacă un grafic este o reţea socială cu n participanţi şi m perechi de „prieteni” ne-am aştepta ca numărul triunghiurilor să fie mult mai mare decât valoarea pentru un grafic aleatoriu. Motivul este că dacă A şi B sunt prieteni, iar A este şi un prieten cu C, ar trebui să existe o şansă mult mai mare decât media ca B şi C sunt şi prieteni. Astfel, numărarea numărului de triunghiuri ne ajută să măsurăm măsura în care un grafic arată ca o reţea socială.
De asemenea, putem privi comunităţile din cadrul unei reţele sociale. A fost a demonstrat că vârsta unei comunităţi este legată de densitatea triunghiurilor. Adică, atunci când un grup tocmai s-a format, oamenii îşi atrag prietenii cu gânduri similare, dar numărul triunghiurilor este relativ mic. Dacă A aduce în prieteni B şi C, se poate ca B şi C să nu se cunoască. Ca comunitate se maturizează, B şi C pot interacţiona din cauza apartenenţei lor la comunitate. Astfel, există mari şanse ca la un moment dat triunghiul {A, B, C} să fie efectuat.

Pagina 420

10.7.2 Un algoritm pentru găsirea triunghiurilor

Vom începe studiul nostru cu un algoritm care rulează cel mai rapid posibil timp pe un singur procesor. Să presupunem că avem un grafic de n noduri şi $m \geq n$ margini. Pentru comoditate, presupuneţi că nodurile sunt numere întregi 1, 2, ..., n. Numiţi un nod un bătător greu dacă gradul său este de cel puţin $\sqrt{m}$. Un bătător greu triunghiul este un triunghi al cărui trei noduri sunt lovitori grei. Folosim separat algoritmi de numărare a triunghiurilor puternice şi a tuturor celorlalte triunghiuri. Reţineţi că numărul de noduri puternice nu este mai mare de $2\sqrt{m}$, deoarece altfel suma din gradele nodurilor de hitter grele ar fi mai mari de 2m. Din moment ce fiecare edge contribuie la gradul de doar două noduri, atunci ar trebui să existe mai mult de m margini.
Presupunând că graficul este reprezentat de muchiile sale, vom prelucra graficul ca urmează:
1. Calculaţi gradul fiecărui nod. Această parte necesită doar examinarea fiecare margine şi adăugaţi 1 la numărul fiecăruia dintre cele două noduri ale sale. Timpul total necesar este O (m).
2. Creaţi un index pe margini, cu perechea de noduri la capetele sale ca cheie. Adică, indexul ne permite să determinăm, având în vedere două noduri, dacă marginea dintre ele există. O masă hash este suficientă. Poate fi construit în O (m) timp şi timpul aşteptat pentru a răspunde la o întrebare despre existenţă a unei margini este o constantă, cel puţin în sensul aşteptat. [2]
3. Creaţi un alt index de margini, acesta cu cheie egal cu un singur nod. Dat fiind un nod v, putem prelua nodurile adiacente lui v în proporţie de timp proporţional cu numărul acestor noduri. Din nou, o masă hash, de data aceasta cu noduri simple ca cheie, sunt suficiente în sensul aşteptat.
Vom ordona nodurile după cum urmează. În primul rând, ordonaţi nodurile în funcţie de grad. Atunci, dacă v şi u au acelaşi grad, reamintim că atât v cât şi u sunt numere întregi, deci ordine le numeric. Adică spunem $v < u$ dacă şi numai dacă oricare
(i) Gradul de v este mai mic decât gradul de u, sau
(ii) Gradele lui u şi v sunt aceleaşi şi $v < u$.
Triunghiuri Heavy-Hitter: Există doar $O(\sqrt{m})$ noduri heavy-hitter, deci noi poate lua în considerare toate seturile a trei dintre aceste noduri. Există $O(m^{3/2})$ posibile grele-hitter şi folosind indexul de pe margini putem verifica dacă există toate cele trei margini în timpul O (1). Prin urmare, este nevoie de timp $O(m^{3/2})$ pentru a găsi toţi cei care lovesc greu triunghiuri.

[2] Astfel, din punct de vedere tehnic, algoritmul nostru este optim numai în sensul timpului de funcţionare aşteptat, nu în cel mai rău caz timpul de funcţionare. Cu toate acestea, hashingul unui număr mare de articole are o extremă probabilitate mare de a ne comporta conform aşteptărilor şi dacă s-a întâmplat să alegem un hash funcţie care a făcut ca unele găleţi să fie prea mari, am putea relua până vom găsi o funcţie hash bună.

Alte triunghiuri: găsim celelalte triunghiuri într-un mod diferit. Luați în considerare fiecare muchie (v_1, v_2). Dacă atât v_1 cât și v_2 sunt lovitori grei, ignorați această margine. Presupune, totuși, v_1 nu este un atacator greu și, în plus, $v_1 < v_2$. Fie u_1, u_2, ..., u_k fie nodurile adiacente v_1. Rețineți că $k < \sqrt{m}$. Putem găsi aceste noduri, folosind indexul pe noduri, în timpul O (k), care este cu siguranță timpul O ($\sqrt{m}$). Pentru fiecare u_i putem folosi primul index pentru a verifica dacă muchia (u_i, v_2) există în timpul O (1). De asemenea, putem determina gradul de u_i în O (1) timp, pentru că am numărat toate gradele nodurilor. Numărăm triunghiul {v_1, v_2, u_i} dacă și numai dacă muchia (u_i, v_2) există și $v_1 < u_i$. În acest fel, un triunghi este numărat o singură dată - când v_1 este nodul triunghiului care precede ambele noduri ale triunghiului conform ordinii <. Astfel, timpul de procesare a tuturor nodurilor adiacente la v_1 este O ($\sqrt{m}$). Deoarece există m muchii, timpul total petrecut numărând altele triunghiurile este O ($m^{3/2}$).

Acum vedem că preprocesarea durează O (m) timp. Timpul pentru a găsi hitter triunghiuri este O ($m^{3/2}$), la fel și timpul pentru a găsi celelalte triunghiuri. Prin urmare, timpul total al algoritmului este O ($m^{3/2}$).

10.7.3 Optimitatea algoritmului de găsire a triunghiului

Se pare că algoritmul descris în secțiunea 10.7.2 este, într-un ordinea de mărime cel mai bine posibil. Pentru a vedea de ce, luați în considerare un grafic complet pe n noduri. Acest grafic are m = ($\binom{n}{2}$) muchii și numărul triunghiurilor este ($\binom{n}{3}$). Deoarece nu putem enumera triunghiuri în mai puțin timp decât numărul acestora triunghiuri, știm că orice algoritm va dura $\Omega(n^3)$ timp pe acest grafic. In orice caz, m = O (n^2), deci orice algoritm ia timp $\Omega(m^{3/2})$ pe acest grafic.

Ne-am putea întreba dacă ar exista un algoritm mai bun care să funcționeze mai rar grafice decât graficul complet. Cu toate acestea, putem adăuga la graficul complet un lanț de noduri cu orice lungime până la n^2. Acest lanț nu mai adaugă triunghiuri. Nu mai mult decât dublează numărul de margini, dar face numărul de noduri la fel de mare pe cât ne place, de fapt scăderea raportului dintre margini și noduri pentru a fi la fel de apropiate la 1 după cum ne place. Deoarece există încă $\Omega(m^{3/2})$ triunghiuri, vedem că aceasta este mai mică reținери legate pentru întreaga gamă de rapoarte posibile de m / n.

10.7.4 Găsirea triunghiurilor folosind MapReduce

Pentru un grafic foarte mare, dorim să folosim paralelismul pentru a accelera calculul. Putem exprima găsirea triunghiului ca o îmbinare pe mai multe căi și putem folosi tehnica Secțiunea 2.5.3 pentru a optimiza utilizarea unei singure lucrări MapReduce pentru a număra triunghiuri. Se pare că această utilizare este cea în care tehnica de îmbinare multiplu a secțiunii respective este, în general, mult mai eficient decât să luați două îmbinări bidirecționale. Mai mult, timpul total de execuție al algoritmului paralel este în esență același cu timpul de execuție pe un singur procesor utilizând algoritmul Secțiunii 10.7.2.

Pentru început, presupuneți că nodurile unui grafic sunt numerotate 1, 2, ..., n. Noi folosiți o relație E pentru a reprezenta muchiile. Pentru a evita reprezentarea fiecărei muchii de două ori, presupunem că dacă E (A, B) este un tuplu al acestei relații, atunci nu numai că există

---

o margine între nodurile A și B, dar și, ca numere întregi, avem A <B. 3

Această cerință elimină, de asemenea, buclele (marginile de la un nod la sine), pe care noi presupunem, în general, că nu există oricum în graficele rețelelor sociale, dar care ar putea conduc la „triunghiuri” care într-adevăr nu implică trei noduri diferite.

Folosind această relație, putem exprima mulțimea triunghiurilor grafului a cărui muchiile sunt E prin îmbinarea naturală

E (X, Y) ⋈ E (X, Z) ⋈ E (Y, Z)

(10,5)

Pentru a înțelege această îmbinare, trebuie să recunoaștem că atributele relației E primește nume diferite în fiecare dintre cele trei utilizări ale lui E. Adică ne imaginăm există trei exemplare ale lui E, fiecare cu aceleași tupluri, dar cu un altul scheme. În SQL, această îmbinare ar fi scrisă folosind o singură relație E (A, B) ca urmează:

Selectați e1.A, e1.B, e2.B
DE LA E e1, E e2, E e3
UNDE e1.A = e2.A ȘI e1.B = e3.A ȘI e2.B = e3.B

În această interogare, atributele echivalente e1.A și e2.A sunt reprezentate în unirea noastră

prin atributul X. De asemenea, e1.B și e3.A sunt reprezentate fiecare de Y; e2.B și e3.B sunt reprezentate de Z.
Observați că fiecare triunghi apare o dată în această îmbinare. Triunghiul constând de noduri v_1, v_2 și v_3 este generat atunci când X, Y și Z sunt aceste trei noduri în ordinea numerică, adică X <Y <Z. De exemplu, dacă ordinea numerică a nodurile este $v_1 < v_2 < v_3$, atunci X poate fi doar v_1, Y este v_2 și Z este v_3.
Tehnica Secțiunii 2.5.3 poate fi utilizată pentru a optimiza îmbinarea Equa-ziunea 10.5. Reamintim ideile din Exemplul 2.15, unde am luat în considerare numărul a modurilor în care valorile fiecărui atribut ar trebui să fie hash. In prezent de exemplu, chestiunea este destul de simplă. Cele trei apariții ale relației E cu siguranță au aceeași dimensiune, deci prin simetrie, atributele X, Y și Z vor fi fiecare hash la același număr de găleți. În special, dacă hash noduri pentru a gălește b, atunci vor exista b^3 reductoare. Fiecare activitate Reduce este asociată cu o secvență din trei numere de găleată (x, y, z), unde fiecare dintre x, y și z este în intervalul 1 până la b.
Sarcinile Hartă împart relația E în atâtea părți câte sunt Hartă sarcini. Să presupunem că unei sarcini de hartă i se dă tuplul E (u, v) pe care să îl trimită anumitor Reduceți sarcinile. În primul rând, gândiți-vă la (u, v) ca la un tuplu al termenului de asociere E (X, Y). Noi poate hash u și v pentru a obține numerele cupei pentru X și Y, dar nu știm găleată la care Z hashes. Astfel, trebuie să trimitem E (u, v) la toate sarcinile Reducer care corespund unei secvențe de trei numere de cupă (h (u), h (v), z) pentru oricare dintre b gălețile posibile z.
[3] Nu confundați această simplă ordonare numerică a nodurilor cu ordinea < pe care noi discutate în secțiunea 10.7.2 și care au implicat gradele nodurilor. Aici, grade de nod nu sunt calculate și nu sunt relevante.

Dar același tuplu E (u, v) trebuie tratat și ca un tuplu pentru termen E (X, Z). Prin urmare, trimitem și tuplul E (u, v) la toate sarcinile Reduce care corespund unui triplu (h (u), y, h (v)) pentru orice y. În cele din urmă, tratăm E (u, v) ca pe un tuplul termenului E (Y, Z) și trimite acel tuplu tuturor Reducerea sarcinilor corespunzătoare la un triplu (x, h (u), h (v)) pentru orice x. Comunicarea totală necesară este astfel 3b perechi cheie-valoare pentru fiecare dintre m tuplurile relației de margine E. Adică costul minim de comunicare este O (mb) dacă folosim b^3 Reduceți sarcinile.
Apoi, să calculăm costul total de execuție la toate sarcinile de reducere. La fel de-presupunem că funcția hash distribuie marginile suficient de aleatoriu încât Reduceți sarcinile, fiecare obținând aproximativ același număr de margini. Din moment ce totalul numărul de margini distribuite către b^3 Reducerea sarcinilor este O (mb), rezultă că fiecare sarcina primește margini O (m / b^2). Dacă folosim algoritmul secțiunii 10.7.2 la fiecare Reduceți sarcina, calculul total la o sarcină este O ($(m / b^2)^{3/2}$) sau O ($m^{3/2} / b^3$).
Deoarece există b^3 Reducerea sarcinilor, costul total de calcul este exact O ($m^{3/2}$) în ceea ce privește algoritmul cu un singur procesor din secțiunea 10.7.2.

10.7.5 Utilizarea mai puținor sarcini de reducere

Printr-o ordonare judicioasă a nodurilor, putem reduce numărul sarcinilor de reducere cu aproximativ un factor de 6. Gândiți-vă la „numele” nodului i ca la pair (h (i), i), unde h este funcția hash pe care am folosit-o în secțiunea 10.7.4 la noduri hash la b găleți. Comandați nodurile după numele lor, luând în considerare numai prima componentă (adică, cupa la care nodul se hashează) și folosind numai a doua componentă pentru a rupe legăturile între nodurile care se află în aceeași găleată.
Dacă folosim această ordonare a nodurilor, atunci sarcina Reduce corespunzătoare lista găleților (i, j, k) va fi necesară numai dacă $i \le j \le k$. Dacă b este mare, atunci aproximativ 1/6 din toate b^3 secvențe de numere întregi, fiecare în intervalul 1 până la b, vor satisfac aceste inegalități. Pentru orice b, numărul acestor secvențe este $\binom{b+2}{3}$ (vezi Exercițiul 10.7.4). Astfel, raportul exact este (b + 2) (b + 1) / ($6b^2$).
Deoarece există mai puține reductoare, obținem o scădere substanțială a numărului a perechilor cheie-valoare care trebuie comunicate. În loc să trebuiască să le trimiteți pe fiecare din marginile m la 3b Reduceți sarcinile, trebuie să trimitem fiecare margine doar la b sarcini. În mod specific, considerați o margine e ale cărei două noduri sunt hash la i și j; aceste găleți ar putea fi la fel sau diferit. Pentru fiecare dintre valorile b ale lui k între 1 și b, ia în considerare lista formată din i, j și k în ordine sortată. Apoi sarcina Reduce care corespunde acestei liste necesită muchia e. Dar nu există alte sarcini de reducere necesită e.
Pentru a compara costul de comunicare al metodei acestei secțiuni cu acela din secțiunea 10.7.4, să stabilim numărul de sarcini de reducere, să zicem k. Apoi metoda din secțiunea 10.7.4 noduri hash la [3]

√k găleți și, prin urmare, comunică
3m 3
√k perechi cheie-valoare. Pe de altă parte, metoda acestei secțiuni hashează
noduri la aproximativ 3
√6k găleți, necesitând astfel m 3
√6 3
√k comunicare.
Astfel, raportul comunicării necesar metodei secțiunii 10.7.4
ceea ce este necesar aici este 3/3
√6 = 1,65.
Exemplul 10.28: Luați în considerare algoritmul simplu din secțiunea 10.7.4

Pagina 424

cu b = 6. Adică, există b 3 = 216 Reduceți sarcinile și comunicarea
costul este de 3 mb = 18 m. Nu putem folosi exact 216 Reducerea sarcinilor cu metoda
această secțiune, dar ne putem apropia foarte mult dacă alegem b = 10. Apoi, numărul
de Reducere sarcini este (12
3) = 220, iar costul comunicării este mb = 10m. Acea
este, costul comunicării este 5/9 din costul metodei simple.
✷

10.7.6 Exerciții pentru secțiunea 10.7

Exercițiul 10.7.1: Câte triunghiuri există în grafice:
(a) Figura 10.1.
(b) Figura 10.9.
! (c) Figura 10.2.
Exercițiul 10.7.2: Pentru fiecare dintre graficele exercițiului 10.7.1 determinați:
(i) Care este gradul minim pentru ca un nod să fie considerat un „bătător greu”?
(ii) Ce noduri sunt puternici?
(iii) Ce triunghiuri sunt triunghiuri cu greutate mare?
! Exercițiul 10.7.3: În acest exercițiu luăm în considerare problema găsirii pătratelor
într-un grafic. Adică vrem să găsim cvadrupluri ale nodurilor a, b, c, d astfel încât
patru grafice (a, b), (b, c), (c, d) și (a, d) există în grafic. Să presupunem graficul
este reprezentat de o relație E ca în secțiunea 10.7.4. Nu este posibil să scrieți un
unire unică a patru copii ale lui E care exprimă toate pătratele posibile din grafic,
dar putem scrie trei astfel de îmbinări. Mai mult, în unele cazuri, trebuie să urmăm
unirea printr-o selecție care elimină „pătratele” în cazul în care o pereche de opuse
colțurile sunt într-adevăr același nod. Putem presupune că nodul a este numeric
mai mici decât vecinii săi b și d, dar există trei cazuri, în funcție de dacă
c este
(i) De asemenea, mai mici decât b și d,
(ii) Între b și d sau
(iii) Mai mare decât b și d.
(a) Scrieți îmbinările naturale care produc pătrate care satisfac fiecare dintre cele trei
condițiile de mai sus. Puteți utiliza patru atribute diferite W, X, Y și Z,
și presupunem că există patru copii ale relației E cu scheme diferite,
deci îmbinările pot fi exprimate fiecare ca îmbinări naturale.
(b) Pentru care dintre aceste îmbinări avem nevoie de o selecție pentru a asigura opusul
colțurile sunt noduri cu adevărat diferite?

Pagina 425

!! (c) Să presupunem că intenționăm să folosim k Reducerea sarcinilor. Pentru fiecare dintre asocierile dvs. de la (a),
în câte găleți ar trebui să hash fiecare dintre W, X, Y și Z în ordine
pentru a minimiza costul comunicării?
(d) Spre deosebire de cazul triunghiurilor, nu este garantat faptul că fiecare pătrat este
redus o singură dată, deși putem fi siguri că fiecare pătrat este produs de
doar una dintre cele trei îmbinări. De exemplu, un pătrat în care se află cele două noduri
la colțurile opuse sunt fiecare mai mici numeric decât fiecare dintre celelalte două
nodurile vor fi produse numai de unirea (i). Pentru fiecare dintre cele trei îmbinări,
de câte ori produce deloc vreun pătrat pe care îl produce?
! Exercițiul 10.7.4: Arătați că numărul secvențelor întregi 1 ≤ i ≤ j ≤

$k \leq b$ este $\binom{b+2}{3}$. Sugestie: arată că aceste secvențe pot fi plasate într-un 1 la 1 corespondență cu șirurile binare de lungime b + 2 având exact trei 1.

10.8 Proprietăți de vecinătate ale graficelor

Există mai multe proprietăți importante ale graficelor care se referă la numărul de noduri la care se poate ajunge de la un nod dat de-a lungul unei căi scurte. In aceasta sectiune ne uităm la algoritmi pentru rezolvarea problemelor despre căi și cartiere pentru grafice foarte mari. În unele cazuri, soluțiile exacte nu sunt fezabile pentru grafice cu milioane de noduri. Prin urmare, ne uităm și la algoritmi de aproximare ca algoritmi exacți.

10.8.1 Grafice și vecinătăți direcționate

În această secțiune vom folosi un grafic direcționat ca model al unei rețele. Un regizat graficul are un set de noduri și un set de arce; acesta din urmă este o pereche de noduri scrise $u \rightarrow v$. Numim u sursa și v ținta arcului. Se spune că arcul este de la u la v.

Multe tipuri de grafice pot fi modelate prin grafice direcționate. Web-ul este un exemplu major, în cazul în care arcul $u \rightarrow v$ este un link de la pagina u la pagina v. Sau, arc $u \rightarrow v$ ar putea însemna că abonatul telefonic u a apelat abonatul v ultima lună. Pentru un alt exemplu, arcul ar putea însemna că u individual este după v individual pe Twitter. Într-un alt grafic, arcul ar putea însemna acea lucrare de cercetare u referă hârtia v.

Mai mult, toate graficele nedirecționate pot fi reprezentate prin grafice direcționate. În-în locul muchiei nedirecționate (u, v), utilizați două arcuri $u \rightarrow v$ și $v \rightarrow u$. Astfel, materialul din această secțiune se aplică și graficelor care sunt inerent nedirecționate, cum ar fi un grafic de prieteni într-o rețea socială.

O cale într-un grafic direcționat este o secvență de noduri $v_0, v_1, ..., v_k$ astfel încât există arce $v_i \rightarrow v_{i+1}$ pentru toate i = 0, 1, ..., k - 1. Lungimea acestei căi este k, numărul de arce de-a lungul căii. Rețineți că există k + 1 noduri pe o cale de lungime k, iar un nod în sine este considerat o cale de lungime 0.

Vecinătatea razei d pentru un nod v este mulțimea nodurilor u pentru care există o cale de lungime cel mult d de la v la u. Denotăm acest cartier de N (v, d). De exemplu, N (v, 0) este întotdeauna {v}, iar N (v, 1) este v plus setul de noduri la care există un arc de la v. Mai general, dacă V este un set de noduri, atunci N (V, d) este setul de noduri u pentru care există o cale de lungime d sau mai mică de la cel puțin un nod din mulțimea V.

Profilul de vecinătate al unui nod v este secvența de dimensiuni a vecinului său-hote | N (v, 1) |, | N (v, 2) |, ... Nu includem vecinătatea distanței 0, deoarece dimensiunea sa este întotdeauna 1.

A
B
D
E
G
F
C

Figura 10.24: Rețeaua noastră socială mică; gândiți-vă la el ca la un grafic direcționat

Exemplul 10.29: Luați în considerare graficul neorientat din Fig. 10.1, pe care îl re-produce aici ca Fig. 10.24. Pentru a-l transforma într-un grafic direcționat, gândiți-vă la fiecare marginea ca o pereche de arcuri, una în fiecare direcție. De exemplu, muchia (A, B) devine arcurile $A \rightarrow B$ și $B \rightarrow A$. Mai întâi, luați în considerare vecinătățile din nodul A. Știm N (A, 0) = {A}. Mai mult, N (A, 1) = {A, B, C}, de atunci sunt arcuri de la A doar la B și C. Mai mult, N (A, 2) = {A, B, C, D} și N (A, 3) = {A, B, C, D, E, F, G}. Cartierele pentru o rază mai mare sunt toate la fel ca N (A, 3).

Pe de altă parte, luați în considerare nodul B. Găsim N (B, 0) = {B}, N (B, 1) = {A, B, C, D} și N (B, 2) = {A, B, C, D, E, F, G}. Știm că B este mai mult central pentru rețea decât A, iar acest fapt este reflectat de vecinătate profilurile celor două noduri. Nodul A are profilul 3, 4, 7, 7, ..., în timp ce B are profil 4, 7, 7, ... Evident, B este mai central decât A, deoarece la fiecare distanță, este cartierul este cel puțin la fel de mare ca cel al lui A. De fapt, D este și mai central decât B, deoarece profilul său de vecinătate 5, 7, 7, ... domină profilul fiecăruia a nodurilor. ✷

10.8.2 Diametrul unui grafic

Diametrul unui grafic direcționat este cel mai mic întreg d astfel încât pentru fiecare

două noduri u și v există o cale de lungime d sau mai mică de la u la v. Într-o direcționată
grafic, această definiție are sens numai dacă graficul este puternic conectat; acea
este, există o cale de la orice nod la orice alt nod. Reamintim discuția noastră
de pe Web în secțiunea 5.1.3, unde am observat că există o mare puternică

---

subsetul conectat al Web în „centru”, dar că Web-ul în ansamblu este
nu puternic conectat. Mai degrabă, există câteva pagini care nu ajung nicăieri
linkuri și câteva pagini care nu pot fi accesate de linkuri.
Dacă graficul nu este direcționat, definiția diametrului este aceeași ca pentru
graficele direcționate, dar calea poate traversa marginile nedirecționate în oricare din
rectie. Adică, tratăm o margine nedirecționată ca o pereche de arcuri, câte una în fiecare
direcție. Noțiunea de diametru are sens într-un grafic nedirectat atât de mult
pe măsură ce acel grafic este conectat.
Exemplul 10.30: Pentru graficul din Fig. 10.24, diametrul este 3. Există
unele perechi de noduri, cum ar fi A și E, care nu au o cale de lungime mai mică de 3.
Dar fiecare pereche de noduri are o cale de la unul la altul cu o lungime de cel mult 3.
✷
Putem calcula diametrul unui grafic calculând dimensiunile acestuia
cartiere cu rază crescătoare, până când la o anumită rază nu reușim să adăugăm niciunul
mai multe noduri. Adică, pentru fiecare nod v, găsiți cel mai mic d astfel încât | N (v, d) | =
| N (v, d + 1) |. Acest d este limita superioară strânsă pe lungimea celui mai scurt
calea de la v la orice nod la care poate ajunge. Spune-i d (v). De exemplu, am văzut din
Exemplul 10.29 că d (A) = 3 și d (B) = 2. Dacă există vreun nod v astfel încât
| N (v, d (v)) | nu este numărul de noduri din întregul grafic, atunci graficul este
nu este conectat puternic și nu putem oferi niciun număr întreg finit ca diametru al acestuia.
Cu toate acestea, dacă graficul este puternic conectat, atunci diametrul graficului este
$\max_v$ (d (v)).
Motivul pentru care funcționează acest calcul este că o modalitate de a exprima N (v, d + 1) este
uniunea lui N (v, d) și mulțimea tuturor nodurilor w astfel încât pentru unele u în N (v, d)
există un arc u → w. Adică începem cu N (v, d) și îi adăugăm țintele
dintre toate arcurile care au o sursă în N (v, d). Dacă toate arcele cu sursă în N (v, d)
sunt deja în N (v, d), atunci nu numai că N (v, d + 1) este egal cu N (v, d), dar toate
N (v, d + 2), N (v, d + 3), ... va fi egal cu N (v, d). În cele din urmă, observăm că de atunci
N (v, d) ⊆ N (v, d + 1) singura cale | N (v, d) | poate fi egal | N (v, d + 1) | este pentru N (v, d)
și N (v, d + 1) să fie același set. Astfel, dacă d este cel mai mic întreg astfel încât
| N (v, d) | = | N (v, d + 1) |, rezultă că fiecare nod pe care îl poate atinge v este atins de a
calea lungimii cel mult d.

10.8.3 Închidere tranzitorie și accesibilitate

Închiderea tranzitivă a unui grafic este setul de perechi de noduri (u, v) astfel încât
există o cale de la u la v de lungime una sau mai multe. Uneori vom scrie
această afirmație ca Calea (u, v). Un concept asociat este accesibilitatea. Spunem nod
u ajunge la nodul v dacă u = v sau Căile (u, v) sunt adevărate. [4] Problema calculelor
închiderea tranzitivă este de a găsi toate perechile de noduri u și v într-un grafic pentru care
Calea (u, v) este adevărată. Problema de accesibilitate este, dată un nod u în grafic,
găsiți toate v = u astfel încât Path (u, v) este adevărat.

[4] Rețineți că ambele pot fi adevărate, dacă există un ciclu care implică nodul u.

Șase grade de separare

Există un joc celebru numit „șase grade ale lui Kevin Bacon”, obiectul
dintre care este de a găsi căi de lungime de cel mult șase în graficul ale cărui noduri
sunt vedete de film și ale căror margini conectează stele care s-au jucat în același
film. Conjectura este că în acest grafic, nici o stea de film nu este la distanță
mai mult de șase de la Kevin Bacon. Mai general, oricare două vedete de film
poate fi conectat printr-o cale de lungime de cel mult șase; adică diametrul
graficul este de șase. Un diametru mic face calculul cartierelor
mai eficient, deci ar fi bine dacă ar fi prezentate toate graficele din rețelele sociale
un diametru mic similar. De fapt, sintagma „șase grade de separare”,
se referă la presupunerea că în rețeaua tuturor oamenilor din lume,
unde o margine înseamnă că cele două persoane se cunosc, diametrul este
șase. Din păcate, așa cum vom discuta în secțiunea 10.8.3, nu toate sunt importante

graficele prezintă astfel de conexiuni strânse.
Aceste două concepte se referă la noțiunile de cartiere pe care le-am văzut
mai devreme. Definiți N (u, ∞) pentru a fi $\bigcup_{i \geq 0}$ N (u, i). Atunci v este în N (u, ∞) dacă și numai
dacă v = u sau Path (u, v) este adevărat. Astfel, problema de accesibilitate este de a calcula
unirea tuturor cartierelor de orice rază pentru un nod dat u. The
discuția din secțiunea 10.8.2 ne reamintește că putem calcula setul accesibil
pentru u prin calcularea vecinătăților sale până la acea rază mai mică d pentru care
N (u, d) = N (u, d + 1).
Cele două probleme - închiderea tranzitivă și accesibilitatea - sunt legate, dar
există multe exemple de grafice în care accesibilitatea este fezabilă și tranzitivă
închiderea nu este. De exemplu, să presupunem că avem un grafic web de un miliard de noduri.
Dacă dorim să găsim paginile (nodurile) accesibile de la o anumită pagină, putem face acest lucru,
chiar și pe o singură mașină cu o memorie principală mare. Cu toate acestea, doar pentru a produce
închiderea tranzitivă a graficului ar putea implica 10^{18} perechi de noduri, adică
nu este practic, chiar și folosind un cluster de calculatoare mare. [5]

10.8.4 Accesibilitate prin MapReduce

Când vine vorba de implementarea paralelă, închiderea tranzitivă este de fapt mai multă
ușor paralelizabil decât accesibilitatea. Dacă vrem să calculăm N (v, ∞),
set de noduri accesibile din nodul v, fără a calcula întregul tranzitiv
închidere, nu avem altă opțiune decât să calculăm succesiunea cartierelor,

[5] Deși nu putem calcula complet închiderea tranzitivă, putem învăța totuși un mare lucru
tratează structura unui grafic, cu condiția să existe componente mari puternic conectate.
De exemplu, experimentele grafice Web discutate în secțiunea 5.1.3 au fost realizate pe un grafic de
aproximativ 200 de milioane de noduri. Deși nu au enumerat niciodată toate perechile de noduri în tranzitiv
de închidere, au putut descrie structura Web-ului. Discutăm algoritmul necesar
în secțiunea 10.8.10.

---

care este în esență o căutare a graficului din v. În relațional
termeni, să presupunem că avem o relație Arc (X, Y) care conține acele perechi (x, y) astfel
că există un arc x → y. Vrem să calculăm iterativ o relație Reach (X)
adică setul de noduri accesibile din nodul v. După i runde, Reach (X) va
conține toate acele noduri din N (v, i).
Inițial, Reach (X) conține doar nodul v. Să presupunem că conține toate
noduri în N (v, i) după o rundă de MapReduce. Pentru a construi N (v, i + 1)
trebuie să ne alăturăm Reach cu relația Arc, apoi să proiectăm pe a doua
componentă și efectuați o unire a rezultatului cu vechea valoare a Reach. În
Termeni SQL, executăm

```
SELECT DISTINCT Arc.Y
DIN Reach, Arc
UNDE Arc.X = Reach.X;
```

Această interogare ne cere să calculăm îmbinarea naturală a Reach (X) și Arc (X, Y),
ceea ce putem face prin MapReduce așa cum este explicat în secțiunea 2.3.7. Atunci noi avem
pentru a grupa rezultatul cu Y și a elimina duplicatele, un proces care se poate face
de către un alt job MapReduce ca în secțiunea 2.3.8.
Luăm apoi uniunea rezultatului acestei interogări cu valoarea actuală de
A ajunge. La un moment dat, vom constata că nu sunt adăugate fapte noi despre Reach,
și în acel moment ne putem opri. Valoarea curentă a Reach va fi N (v, ∞).
Câte runde necesită acest proces depinde de cât de departe este v
cel mai îndepărtat nod pe care v îl poate atinge. În multe grafice ale rețelelor sociale, diametrul este
mici, așa cum sa discutat în caseta de pe „Șase grade de separare”. Dacă da, calculează
accesibilitatea în paralel, folosind MapReduce sau o altă abordare, este fezabilă. Puțini
vor fi necesare runde de calcul, iar cerințele de spațiu nu sunt
mai mare decât spațiul necesar pentru a reprezenta graficul.
Cu toate acestea, există câteva grafice în care numărul de runde este grav
impediment. De exemplu, într-o porțiune tipică a Web-ului, a fost găsit
că majoritatea paginilor accesibile de la o anumită pagină sunt accesibile prin căi de lungime
10–15. Cu toate acestea, există câteva perechi de pagini, astfel încât prima ajunge la a doua,
ci numai prin cărări a căror lungime se măsoară în sute. De exemplu,
blogurile sunt uneori structurate, astfel încât fiecare răspuns este accesibil numai prin intermediul
comentariu la care răspunde. Rularea argumentelor duce la căi lungi cu
nici o cale de a „scurta” în jurul acelei căi. Sau un tutorial pe web, cu 50
capitole, pot fi structurate astfel încât să puteți accesa capitolul I doar prin intermediul paginii
pentru capitolul I - 1.

10.8.5 Evaluare seminaivă

Există un truc comun care poate face evaluări iterative, cum ar fi cel al
Secțiunea 10.8.4 mai eficientă. Acolo, la fiecare rundă ne-am alăturat întregului set de

noduri accesibile accesibile cu arcele. Cu toate acestea, dacă am ști la o rundă anterioară
acel nod u era accesibil și există un arc de la u la un anumit nod w, atunci
am descoperit deja că w este accesibil. Prin urmare, nu este necesar

---

Pagina 430

să ia în considerare din nou nodul în runda curentă. Prin luarea în considerare a fiecărui membru al
Ajungeți doar la rundă după ce a fost descoperită accesibilitatea sa, economisim o mulțime de
muncă.
Algoritmul îmbunătățit, care este o instanță a ceea ce se numește seminaiv
evaluare, [6] utilizează nu numai un set Reach (X), ci și un set NewReach (X), con-
rezistență la toate și numai la acele noduri care au fost descoperite pentru prima oară la care se poate ajunge
runda anterioară. Inițial, am setat NewReach să fie {v}, nodul sursă, în timp ce
Reach este gol. Repetăm pașii următori până când la o rundă, NewReach
este un subset al Reach; adică, nu se pot descoperi fapte noi de Reach.
1. Introduceți fiecare nod în NewReach în Reach. Dacă nu are loc nicio modificare la Reach,
apoi am descoperit toate nodurile accesibile, iar iterația se încheie.
2. În caz contrar, calculați o nouă valoare pentru relația NewReach (X) setând-o
să fie rezultatul următoarei interogări:
SELECT DISTINCT Arc.Y
DIN NewReach, Arc
UNDE Arc.X = NewReach.X;

10.8.6 Închidere tranzitivă liniară

Metoda din secțiunea 10.8.4 poate fi aplicată pentru a calcula întregul tranzitiv
închidere în paralel, dacă calculăm setul accesibil din fiecare nod. In orice caz,
există modalități mai simple de a calcula relația Calea (X, Y) din
relația Arc (X, Y), precum și modalități care sunt mai eficiente. Cel mai simplu mod
a calcula închiderea tranzitivă înseamnă a începe cu Calea (x, Y) egală cu Arc (x, Y),
și în fiecare rundă paralelă extindeți lungimile căilor despre care știm
prin extinderea căilor printr-un singur arc. Această abordare se numește tranzitivă liniară
închidere. În termeni SQL, fiecare rundă calculează noile căi după
SELECT DISTINCT Path.X, Arc.Y
DIN Cale, Arc
UNDE Arc.X = Path.Y;
În ceea ce privește accesibilitatea, folosim algoritmii MapReduce pentru unire și grupare-
și-agregare pentru a executa această interogare și apoi luați uniunea interogării
rezultă cu vechea valoare a Căii. Rezultatul va fi valoarea Căii pentru
runda următoare. Numărul de runde pe care trebuie să le executăm este același ca pentru
accesibilitate. După d runde, unde d este diametrul graficului, vom avea

[6] Acest termen destul de ciudat are o istorie în spate. În anii 1980, când a fost observat pentru prima dată
că algoritmii „naivi” pentru calcularea relațiilor recursiv prin utilizarea întregului recursiv
relația la fiecare rundă ar putea fi îmbunătățită doar luând în considerare noile tupluri la fiecare rundă
sa presupus că în scurt timp o abordare chiar mai bună, care ar putea justifica numele „nu naiv”,
ar fi descoperit. Din păcate, nu s-a găsit niciodată o astfel de îmbunătățire, așa că am rămas cu
termenul „seminaiv”.

Pagina 431

a urmat toate căile de lungime până la d + 1 și, prin urmare, nu va descoperi nici o cale nouă
fapte din ultima rundă. În acel moment, știm că ne putem opri.
În ceea ce privește recuperabilitatea, putem folosi evaluarea seminaivă pentru a accelera unirea la
fiecare rundă. O cale de fapt (v, u) care a fost descoperită acum mai mult de o rundă
va fi deja alăturat tuturor faptelor utile Arc (u, w) și, prin urmare, va fi
nu adăugați fapte noi de cale la runda curentă. Prin urmare, putem implementa
TC liniar prin următorul algoritm seminaiv.
Algoritmul îmbunătățit folosește relația Path (X, Y), precum și o relație
NewPath (X, Y), care conține toate și numai faptele Căii care au fost
descoperit la runda anterioară. Inițial, am setat NewPath să fie Arc, în timp ce
Calea este goală. Repetăm pașii următori până când la o rundă, NewPath este
un subset de Cale.
1. Introduceți fiecare nod în NewPath în Cale. Dacă nu are loc nicio modificare la Calea, atunci
am descoperit toate faptele Căii, iar iterația se încheie.
2. În caz contrar, calculați o nouă valoare pentru relația NewPath (X, Y) prin setare
să fie rezultatul următoarei interogări:
SELECT DISTINCT NewPath.X, Arc.Y
DIN NewPath, Arc

UNDE Arc.X = NewPath.Y;

10.8.7 Închiderea tranzitivă prin dublare recursivă

Interesant este faptul că închiderea tranzitivă poate fi calculată mult mai rapid în paralel decât accesibilitatea strictă sau închiderea tranzitivă liniară. Printr-o dublare recursivă tehnică, putem dubla lungimea căilor pe care le cunoaștem într-o singură rundă.

Astfel, pe un grafic al diametrului d, avem nevoie doar de jurnalul de 2 runde, mai degrabă decât d runde. Dacă d = 6, diferența nu este importantă, dar dacă d = 1000, $\log_2 d$ este aproximativ 10, deci există o reducere de sută de ori a numărului de runde.

Cea mai simplă abordare de dublare recursivă este de a începe cu relația Path (X, Y) egal cu relația arc Arc (X, Y). Să presupunem că după i runde, Calea (X, Y) conține toate perechile (x, y) astfel încât există o cale de la x la y de lungime cel mult 2^i. Apoi, dacă ne alăturăm Calea cu sine în următoarea rundă, îi vom descoperi pe toți perechi (x, y) astfel încât există o cale de la x la y de lungime cel mult $2 \times 2^i = 2^{i+1}$.

Interogarea recursivă-dublare în SQL este

SELECT DISTINCT p1.X, p2.Y
DE la Calea p1, Calea p2
UNDE p1.Y = p2.X;

După calculul acestei interogări, obținem toate perechile conectate printr-o cale de lungime între 2 și 2^{i+1}, presupunând că Calea conține perechi conectate prin căi de lungime între 1 și 2^i. Dacă luăm uniunea rezultatului acestei interogări cu Arc relația în sine, atunci obținem toate căile de lungime cuprinse între 1 și 2^{i+1} și putem folosi

---

Pagina 432

Accesibilitate de la dublarea recursivă

Am susținut că accesibilitatea de calcul necesită în mod inerent un număr de runde egale cu diametrul graficului. Acest lucru nu este strict adevărat, dar există un dezavantaj în ceea ce privește abaterea de la metoda dată în secțiunea 10.8.4 pentru accesibilitate. Dacă dorim setul Reach (v), putem calcula tranzitivul închiderea întregului grafic prin dublare recursivă și apoi aruncați-le pe toate perechi care nu au v ca primă componentă. Dar nu putem arunca îndepărtează toate acele perechi până când terminăm. În timpul calculului închidere tranzitivă, am putea încheia calculând multe fapte Calea (x, y), unde nici x, nici y nu sunt accesibile din v și chiar dacă sunt accesibile din v, este posibil să nu fie nevoie să știm că x poate ajunge la y. Dacă graficul este mare, acesta poate fi imposibil de stocat toate faptele Căii, chiar dacă am putea stoca faptele Reach.

uniunea ca relație Cale în următoarea rundă de dublare recursivă. Interogarea el însuși poate fi implementat prin două joburi MapReduce, unul pentru a face join și cel altul să facă unirea și să elimine duplicatele. După cum am observat pentru paralelă calculul accesibilității, metodele din secțiunile 2.3.7 și 2.3.8 sunt suficiente. The union, discutat în secțiunea 2.3.6 nu necesită cu adevărat o sarcină MapReduce proprii; poate fi combinat cu duplicatul-eliminare.

Dacă un grafic are diametrul d, atunci după $\log_2 d$ runde ale algoritmului de mai sus Calea conține toate perechile (x, y) conectate printr-o cale de lungime cel mult d; acesta este, conține toate perechile din închiderea tranzitivă. Dacă nu știm deja d, unul va fi necesară mai multă rundă pentru a verifica dacă nu mai pot fi găsite perechi, dar pentru d mare, acest proces durează mult mai puține runde decât căutarea în lățime pe care le-am folosit pentru accesibilitate.

Cu toate acestea, metoda de dublare recursivă de mai sus face multă muncă redundantă. Un exemplu ar trebui să clarifice ideea.

Exemplul 10.31: Să presupunem că cea mai scurtă cale de la x_0 la x_{17} are lungimea 17; în în special, să existe o cale $x_0 \rightarrow x_1 \rightarrow \cdots \rightarrow x_{17}$. Vom descoperi fapt Path (x_0, x_{17}) în runda a cincea, când Path conține toate perechile conectate prin căi de lungime până la 16. Aceeași cale de la x_0 la x_{17} va fi descoperită 16 momente în care ne alăturăm Calea cu ea însăși. Adică, putem lua calea faptului (x_0, x_{16}) și combinați-l cu Path (x_{16}, x_{17}) pentru a obține Path (x_0, x_{17}). Sau ne putem combina Calea (x_0, x_{15}) cu Calea (x_{15}, x_{17}) pentru a descoperi aceleași fapt și așa mai departe. Pe pe de altă parte, TC liniar ar descoperi această cale o singură dată: când Path (x_0, x_{16}) a fost în NewPath. ✷

În ceea ce privește algoritmii anteriori, algoritmul recursiv-dublare are mai mult versiune eficientă care utilizează evaluarea seminaivă. Din nou, folosim două relații, Calea (X, Y) și NewPath (X, Y). În loc să alăturăm două copii ale cărării, noi

---

Pagina 433

înlocuiți a doua relație Path cu NewPath. Rețineți că, dacă cea mai scurtă cale de la u la v are lungimea k, atunci putem rupe această cale într-un cap de lungime ⌊K / 2⌋ la început și o coadă de lungime ⌈k / 2⌉ la sfârșit. Coada nu o va face fi descoperit până la runda ⌈k / 2⌉, după care apare în NewPath, în timp ce capul va fi cu siguranță descoperit la acea rundă (dacă k este par) sau înainte (dacă k este ciudat), și va fi astfel în Cale pentru următoarea alăturare cu NewPath. Astfel, urmând versiunea seminaivă a dublării recursive va funcționa.
Inițial, setăm NewPath să fie Arc, în timp ce Calea este goală. Repetăm urmând pașii până la o rundă nouă, NewPath este un subset al Căii.
1. Introduceți fiecare nod în NewPath în Cale. Dacă nu are loc nicio modificare la Calea, atunci am descoperit toate faptele Căii, iar iterația se încheie.
2. În caz contrar, calculați o nouă valoare pentru relația NewPath (X, Y) prin setare să fie rezultatul următoarei interogări:
SELECT DISTINCT Path.X, NewPath.Y
FROM Path, NewPath
UNDE NewPath.X = Path.Y;

10.8.8 Închidere tranzitivă inteligentă

Există o variantă de dublare recursivă, numită închidere tranzitivă inteligentă, care evită să descopere aceeași cale de mai multe ori. Fiecare cale de lungime mai mare mai mult de 1 poate fi spart într-un cap a cărui lungime este o putere de 2, urmată de a coada a cărei lungime nu este mai mare decât lungimea capului.
Exemplul 10.32: O cale de lungime 13 are un cap format din primele 8 arce, urmat de o coadă formată din ultimele 5 arce. O cale de lungime 2 este un cap de lungimea 1 urmată de o coadă de lungimea 1. Rețineți că 1 este o putere de 2 (a 0-a putere), iar coada va fi la fel de lungă ca și capul ori de câte ori calea în sine are o lungimea care este o putere de 2. ✷
Pentru a implementa închiderea tranzitivă inteligentă în SQL, introducem o relație Q (X, Y) a cărui funcție după runda a-a este de a ține toate perechile de noduri (x, y) astfel încât cea mai scurtă cale de la x la y are lungimea exact 2^i. De asemenea, după ith round, Path (x, y) va fi adevărat dacă cea mai scurtă cale de la x la y este de lungime cel mult $2^{i+1} - 1$. Rețineți că această interpretare a Căii este ușor diferită din interpretarea Path în metoda simplă recursivă-dublare dată în secțiunea 10.8.7.
Inițial, setați atât Q cât și Path pentru a fi copii ale relației Arc. După ith round, presupunem că Q și Path au conținutul descris în precedent paragraf. Rețineți că pentru runda i = 1, valorile inițiale ale Q și Cale îndeplinesc inițial condițiile descrise pentru i = 0. Pe (i + 1) runda, facem următoarele:

Pagina 434

1. Calculați o nouă valoare pentru Q unindu-l cu sine și descoperind astfel căi de două ori mai mari decât cele din Q, utilizând interogarea SQL
SELECT DISTINCT q1.X, q2.Y
DE LA Q q1, Q q2
UNDE q1.Y = q2.X;
2. Scădeți Calea din relația Q calculată în Pasul (1) Rețineți că Pasul (1) descoperă toate căile de lungime 2^{i+1}. Dar unele perechi conectate prin aceste căi poate avea și căi mai scurte. Rezultatul pasului (2) este de a lăsa în Q tot și numai acele perechi (u, v) astfel încât cea mai scurtă cale de la u la v are lungime exact 2^{i+1}.
3. Alăturați Path cu noua valoare a Q calculată la Pasul (2), utilizând SQL interogare
ALEGE DISTINCT QX, Path.Y
DE LA Q, Cale
UNDE QY = Path.X;
La începutul rundei, Calea conține toate (y, z) astfel încât cea mai scurtă cale de la y la z are o lungime de până la $2^{i+1} - 1$, iar noua valoare din Q conține toate perechile (x, y) pentru care este cea mai scurtă cale de la x la y lungimea 2^{i+1}. Astfel, rezultatul acestei interogări este setul de perechi (x, y) astfel că cea mai scurtă cale de la x la y are o lungime cuprinsă între $2^{i+1} + 1$ și $2^{i+2} - 1$, inclusiv.
4. Setați noua valoare Path pentru a fi uniunea relației calculate în Pasul (3), noua valoare a lui Q calculată în pasul (1) și vechea valoare a Cale. Acești trei termeni ne oferă toate perechile (x, y) a căror cale cea mai scurtă este de

lungimea $2^{i+1} + 1$ până la $2^{i+2} - 1$, exact 2^{i+1} și 1 până la $2^{i+1} - 1$,
respectiv. Astfel, uniunea ne oferă tuturor celor mai scurte căi până la lungime
$2^{i+2} - 1$, așa cum este cerut de ipoteza inductivă despre ceea ce este adevărat după
fiecare rundă.
Fiecare rundă a algoritmului de închidere tranzitivă inteligentă folosește pași care sunt îmbinări,
agregări (eliminări duplicat) sau uniuni. O rundă poate fi astfel implementată
menționate ca o succesiune scurtă de joburi MapReduce. Mai mult, multă muncă
poate fi salvat dacă aceste operații pot fi combinate, să zicem folosind cele mai generale
modele de comunicare permise de un sistem de flux de lucru (a se vedea secțiunea 2.4.1).
Interesant este faptul că TC inteligent încorporează deja trucul seminaiv. La fiecare
rotund, relația Q conține doar fapte Q (u, v) care nu au fost niciodată în Q
înainte, deoarece cea mai scurtă cale de la U la v este mai lungă decât cele mai scurte căi
care au fost reprezentate în Q la orice rundă anterioară. Mai departe, când ne alăturăm Q
cu Path, folosim doar fapte în Q care nu erau în Q în rundele anterioare,
deci Q este un subset al ceea ce ar fi NewPath în algoritmul Secțiunii 10.8.7.

Pagina 435

Fapte despre cale și căi

Ar trebui să fim atenți să distingem între o cale, care este o secvență de
arcuri și un fapt de cale, care este o afirmație că există o cale de la
unele nod x la unele nod y. Faptul căii a fost prezentat de obicei ca
Calea (x, y). Închiderea tranzitivă inteligentă descoperă fiecare cale o singură dată, dar ea
poate descoperi o cale de mai multe ori. Motivul este că adesea un grafic
va avea multe căi de la x la y și poate avea chiar și multe căi diferite
de la x la y care au aceeași lungime.
Nu toate căile sunt descoperite independent prin închidere tranzitivă inteligentă.
De exemplu, dacă există arce w → x → y → z și, de asemenea, arce x → u → z,
atunci calea faptului Calea (w, z) va fi descoperită de două ori, o dată prin combinare
Path (w, y) cu Path (y, z) și din nou atunci când combinați Path (w, u) cu
Calea (u, z). Pe de altă parte, dacă arcurile ar fi w → x → y → z și w →
v → y, atunci Calea (w, z) ar fi descoperită o singură dată, la combinare
Calea (w, y) cu Calea (y, z).

10.8.9 Compararea metodelor

Metodele discutate în secțiunile 10.8.4 până la 10.8.8 au avantajele lor
și dezavantaje. Problemele principale sunt numărul de runde necesare și
timpul necesar executării fiecărei runde. Vom lua în considerare aceste costuri pe un
grafic direcționat cu n noduri, e arcuri și diametru d.
Pentru toți algoritmii discutați, costul îmbinărilor domină costul
a celorlalte operațiuni la o rundă - grupare, agregare și unire. Prin urmare,
ne vom gândi doar la costul îmbinărilor. Mai mult, vom presupune că
unirea este implementată eficient, ca în algoritmul Secțiunii 2.3.7. In aceea
algoritm, fiecare tuplu din fiecare dintre cele două relații este trimis la un reductor și
lucrul la fiecare reductor este produsul numărului de tupluri din fiecare dintre
cele două relații la acel reductor. În cele din urmă, vom presupune toți algoritmii
sunt implementate prin versiunile lor seminaive, deoarece acest lucru este mai eficient decât
versiunea „naivă”. Figura 10.25 rezumă analiza.

Algoritm
Runde de calcul
Accesibilitate
O (e)
d
TC liniar
Unu)
d
Dublare recursivă
$O(n^3)$
jurnal $_2$ d
Smart TC
$O(n^3)$
jurnal $_2$ d

Figura 10.25: Costurile algoritmilor de închidere tranzitivă
Numărul de runde pentru acești algoritmi a fost deja discutat.

Pagina 436

Accesibilitatea și închiderea tranzitivă liniară vor găsi noi informații despre Reach sau Path până la căutarea implicată de iterațiile lor permite accesarea oricărui nod din oricare alte. Acest număr de runde este delimitat de diametrul graficului. Cand calculând închiderea tranzitivă completă, vor exista întotdeauna unele noduri u și v astfel încât cea mai scurtă cale de la u la v are distanța d. Astfel, tranzitiv liniar închiderea necesită întotdeauna d runde. Cu toate acestea, atunci când căutăm dintr-un anume nodul u, este posibil să poată ajunge la toate nodurile pe care le atinge vreodată folosind căi mult mai scurte decât diametrul graficului. Adică valoarea lui d ar putea fi determinată de o pereche de noduri care nu include u. Pentru cei doi 10.25, am observat că în jurnal 2 d runde, descoperim toate căile de lungime până la d și, prin urmare, descoperim toate Fapte de cale.

Acum, luați în considerare costul îmbinării în versiunea seminaivă a accesibilității algoritmul secțiunii 10.8.4. Fiecare nod u poate fi în NewReach la o singură rundă. În acea rundă, va întâlni toate arcele cu capul u la un reductor. Astfel, peste toate rundele și toate reductoarele la fiecare rundă, costul aderării nodului u cu arcuri este egal cu gradul exterior al lui u. Observați că fiecare arc contribuie la gradul exterior al unui singur nod - nodul din coada sa. Astfel, suma gradele exterioare ale nodurilor unui grafic direcționat este egal cu numărul de arce din grafic. Această observație justifică O (e) legat de costul unei runde pentru accesibilitate.

Pentru închiderea tranzitivă liniară seminaivă, luați în considerare un nod u și reductoarele corespunzând lui u peste toate rundele. Pentru fiecare nod v, doar unul dintre acestea reductoarele pot primi faptul NewPath (v, u). La acea rundă, acest reductor va alăturați-vă acestui fapt cu toate faptele Arc (u, w) și astfel veți suporta costuri egale cu în afara gradului de u. Dacă reparăm v și lăsăm să varieze, atunci lucrarea totală a tuturor reductoarelor tratarea faptelor de forma NewPath (v, u) pentru orice u este suma gradelor externe dintre toate nodurile u, care este numărul total de margini, e. Atunci, când vom rezuma acest lucru cost pentru toate n nodurile v, constatăm că numărul total de fapte unite printre toate reductoarele de la toate rundele este O (ne).

Apoi, uită-te la asocierea asociată cu seminaive recursive-dublare algoritmul secțiunii 10.8.7. Fiecare fapt NewPath (u, w) apare în NewPath la o singură rundă. În acea rundă, acesta poate întâlni orice fapt de cale (v, u). Astfel, lucrul total al tuturor reductoarelor la toate rundele este n 2 (pentru numărul diferitelor fapte care pot apărea vreodată în NewPath) ori n (pentru numărul de fapte ale căii cu care se poate uni când apare. Acest argument justifică O (n 3) destinate dublării recursive seminaive.

În cele din urmă, uitați-vă la închiderea tranzitivă inteligentă. Acum, facem două uniri la fiecare rundă: Q cu ea însăși și Q cu Calea. Un fapt Q (u, w) poate apărea numai la o rundă și, în cel mai rău caz, va fi alăturat cu n alte fapte Q (v, u) rundă. Astfel, lucrarea totală este cel mult O (n 3), care este produsul a n 2 fapte ori o alăturare cu n fapte. Argumentul pentru al doilea join, Q cu Path, este similar. Rețineți că limita superioară a calculului pentru TC inteligent este aceeași în ceea ce privește dublarea recursivă. Cu toate acestea, așa cum am discutat în secțiunea 10.8.8, există o mare probabilitate ca mai puține perechi de fapte ale Căii să fie conectate folosind

---

Pagina 437

abordare smart-TC decât dacă folosim dublarea recursivă a forței brute, chiar dacă cele mai grave cazuri teoretice sunt aceleași.

10.8.10 Închiderea tranzitivă prin reducerea graficului

O componentă puternic conectată (SCC) a unui grafic este un set de noduri S astfel încât:

1. Fiecare nod din S ajunge la orice alt nod din S.
2. S este maxim, în sensul că nu există un nod în afara lui S care să fie ambele ajunge la fiecare nod din S și este atins de fiecare nod din S.

Un grafic direcționat tipic, cum ar fi Web, conține multe conectări puternice componente (SCC). Putem prăbuși un SCC la un singur nod în măsura în care punerea închiderii tranzitive este în cauză, deoarece toate nodurile dintr-un SCC ajung exact aceleași noduri. Există un algoritm elegant pentru găsirea SCC-urilor un grafic în timp liniar în mărimea graficului, datorită JE Hopcroft și RE Tarjan. Cu toate acestea, acest algoritm este în mod inerent secvențial, bazat pe profunzimea întâi căutarea și, prin urmare, nu sunt potrivite pentru impelementarea paralelă pe grafice mari.

Putem găsi majoritatea SCC-urilor într-un grafic prin câteva selecții de noduri aleatorii, cu două calcule de accesibilitate pentru fiecare nod selectat. Mai mult, cu atât mai mare un SCC este, cu atât este mai probabil să fie prăbușit devreme, reducând astfel dimensiunea

a graficului rapid. Algoritmul pentru reducerea SCC-urilor la noduri simple este la fel
urmează. Fie G graficul care urmează a fi redus, și G ′ G cu toate arcele
inversat.
1. Alegeți un nod v din G la întâmplare.
2. Găsiți N G (v, ∞), setul de noduri accesibil din v în G.
3. Găsiți N G ′ (v, ∞), ansamblul de noduri pe care v ajunge în graficul G ′ care are
arcurile lui G inversate. În mod echivalent, acest set este acele noduri care ajung la v
în G.
4. Construiți SCC S care conține v, care este N G (v, ∞) ∩ N G ′ (v, ∞). Acea
este, v și u sunt în același SCC al lui G și dacă numai dacă v poate ajunge la u și u
poate ajunge la v.
5. Înlocuiți SCC S cu un singur nod s în G. Pentru aceasta, ștergeți toate nodurile din S
din G și adăugați s la setul de noduri al lui G. Ștergeți din G toate arcele unul sau
ambele capete sunt în S. Apoi, adăugați la setul de arc al lui G un arc s → x
ori de câte ori a existat un arc în G de la orice membru al lui S la x. În cele din urmă, adăugați
un arc x → s dacă a existat un arc de la x la orice membru al lui S.
Putem itera pașii de mai sus de un număr fix de ori. Alternativ, noi
poate itera până când graficul devine suficient de mic. Am putea chiar examina

---

Pagina 438

toate nodurile până când fiecare este atribuit unui SCC. Rețineți că, în caz extrem, a
nodul v se află într-un SCC format doar din el însuși, dacă
N G (v, ∞) ∩ N G ′ (v, ∞) = {v}
Dacă facem ultima alegere, graficul rezultat se numește re tranzitiv
producția graficului original G. Reducerea tranzitivă este întotdeauna aciclică, deoarece
dacă ar avea un ciclu, ar rămâne un SCC de mai mult de un nod. Cu toate acestea, ea
nu este necesar să se reducă la un grafic aciclic, atâta timp cât are graficul rezultat
suficient de puține noduri încât să fie posibil să se calculeze închiderea tranzitivă completă a
acest grafic; adică numărul de noduri este suficient de mic cu care ne putem ocupa
un rezultat a cărui dimensiune este proporțională cu pătratul acelui număr de noduri.
În timp ce închiderea tranzitivă a graficului redus nu este exact aceeași
ca închidere tranzitivă a graficului original, primul, plus informațiile
despre ce SCC aparține fiecărui nod original este suficient pentru a ne spune ceva
că ne spune închiderea tranzitivă a graficului original. Dacă vrem să știm
dacă Path (u, v) este adevărat în graficul original, găsiți SCC-urile care conțin u
și v. Dacă u și v aparțin aceluiași SCC, atunci cu siguranță u pot ajunge la v. Dacă ele
aparțin diferitelor SCC s și respectiv t, determină dacă s atinge t
în graficul redus. Dacă da, atunci u atinge v în graficul original și, dacă nu,
atunci nu.
Exemplul 10.33: Să reconsiderăm imaginea „papion” de pe web
Secțiunea 5.1.3. Numărul de noduri din partea graficului examinat a fost
peste 200 de milioane; cu siguranță prea mare pentru a face față datelor cu dimensiuni proporționale cu
pătratul acelui număr. A existat un set mare de noduri numit „SCC” care
a fost considerat ca fiind centrul graficului. Din moment ce aproximativ un nod din patru a fost
în acest SCC, acesta ar fi prăbușit la un singur nod de îndată ce oricare dintre acestea
membrii au fost aleși la întâmplare. Dar există multe alte SCC-uri pe web,
chiar dacă nu au fost arătate în mod explicit în „papion”. De exemplu,
in-component ar avea în interiorul său multe SCC-uri mari. Nodurile dintr-unul din
aceste SCC pot ajunge unul la altul și pot ajunge la unele dintre celelalte noduri din
în componentă și, desigur, pot ajunge la toate nodurile din SCC central.
SCC-urile din componentele de intrare și ieșire, tuburile și alte structuri pot
toate să fie prăbușite, ducând la un grafic mult mai mic. ✷

10.8.11 Aproximarea dimensiunilor vecinătăților

În această secțiune vom aborda problema calculării vecinătății
profil pentru fiecare nod al unui grafic mare. O variantă a problemei, care cedează
la aceeași tehnică, este de a găsi dimensiunea setului accesibil pentru fiecare nod v,
adică mulțimea pe care am numit-o N (v, ∞). Reamintim că pentru un grafic de un miliard de noduri,
este total imposibil de calculat vecinătățile pentru fiecare nod, chiar folosind
un grup foarte mare de mașini. Chiar dacă vrem doar un număr al nodurilor din
fiecare cartier, mai degrabă decât nodurile în sine, trebuie să ne amintim

---

Pagina 439

noduri găsite până când explorăm graficul, altfel nu vom ști dacă
un nod găsit este nou sau unul pe care l-am văzut deja.
Pe de altă parte, nu este atât de greu să găsești o aproximare la dimensiune
din fiecare cartier, folosind tehnica Flajolet-Martin discutată în secțiunea
4.4.2. Reamintim că această tehnică folosește un număr mare de funcții hash;
în acest caz, funcțiile hash sunt aplicate nodurilor grafului. The
proprietatea importantă a șirului de biți pe care o obținem atunci când aplicăm funcția hash h
nodul v este „lungimea cozii” - numărul de 0 la sfârșitul șirului. Pentru
orice set de noduri, o estimare a dimensiunii setului este de 2^R, unde R este lungimea
de cea mai lungă coadă pentru orice membru al setului. Astfel, în loc să stochezi toate
membri ai setului, putem înregistra în schimb doar valoarea lui R pentru acel set. De
desigur, tehnica Flajolet-Martin folosește multe funcții hash diferite și
combină valoarea lui R obținută din fiecare, deci trebuie să înregistrăm valoarea lui
R pentru fiecare funcție hash. Cu toate acestea, spațiul total necesar pentru (să zicem) mai mulți
sutele de valori ale lui R sunt mult mai mici decât specificațiile necesare pentru listarea tuturor membrilor
a unui cartier mare.
Exemplul 10.34: Dacă folosim o funcție hash care produce un șir de 64 de biți, atunci
șase biți sunt tot ceea ce avem nevoie pentru a stoca fiecare valoare a lui R. De exemplu, dacă există
un miliard de noduri și vrem să estimăm dimensiunea cartierului pentru fiecare,
putem stoca valoarea lui R pentru 20 de funcții hash pentru fiecare nod în 15 gigaocteți.
✷
Dacă stocăm lungimi de coadă pentru fiecare cartier, putem folosi aceste informații
pentru a calcula estimări pentru cartierele mai mari din estimările noastre pentru
cartiere mai mici. Adică, să presupunem că am calculat estimările noastre pentru
| N (v, d) | pentru toate nodurile v și vrem să calculăm estimări pentru vecinul-
hote cu raza d + 1. Pentru fiecare funcție hash h, valoarea lui R pentru N (v, d + 1)
este cea mai mare dintre:
1. Coada lui v în sine și
2. Valorile lui R asociate cu h și N (u, d), unde v → u este un arc al lui
graficul.
Observați că nu contează dacă un nod este accesibil printr-un singur
succesorul lui v în grafic sau prin mulți succesori diferiți. Primim
aceeași estimare în ambele cazuri. Această proprietate utilă a fost aceeași ca și noi
exploatat în secțiunea 4.4.2 pentru a evita să știe dacă un element de flux
a apărut o dată sau de multe ori în flux.
Vom descrie acum algoritmul complet, numit ANF (Aproximativ
Funcția de vecinătate). Alegem funcțiile K hash h_1, h_2, ..., h_k. Pentru fiecare
nodul v și raza d, fie R_i (v, d) să indice lungimea maximă a cozii oricărui nod
în N (v, d) folosind funcția hash h_i.
BAZĂ : Pentru a inițializa, să fie R_i (v, 0) lungimea cozii lui h_i (v), pentru toate i și
v.

INDUCȚIE : Să presupunem că am calculat R_i (v, d) pentru tot i și v. Inițializați
R_i (v, d + 1) să fie R_i (v, d), pentru tot i și v. Apoi, considerați toate arcele x → y în
grafic, în orice ordine. Pentru fiecare x → y, setați R_i (x, d + 1) la cel mai mare curent
valoare și R_i (y, d).
Observați că ne poate oferi faptul că putem considera arcurile în orice ordine
o accelerare mare în cazul în care putem stoca R_i în memoria principală, în timp ce
setul de arce este atât de mare încât trebuie stocat pe disc. Putem transmite în flux toate
blocuri de discuri care conțin arcuri pe rând, folosind astfel un singur acces pe disc per
iterație pe bloc de disc utilizat pentru stocarea arcului. Acest avantaj este similar cu
una pe care am observat-o în secțiunea 6.2.1, unde am arătat cât de frecvent este setul de articole
algoritmi precum A-priori ar putea profita de citirea datelor din coșul de piață
un flux, unde fiecare bloc de disc a fost citit o singură dată pentru fiecare rundă.
Pentru a estima dimensiunea lui N (v, d), combinați valorile lui R_i (v, d) pentru i =
1, 2, ..., k, așa cum sa discutat în secțiunea 4.4.3. Adică, grupați R-urile în mici
grupuri, luați media și luați mediana mediilor.
O altă îmbunătățire a algoritmului ANF poate fi obținută dacă suntem numai noi
interesat de estimarea dimensiunilor seturilor accesibile, N (v, ∞). Nu este atunci
necesar pentru a salva R_i (v, d) pentru diferite raze d. Putem menține o valoare R_i (v)
pentru fiecare funcție hash h_i și pentru fiecare nod v. Când se ia în orice rundă, luăm în considerare
arc x → y, pur și simplu atribuim
R_i (x): = max (R_i (x), R_i (y))
Putem opri iterația atunci când la o rundă nu se modifică nicio valoare a vreunui R_i (v).
Sau dacă știm că d este diametrul graficului, putem itera doar d ori.

A
B
C
D
E
G
H
F
Eu
Figura 10.26: Un grafic pentru exerciții asupra cartierelor și închiderea tranzitivă

10.8.12 Exerciții pentru secțiunea 10.8

Exercițiul 10.8.1: Pentru graficul din Fig. 10.9, pe care îl repetăm aici ca Fig. 10.26:

(a) Dacă graficul este reprezentat ca un grafic direcționat, câte arce există?
(b) Care sunt profilurile de vecinătate pentru nodurile A și B?
(c) Care este diametrul graficului?
(d) Câte perechi sunt în închiderea tranzitivă? Indicație: Nu uitați asta
există căi de lungime mai mare decât zero de la un nod la sine în acest
grafic.
(e) Dacă calculăm închiderea tranzitivă prin dublare recursivă, câte
rundele sunt necesare?
Exercițiul 10.8.2: Algoritmul inteligent de închidere tranzitivă rupe căile oricăruia
lungimea în cap și coadă de lungimi specifice. Care sunt lungimile capului și cozii
pentru căi de lungime 7, 8 și 9?
! Exercițiul 10.8.3: Completați valorile tabelului din Fig. 10.25 pentru versiunile de
algoritmii pentru accesibilitate, TC liniar și dublarea recursivă dacă seminaivul
truc nu este folosit.
! Exercițiul 10.8.4: În secțiunea 10.8.9 am observat că suma gradelor exterioare
dintre nodurile dintr-un grafic direcționat este egal cu numărul de arce din grafic. Acolo
este o relație similară pentru un grafic neorientat, între gradele lui
noduri și numărul său de margini. Găsiți și demonstrați această relație.
Exercițiul 10.8.5: Luați în considerare ultimul exemplu al unei rețele sociale
prezentat în Fig. 10.24. Să presupunem că folosim o funcție hash h care mapează fiecare
nod (litere mari) la codul ASCII. Rețineți că codul ASCII pentru A este
01000001 și codurile pentru B, C, ... sunt secvențiale, 01000010, 01000011, ...
(a) Folosind această funcție hash, calculați valorile lui R pentru fiecare nod și
raza 1. Care sunt estimările dimensiunilor fiecărui cartier? Cum
estimările se compară cu realitatea?
(b) Apoi, calculați valorile lui R pentru fiecare nod și rază 2. Calculați din nou
estimările dimensiunilor cartierului și se compară cu realitatea.
(c) Diametrul graficului este 3. Calculați valoarea lui R și dimensiunea
estimare pentru setul de noduri accesibile pentru fiecare dintre nodurile grafului.
(d) O altă funcție hash g este una plus codul ASCII pentru o literă. Repeta
(a) până la (c) pentru funcția hash g. Luați estimarea dimensiunii unui
vecinătatea să fie media estimărilor date de h și g. Cum
aproape sunt aceste estimări?

10.9 Rezumatul capitolului 10

♦ Grafice de rețea socială: grafice care reprezintă conexiunile într-un social
rețeaua nu este doar mare, ci prezintă o formă de localitate, unde este mică
subseturile de noduri (comunități) au o densitate de margini mult mai mare decât
densitatea medie.

♦ Comunități și clustere: în timp ce comunitățile seamănă cu clustere în unele
moduri, există, de asemenea, diferențe semnificative. Indivizi (noduri) în mod normal
aparțin mai multor comunități, iar măsurile obișnuite la distanță nu reușesc
reprezintă apropierea între nodurile unei comunități. Ca urmare, standard
algoritmii pentru găsirea clusterelor în date nu funcționează bine pentru comunitate
constatare.

♦ Betweenness: Un mod de a separa nodurile în comunități este de a măsura

intervalul de margini, care este suma peste toate perechile de noduri ale
fracțiune dintre cele mai scurte căi între acele noduri care trec prin date
margine. Comunitățile se formează prin ștergerea marginilor a căror distanță
este peste un prag dat.

♦ Algoritmul Girvan-Newman: Algoritmul Girvan-Newman este un
tehnică eficientă pentru calcularea intermediarității muchiilor. O lățime-
se efectuează prima căutare din fiecare nod și o secvență de pași de etichetare
calculează cota de căi de la rădăcină la celălalt nod care merge
prin fiecare dintre margini. Acțiunile pentru un avantaj care sunt calculate pentru
fiecare rădăcină este însumată pentru a obține intermediaritatea.

♦ Comunități și grafice bipartite complete: un grafic bipartit complet
are două grupuri de noduri, toate marginile posibile între perechi de noduri alese
una din fiecare grup și fără margini între nodurile aceluiași grup.
Orice comunitate suficient de densă (un set de noduri cu multe margini între
ele) vor avea un grafic mare bipartit complet.

♦ Găsirea graficelor bipartite complete: Putem găsi grafice bipartite complete
prin aceleași tehnici pe care le-am folosit pentru a găsi obiecte frecvente. Noduri de
graficul poate fi considerat atât ca elemente, cât și ca coșuri. The
coșul corespunzător unui nod este setul de noduri adiacente, gândit ca fiind
obiecte. Un grafic bipartit complet cu grupuri de noduri de mărimea t și s poate
să fie gândit ca găsind obiecte frecvente de mărime t cu suport.

♦ Partiționarea graficului: o modalitate de a găsi comunități este partiționarea unui grafic
repetat în bucăți de dimensiuni aproximativ asemănătoare. O tăietură este o partiție a
nodurile grafului în două seturi, iar dimensiunea acestuia este numărul de muchii care
au un capăt în fiecare set. Volumul unui set de noduri este numărul de
muchii cu cel puțin un capăt în setul respectiv.

♦ Tăieturi normalizate: putem normaliza dimensiunea unei tăieturi luând raportul
a mărimii tăieturii și a volumului fiecăruia dintre cele două seturi formate

---

Pagina 443

prin tăietură. Apoi adăugați aceste două rapoarte pentru a obține valoarea de tăiere normalizată.
Tăierile normalizate cu o sumă mică sunt bune, în sensul în care tind
împarte nodurile în două părți aproximativ egale și au o relativ mică
mărimea.

♦ Matrici de adiacență: Aceste matrice descriu un grafic. Intrarea în rând
i și coloana j este 1 dacă există o margine între nodurile i și j și 0
in caz contrar.

♦ Matrici de grade: matricea de grade pentru un grafic are d în diagonala i
intrare dacă d este gradul nodului ith. În afara diagonalei, toate intrările sunt 0.

♦ Matrice Laplaciene: Matricea Laplaciană pentru un grafic este matricea sa de grad
minus matricea sa de adiacență. Adică intrarea din rândul i și coloana i din
matricea laplaciană este gradul nodului ith al graficului și
intrarea în rândul i și coloana j, pentru i = j, este -1 dacă există o margine între
nodurile i și j, și 0 în caz contrar.

♦ Metoda spectrală pentru partiționarea graficelor: cea mai mică valoare proprie pentru oricare
Matricea laplaciană este 0, iar vectorul propriu corespunzător este format din toate
1. Vectoarele proprii corespunzătoare valorilor proprii mici pot fi utilizate
ghidați o partiție a graficului în două părți de dimensiuni similare cu o mică
valoare redusă. Pentru un exemplu, punerea nodurilor cu o componentă pozitivă
în vectorul propriu cu valoarea a doua cea mai mică într-un singur set și
cei cu o componentă negativă în cealaltă este de obicei bună.

♦ Comunități suprapuse: de obicei, indivizii sunt membri ai mai multor
comunitățile. În graficele care descriu rețelele sociale, este normal pentru
probabilitatea ca doi indivizi să fie prieteni să crească ca număr de
crește comunitățile din care ambii sunt membri.

♦ Modelul grafic de afiliere: un model adecvat pentru calitatea de membru
comunități este să presupunem că pentru fiecare comunitate există o probabilitate
că din cauza acestei comunități doi membri devin prieteni (au un
marginea în graficul rețelei sociale). Astfel, probabilitatea ca două noduri
au o margine este 1 minus produsul probabilităților pe care niciuna dintre
comunitățile ale căror membri sunt amândoi determină o margine
între ele. Găsim apoi atribuirea de noduri către comunități și

valorile acelor probabilități care descriu cel mai bine socialul observat
grafic.

♦ Estimarea maximă a probabilității: o tehnică de modelare importantă,
pentru modelarea comunităților, precum și pentru multe alte lucruri, este de a calcula,
ca o funcție a tuturor opțiunilor de valori ale parametrilor pe care le permite modelul,
probabilitatea ca datele observate să fie generate. Valorile care
randament cea mai mare probabilitate sunt presupuse a fi corecte și numite
estimarea probabilității maxime (MLE).

Pagina 444

♦ Utilizarea descendenței gradientului: dacă știm apartenența la comunități, putem
găsiți MLE prin coborâre în gradient sau alte metode. Cu toate acestea, nu putem
găsiți cel mai bun apartenență la comunități după descendență gradientă, deoarece
calitatea de membru este discretă, nu continuă.

♦ Modelare comunitară îmbunătățită prin puterea de membru: Putem for-
mulate problema de a găsi MLE de comunități într-un grafic social
presupunând că indivizii au o forță de membru în fiecare comunitate
nitate, eventual 0 dacă nu sunt membri. Dacă definim probabilitatea unui
marginea dintre două noduri pentru a fi o funcție a punctelor forte de apartenență la
comunitățile lor comune, putem transforma problema găsirii MLE
într-o problemă continuă și rezolvați-o folosind coborârea în gradient.

♦ Simrank: o modalitate de a măsura similaritatea nodurilor într-un grafic cu
mai multe tipuri de noduri este să porniți un walker aleatoriu la un singur nod și să permiteți
să rătăcească, cu o probabilitate fixă de a reporni la același nod. The
distribuția locului în care se poate aștepta să fie mersul este o măsură bună
de asemănarea nodurilor cu nodul de pornire. Acest proces trebuie să fie
repetat cu fiecare nod ca nod de pornire dacă dorim să obținem toate perechile
similitudine.

♦ Triunghiuri în rețelele sociale: numărul de triunghiuri pe nod este un
măsură importantă a apropierii unei comunități și adesea o reflectă
maturitate. Putem enumera sau număra triunghiurile dintr-un grafic cu m
muchii în timp O (m 3/2), dar nu există un algoritm mai eficient în general.

♦ Găsirea triunghiului prin MapReduce: Putem găsi triunghiuri într-o singură rundă
din MapReduce tratându-l ca o îmbinare cu trei căi. Fiecare margine trebuie trimisă
la un număr de reductoare proporționale cu rădăcina cubică a numărului total
de reductoare, iar timpul total de calcul petrecut la toate reductoarele este
proporțional cu timpul algoritmului serial pentru găsirea triunghiului.

♦ Vecinătăți: vecinătatea razei d pentru un nod v într-o direcționată
sau grafic nedirecționat este setul de noduri accesibile din v de-a lungul căilor de
lungime cel mult d. Profilul de vecinătate al unui nod este secvența
dimensiuni de cartier pentru toate distanțele de la 1 în sus. Diametrul unui
graficul conectat este cel mai mic d pentru care vecinătatea razei d
pentru orice nod de pornire include întregul grafic.

♦ Închidere tranzitivă: un nod v poate ajunge la nodul u dacă u se află în vecinătate
de v pentru o anumită rază. Închiderea tranzitivă a unui grafic este setul de perechi
de noduri (v, u) astfel încât v poate ajunge la u.

♦ Închiderea tranzitivă de calcul: Deoarece închiderea tranzitivă poate avea un
numărul de fapte egal cu pătratul numărului de noduri ale unui grafic, acesta
este imposibil de calculat închiderea tranzitivă direct pentru grafice mari. unu
abordarea este de a găsi componente puternic conectate ale graficului și
descărcați-le fiecare la un singur nod înainte de a calcula închiderea tranzitivă.

Pagina 445

♦ Închidere tranzitivă și MapReduce: Putem vizualiza com
putation ca îmbinare iterativă a unei relații de cale (perechi de noduri v și u
astfel încât u este cunoscut a fi accesibil din v) și relația arc a
grafic. O astfel de abordare necesită un număr de runde MapReduce egale
la diametrul graficului.

♦ Evaluare seminaivă: la calculul închiderii tranzitive pentru un grafic,

putem accelera evaluarea iterativă a relației Cale prin recunoașterea afirmând că un fapt de cale este util numai în runda de după runda respectivă atunci când a fost descoperit pentru prima dată. O idee similară accelerează calculul de accesibilitate și mulți algoritmi iterați similari.

♦ Închiderea tranzitivă prin dublare recursivă: o abordare care folosește mai puține Rundele MapReduce este de a uni relația de cale cu ea însăși la fiecare rundă. La fiecare rundă, dublăm lungimea căilor care pot contribui la închiderea tranzitivă. Astfel, numărul de runde necesare este doar logaritmul de bază-2 al diametrului graficului.

♦ Închidere tranzitivă inteligentă: în timp ce dublarea recursivă poate provoca același lucru calea care trebuie luată în considerare de multe ori și astfel crește calculul total timpul de operație (comparativ cu îmbinarea iterativă a căilor cu arcuri simple), a varianta numită închidere tranzitivă inteligentă evită descoperirea aceleiași căi mai mult de o dată. Trucul este să cereți ca atunci când uniți două căi, mai întâi are o lungime care este o putere de 2.

♦ Aproximarea dimensiunilor de vecinătate: prin utilizarea tehnologiei Flajolet-Martin unic pentru aproximarea numărului de elemente distincte dintr-un flux, putem găsi dimensiunile cartierului la raze diferite aproximativ. Noi mențineți un set de lungimi de coadă pentru fiecare nod. Pentru a mări raza cu 1, examinăm fiecare margine (u, v) și pentru fiecare lungime a cozii pentru u o setăm egală cu lungimea cozii corespunzătoare pentru v dacă aceasta din urmă este mai mare decât fost.

10.10 Referințe pentru capitolul 10

Simrank provine din [8]. O abordare alternativă în [11] vede similaritatea a două noduri ca propabilitatea că mersul aleatoriu de la cele două noduri va fi la același nod. [3] combină plimbările aleatorii cu clasificarea nodurilor pentru a prezice legături într-un grafic de rețea socială. [16] privește eficiența calculului simrank ca un PageRank personalizat.
Algoritmul Girvan-Newman este din [6]. Găsirea comunităților prin căutare pentru graficele bipartite complete apare în [9].
Tăieturi normalizate pentru analiza spectrală au fost introduse în [13]. [19] este o sursă metodele spectrale pentru găsirea clusterelor și [5] este un studiu mai general de găsire a comunității în grafice. [10] este o analiză a comunităților din multe rețele întâlnite în practică.

Detectarea comunităților suprapuse este explorată în [20], [21] și [22].
Numărarea triunghiurilor folosind MapReduce a fost discutată în [15]. Metoda descris aici este din [1], care oferă, de asemenea, o tehnică care funcționează pentru orice subgraf. [17] discută algoritmii randomizați pentru găsirea triunghiului.
Algoritmul ANF a fost investigat pentru prima dată în [12]. [4] oferă un supliment accelerarea la ANF.
Algoritmul inteligent de închidere tranzitivă a fost descoperit de [7] și [18] independent. Implementarea închiderii tranzitive folosind MapReduce sau similar sisteme este discutat în [2].
O implementare open source C ++ a multor algoritmi descriși în acest capitol puteți găsi în biblioteca SNAP [14].

1. FN Afrati, D. Fotakis și JD Ullman, „Enumerarea subgrafului în poziții prin reducerea hărții ”
http://ilpubs.stanford.edu:8090/1020
2. FN Afrati și JD Ullman, „Închidere tranzitivă și Datalog recursiv implementate pe clustere ”, în Proc. EDBT (2012).
3. L. Backstrom și J. Leskovec, „Plimbări aleatorii supravegheate: predicție și recomandând legături în rețelele sociale ”, Proc. Al patrulea ACM Intl. Conf. on Web Search and Data Mining (2011), pp. 635-644.
4. P. Boldi, M. Rosa și S. Vigna, „HyperANF: aproximarea vecinilor-funcția de burghezie a graficelor foarte mari pe un buget, ”Proc. WWW Conference (2011), pp. 625-634.
5. S. Fortunato, „Detectarea comunității în grafice”, Physics Reports 486: 3-5 (2010), pp. 75–174.
6. M. Girvan și MEJ Newman, „Structura comunitară în domeniul social și bi-rețele ologice, ”Proc. Natl. Acad. Știință. 99 (2002), pp. 7821-7826.
7. YE Ioannidis, „Despre calculul închiderii tranzitive a relațiilor operatori, ”Proc. A 12-a Intl. Conf. pe baze de date foarte mari, pp. 403–411.
8. G. Jeh și J. Widom, „Simrank: o măsură a contextului structural similar-

”, Lucrările celei de-a opta conferințe internaționale ACM SIGKDD
pe Discovery Knowledge and Data Mining (2002), pp. 538-543.
9. R. Kumar, P. Raghavan, S. Rajagopalan și A. Tomkins, „Trawling
Web pentru comunități cibernetice emergente, Rețele de calculatoare 31: 11-16
(Mai, 1999), pp. 1481–1493.
10. J. Leskovec, KJ Lang, A. Dasgupta și MW Mahoney, „Comunitate
structura în rețele mari: dimensiuni de cluster naturale și absența unor mari
clustere bine definite ", http://arxiv.org/abs/0810.1355.

Pagina 447

11. S. Melnik, H. Garcia-Molina și E. Rahm, „Inundații similare: o ver-
algoritmul de potrivire a graficului satilic și aplicarea acestuia la potrivirea schemelor,
Proc. Intl. Conf. on Data Engineering (2002), pp. 117-128.
12. CR Palmer, PB Gibbons și C. Faloutsos, „ANF: un rapid și scalabil
instrument pentru extragerea datelor în grafice masive, ”Proc. Al optulea ACM SIGKDD
Intl. Conf. pe Knowledge Discovery and Data Mining (2002), pp. 81–90.
13. J. Shi și J. Malik, „Cutii normalizate și segmentarea imaginii”, IEEE
Trans. on Pattern Analysis and Machine Intelligence ”, 22: 8 (2000), pp. 888–
905.
14. Platforma de analiză a rețelei Stanford, http://snap.stanford.edu.
15. S. Suri și S. Vassilivitskii, „Numărarea triunghiurilor și blestemul ultimului
reductor, ”Proc. Conferința WWW (2011).
16. H. Tong, C. Faloutsos și J.-Y. Pan, „Plimbare rapidă aleatorie cu repornire
și aplicațiile sale ", ICDM 2006, pp. 613-622.
17. CE Tsourakakis, U. Kang, GL Miller și C. Faloutsos, „DOULION:
numărând triunghiurile în grafice masive cu o monedă, ”Proc. Al cincisprezecelea ACM
SIGKDD Intl. Conf. pe Discovery Knowledge and Data Mining (2009).
18. P. Valduriez și H. Boral, „Evaluarea interogărilor recursive folosind join
indici, ”Expert Database Conf. (1986), pp. 271–293.
19. U. von Luxburg, „Un tutorial despre clusterizarea spectrală”, Statistică și Com-
punerea bf17: 4 (2007), 2007, pp. 395-416.
20. J. Yang și J. Leskovec, „Suprapunerea detectării comunității la scară: a
abordare de factorizare a matricii non-negative ”, ACM International Conference-
în căutare web și minerit de date, 2013.
21. J. Yang, J. McAuley, J. Leskovec, „Detectarea coeziunii și a modului 2
comunități în rețelele direcționate și nedirecționate ", ACM International Con-
ference on Web Search and Data Mining, 2014.
22. J. Yang, J. McAuley, J. Leskovec, „Detectarea comunității în rețele cu
atributele nodului ”, IEEE International Conference On Data Mining, 2013.

Pagina 448

Pagina 449

Capitolul 11
Reducerea dimensiunii

Există multe surse de date care pot fi privite ca o matrice mare. Noi
am văzut în Capitolul 5 cum Web-ul poate fi reprezentat ca o matrice de tranziție. În
Capitolul 9, matricea utilitară a fost un punct de concentrare. Și în capitolul 10 noi
au examinat matrici care reprezintă rețelele sociale. În multe dintre aceste matrice
aplicații, matricea poate fi rezumată prin găsirea matricilor „mai înguste”
care într-un anumit sens sunt apropiate de original. Aceste matrice înguste au doar a
un număr mic de rânduri sau un număr mic de coloane și, prin urmare, pot fi utilizate
mult mai eficient decât poate matricea mare originală. Procesul de găsire
aceste matrice înguste se numesc reducerea dimensionalității.
Am văzut un exemplu preliminar de reducere a dimensionalității în secțiunea 9.4.
Acolo, am discutat despre descompunerea UV a unei matrice și am dat un algoritm simplu
pentru găsirea acestei descompuneri. Amintiți-vă că o matrice mare M a fost descompusă

în două matrice U și V al căror produs UV a fost aproximativ M. The
matricea U avea un număr mic de coloane, în timp ce V avea un număr mic de rânduri,
deci fiecare era semnificativ mai mic decât M și totuși împreună reprezentau
majoritatea informațiilor din M care au fost utile în prezicerea evaluărilor articolelor după
indivizi.
În acest capitol vom explora ideea reducerii dimensionalității în
mai multe detalii. Începem cu o discuție despre valorile proprii și utilizarea lor în „principii
analiza componentelor cipale ”(PCA). Acoperim descompunerea cu valoare singulară, a
versiune mai puternică a descompunerii UV. În sfârșit, pentru că suntem întotdeauna
interesați de cele mai mari dimensiuni de date pe care le putem gestiona, ne uităm la un alt formular
de descompunere, numită CUR-descompunere, care este o variantă de singular-
descompunerea valorii care menține matricele descompunerii rare dacă
matricea originală este rară.

11.1 Valori proprii și vectori proprii de simetrie ric Matrici

Vom presupune că sunteți familiarizați cu elementele de bază ale algebrei matriciale: multi-
plicarea, transpunerea, determinanții și rezolvarea ecuațiilor liniare, de exemplu. În
în această secțiune, vom defini valorile proprii și vectorii proprii ai unei matrici simetrice
și arată cum să le găsești. Reamintim o matrice este simetrică dacă elementul din rând
i și coloana j este egal cu elementul din rândul j și coloana i.

11.1.1 Definiții

Fie M o matrice pătrată. Fie λun vector de coloană constant și ea diferit de zero
cu același număr de rânduri ca M. Atunci λeste o valoare proprie a lui M și e este
vectorul propriu corespunzător al lui M dacă Me = λe.
Dacă e este un vector propriu al lui M și c este orice constantă, atunci este de asemenea adevărat că
ce este un vector propriu al lui M cu aceeași valoare proprie. Înmulțirea unui vector cu a
constantă modifică lungimea unui vector, dar nu și direcția acestuia. Astfel, pentru a evita
ambiguitate cu privire la lungime, vom cere ca fiecare vector propriu să fie a
vector unitate, adică suma pătratelor componentelor
vectorul este 1. Chiar și asta nu este suficient pentru a face vectorul propriu unic,
deoarece putem încă să ne înmulțim cu -1 fără a schimba suma pătratelor lui
componente. Astfel, în mod normal, vom cere ca prima componentă diferită de zero
a unui vector propriu să fie pozitiv.
Exemplul 11.1: Fie M matricea
[3 2
2 6]
Unul dintre vectorii proprii ai lui M este
[1 / $\sqrt{5}$
2 / $\sqrt{5}$]
iar valoarea sa corespunzătoare este 7. Ecuația
[3 2
2 6] [
1 / $\sqrt{5}$
2 / $\sqrt{5}$] = 7 [
1 / $\sqrt{5}$
2 / $\sqrt{5}$]
demonstrează adevărul acestei afirmații. Rețineți că ambele părți sunt egale cu
[7 / $\sqrt{5}$
14 / $\sqrt{5}$]
De asemenea, observați că vectorul propriu este un vector unitar, deoarece $(1 / \sqrt{5})^2 + (2 / \sqrt{5})^2 =$
1/5 + 4/5 = 1. ✷

11.1.2 Calcularea valorilor proprii și a vectorilor proprii

Am văzut deja o abordare pentru găsirea unei perechi proprii (o valoare proprie și
vectorul propriu corespunzător) pentru o matrice adecvată M în secțiunea 5.1: începeți cu
orice vector unitate v de lungimea corespunzătoare și calculați M^i v iterativ până la el
converge. 1 Când M este o matrice stocastică, vectorul limitativ este principalul
eigenvector (vectorul propriu cu cea mai mare valoare proprie) și corespunzător

valoarea proprie este 1. [2] Această metodă pentru găsirea vectorului propriu principal, numită putere iterație, funcționează destul de general, deși dacă valoarea proprie principală (valoarea proprie asociat cu vectorul propriu principal) nu este 1, atunci pe măsură ce crește, raportul a lui $M^{i+1}v$ la $M^{i}v$ se apropie de valoarea proprie principală în timp ce $M^{i}v$ se apropie un vector (probabil nu un vector unitar) cu aceeași direcție ca principalul vector propriu.

Vom prelua generalizarea metodei de iterație a puterii pentru a găsi toate perechi proprii în secțiunea 11.1.3. Cu toate acestea, există o metodă de timp de rulare $O(n^3)$ pentru a calcula exact toate perechile proprii ale unei matrici simetrice $n \times n$, și asta metoda va fi prezentată mai întâi. Vor exista întotdeauna n perechi proprii, deși în unele cazuri, unele dintre valorile proprii vor fi identice. Metoda începe prin reformulând ecuația care definește perechi proprii, $Me = \lambda e$ca $(M - \lambda I)e = 0$,

Unde

1. I este matricea de identitate $n \times n$ cu 1 de-a lungul diagonalei principale și 0 în altă parte.
2. 0 este un vector al tuturor celor 0.

Un fapt al algebrei liniare este că, pentru ca $(M - \lambda I)e = 0$ să se mențină pentru a vectorul $e = 0$, determinantul lui $M - \lambda I$trebuie să fie 0. Observați că $(M - \lambda I)$ arată aproape ca matricea M, dar dacă M are c într-unul din elementele sale diagonale, atunci $(M - \lambda I)$are $c - \lambda$acolo. În timp ce determinantul unei matrice $n \times n$ are $n!$ în termeni, poate fi calculat în diferite moduri în timpul $O(n^3)$; un exemplu este metoda „condensării pivotante”.

Determinantul lui $(M - \lambda I)$este un polinom de gradul n în λ,din care putem obține n valorile lui λcare sunt valorile proprii ale lui M. Pentru orice astfel de valoare, să spunem c, putem rezolva apoi ecuația $Me = c\,e$. Există n ecuații în n necunoscute (cele n componente ale e), dar din moment ce nu există niciun termen constant în niciunul ecuație, putem rezolva doar pentru e într-un factor constant. Cu toate acestea, folosind orice soluție, o putem normaliza astfel încât suma pătratelor componentelor este 1, obținându-se astfel vectorul propriu care corespunde valorii proprii c.

Exemplul 11.2: Să găsim perechile proprii pentru matricea 2×2 M din Exemplu 11.1. Reamintim M =

[3 2
2 6]

[1] Reamintim M^{i} înseamnă multiplicarea cu matricea M i ori, așa cum sa discutat în secțiunea 5.1.2.

[2] Rețineți că o matrice stocastică nu este în general simetrică. Matrici simetrice și matricile stochastice sunt două clase de matrice pentru care există perechi proprii și pot fi exploatate. În acest capitol, ne concentrăm asupra tehnicilor pentru matrici simetrice.

Pagina 452

Atunci $M - \lambda I$este

[3 - λ
2
2
6 - λ]

Determinantul acestei matrice este $(3 - \lambda)(6 - \lambda) - 4$, pe care trebuie să îl setăm la 0. Ecuația din λde rezolvat este astfel $\lambda^2 - 9\lambda + 14 = 0$. Rădăcinile acestei ecuații sunt $\lambda = 7$ și $\lambda = 2$; primul este valoarea proprie principală, deoarece este mai mare.

Fie e vectorul necunoscutelor

[X y]

Trebuie să rezolvăm

[3 2
2 6] [
X
y] = 7 [
X
y]

Când înmulțim matricea și vectorul obținem două ecuații

$3x + 2y = 7x$
$2x + 6y = 7y$

Observați că ambele ecuații spun cu adevărat același lucru: $y = 2x$. Astfel, a posibil vectorul propriu este

[12]

Dar acel vector nu este un vector unitate, deoarece suma pătratelor compoziției sale nents este 5, nu 1. Astfel, pentru a obține vectorul unitar în aceeași direcție, împărțim fiecare componentă cu $\sqrt{5}$. Adică, vectorul propriu principal este

[$1/\sqrt{5}$
$2/\sqrt{5}$]

iar valoarea sa proprie este 7. Rețineți că aceasta a fost perechea proprie pe care am explorat-o în
ple 11.1.
Pentru a doua pereche proprie, repetăm cele de mai sus cu valoarea proprie 2 în locul lui
7. Ecuația care implică componentele lui e este x = −2y, iar a doua
vectorul propriu este
[
2 / √5
−1 / √5]
Valoarea sa corespunzătoare este 2, desigur. ✷

11.1.3 Găsirea perechilor proprii prin iterație de putere

Acum examinăm generalizarea procesului pe care l-am folosit în secțiunea 5.1 până la
găsiți vectorul propriu principal, care în acea secțiune era vectorul PageRank -
tot ce ne trebuia dintre diferiții vectori proprii ai matricei stochastice a
internetul. Începem prin a calcula vectorul propriu principal cu o ușoară generală-
a abordării utilizate în secțiunea 5.1. Apoi modificăm matricea la, în

---

efect, eliminați vectorul propriu principal. Rezultatul este o nouă matrice ale cărei principii
vectorul propriu cipal este al doilea vector propriu (vectorul propriu cu al doilea ca mărime
valoare proprie) a matricei originale. Procesul se desfășoară în acest mod, re-
mutând fiecare vector propriu așa cum îl găsim și apoi folosind iterația de putere pentru a găsi
principalul vector propriu al matricei care rămâne.
Fie M matricea ale cărei perechi proprii am dori să le găsim. Începeți cu oricare
vector nenul x_0 și apoi iterați:
$x_{k+1} :=$
Mx_k
Mx_k
unde N pentru o matrice sau vector N denotă norma Frobenius; adică
rădăcină pătrată a sumei pătratelor elementelor lui N. Înmulțim
vectorul curent x_k de matricea M până la convergență (adică, $x_k - x_{k+1}$
este mai puțin
decât o mică constantă aleasă). Fie x x_k pentru acea valoare a lui k la care
se obține convergența. Atunci x este (aproximativ) vectorul propriu principal
lui M. Pentru a obține valoarea proprie corespunzătoare, calculăm pur și simplu $\lambda_1 = x^T Mx$,
care este ecuația Mx = λxrezolvată pentru λ,deoarece x este un vector unitar.
Exemplul 11.3: Luați matricea din Exemplul 11.2:
M = [
3 2
2 6]
și să începem cu x_0 un vector cu 1 pentru ambele componente. Pentru a calcula x_1 ,
înmulțim Mx_0 pentru a obține
[3 2
2 6] [
1
1] = [
5
8]
Norma Frobenius a rezultatului este $\sqrt{5^2 + 8^2} = \sqrt{89} = 9.434$. Obținem x_1 cu
împărțirea 5 și 8 la 9.434; acesta este:
x_1 = [
0,530
0,848]
Pentru următoarea iterație, calculăm
[3 2
2 6] [
0,530
0.848] = [
3.286
6.148]
Norma Frobenius a rezultatului este 6,971, deci ne împărțim pentru a obține
x_2 = [
0,471
0,882]
Convergem către un vector normal a cărui a doua componentă este de două ori
primul. Adică valoarea limitativă a vectorului pe care îl obținem prin iterație de putere
este vectorul propriu principal:
x = [

0,447
0,884]

Pagina 454

În cele din urmă, calculăm valoarea proprie principală prin
$\lambda = x^T Mx$ = [0,447 0,894]
[3 2
2 6] [
0,447
0,884]
= 6,993
Reamintim din Exemplul 11.2 că adevărata valoare principală principală este 7. Iterație de putere va introduce mici erori datorate fie preciziei limitate, cum a fost cazul aici, sau datorită faptului că oprim iterația înainte de a atinge valoarea exactă a vectorul propriu. Când am calculat PageRank, micile inexactități nu au făcut-o contează, dar când încercăm să calculăm toate perechile proprii, inexactitățile se acumulează dacă nu suntem atenți. ✷
Pentru a găsi al doilea eigenpair vom crea o nouă matrice $M^* = M - \lambda_1 xx^T$.
Apoi, utilizați iterația de putere pe M^* pentru a calcula cea mai mare valoare proprie. Obmenținute x^* și λ^* corespund celei de-a doua mari valori proprii și vectorul propriu sponding al matricei M.
Intuitiv, ceea ce am făcut este să eliminăm influența unui anumit eigenvector setând valoarea sa asociată la zero. Justificarea formală este următoarele două observații. Dacă $M^* = M - \lambda xx^T$, unde x și λ sunt pereche proprie cu cea mai mare valoare proprie, apoi:
1. x este, de asemenea, un vector propriu al lui M^*, iar valoarea sa corespunzătoare este 0. In dovadă, observă asta
$M^* x = (M - \lambda xx^T) x = Mx - \lambda xx^T x = Mx - \lambda x = 0$
La pas-la-ultimul pas folosim faptul că $x^T x = 1$ deoarece x este o unitate vector.
2. În schimb, dacă v și λ_v sunt o pereche proprie a unei matrice simetrice M altele decât prima pereche proprie (x, λ), atunci ele sunt, de asemenea, o pereche proprie de M^*.
Dovadă:
$M^* v = (M^*)^T v = (M - \lambda xx^T)^T v = M^T v - \lambda x(x^T v) = M^T v = \lambda_v v$
Această succesiune de egalități are nevoie de următoarele justificări:
(a) Dacă M este simetrică, atunci $M = M^T$.
(b) Vectorii proprii ai unei matrici simetrice sunt ortogonali. Adică produsul punct al oricăror doi vectori proprii diferiți ai unei matrice este 0. Noi facem nu dovedește această afirmație aici.
Exemplul 11.4: Continuând exemplul 11.3, calculăm
M^* = [
3 2
2 6] - 6,993 [
0,447
0,894] [0,447 0,894] =
[3 2
2 6] - [
1,397 2,795
2,795 5,589] = [
1,603 −0,795
−0,795
0.411]
Putem găsi a doua pereche proprie prin procesarea matricei de mai sus așa cum am făcut matrice originală M. ✷

Pagina 455

11.1.4 Matricea vectorilor proprii

Să presupunem că avem o matrice simetrică n × n ale cărei vectori proprii, priviți ca vectori coloană, sunt $e_1, e_2, ..., e_n$. Fie E matricea a cărei coloană este e_i. Atunci $EE^T = E^T E = I$. Explicația este că vectorii proprii ai lui a matricea simetrică este ortonormală. Adică sunt vectori unitari ortogonali.
Exemplul 11.5: Pentru matricea M din Exemplul 11.2, matricea E este
[

2 / √51 / √5
−1 / √52 / √5]
E T este deci
[2 / √5−1 / √5
1 / √5
2 / √5]
Când calculăm EE T obținem
[
4/5 + 1/5 −2 / 5 + 2/5
−2 / 5 + 2/5
1/5 + 4/5] = [
1 0
0 1]
Calculul este similar atunci când calculăm E T E. Observați că 1 este de-a lungul diagonala principală sunt sumele pătratelor componentelor fiecăruia dintre vectorii proprii, ceea ce are sens deoarece sunt vectori unitari. 0 este în afara diagonala reflectă faptul că intrarea în al doilea rând și coloana jth este produs dot al vectorilor proprii ith și jth. Deoarece vectorii proprii sunt ortogonali, aceste produse dot sunt 0. ✷

11.1.5 Exerciții pentru secțiunea 11.1

Exercițiul 11.1.1: Găsiți vectorul unitar în aceeași direcție ca vectorul [1, 2, 3].

Exercițiul 11.1.2: Completați Exemplul 11.4 calculând propriul principal-vectorul matricei care a fost construit în acest exemplu. Cât de aproape de soluție corectă (din Exemplul 11.2) ești?

Exercițiul 11.1.3: Pentru orice matrice simetrică 3 × 3
□
□
a - λ
b
c
b
d - λ
e
c
e
f - λ
□
□
există o ecuație cubică în λcare spune că determinantul acestei matrice este 0. În termenii de la a până la f, găsiți această ecuație.

Pagina 456

Exercițiul 11.1.4: Găsiți perechile proprii pentru următoarea matrice:
□
□
1 1 1
1 2 3
1 3 5
□
□
folosind metoda secțiunii 11.1.2.

! Exercițiul 11.1.5: Găsiți perechile proprii pentru următoarea matrice:
□
□
1 1 1
1 2 3
1 3 6
□
□
folosind metoda secțiunii 11.1.2.

Exercițiul 11.1.6: Pentru matricea exercițiului 11.1.4:
(a) Începând cu un vector de trei 1, utilizați iterația de putere pentru a găsi o aprox.
imitați valoarea vectorului propriu principal.
(b) Calculați o estimare a valorii proprii principale pentru matrice.
(c) Construiți o nouă matrice scăzând efectul principalului
pereche proprie, ca în secțiunea 11.1.3.
(d) Din matricea dvs. de (c), găsiți a doua pereche proprie pentru matricea originală
a exercițiului 11.1.4.
(e) Repetați (c) și (d) pentru a găsi a treia pereche proprie pentru matricea originală.
Exercițiul 11.1.7: Repetați exercițiul 11.1.6 pentru matricea exercițiului 11.1.5.

11.2 Analiza componentelor principale

Analiza componentelor principale, sau PCA, este o tehnică pentru luarea unui set de date con-
rezistență la un set de tupluri reprezentând puncte într-un spațiu de înaltă dimensiune și
găsind direcțiile de-a lungul cărora se aliniază cel mai bine tuplurile. Ideea este să tratezi
setul de tupluri ca matrice M și găsiți vectorii proprii pentru MM T sau M T M.
Matricea acestor vectori proprii poate fi gândită ca o rotație rigidă într-o
spațiul dimensional. Când aplicați această transformare datelor originale,
axa corespunzătoare vectorului propriu principal este cea de-a lungul căreia
punctele sunt cele mai „întinse”, mai exact, această axă este cea de-a lungul căreia
varianța datelor este maximizată. Altfel spus, punctele pot fi cel mai bine
privită ca situată de-a lungul acestei axe, cu mici abateri de la această axă. De asemenea,
axa corespunzătoare celui de-al doilea vector propriu (vectorul propriu corespunde
a doua cea mai mare valoare proprie) este axa de-a lungul căreia variația lui
distanțele față de prima axă sunt mai mari și așa mai departe.

Putem privi PCA ca o tehnică de extragere a datelor. Datele de înaltă dimensiune
poate fi înlocuit prin proiecția sa pe cele mai importante axe. Aceste axe
sunt cele care corespund celor mai mari valori proprii. Astfel, datele originale
este aproximat de date care au mult mai puține dimensiuni și care rezumă
bine datele originale.

11.2.1 Un exemplu ilustrativ

Vom începe expunerea cu un exemplu simplu și inventat. In acest
de exemplu, datele sunt bidimensionale, un număr de dimensiuni prea mic
pentru a face PCA cu adevărat utilă. Mai mult, datele, prezentate în Fig. 11.1 au numai
patru puncte și sunt aranjate într-un model simplu de-a lungul liniei de 45 de grade
pentru a face calculele noastre ușor de urmat. Adică pentru a anticipa rezultatul,
punctele pot fi privite cel mai bine ca situate de-a lungul axei care este la un unghi de 45 de grade,
cu mici abateri în direcția perpendiculară.
(2,1)
(3,4)
(4,3)
(1,2)
Figura 11.1: Patru puncte într-un spațiu bidimensional
Pentru început, să reprezentăm punctele printr-o matrice M cu patru rânduri - unul
pentru fiecare punct - și două coloane, corespunzătoare axei x și axa y. Acest
matricea este
M =
□
□
1 2
2 1
3 4
4 3
□
□
□
□
Calculați M T M, care este
M T M = [
1 2 3 4
2 1 4 3]
□

□
□
□
1 2
2 1
3 4
4 3
□
□
□
□
= [
30 28
28 30]
Putem găsi valorile proprii ale matricei de mai sus rezolvând ecuația
(30 - λ)(30 - λ)- 28 × 28 = 0

Pagina 458

așa cum am făcut în exemplul 11.2. Soluția este λ= 58 și λ= 2.
Urmând aceeași procedură ca în Exemplul 11.2, trebuie să rezolvăm
[30 28
28 30] [
X
y] = 58 [
X
y]
Când înmulțim matricea și vectorul, obținem două ecuații
30x + 28y =
58x
28x + 30y =
58y
Ambele ecuații ne spun același lucru: x = y. Astfel, vectorul propriu al unității
corespunzând valorii proprii principale 58 este
[1 / √2
1 / √2]
Pentru a doua valoare proprie, 2, efectuăm același proces. Înmulțiți-vă
[30 28
28 30] [
X
y] = 2 [
X
y]
pentru a obține cele două ecuații
30x + 28y = 2x
28x + 30y = 2y
Ambele ecuații ne spun același lucru: x = −y. Astfel, vectorul propriu al unității
corespunzând valorii proprii principale 2 este
[−1 / √2
1 / √2]
În timp ce am promis să scriem vectori proprii cu prima lor componentă pozitivă,
alegem aici opusul deoarece face transformarea coordonatelor
mai ușor de urmărit în acest caz.
Acum, să construim E, matricea vectorilor proprii pentru matricea M T M.
Plasând primul vectorul propriu principal, matricea vectorilor proprii este
E = [
1 / √2−1 /
√2
1 / √2
1 / √2]
Orice matrice de vectori ortonormali (vectori unitari care sunt ortogonali la unul
alta) reprezintă o rotație și / sau reflexie a axelor unui spațiu euclidian.
Matricea de mai sus poate fi privită ca o rotație de 45 de grade în sens invers acelor de ceasornic. Pentru
de exemplu, să înmulțim matricea M care reprezintă fiecare dintre punctele lui

Fig. 11.1 de E. Produsul este
ME =
□
□
1 2
2 1
3 4
4 3
□
□
□
□
[1 / √2−1 / √2
1 / √2
1 / √2] =
□
□
3 / √2
1 / √2
3 / √2−1 /
√2
7 / √2
1 / √2
7 / √2−1 /
√2
□
□
□
□

11.2. ANALIZA COMPONENTELOR PRINCIPALE
439
(2,1)
(3,4)
(4,3)
(1,2)
(1,5,1,5)
(3,5,3,5)
Figura 11.2: Figura 11.1 cu axele rotite cu 45 de grade în sens invers acelor de ceasornic
Vedem că primul punct, [1, 2], a fost transformat în punct
[3 / √2, 1 / √2]
Dacă examinăm Fig. 11.2, cu linia întreruptă reprezentând noua axă x, vom
vezi că proiecția primului punct pe acea ax o plasează la distanță
3 / √2de la origine. Pentru a verifica acest fapt, observați că punctul de proiecție pentru
atât primul, cât și al doilea punct este [1.5, 1.5] în sistemul de coordonate original,
iar distanța de la origine până la acest punct este
√ (1.5) 2 + (1.5) 2 = √9/ 2 = 3 /
√2
Mai mult, noua axă y este, desigur, perpendiculară pe linia punctată. The
primul punct este la distanța 1 / √2deasupra noii axe x în direcția
axa y. Adică distanța dintre punctele [1, 2] și [1.5, 1.5] este
√ (1 - 1,5) 2 + (2 - 1,5) 2 = √(−1/2) 2 + (1/2) 2 = √1/ 2 = 1 /
√2
Figura 11.3 prezintă cele patru puncte din sistemul de coordonate rotit.
2
2
(3 /
, −1 /
)
2
2
(3 /

, 1 /
)
2
2
(7 /
, 1 /
)
2
2
(7 /
, −1 /
)
Figura 11.3: Punctele din Fig. 11.1 în noul sistem de coordonate
Al doilea punct, [2, 1] se întâmplă prin coincidență să se proiecteze pe același
punctul noii axe x. Este 1 / √2sub acea axă de-a lungul noii axe y, așa cum este

---

confirmat de faptul că al doilea rând din matricea punctelor transformate
este [3 / √2, −1 /
√2]. Al treilea punct, [3, 4] este transformat în [7 / √2, 1 / √2] și
al patrulea punct, [4, 3], este transformat în [7 / √2, −1 /
√2]. Adică amândoi
proiectează pe același punct al noii axe x, iar acel punct se află la distanță
7 / √2de la origine, în timp ce sunt 1 / √2deasupra și sub noua axă x din
direcția noii axe y.

11.2.2 Utilizarea vectorilor proprii pentru reducerea dimensiunii

Din exemplul pe care tocmai l-am elaborat, putem vedea un principiu general. Dacă
M este o matrice ale cărei rânduri reprezintă fiecare un punct într-un spațiu euclidian cu
orice număr de dimensiuni, putem calcula $M^T M$ și putem calcula propriile perechi.
Fie E matricea ale cărei coloane sunt vectorii proprii, ordonați ca fiind cei mai mari
valoarea proprie mai întâi. Definiți matricea L pentru a avea valorile proprii ale lui $M^T M$ de-a lungul
diagonala, prima cea mai mare și 0 în toate celelalte intrări. Apoi, din moment ce $M^T Me =$
$\lambda e = e\lambda$ pentru fiecare vector propriu e și valoarea sa corespunzătoare λ,urmează
că $M^T ME = EL$.
Am observat că ME reprezintă punctele lui M transformate într-o nouă coordonare
spațiu nate. În acest spațiu, prima axă (cea corespunzătoare celei mai mari
valoarea proprie) este cea mai semnificativă; formal, varianța punctelor de-a lungul aceleia
axa este cea mai mare. A doua axă, corespunzătoare celei de-a doua perechi proprii,
este în continuare cel mai semnificativ în același sens și modelul continuă pentru fiecare
a perechilor proprii. Dacă vrem să transformăm M într-un spațiu cu mai puține dimensiuni
alegeri care păstrează cea mai mare semnificație este cea care folosește
vectorii proprii asociați cu cele mai mari valori proprii și îl ignoră pe celălalt
valori proprii.
Adică, să fie E_k primele k coloane ale lui E. Atunci ME_k este o k-dimensională
reprezentarea lui M.
Exemplul 11.6: Fie M matricea din secțiunea 11.2.1. Aceste date au doar
două dimensiuni, deci singura reducere a dimensionalității pe care o putem face este să folosim k = 1;
adică, proiectați datele pe un spațiu unidimensional. Adică calculăm ME_1
de
□
□
□
□
1 2
2 1
3 4
4 3
□
□
□
□
[1 / √2
1 / √2] =

□
□
3 / √2
3 / √2
7 / √2
7 / √2
□
□
□
□
Efectul acestei transformări este de a înlocui punctele lui M cu pro-
jecții pe axa x din Fig. 11.3. În timp ce primele două puncte se proiectează către
același punct, ca și al treilea și al patrulea punct, această reprezentare face ca
cele mai bune distincții unidimensionale posibile între puncte. ✷

11.2.3 Matricea distanțelor

Să ne întoarcem la exemplul Secțiunii 11.2.1, dar în loc să începem cu
$M^T M$, să examinăm valorile proprii ale MM^T. Întrucât exemplul nostru M are
mai multe rânduri decât coloane, acesta din urmă este o matrice mai mare decât primul, dar dacă
M avea mai multe coloane decât rânduri, de fapt am obține o matrice mai mică. În
exemplul care rulează, îl avem
MM^T =
□
□
1 2
2 1
3 4
4 3
□
□
□
□
[1 2 3 4
2 1 4 3] =
□
□
5
4 11 10
4
5 10 11
11 10 25 24
10 11 24 25
□
□
□
□
La fel ca $M^T M$, vedem că MM^T este simetric. Intrarea în al doilea rând și
coloana a X-a are o interpretare simplă; este produsul punct al vectorilor
reprezentate de punctele ith și jth (rânduri de M).
Există o relație puternică între valorile proprii ale lui $M^T M$ și MM^T.
Să presupunem că e este un vector propriu al lui $M^T M$; acesta este,
$M^T Me = \lambda e$
Înmulțiți ambele părți ale acestei ecuații cu M în stânga. Apoi
$MM^T (Me) = M\lambda e = \lambda(Me)$
Astfel, atâta timp cât Me nu este vectorul zero 0, acesta va fi un vector propriu al MM^T
și λ va fi o valoare proprie a MM^T, precum și a lui $M^T M$.
Și conversația se menține. Adică, dacă e este un vector propriu al MM^T cu
valoarea proprie corespunzătoare λ, apoi începeți cu $MM^T e = \lambda e$ și multiplicați pe
lăsat de M^T pentru a concluziona că $M^T M (M^T e) = \lambda(M^T e)$. Astfel, dacă $M^T e$ nu este 0,

atunci λeste, de asemenea, o valoare proprie a lui M T M.
Ne-am putea întreba ce se întâmplă când M T e = 0. În acest caz, MM T e
este, de asemenea, 0, dar e nu este 0, deoarece 0 nu poate fi un vector propriu. Cu toate acestea, din moment ce
0 = λe, concluzionăm că λ= 0.
Concluzionăm că valorile proprii ale MM T sunt valorile proprii ale lui M T M
plus 0 suplimentare. Dacă dimensiunea MM T ar fi mai mică decât dimensiunea
din M T M, atunci s-ar întâmpla contrariul; valorile proprii ale lui M T M ar fi
cele ale MM T plus 0 suplimentare.

□
□
□
□
3 / √116
1/2
7 / √116
1/2
3 / √116 −1/2
7 / √116 −1/2
7 / √116
1/2 −3 /
√116
−1/2
7 / √116 −1/2 −3 /
√116
1/2
□
□
□
□

Figura 11.4: Matricea vectorului propriu pentru MM T

Exemplul 11.7: Valorile proprii ale MM T pentru exemplul nostru de rulare trebuie să includă
includ 58 și 2, deoarece acestea sunt valorile proprii ale M T M așa cum am observat în
Secțiunea 11.2.1. Deoarece MM T este o matrice 4 × 4, are alte două valori proprii,
care trebuie să fie ambele 0. Matricea vectorilor proprii corespunzătoare 58, 2, 0,
iar 0 este prezentat în Fig. 11.4. ✷

11.2.4 Exerciții pentru secțiunea 11.2

Exercițiul 11.2.1: Fie M matricea punctelor de date

□
□
□
□
1
1
2
4
3
9
4 16
□
□
□
□

(a) Ce sunt M T M și MM T ?
(b) Calculați perechile proprii pentru M T M.
! (c) Care vă așteptați să fie valorile proprii ale MM T ?
! (d) Găsiți vectorii proprii ai MM T , folosind valorile proprii din partea (c).
! Exercițiul 11.2.2: Dovediți că dacă M este orice matrice, atunci M T M și MM T sunt
simetric.

11.3 Descompunerea valorii singulare

Acum luăm o a doua formă de analiză matricială care duce la o dimensiune redusă reprezentarea unei matrice de înaltă dimensiune. Această abordare, numită singular-descompunerea valorii (SVD), permite o reprezentare exactă a oricărei matrice și facilitează, de asemenea, eliminarea părților mai puțin importante ale acelei reprezentări produce o reprezentare aproximativă cu orice număr dorit de dimensiuni.

Desigur, cu cât alegem mai puține dimensiuni, cu atât va fi mai puțin precisă apropiere.

Începem cu definițiile necesare. Apoi, explorăm ideea că SVD definește un număr mic de „concepte” care conectează rândurile și coloanele a matricei. Arătăm cum eliminarea conceptelor cel mai puțin importante ne oferă o reprezentare mai mică care se apropie îndeaproape de matricea originală. Apoi, noi vezi cum pot fi folosite aceste concepte pentru a interoga matricea originală mai eficient, și în cele din urmă oferim un algoritm pentru efectuarea SVD în sine.

11.3.1 Definiția SVD

Fie M o matrice m × n, iar rangul lui M să fie r. Reamintim că rangul de o matrice este cel mai mare număr de rânduri (sau echivalent coloane) pe care le putem alege

---

Pagina 463

pentru care nicio combinație liniară nulă a rândurilor nu este vectorul zero 0 (noi să spunem că un set de astfel de rânduri sau coloane este independent). Atunci putem găsi matrici U, Σ și V așa cum se arată în Fig. 11.5 cu următoarele proprietăți:

1. U este o m × r coloană-matrice ortonormală; adică fiecare dintre coloanele sale este un vector unitate și produsul punct al oricăror două coloane este 0.
2. V este o coloană n × r-matrice ortonormală. Rețineți că folosim întotdeauna V în forma sa transpusă, deci rândurile lui V^T sunt ortonormale.
3. Σ este o matrice diagonală; adică toate elementele care nu sunt pe diagonala principală sunt 0. Elementele lui Σ se numesc valorile singulare ale lui M.

M
m
n
V
T
r
U
r
Σ
=
n
r

Figura 11.5: Forma unei descompuneri cu valoare singulară

Exemplul 11.8: Figura 11.6 oferă o matrice de rang 2 care reprezintă ratinguri de filme de utilizatori. În acest exemplu inventat există două „concepte” care stau la baza filmele: science-fiction și romantism. Toți băieții apreciază doar science-fiction, iar toate fetele votează doar romantism. Este această existență a doi respectați cu strictețe concepte care conferă matricei un rang de 2. Adică, putem alege unul dintre primele patru rânduri și unul dintre ultimele trei rânduri și observați că nu există niciun zero suma liniară a acestor rânduri care este 0. Dar nu putem alege trei rânduri independente. De exemplu, dacă alegem rândurile 1, 2 și 7, atunci de trei ori primul minus al doilea, plus de zero ori al șaptelea este 0.

Putem face o observație similară despre coloane. Putem alege unul din primele trei coloane și una din ultimele două coloane și vor fi independent, dar niciun set de trei coloane nu este independent.

Descompunerea matricei M din Fig. 11.6 în U, Σ și V, cu toate elementele corecte la două cifre semnificative, este prezentat în Fig. 11.7. Din moment ce rangul lui M este 2, putem folosi r = 2 în descompunere. Vom vedea cum calculați această descompunere în secțiunea 11.3.6. ✷

Pagina 464

1 1 1 0 0
3 3 3 0 0
4 4 4 0 0
5 5 5 0 0

0 0 0 4 4
0 0 0 5 5
0 0 0 2 2
Joe
Jim
Ioan
Jack
Jill
Jenny
Jane
Casablanca
Matrice
Titanic
Străin
Razboiul Stelelor
Figura 11.6: Evaluările filmelor de către utilizatori
□
□
□
□
□
□
□
□
□
□
1 1 1 0 0
3 3 3 0 0
4 4 4 0 0
5 5 5 0 0
0 0 0 4 4
0 0 0 5 5
0 0 0 2 2
□
□
□
□
□
□
□
□
□
□
=
□
□
□
□
□
□
□
□
□
□
.14
0
.42
0
.56
0
.70

0
0
.60
0
.75
0
.30
□
□
□
□
□
□
□
□
□
□
[12.4
0
0
9.5] [
.58 .58 .58
0
0
0
0
0
.71 .71]
M
U
Σ
V T
Figura 11.7: SVD pentru matricea M din Fig. 11.6

11.3.2 Interpretarea SVD

Cheia pentru a înțelege ce oferă SVD este vizualizarea coloanelor r din U, Σși V ca reprezentând concepte care sunt ascunse în matricea originală M. In Exemplul 11.8, aceste concepte sunt clare; una este „science fiction” și cealaltă este „romantism”. Să ne gândim la rândurile lui M ca la oameni și la coloanele lui M ca filme. Apoi matricea U conectează oamenii la concepte. De exemplu, persoanei Joe, care corespunde rândului 1 din M din Fig. 11.6, îi place doar conceptul operă științifico-fantastică. Valoarea 0,14 din primul rând și prima coloană din U este mai mică decât unele dintre celelalte intrări din acea coloană, pentru că în timp ce Joe vizionează numai science fiction, el nu apreciază acele filme. A doua coloană a primul rând din U este 0, deoarece Joe nu evaluează deloc filmele romantice. Matricea V corelează filmele cu conceptele. Cei 0,58 din fiecare dintre primele trei coloanele din primul rând al lui V T indică faptul că primele trei filme - The Matrix, Alien și Star Wars - fiecare face parte din genul science-fiction, în timp ce 0 este în ultimele două coloane din primul rând spun că aceste filme nu participă conceptul romantism deloc. La fel, al doilea rând al lui V T ne spune că

---

filmele Casablanca și Titanic sunt exclusiv romantism.
În cele din urmă, matricea Σdă puterea fiecăruia dintre concepte. În a noastră de exemplu, puterea conceptului de science-fiction este 12,4, în timp ce puterea al conceptului de romantism este 9.5. Intuitiv, conceptul science-fiction este mai puternic deoarece datele oferă mai multe informații despre filmele din acel gen și oamenii cărora le plac.
În general, conceptele nu vor fi delimitate atât de clar. Vor fi mai puțini 0 în U și V, deși Σeste întotdeauna o matrice diagonală și va avea întotdeauna 0 este în afara diagonalei. Entitățile reprezentate de rândurile și coloanele lui M (similar cu oamenii și filmele din exemplul nostru) vor participa la mai multe concepte în diferite grade. De fapt, descompunerea exemplului 11.8 a fost mai ales simplu, deoarece rangul matricei M a fost egal cu cel dorit

numărul de coloane din U, Σ și V. Prin urmare, am putut obține o precizie
descompunerea lui M cu doar două coloane pentru fiecare dintre cele trei matrice U, Σ,
și V; produsul $U\Sigma V^T$, dacă ar fi realizat cu o precizie infinită, ar fi exact
M. În practică, viața nu este atât de simplă. Când rangul lui M este mai mare decât
numărul de coloane pe care le dorim pentru matricile U, Σ și V, descompunerea este
nu exact. Trebuie să eliminăm din descompunerea exactă acele coloane de
U și V care corespund celor mai mici valori singulare, pentru a obține cel mai bun
apropiere. Următorul exemplu este o ușoară modificare a Exemplului 11.8
care va ilustra punctul.

1 1 1 0 0
3 3 3 0 0
4 4 4 0 0
5 5 5 0 0
0 2 0 4 4
0 0 0 5 5
0 1 0 2 2
Joe
Jim
Ioan
Jack
Jill
Jenny
Jane
Casablanca
Matrice
Titanic
Străin
Razboiul Stelelor

Figura 11.8: Noua matrice M′, cu evaluări pentru Alien de către doi evaluatori suplimentari

Exemplul 11.9: Figura 11.8 este aproape la fel ca Fig. 11.6, dar Jill și Jane
a apreciat Alien, deși niciunui dintre ei nu i-a plăcut foarte mult. Rangul matricei în
Fig. 11.8 este 3; de exemplu primul, al șaselea și al șaptelea rând sunt independente,
dar puteți verifica dacă nu există patru rânduri independente. Figura 11.9 prezintă
descompunerea matricei din Fig. 11.8.

Am folosit trei coloane pentru U, Σ și V, deoarece acestea descompun a
matrice de rangul trei. Coloanele U și V corespund în continuare conceptelor.
Primul este încă „science fiction”, iar al doilea este „romantism”. Este mai greu să

□
□
□
□
□
□
□
□
□
□

1 1 1 0 0
3 3 3 0 0
4 4 4 0 0
5 5 5 0 0
0 2 0 4 4
0 0 0 5 5
0 1 0 2 2

□
□
□
□
□
□

□
□
□
□
=
M′
□
□
□
□
□
□
□
□
□
□
.13
.02 −.01
.41
.07 −.03
.55
.09 −.04
.68
.11 −.05
.15 −.59
.65
.07 −.73 −.67
.07 −.29
.32
□
□
□
□
□
□
□
□
□
□
□
□
12.4
0
0
0
9.5
0
0
0
1.3
□
□
□
□
.56
.59 .56
.09
.09
.12 −.02 .12 −.69 −.69
.40 −.80 .40

.09
.09
□
□
U
Σ
V T
Figura 11.9: SVD pentru matricea M ′ din Fig. 11.8
explica conceptul celei de-a treia coloane, dar nu contează atât de mult, deoarece greutatea sa, dată de a treia intrare diferită de zero în Σ, este foarte mică în comparație cu greutățile primelor două concepte. ✷

În secțiunea următoare, ne gândim să eliminăm unele dintre cele mai puțin importante concepte. De exemplu, am putea dori să eliminăm cel de-al treilea concept din Exemplu 11.9, deoarece într-adevăr nu ne spune multe și faptul că este asociat valoarea singulară este atât de mică confirmă lipsa de importanță a acesteia.

11.3.3 Reducerea dimensiunii utilizând SVD

Să presupunem că vrem să reprezentăm o matrice foarte mare M prin componentele sale SVD U, Σ și V, dar aceste matrice sunt, de asemenea, prea mari pentru a fi stocate convenabil. Cel mai bun modalitatea de a reduce dimensionalitatea celor trei matrice este de a seta cea mai mică dintre valorile singulare la zero. Dacă stabilim cele mai mici valori la 0, atunci putem elimina și coloanele s corespunzătoare ale lui U și V.

Exemplul 11.10: Descompunerea exemplului 11.9 are trei valori singulare ues. Să presupunem că dorim să reducem numărul de dimensiuni la două. Atunci noi setați cea mai mică dintre valorile singulare, care este 1,3, la zero. Efectul asupra expresia din Fig. 11.9 este că a treia coloană a lui U și al treilea rând al lui V T sunt

---

Pagina 467

înmulțit doar cu 0 atunci când efectuăm înmulțirea, deci acest rând și acesta coloana poate să nu fie acolo. Adică aproximarea la M ′ obținută folosind doar cele două mari valori la singular este cea prezentată în Fig. 11.10.

□
□
□
□
□
□
□
□
□
□
.13
.02
.41
.07
.55
.09
.68
.11
.15 −.59
.07 −.73
.07 −.29
□
□
□
□
□
□
□
□

□
□
[12.4
0
0
9.5] [
.56
.59 .56
.09
.09
.12 −.02 .12 −.69 −.69]
=
□
□
□
□
□
□
□
□
□
□
0,93 0,95 0,93 0,014 0,014
2,93 2,99 2,93 .000 .000
3,92 4,01 3,92 .026 .026
4,84 4,96 4,84 .040 .040
0,37 1,21 0,37 4,04 4,04
0,35 0,65 0,35 4,87 4,87
0,16 0,57 0,16 1,98 1,98
□
□
□
□
□
□
□
□
□
□

Figura 11.10: Eliminarea celei mai mici valori singulare din descompunerea
Fig. 11.7

Matricea rezultată este destul de apropiată de matricea M ′ din Fig. 11.8. În mod ideal, întreaga diferență este rezultatul faptului că ultima valoare singulară va fi 0. Cu toate acestea, în acest exemplu simplu, o mare parte din diferență se datorează erorii de rotunjire cauzate prin faptul că descompunerea lui M ′ a fost corectă doar la două semnificative cifre. ✷

11.3.4 De ce funcționează reducerea la zero a valorilor singulare scăzute

Alegerea celor mai mici valori la singular pentru a scădea atunci când reducem numărul de dimensiunile pot fi afișate pentru a minimiza eroarea rădăcină-medie-pătrat între matricea originală M și aproximarea acesteia. Deoarece numărul de intrări este fix, iar rădăcina pătrată este o operație monotonă, putem simplifica și compara normele Frobenius ale matricilor implicate. Reamintim că norma Frobenius dintr-o matrice M, notată M, este rădăcina pătrată a sumei pătratelor lui elementele lui M. Rețineți că dacă M este diferența dintre o matrice și aproximarea sa, atunci M este proporțională cu RMSE (rădăcină-medie-pătrat eroare) între matrici.

Pentru a explica de ce alegerea celor mai mici valori singular pentru a seta la 0 minimizează RMSE sau norma Frobenius a diferenței dintre M și aproximația sa zie, să începem cu o mică algebră matricială. Să presupunem că M este produsul trei matrici M = P QR. Fie m_{ij} , p_{ij} , q_{ij} și r_{ij} elementele din rândul i și coloana j din M, P, Q și, respectiv, R. Apoi definiția matricei

Câte valori unice ar trebui să păstrăm?

O regulă utilă este de a păstra suficiente valori singulare pentru a compune 90% din energia din Σ. Adică suma pătratelor celor reținuți valorile singulare ar trebui să fie cel puțin 90% din suma pătratelor tuturor valori singulare. În exemplul 11.10, energia totală este $(12,4)^2 + (9,5)^2 + (1,3)^2 = 245,70$, în timp ce energia reținută este $(12,4)^2 + (9,5)^2 = 244,01$. Astfel, am reținut peste 99% din energie. Cu toate acestea, am fost noi eliminați a doua valoare singulară, 9,5, energia reținută ar fi numai $(12,4)^2 / 240,70$ sau aproximativ 63%.

multiplicarea ne spune

$$m_{ij} = \sum_k \sum_\ell p_{ik} q_{k\ell} r_{\ell j}$$

Apoi

$$M^2 = \sum_i \sum_j (m_{ij})^2 = \sum_i \sum_j \left(\sum_k \sum_\ell p_{ik} q_{k\ell} r_{\ell j}\right)^2 \quad (11.1)$$

Când pătrăm o sumă de termeni, așa cum facem în partea dreaptă a ecuației 11.1, noi creați efectiv două copii ale sumei (cu indici de însumare diferiți) și înmulțiți fiecare termen din prima sumă cu fiecare termen din a doua sumă. Acea este,

$$\left(\sum_k \sum_\ell p_{ik} q_{k\ell} r_{\ell j}\right)^2 = \sum_k \sum_\ell \sum_m \sum_n p_{ik} q_{k\ell} r_{\ell j} p_{in} q_{nm} r_{mj}$$

putem rescrie astfel ecuația 11.1 ca.

$$M^2 = \sum_i \sum_j \sum_k \sum_\ell \sum_n \sum_m p_{ik} q_{k\ell} r_{\ell j} p_{in} q_{nm} r_{mj} \quad (11.2)$$

Acum, să examinăm cazul în care P, Q și R sunt într-adevăr SVD-ul lui M. Adică, P este o matrice ortonormală a coloanei, Q este o matrice diagonală și R este transpunerea unei coloane-matrice ortonormală. Adică R este rând-ortonormal; rândurile sale sunt vectori unitari și produsul punct al oricăror două rânduri diferite este 0. To începe, deoarece Q este o matrice diagonală, $q_{k\ell}$ și q_{nm} vor fi zero cu excepția cazului în care $k = \ell$ și $n = m$. Putem astfel să renunțăm la sumele pentru ℓ și m în ecuația 11.2 și set $k = \ell$ și $n = m$. Adică, ecuația 11.2 devine

$$M^2 = \sum_i \sum_j \sum_k \sum_n p_{ik} q_{kk} r_{kj} p_{in} q_{nn} r_{nj} \quad (11.3)$$

Apoi, reordonați suma, deci i este suma cea mai interioară. Ecuația 11.3 are numai doi factori p_{ik} și p_{in} care implică i; toți ceilalți factori sunt constante până acum în ceea ce privește însumarea peste i. Deoarece P este ortonormal pe coloană, știm

că $\sum_i p_{ik} p_{in}$ este 1 dacă $k = n$ și 0 în caz contrar. Adică, în ecuația 11.3 putem set $k = n$, picătură factorii p_{IK} și p_{in}, și să elimine sumele de peste i și n, cedând

$$M^2 = \sum_j \sum_k q_{kk} r_{kj} q_{kk} r_{kj} \quad (11.4)$$

Deoarece R este rând-ortonormal, $\sum_j r_{kj} r_{kj}$ este 1. Astfel, putem elimina termenii r_{kj} și suma peste j, lăsând o formulă foarte simplă pentru Frobenius normă:

$$M^2 = \sum_k (q_{kk})^2 \quad (11,5)$$

În continuare, să aplicăm această formulă unui M matrice a cărui SVD este $M = U\Sigma V^T$. Fie elementul diagonal ith al lui Σ să fie σ_i și să presupunem că păstrăm primul n din elementele r diagonale ale lui Σ, setând restul la 0. Fie Σ' rezultatul matrice diagonală. Fie $M' = U\Sigma' V^T$ aproximarea rezultată la M. Apoi

$M - M' = U(\Sigma - \Sigma')V^T$ este matricea care dă erorile care rezultă din apropiere.
Dacă aplicăm ecuația 11.5 matricei $M - M'$, vedem că $M - M'^2$ este egală cu suma pătratelor elementelor diagonale ale lui $\Sigma - \Sigma'$. Dar $\Sigma - \Sigma'$ are 0 pentru primele n elemente diagonale și σ_i pentru ith elementul diagonal, unde $n < i \leq r$. Adică, $M - M'^2$ este suma pătratelor elementelor din Σ care au fost setate la 0. Pentru a minimiza $M - M'^2$, alegeți acele elemente pentru a fi cel mai mic din Σ. Procedând astfel, se obține cea mai mică valoare posibilă a lui $M - M'^2$ sub constrângerea pe care o păstrăm n a elementelor diagonale și, prin urmare, aceasta minimizează RMSE sub aceeași constrângere.

11.3.5 Interogarea utilizând concepte

În această secțiune vom analiza modul în care SVD ne poate ajuta să răspundem la anumite întrebări eficient, cu o precizie bună. Să presupunem, de exemplu, că avem a compus datele noastre originale de evaluare a filmelor (datele de rang 2 din Fig. 11.6) în Forma SVD din Fig. 11.7. Quincy nu este unul dintre persoanele reprezentate de matrice originală, dar vrea să folosească sistemul pentru a ști ce filme ar face ca. El a văzut un singur film, The Matrix, și l-a evaluat cu 4. Astfel, putem reprezintă Quincy prin vectorul q = [4, 0, 0, 0, 0], ca și când acesta ar fi unul dintre rânduri a matricei originale.
Dacă am folosi o abordare de filtrare colaborativă, am încerca să comparăm Quincy cu ceilalți utilizatori reprezentați în matricea originală M. În schimb, îl putem mapa pe Quincy în „spațiu conceptual" înmulțindu-l cu matricea V a descompunerii. Găsim $qV = [2.32, 0]$. [3] Adică Quincy este mare în interesul science-fiction și deloc interesat de romantism.
Acum avem o reprezentare a lui Quincy în spațiul conceptual, derivat din, dar diferită de reprezentarea sa în „spațiul filmului" original. Un lucru util ceea ce putem face este să mapăm reprezentarea sa înapoi în spațiul filmului prin multiplicare

[3] Rețineți că Fig. 11.7 arată V^T, în timp ce această multiplicare necesită V.

[2,32, 0] de V^T. Acest produs este [1,35, 1,35, 1,35, 0, 0]. Acesta sugerează că Quincy mi-ar plăcea Alien și Star Wars, dar nu Casablanca sau Titanic.
Un alt tip de interogare pe care îl putem efectua în spațiul conceptual este acela de a găsi utilizatori similari către Quincy. Putem folosi V pentru a mapa toți utilizatorii în spațiul conceptual. De exemplu, Joe mapează la [1.74, 0], iar Jill mapează la [0, 5.68]. Observați că în acest simplu de exemplu, toți utilizatorii sunt fie fani 100% science-fiction, fie fani romantici 100%, deci fiecare vector are un zero într-o componentă. În realitate, oamenii sunt mai complexi, și vor avea niveluri de interes diferite, dar diferite de zero, pentru diverse concepte.
În general, putem măsura similitudinea utilizatorilor după distanța lor de cosinus în concept spațiu.
Exemplul 11.11: Pentru cazul introdus mai sus, rețineți că vectorii conceptului pentru Quincy și Joe, care sunt [2.32, 0] și [1.74, 0], respectiv, nu sunt la fel, dar au exact aceeași direcție. Adică distanța lor de cosinus este 0. Pe de altă parte, vectorii pentru Quincy și Jill, care sunt [2.32, 0] și [0, 5.68], respectiv, au un produs punct 0 și, prin urmare, unghiul lor este 90 grade. Adică, distanța lor la cosinus este 1, maximul posibil. ✷

11.3.6 Calcularea SVD-ului unei matrice

SVD-ul unei matrice M este puternic conectat la valorile proprii ale simetrei-matrici ric $M^T M$ și MM^T. Această relație ne permite să obținem SVD a lui M din perechile proprii ale ultimelor două matrice. Pentru a începe explicația, începeți cu $M = U\Sigma V^T$, expresia pentru SVD a lui M. Apoi

$$M^T = (U\Sigma V^T)^T = (V^T)^T \Sigma^T U^T = V \Sigma^T U^T$$

Deoarece Σ este o matrice diagonală, transpunerea acesteia nu are niciun efect. Astfel, $M^T = V \Sigma U^T$.
Acum, $M^T M = V \Sigma U^T U \Sigma V^T$. Amintiți-vă că U este o matrice ortonormală, deci $U^T U$ este matricea de identitate a mărimii corespunzătoare. Acesta este,

$$M^T M = V \Sigma^2 V^T$$

Înmulțiți ambele părți ale acestei ecuații din dreapta cu V pentru a obține

$$M^T MV = V \Sigma^2 V^T V$$

Deoarece V este, de asemenea, o matrice ortonormală, știm că $V^T V$ este identitatea. Prin urmare

$$M^T MV = V \Sigma^2 \tag{11,6}$$

Deoarece Σ este o matrice diagonală, Σ^2 este, de asemenea, o matrice diagonală a cărei intrare în ith rândul și coloana este pătratul intrării în aceeași poziție a lui Σ. Acum, Ecuația (11.6) ar trebui să fie familiară. Se spune că V este matricea vectorilor proprii lui $M^T M$ și Σ^2 este matricea diagonală ale cărei intrări sunt corespunzătoare valori proprii.

Astfel, același algoritm care calculează perechile proprii pentru $M^T M$ ne oferă matricea V pentru SVD-ul lui M în sine. De asemenea, ne oferă valorile singulare pentru acest SVD; pur și simplu luați rădăcinile pătrate ale valorilor proprii pentru $M^T M$. Doar U rămâne de calculat, dar poate fi găsit la fel ca noi găsit V. Începe cu

$MM^T = U\Sigma V^T (U\Sigma V^T)^T = U\Sigma V^T V \Sigma U^T = U\Sigma^2 U^T$

Apoi, printr-o serie de manipulări analoage celor de mai sus, aflăm că

$MM^T U = U\Sigma^2$

Adică, U este matricea vectorilor proprii de MM^T.

Un mic detaliu trebuie explicat cu privire la U și V. Fiecare dintre acestea matricile au r coloane, în timp ce $M^T M$ este o matrice $n \times n$ și MM^T este un $m \times m$ matrice. Atât n cât și m sunt cel puțin la fel de mari ca r. Astfel, $M^T M$ și MM^T ar trebui au o pereche proprie n - r și respectiv m - r, respectiv aceste perechi au nu apare în U, V și Σ.Deoarece rangul lui M este r, toate celelalte valori proprii va fi 0, iar acestea nu sunt utile.

11.3.7 Exerciții pentru secțiunea 11.3

Exercițiul 11.3.1: În Fig. 11.11 este o matrice M. Are rangul 2, după cum puteți vedea observând că prima coloană plus a treia coloană minus de două ori a doua coloana este egală cu 0.

□
□
□
□
□
□

1 2 3
3 4 5
5 4 3
0 2 4
1 3 5

□
□
□
□
□
□

Figura 11.11: Matricea M pentru exercițiul 11.3.1

(a) Compute matricele $M^T M$ și MM^T.

! (b) Găsiți valorile proprii pentru matricile dvs. din partea (a).

(c) Găsiți vectorii proprii pentru matricele părții (a).

(d) Găsiți SVD pentru matricea originală M din părțile (b) și (c). Notă că există doar două valori proprii diferite de zero, deci matricea Σar trebui să aibă doar două valori singulare, în timp ce U și V au doar două coloane.

(e) Setați valoarea dvs. singulară mai mică la 0 și calculați unidimensionalul aproximare la matricea M din Fig. 11.11.

(f) Cât din energia valorilor originale originale este reținută de aproximare unidimensională?

Exercițiul 11.3.2: Utilizați SVD din Fig. 11.7. Să presupunem că Leslie acordă ratingul 3 pentru Alien și nota 4 pentru Titanic, oferindu-ne o reprezentare a lui Leslie în „film spațiu ”din [0, 3, 0, 0, 4]. Găsiți reprezentarea lui Leslie în spațiul conceptual. Ce prezice această reprezentare despre cât de bine și-ar dori Leslie celuilalt filme care apar în datele noastre de exemplu?

! Exercițiul 11.3.3: Demonstrați că rangul matricei din Fig. 11.8 este 3.

! Exercițiul 11.3.4: Secțiunea 11.3.5 a arătat cum să ghicească filmele unei persoane

mi-ar plăcea cel mai mult. Cum ați folosi o tehnică similară pentru a ghici oamenii asta ar dori cel mai mult un film dat, dacă tot ceea ce ai avea ar fi fost ratingurile filmului respectiv de câțiva oameni?

11.4 Descompunerea CUR

Există o problemă cu SVD care nu apare în exemplul de rulare din secțiunea 11.3. În aplicațiile cu date mari, este normal ca matricea M să fie descompus a fi foarte rar; adică majoritatea intrărilor sunt 0. De exemplu, a matrice reprezentând multe documente (ca rânduri) și cuvintele pe care le conțin (ca coloane) vor fi rare, deoarece majoritatea cuvintelor nu sunt prezente în majoritatea documentelor. În mod similar, o matrice de clienți și produse va fi rară, deoarece majoritatea oamenii nu cumpără majoritatea produselor.

Nu ne putem ocupa de matrici dense care au milioane sau miliarde de rânduri și / sau coloane. Cu toate acestea, cu SVD, chiar dacă M este rar, U și V vor fi dens. [4] Deoarece Σeste diagonală, acesta va fi rar, dar Σeste de obicei mult mai mic decât U și V, deci limita sa nu ajută.

În această secțiune, vom lua în considerare o altă abordare a descompunerii, numită CUR-descompunere. Meritul acestei abordări constă în faptul că dacă M este rare, apoi cele două matrice mari (numite C și R pentru „coloane” și „rânduri”) analogi cu U și V în SVD sunt, de asemenea, rare. Doar matricea din mijloc (analog cu Σîn SVD) este dens, dar această matrice este mică, astfel încât densitatea o face nu doare prea mult.

Spre deosebire de SVD, care oferă o descompunere exactă atâta timp cât parametrul r este considerat a fi cel puțin la fel de mare ca rangul matricei M, CUR-descompunere este o aproximare oricât de mare am face r. Există o teorie care garantează convergența către M pe măsură ce r devine mai mare, dar de obicei trebuie să faceți r atât de mare de obținut, spuneți în limita a 1%, încât metoda devine impracticabilă. Niciodată mai puțin, o descompunere cu o valoare relativ mică de r are o bună probabilitate de a fi o descompunere utilă și precisă.

[4] În Fig. 11.7, se întâmplă ca U și V să aibă un număr semnificativ de 0. Cu toate acestea, asta este un artefact al naturii foarte regulate a matricei noastre de exemplu M și nu este cazul în general.

De ce funcționează pseudoinversa

În general, să presupunem că o matrice M este egală cu un produs al matricilor XZY. Dacă există toate inversele, atunci ne spune regula pentru inversarea unui produs $M^{-1} = Y^{-1} Z^{-1} X^{-1}$. Deoarece în cazul în care ne interesează, XZY este un SVD, știm că X este ortonormal pe coloană și Y este ortonormal pe rând. În oricare dintre aceste cazuri, inversul și transpunerea sunt aceleași. Acea este, XX^T este o matrice identitate de dimensiune corespunzătoare, și așa este YY^T. Astfel, $M^{-1} = Y^T Z^{-1} X^T$.

Știm, de asemenea, că Z este o matrice diagonală. Dacă nu există 0 de-a lungul diagonală, atunci Z^{-1} se formează din Z luând inversul numeric al lui fiecare element diagonal. Este doar atunci când există 0 de-a lungul diagonalei din Z că nu putem găsi un element pentru aceeași poziție în invers astfel încât să putem obține o matrice de identitate atunci când înmulțim Z cu inversul său. De aceea recurgem la un „pseudoinvers”, acceptând faptul că produsul ZZ^+ nu va fi o matrice de identitate, ci mai degrabă un matrice diagonală unde intrarea diagonală ith este 1 dacă elementul ith al lui Z este diferit de zero și 0 dacă al zecelea element al lui Z este 0.

11.4.1 Definiția CUR

Fie M o matrice de m rânduri și n coloane. Alegeți un număr țintă de „concepte” r pentru a fi utilizat în descompunere. O descompunere CUR a lui M este o întâmplare set ales de r coloane ale lui M, care formează matricea $m \times r$ C și a aleatoriu set ales de r rânduri de M, care formează matricea $r \times n$ R. Există și un $r \times r$ matricea U care este construită din C și R după cum urmează:

1. Fie W matricea $r \times r$ care este intersecția coloanelor alese din C și rândurile alese ale lui R. Adică elementul din rândul i și coloana j din W este elementul lui M a cărui coloană este a j-a coloană a lui C și al cărui rând este al doilea rând al lui R.
2. Calculați SVD-ul lui W; spun $W = X\Sigma Y^T$.
3. Calculați Σ^+, pseudoinversul Moore-Penrose al matricei diagonale Σ.Adică, dacă al doilea element diagonal al lui Σeste $\sigma = 0$, atunci înlocuiți-l cu $1 / \sigma$.Dar dacă elementul ith este 0, lăsați-l ca 0.
4. Fie $U = Y (\Sigma^+)^2 X^T$.

Vom amâna la Secțiunea 11.4.3 un exemplu în care ilustrăm întregul CUR proces, inclusiv chestiunea importantă a modului în care ar trebui matricile C și R

să fie aleasă pentru ca aproximarea la M să aibă o mică valoare așteptată.

Pagina 474

11.4.2 Alegerea corectă a rândurilor și coloanelor

Reamintim că alegerea rândurilor și coloanelor este aleatorie. Cu toate acestea, această alegere trebuie să fie părtinitoare, astfel încât rândurile și coloanele mai importante să aibă o mai bună sansa de a fi ales. Măsura importanței pe care trebuie să o folosim este pătratul din norma Frobenius, adică suma pătratelor elementelor din rând sau coloană. Fie $f = \Sigma_{i,j} m_{ij}^2$, pătratul normei Frobenius a lui M.

Apoi, de fiecare dată când selectăm un rând, probabilitatea p_i cu care selectăm rândul i este $\Sigma_j m_{ij}^2 / f$. De fiecare dată când selectăm o coloană, probabilitatea q_j cu care noi selectați coloana j este $\Sigma_i m_{ij}^2 / f$.

1 1 1 0 0
3 3 3 0 0
4 4 4 0 0
5 5 5 0 0
0 0 0 4 4
0 0 0 5 5
0 0 0 2 2
Joe
Jim
Ioan
Jack
Jill
Jenny
Jane
Casablanca
Matrice
Titanic
Străin
Razboiul Stelelor

Figura 11.12: Matricea M, repetată din Fig. 11.6

Exemplul 11.12: Să reconsiderăm matricea M din Fig. 11.6, pe care noi repetați aici ca Fig. 11.12. Suma pătratelor elementelor lui M este 243. Cele trei coloane pentru filmele science-fiction The Matrix, Alien și Star Războaiele au fiecare o normă Frobenius pătrată de $1^2 + 3^2 + 4^2 + 5^2 = 51$, deci probabilitățile sunt fiecare 51/243 = .210. Cele două coloane rămase au câte o norma Frobenius pătrată de $4^2 + 5^2 + 2^2 = 45$ și, prin urmare, probabilitățile lor sunt fiecare 45/243 = .185.

Cele șapte rânduri de M au pătrat normele Frobenius de 3, 27, 48, 75, 32, 50, respectiv 8. Astfel, probabilitățile lor respective sunt .012, .111, .198, .309, .132, .206 și .033. ✷

Acum, să selectăm r coloane pentru matricea C. Pentru fiecare coloană, alegem aleatoriu din coloanele lui M. Cu toate acestea, selecția nu este uniformă probabilitate; mai degrabă, coloana jth este selectată cu probabilitatea q_j. Reamintim că probabilitatea este suma pătratelor elementelor din acea coloană împărțită la suma pătratelor tuturor elementelor matricei. Fiecare coloană a lui C este alese independent de coloanele lui M, deci există unele șanse ca a coloana va fi selectată de mai multe ori. Vom discuta despre cum să rezolvăm acest lucru situație după explicarea elementelor de bază ale descompunerii CUR.

Pagina 475

După ce am selectat fiecare dintre coloanele lui M, scalăm fiecare coloană împărțind elementele sale după rădăcina pătrată a numărului de ori așteptat de această coloană ar fi ales. Adică împărțim elementele coloanei a X-a a lui M, dacă este este selectat, de $\sqrt{rq_j}$. Coloana scalată a lui M devine o coloană a lui C. Rândurile lui M sunt selectate pentru R în mod analog. Pentru fiecare rând de R noi selectați din rândurile lui M, alegând rândul i cu probabilitatea p_i. Reamintim p_i este suma pătratelor elementelor celui de-al doilea rând împărțit la suma lui pătrate ale tuturor elementelor lui M. Scalăm apoi fiecare rând ales împărțind de $\sqrt{rp_i}$ dacă a fost ales al doilea rând al lui M.

Exemplul 11.13: Fie r = 2 pentru descompunerea noastră CUR. Să presupunem că a noastră selectarea aleatorie a coloanelor din matricea M din Fig. 11.12 este mai întâi Alien (the a doua coloană) și apoi Casablanca (a patra coloană). Coloana pentru Alien este [1, 3, 4, 5, 0, 0, 0] ^{T} și trebuie să scalăm această coloană împărțind la $\sqrt{rq_2}$. Reamintim din Exemplul 11.12 că probabilitatea asociată coloanei Alien este .210, deci împărțirea este cu $\sqrt{2}\times$ 0.210 = 0.648. Cu două zecimale, coloană scalat pentru Alien este [1,54, 4,63, 6,17, 7,72, 0, 0, 0] ^{T} . Această coloană devine prima coloană a lui C.

A doua coloană a lui C este construită luând coloana lui M pentru Casablanca, care este [0, 0, 0, 0, 4, 5, 2] ^{T} și o împarte cu $\sqrt{rp_4} = \sqrt{2}\times$ 0,185 = 0,608. Astfel, a doua coloană a lui C este [0, 0, 0, 0, 6,58, 8,22, 3,29] ^{T} la două decizi mal locuri.

Acum, să alegem rândurile pentru R. Cele mai probabile rânduri care vor fi alese sunt cele pentru Jenny și Jack, deci să presupunem că aceste rânduri sunt într-adevăr alese, Jenny primul. Rândurile nescalate pentru R sunt astfel

[0 0 0 5 5
5 5 5 0 0]

Pentru a scala rândul pentru Jenny, observăm că probabilitatea sa asociată este 0,206, deci împărțim la $\sqrt{2}\times$ 0,206 = 0,642. Pentru a scala rândul pentru Jack, al cărui asociat probabilitatea este 0,309, împărțim la $\sqrt{2}\times$ 0,309 = 0,786. Astfel, matricea R este

[0
0
0
7.79 7.79
6,36 6,36 6,36
0
0]

✷

11.4.3 Construirea matricei de mijloc

În cele din urmă, trebuie să construim matricea U care leagă C și R în decom-poziție. Reamintim că U este o matrice r × r. Începem construcția U cu o altă matrice de aceeași dimensiune, pe care o numim W. Intrarea din rândul i și coloana j din W este intrarea lui M al cărui rând este cel din care am selectat al doilea rând al lui R și a cărui coloană este cea din care am selectat jth coloana lui C.

Pagina 476

Exemplul 11.14: Să urmărim selecțiile de rânduri și coloane realizate în Exemplul 11.13. Noi pretindem

W = [
0 5
5 0]

Primul rând al lui W corespunde primului rând al lui R, care este rândul pentru Jenny în matricea M din Fig. 11.12. 0 din prima coloană este acolo pentru că este intrarea în rândul M pentru Jenny și coloana pentru Alien; amintim că prima coloană C a fost construită din coloana M pentru Alien. Cele 5 din a doua coloană reflectă 5 din rândul lui M pentru Jenny și coloana pentru Casablanca; aceasta din urmă este coloana lui M din care a derivat a doua coloană a lui C. În mod similar, al doilea rând de W este intrările din rând pentru Jack și coloane pentru Alien și, respectiv, pentru Casablanca. ✷

Matricea U este construită din W de pseudoin Moore-Penrose versetul descris în secțiunea 11.4.1. Consistă în luarea SVD a lui W, să zicem $W = X\Sigma Y^T$ și înlocuirea tuturor elementelor nenule din matricea Σa valorile larilor prin inversele lor numerice, pentru a obține pseudoinversa Σ_+ . Apoi $U = Y (\Sigma_+)^2 X^T$.

Exemplul 11.15: Să construim U din matricea W pe care am construit-o în exemplul 11.14. În primul rând, iată SVD pentru W:

W = [
0 5
5 0] = [
0 1
1 0] [
5 0
0 5] [
1 0
0 1]

Cu alte cuvinte , cele trei matrice din dreapta sunt X, Σ,și Y T , respectiv. The
matricea Σnu are zerouri de-a lungul diagonalei, deci fiecare element este înlocuit cu al său
invers invers numeric pentru a obține pseudoinversul său Moore-Penrose:
Σ_+ = [
1/5
0
0
1/5]
Acum X și Y sunt simetrice, deci sunt propriile lor transpuneri. Prin urmare,
U = Y $(\Sigma_+)^2$ X^T = [
1 0
0 1] [
1/5
0
0
1/5]
2
[0 1
1 0] = [
0
1/25
1/25
0
]
✷

11.4.4 Descompunerea completă a CUR

Acum avem o metodă de selectare aleatorie a celor trei matrice componente C,
U și R. Produsul lor va aproxima matricea originală M. Ca și noi
menționată la începutul discuției, aproximarea este doar pentru
garantat masculin să fie aproape atunci când un număr foarte mare de rânduri și coloane
sunt selectate. Cu toate acestea, intuiția este aceea prin selectarea rândurilor și coloanelor care
tind să aibă o „importanță” ridicată (adică o normă Frobenius ridicată), pe care o extragem

---

CUR =
□
□
□
□
□
□
□
□
□
□
1,54
0
4.63
0
6.17
0
7,72
0
0
9.30
0 11,63
0
4.65
□
□
□

□
□
□
□
□
□
□
[
0
1/25
1/25
0
] [
0
0
0 11.01 11.01
8,99 8,99 8,99
0
0]
=
□
□
□
□
□
□
□
□
□
□
0,55 0,55 0,55
0
0
1,67 1,67 1,67
0
0
2.22 2.22 2.22
0
0
2,78 2,78 2,78
0
0
0
0
0 4.10 4.10
0
0
0 5.12 5.12
0
0
0 2,05 2,05
□
□
□
□
□
□
□
□
□
□

Figura 11.13: CUR-descompunerea matricei din Fig. 11.12
cele mai semnificative părți ale matricei originale, chiar și cu un număr mic de
rânduri și coloane. De exemplu, să vedem cât de bine ne descurcăm cu alergarea
exemplu al acestei secțiuni.
Exemplul 11.16: Pentru exemplul nostru de rulare, descompunerea este prezentată în
Fig. 11.13. Deși există o diferență considerabilă între acest rezultat și
matricea originală M, mai ales în numerele science-fiction, valorile sunt în
proporțional cu originalele lor. Acest exemplu este mult prea mic, iar selecția
din numărul mic de rânduri și coloane a fost mai degrabă arbitrar decât aleatoriu, pentru
ne așteptăm la o convergență strânsă a descompunerii CUR la valorile exacte.
✷

11.4.5 Eliminarea rândurilor și coloanelor duplicate

Este foarte posibil ca un singur rând sau coloană să fie selectat de mai multe ori.
Nu există un mare rău în utilizarea aceluiași rând de două ori, deși rangul
matricile descompunerii vor fi mai mici decât numărul de rând și coloană
alegerile făcute. Cu toate acestea, este de asemenea posibil să combinați k rânduri de R care sunt fiecare
același rând al matricei M într-un singur rând de R, lăsând astfel R cu mai puțini
rânduri. La fel, k coloane de C care provin fiecare din aceeași coloană a lui M
poate fi combinat într-o singură coloană de C. Cu toate acestea, fie pentru rânduri, fie pentru coloane,
vectorul rămas ar trebui să aibă fiecare dintre elementele sale multiplicat cu $\sqrt{k}$.
Când îmbinăm câteva rânduri și / sau coloane, este posibil ca R să aibă mai puține
rândurile decât C au coloane sau invers. În consecință, W nu va fi un
matrice pătrată. Cu toate acestea, putem lua pseudoinversul său descompunându-l
în $W = X\Sigma Y^T$, unde Σ este acum o matrice diagonală cu câteva rânduri all-0 sau

---

Pagina 478

coloane, oricare ar avea mai mult. A lua pseudoinversa unei astfel de diagonale
matrice, tratăm fiecare element de pe diagonală ca de obicei (inversați elementele nenule
și lăsați 0 așa cum este), dar atunci trebuie să transpunem rezultatul.
Exemplul 11.17: Să presupunem
$\Sigma =$
□
2 0 0 0
0 0 0 0
0 0 3 0
□
□
Apoi
$\Sigma_+ =$
□
□
1/2 0
0
0
0
0
0
0 1/3
0
0
0
□
□
□
□
✷

11.4.6 Exerciții pentru secțiunea 11.4

Exercițiul 11.4.1: SVD pentru matrice
M = [
48
14
14 −48]

este
[48
14
14 −48] = [
3/5
4/5
4/5 −3/5] [
50 0
0 25] [
4/5 −3/5
3/5
4/5]
Găsiți pseudoinversa lui Moore-Penrose a lui M.
! Exercițiul 11.4.2: Găsiți descompunerea CUR a matricei din Fig. 11.12
când alegem două rânduri și coloane „aleatorii” după cum urmează:
(a) Coloanele pentru Matrix și Alien și rândurile pentru Jim și John.
(b) Coloanele pentru Alien și Star Wars și rândurile pentru Jack și Jill.
(c) Coloanele pentru Matrix și Titanic și rândurile pentru Joe și Jane.
! Exercițiul 11.4.3: Găsiți descompunerea CUR a matricei din Fig. 11.12 dacă
cele două rânduri „aleatorii” sunt atât Jack, cât și cele două coloane sunt Star Wars și
Casablanca.

11.5 Rezumatul capitolului 11

♦ Reducerea dimensionalității: Scopul reducerii dimensionalității este de a re-
plasați o matrice mare cu două sau mai multe alte matrice ale căror dimensiuni sunt mult
mai mic decât originalul, dar din care originalul poate fi de aproximativ
reconstituite matematic, de obicei prin luarea produsului lor.

Pagina 479

♦ Valori proprii și vectori proprii: o matrice poate avea mai mulți vectori proprii
astfel încât atunci când matricea înmulțește vectorul propriu, rezultatul este un con-
multiplu stant al vectorului propriu. Această constantă este valoarea proprie as-
ciat cu acest vector propriu. Împreună vectorul propriu și valoarea sa proprie
sunt numite perechi proprii.

♦ Găsirea perechilor proprii prin iterația puterii: Putem găsi principalele proprii-
vector (vector propriu cu cea mai mare valoare proprie) începând cu orice vec-
tor și înmulțind în mod repetat vectorul curent cu matricea pentru a obține un
vector nou. Când modificările vectorului devin mici, putem trata
rezultatul ca o apropiere apropiată de vectorul propriu principal. Prin mod-
dacă selectăm matricea, putem folosi aceeași iterație pentru a obține a doua
pereche proprie (cea cu cea de-a doua cea mai mare valoare proprie) și obțineți în mod similar fiecare
la rândul lor al perechilor proprii, în ordinea valorii descrescătoare a valorii proprii.

♦ Analiză componentă principală: Această tehnică pentru reducerea dimensionalității
vizualizarea datelor constă dintr-o colecție de puncte într-un multidimensional
spațiu ca matrice, cu rânduri corespunzătoare punctelor și coloanelor la
dimensiunile. Produsul acestei matrice și transpunerea sa are proprii
perechi și vectorul propriu principal poate fi văzut ca direcția în
spațiu de-a lungul căruia se aliniază cel mai bine punctele. Al doilea vector propriu reprezintă
trimite direcția în care sunt abaterile de la vectorul propriu principal
cel mai mare și așa mai departe.

♦ Reducerea dimensiunii prin PCA: Prin reprezentarea matricei de puncte
printr-un număr mic de vectori proprii, putem aproxima datele într-un
mod care minimizează eroarea rădăcină-medie-pătrat pentru numărul dat de
coloane din matricea reprezentativă.

♦ Descompunere cu valoare singulară: descompunerea cu valoare singulară a unei ma-
trix constă din trei matrice, U, Σ și V. Matricile U și V sunt
coloană-ortonormală, ceea ce înseamnă că, în calitate de vectori, coloanele sunt ortogo-
nal, iar lungimile lor sunt 1. Matricea Σ este o matrice diagonală și
valorile de-a lungul diagonalei sale se numesc valori singulare. Produsul lui U, Σ,
iar transpunerea lui V este egală cu matricea originală.

♦ Concepte: SVD este util atunci când există un număr mic de concepte care
conectați rândurile și coloanele matricei originale. De exemplu, dacă
matricea originală reprezintă evaluările acordate de spectatorii de filme (rânduri) către
filme (coloane), conceptele ar putea fi genurile filmelor. The

matricea U conectează rândurile la concepte, Σreprezintă punctele forte ale concepte, iar V conectează conceptele la coloane.

♦ Interogări folosind descompunerea valorii unice: Putem folosi descompunerea poziția de a relaționa rânduri noi sau ipotetice ale matricei originale cu concepte reprezentate de descompunere. Înmulțiți un rând cu matricea V a descompunerii pentru a obține un vector care indică măsura în care rândul se potrivește cu fiecare dintre concepte.

Pagina 480

♦ Utilizarea SVD pentru reducerea dimensiunii: într-un SVD complet pentru o matrix, U și V sunt de obicei la fel de mari ca originalul. Pentru a utiliza mai puține coloane pentru U și V, ștergeți coloanele corespunzătoare celui mai mic singular valori din U, V și Σ.Această alegere minimizează eroarea în reconstituire-introducând matricea originală din U, Σși V modificate.

♦ Descompunerea matricelor rare: chiar și în cazul obișnuit în care este dat matricea este rară, matricile construite de SVD sunt dense. CUR descompunerea urmărește descompunerea unei matrici rare în rare, mai mici matrici al căror produs se apropie de matricea originală.

♦ Descompunere CUR: Această metodă alege dintr-o anumită matrice rar un set de coloane C și un set de rânduri R, care joacă rolul lui U și V T în SVD; utilizatorul poate alege orice număr de rânduri și coloane. The alegerea rândurilor și coloanelor se face aleatoriu cu o distribuție care depinde de norma Frobenius sau de rădăcina pătrată a sumei pătrate ale elementelor. Între C și R este o matrice pătrată numită U care este construit printr-un pseudo-invers al intersecției celui ales rânduri și coloane.

11.6 Referințe pentru capitolul 11

Un text bine apreciat despre algebra matricială este [4].
Analiza componentelor principale a fost discutată pentru prima dată în urmă cu peste un secol, în [6].
SVD este de la [3]. Au existat multe aplicații ale acestei idei. Două demne de menționat sunt [1] care se ocupă cu analiza documentelor și [8] care se ocupă cu aplicații în biologie.
Descompunerea CUR este de la [2] și [5]. Descrierea noastră urmează mai târziu munca [7].

1. S. Deerwester, ST Dumais, GW Furnas, TK Landauer și R. Harshman, „Indexing by latent semantic analysis”, J. American Society for Știința informației 41: 6 (1990).
2. P. Drineas, R. Kannan și MW Mahoney, „Fast Monte-Carlo algoritmi pentru matrici III: Calcularea unei matrice aproximate comprimate descompunere, ”SIAM J. Computing 36: 1 (2006), pp. 184–206.
3. GH Golub și W. Kahan, „Calculul valorilor singulare și pseudo-inversul unei matrice, ”J. SIAM Seria B 2: 2 (1965), pp. 205–224.
4. GHGolub și CF Van Loan, Matrix Computations, JHU Press, 1996.
5. MW Mahoney, M. Maggioni și P. Drineas, Tensor-CUR decomposi-pentru date bazate pe tensori, SIGKDD, pp. 327-336, 2006.
6. K. Pearson, „Pe linii și planuri de cea mai apropiată potrivire cu sistemele de puncte din spațiu ”, Philosophical Magazine 2:11 (1901), pp. 559-572.

Pagina 481

7. J. Sun, Y. Xie, H. Zhang și C. Faloutsos, „Mai puțin este mai mult: compact descompunerea matricei pentru grafice mari rare, ”Proc. SIAM Intl. Conf. pe Data Mining, 2007.
8. ME Wall, A. Reichtsteiner și LM Rocha, „Singular value decom-analiza poziției și componentelor principale ”, în A Practical Approach to Analiza datelor Microarray (DP Berrar, W. Dubitzky și M. Granzow eds.), pp. 91–109, Kluwer, Norwell, MA, 2003.

Pagina 482

Capitolul 12
Mașină pe scară largă Învățare

Mulți algoritmi sunt astăzi clasificați ca „învățare automată”. Acești algoritmi
împărtășiți, împreună cu ceilalți algoritmi studiați în această carte, scopul extragerii
informații din date. Toți algoritmii pentru analiza datelor sunt concepuți pentru
produce un rezumat util al datelor, din care se iau decizii. Printre
multe exemple, produce analiza de elemente frecvente pe care am făcut-o în capitolul 6
informații precum regulile de asociere, care pot fi apoi utilizate pentru planificarea vânzărilor
strategie sau în multe alte scopuri.
Cu toate acestea, algoritmii numiți „învățare automată” nu doar o rezumă pe a noastră
date; sunt percepuți ca învățând un model sau clasificator din date, și astfel
descoperiți ceva despre datele care vor fi văzute în viitor. De exemplu,
algoritmii de grupare discutați în capitolul 7 produc clustere care nu numai că spun
ceva despre datele analizate (setul de instruire), dar ele ne permit
să clasificăm datele viitoare într-unul dintre clusterele care rezultă din clustering
algoritm. Astfel, pasionații de învățare automată vorbesc adesea despre clustering cu
neologismul „învățare nesupravegheată”; termenul nesupravegheat se referă la fapt
că datele de intrare nu indică algoritmului de clusterizare ce sunt clusterele
ar trebui să fie. În învățarea automată supravegheată, care face obiectul acestui capitol,
datele disponibile includ informații despre modul corect de clasificare cel puțin
unele dintre date. Datele clasificate deja se numesc setul de antrenament.
Acest capitol nu este destinat să fie o discuție completă despre învățarea automată.
Ne concentrăm pe un număr mic de idei și subliniem modul de abordare
seturi de date foarte mari. Deosebit de important este modul în care exploatăm paralelismul pentru a construi
modele de date. Considerăm abordarea clasică „perceptron” a învățării
un clasificator de date, unde se caută un hiperplan care separă două clase. Apoi,
ne uităm la tehnici mai moderne care implică mașini vector-suport. Similar
la perceptroni, aceste metode caută hiperplane care împart cel mai bine clasele,
astfel încât puțini, dacă există, membri ai setului de antrenament se află aproape de hiperplan. Noi
apoi luați în considerare tehnicile cele mai apropiate, în cazul în care datele sunt clasificate în funcție de

clasa (clasele) celor mai apropiați vecini într-un spațiu. Încheiem cu o discuție
de arbori de decizie, care sunt programe de ramificare pentru prezicerea clasei unui
exemplu.

12.1 Modelul de învățare automată

În această secțiune introducem cadrul pentru algoritmi de învățare automată și
dați definițiile de bază.

12.1.1 Seturi de antrenament

Datele la care se referă un algoritm de învățare automată (adesea prescurtată ML)
aplicat se numește set de antrenament. Un set de antrenament constă dintr-un set de perechi (x, y),
numite exemple de instruire, unde

- x este un vector de valori, adesea numit vector de caracteristică sau pur și simplu o intrare.
Fiecare valoare sau caracteristică poate fi categorică (valorile sunt luate dintr-un set
de valori discrete, cum ar fi {roșu, albastru, verde}) sau numerice (valorile sunt
numere întregi sau numere reale).
- y este eticheta clasei, sau pur și simplu rezultatul, valoarea de clasificare pentru x.

Obiectivul procesului ML este de a descoperi o funcție y = f (x) care
cel mai bun prezice valoarea lui y asociată cu fiecare valoare a lui x. Tipul de y este în
principiu arbitrar, dar există mai multe cazuri comune și importante.
1. y este un număr real. În acest caz, problema ML se numește regresie.
2. y este o valoare booleană: adevărat sau fals, mai frecvent scris ca +1 și −1,
respectiv. În această clasă problema este clasificarea binară.

3. y este un membru al unei mulțimi finite. Membrii acestui set pot fi gândiți ca „clase” și fiecare membru reprezintă o clasă. Problema este că clasificare multiclass.
4. y este membru al unui set potențial infinit, de exemplu, un arbore analizat pentru x, care este interpretat ca o propoziție.

12.1.2 Câteva exemple ilustrative

Exemplul 12.1: Reamintim Fig. 7.1, repetată ca Fig. 12.1, unde am reprezentat graficul înălțimea și greutatea câinilor din trei clase: Beagles, Chihuahuas și Dachshunds. Ne putem gândi la aceste date ca la un set de instruire, cu condiția ca datele să includă varietatea câinelui împreună cu fiecare pereche înălțime-greutate. Fiecare pereche (x, y) în setul de antrenament constă dintr-un vector caracteristic x al formei [înălțime, greutate]. The eticheta asociată y este varietatea câinelui. Un exemplu de pereche de seturi de antrenament ar fi ([5 inch, 2 pounds], Chihuahua).

Beagles
Greutate
Înălțime
Chihuahua
Teckelii
înălțime = 7
greutate = 3

Figura 12.1: Repetați figura 7.1, indicând înălțimile și greutățile anumitor câini

O modalitate adecvată de implementare a funcției de decizie f ar fi imaginația în două linii, prezentate în fig. 12.1. Linia orizontală reprezintă o înălțime de 7 inci și separă Beagles de Chihuahuas și Teckel. Verti-linia cal reprezintă o greutate de 3 kilograme și separă Chihuahuas de Beagles și Dachshunds. Algoritmul care implementează f este:

dacă (înălțime> 7) tipăriți Beagle
altfel dacă (greutate <3) tipărește Chihuahua
altfel tipărește Dachshund;

Reamintim că intenția inițială din Fig. 7.1 a fost de a grupa puncte fără știind ce varietate de câine au reprezentat. Adică eticheta asociată cu un anumit vector înălțime-greutate nu a fost disponibil. Aici, performăm învățare supravegheată cu aceleași date mărite de clasificări pentru date de instruire. ✷

Exemplul 12.2: Ca exemplu de învățare supravegheată, cele patru puncte (1, 2), (2, 1), (3, 4) și (4, 3) din Fig.11.1 (repetat aici ca Fig. 12.2), poate fi gândit ca un set de antrenament, unde vectorii sunt unidimensionali. Aceasta este ideea (1, 2) poate fi gândit ca o pereche ([1], 2), unde [1] este caracteristica unidimensională vectorul x și 2 este eticheta asociată y; celelalte puncte pot fi interpretate în mod similar.

Să presupunem că vrem să „învățăm” cel mai bun model pentru setul complet de puncte, de la pe care aceste patru puncte sunt luate ca exemple. O formă simplă de model, cea vom considera, este o funcție liniară f (x) = ax + b. O interpretare firească

(2,1)
(3,4)
(4,3)
(1,2)

Figura 12.2: Repetați figura 11.1, pentru a fi folosit ca set de antrenament

de „cel mai bun” este că RMSE al valorii lui f (x) în comparație cu valoarea dată de y este minimizat. Adică vrem să reducem la minimum

$$\sum_{x=1}^{4} (ax + b - y_x)^2$$

unde y_x este valoarea y asociată cu x. Această sumă este

$(a + b - 2)^2 + (2a + b - 1)^2 + (3a + b - 4)^2 + (4a + b - 3)^2$

Simplificând, suma este $30a^2 + 4b^2 + 20ab - 56a - 20b + 30$. Dacă luăm apoi derivate cu privire la a și b și le setăm la 0, obținem

60a + 20b - 56 = 0
20a + 8b - 20

= 0
Soluția la aceste ecuații este a = 3/5 și b = 1. Pentru aceste valori,
RMSE este 3,2.
Rețineți că linia dreaptă învățată nu este axa principală care a fost
acoperite pentru aceste puncte în secțiunea 11.2.1. Axa respectivă era linia cu pantă
1, parcurgând originea, adică linia y = x. Pentru această linie, RMSE este
4. Diferența este că PCA discutat în secțiunea 11.2.1 minimizează suma
a pătratelor lungimilor proiecțiilor pe axa aleasă, care este
constrâns să treacă prin origine. Aici, minimizăm suma
pătrate ale distanțelor verticale dintre puncte și linie. De fapt, chiar
Dacă am fi încercat să învățăm linia prin originea cu cel mai mic RMSE, am face-o
nu alege linia y = x. Puteți verifica dacă y = 14
15
x are un RMSE mai mic decât
4. ✷
Exemplul 12.3: O aplicație comună a învățării automate implică o instruire
set în cazul în care vectorii de caracteristici x sunt booleeni și de dimensiuni foarte mari.
Ne vom concentra pe date constând din documente, de exemplu, e-mailuri, pagini web sau
articole de ziar. Fiecare componentă reprezintă un cuvânt într-o dicțiune mare
nary. Probabil am elimina din aceasta cuvintele stop (cuvinte foarte frecvente)

---

dicționar, deoarece aceste cuvinte tind să nu ne spună prea multe despre subiectul
ter din document. În mod similar, s-ar putea să restrângem și dicționarul la cuvinte
cu scoruri TF.IDF ridicate (a se vedea secțiunea 1.3.1), astfel încât cuvintele luate în considerare ar avea tendința
pentru a reflecta subiectul sau substanța documentului.
Setul de antrenament este format din perechi, unde vectorul x reprezintă prezența
sau absența fiecărui cuvânt din dicționar în document. Eticheta y ar putea fi +1 sau
−1, cu +1 reprezentând faptul că documentul (un e-mail, de ex.) Este spam. Scopul nostru
ar fi pregătirea unui clasificator pentru examinarea e-mailurilor viitoare și pentru a decide dacă sau nu
sunt spam. Vom ilustra această utilizare a învățării automate în Exemplul 12.4.
Alternativ, y ar putea fi ales dintr-un set finit de subiecte, de exemplu, „sport”
,,Politică” și așa mai departe. Din nou, x ar putea reprezenta un document, poate un web
pagină. Scopul ar fi crearea unui clasificator pentru paginile Web care au atribuit un
subiect pentru fiecare. ✷

12.1.3 Abordări ale învățării automate

Există multe forme de algoritmi ML și nu le vom acoperi pe toate aici.
Iată principalele clase de astfel de algoritmi, fiecare dintre care se distinge prin
forma prin care este reprezentată funcția f.
1. Arborii de decizie au fost discutați pe scurt în secțiunea 9.2.7 și vor fi acoperiți
mai detaliat în secțiunea 12.5. Forma lui f este un copac și fiecare nod al lui
arborele are o funcție de x care determină la ce copil sau copii
căutarea trebuie să continue. Arborii de decizie sunt potriviți pentru binare și multiclase
clasificare, mai ales atunci când dimensiunea vectorului caracteristică nu este
prea mare (un număr mare de funcții poate duce la supra-montare).
2. Perceptronii sunt funcții de prag aplicate componentelor vecului
tor $x = [x_1, x_2, ..., x_n]$. O greutate w_i este asociată cu componenta ith,
pentru fiecare i = 1, 2, ..., n și există un prag θ.Ieșirea este +1 dacă

$$\sum_{i=1}^{n} w_i x_i > \theta$$

iar ieșirea este -1 dacă suma respectivă este mai mică decât θ.Un perceptron este potrivit
pentru clasificarea binară, chiar și atunci când numărul de caracteristici este foarte mare,
de exemplu, prezența sau absența cuvintelor într-un document. Perceptronii sunt
subiectul secțiunii 12.2.
3. Rețelele neuronale sunt rețele aciclice de perceptroni, cu ieșiri ale unora
perceptroni folosiți ca intrări pentru alții. Acestea sunt potrivite pentru binare sau
clasificare multiclass, deoarece pot exista mai multe perceptroni utilizate ca
ieșire, cu una sau mai multe indicând fiecare clasă.
4. Învățarea bazată pe instanțe folosește întregul set de instruire pentru a reprezenta funcțiile
tion f. Calculul etichetei y asociat cu un nou vector de caracteristici
x poate implica examinarea întregului set de antrenament, deși de obicei unele

---

preprocesarea setului de antrenament permite calcularea lui f (x) pentru a
eficient. Vom lua în considerare un tip important de instanță
învățare, k-cel mai apropiat-vecin, în secțiunea 12.4. De exemplu, 1-cel mai apropiat-
vecinul clasifică datele oferindu-i aceeași clasă ca cea a celui mai apropiat
exemplu de instruire. Există k-cei mai apropiați algoritmi vecini care sunt
proprie pentru orice fel de clasificare, deși ne vom concentra asupra
cazul în care y și componentele lui x sunt numere reale.
5. Mașinile vector-suport reprezintă un avans față de algoritmii tradiționali
folosit pentru a selecta greutățile și pragul. Rezultatul este un clasificator care
tinde să fie mai precisă asupra datelor nevăzute. Discutăm vectorul de sprijin
mașini din secțiunea 12.3.

12.1.4 Arhitectura de învățare automată

Algoritmii de învățare automată pot fi clasificați nu numai după algoritmul lor general
abordarea microfonului așa cum am discutat în secțiunea 12.1.3. dar și prin baza lor
arhitectură - modul în care sunt tratate datele și modul în care sunt utilizate pentru a construi modelul.
Instruire, validare și testare
O problemă generală cu privire la manipularea datelor este că există un motiv întemeiat
reține unele dintre datele disponibile din setul de instruire. Datele rămase
se numește set de testare. În unele cazuri, reținem două seturi de exemple de instruire,
un set de validare (numit uneori un set de dezvoltare), precum și setul de testare.
Diferența este că setul de validare este utilizat pentru a ajuta la proiectarea modelului, în timp ce
setul de testare este folosit doar pentru a determina cât de bun este modelul. Problema
abordat de un set de validare este că mulți algoritmi de învățare automată au tendința de a
depășește datele; ridică artefacte care apar în setul de antrenament, dar care
sunt atipice pentru populația mai mare de date posibile. Prin utilizarea validării
set, și văzând cât de bine funcționează clasificatorul în acest sens, putem spune dacă clasificatorul
supradaptează datele. Dacă da, putem restricționa algoritmul de învățare automată
într-un fel. De exemplu, dacă construim un arbore de decizie, putem limita
numărul de nivele ale copacului.
Figura 12.3 ilustrează arhitectura tren-și-test. Ne asumăm toate
datele sunt potrivite pentru instruire (de exemplu, informațiile despre clasă sunt atașate la date),
dar separăm o mică parte din datele disponibile ca set de testare. Noi
utilizați datele rămase pentru a construi un model sau clasificator adecvat. Apoi ne hrănim
datele de testare la acest model. Din moment ce cunoaștem clasa fiecărui element al
date de testare, putem spune cât de bine merge modelul pe datele de testare. Dacă eroarea
rata de date de testare nu este mult mai slabă decât rata de eroare a modelului de pe
datele de instruire în sine, atunci ne așteptăm să fie puțină, dacă există, prea multă dotare și
modelul poate fi folosit. Pe de altă parte, dacă clasificatorul are rezultate mult mai slabe
pe datele de testare decât pe datele de antrenament, ne așteptăm să existe o supra-dotare și
trebuie să regândim modul în care construim clasificatorul.

Test
Model
Model
Generație
Set de antrenament
A stabilit
Rata de eroare
Figura 12.3: Setul de instruire ajută la construirea modelului, iar setul de testare se validează
aceasta

Generalizare

Ar trebui să ne amintim că scopul creării unui model sau clasificator este
nu pentru a clasifica setul de antrenament, ci pentru a clasifica datele a căror clasă o facem
nu știu. Vrem ca aceste date să fie clasificate corect, dar de multe ori avem
nici o modalitate de a ști dacă modelul o face sau nu. Dacă natura
datele se schimbă în timp, de exemplu, dacă încercăm să detectăm spamul
e-mailuri, atunci trebuie să măsurăm performanța în timp, la fel de bine ca noi
poate sa. De exemplu, în cazul e-mailurilor spam, putem nota rata de
rapoarte de e-mailuri spam care nu au fost clasificate ca spam.
Nu este nimic special în selectarea datelor de testare. De fapt, noi
putem repeta procesul de antrenare-testare de mai multe ori folosind aceleași date, dacă noi
împărțiți datele în k bucăți de dimensiuni egale. La rândul nostru, lăsăm fiecare bucată să fie testul

date și utilizați restul de k-1 bucăți ca date de antrenament. Acest antrenament
arhitectura se numește validare încrucișată.
Lot versus învățare on-line
Adesea, ca în Exemplele 12.1 și 12.2, folosim o arhitectură de învățare în serie. Acea
este, întregul set de instruire este disponibil la începutul procesului și este
toate folosite în orice mod necesită algoritmul pentru a produce un model o dată și
pentru toți. Alternativa este învățarea on-line, unde setul de antrenament ajunge într-un
și, ca orice flux, nu poate fi revizuit după ce este procesat. În linie
învățând, menținem un model în orice moment. Pe măsură ce sosesc noi exemple de instruire,
putem alege să modificăm modelul pentru a explica noile exemple. Pe net
învățarea are avantajele pe care le poate
1. Luați cu seturi de antrenament foarte mari, deoarece nu accesează mai mult de
câte un exemplu de antrenament la un moment dat.

Pagina 490

2. Adaptați-vă la schimbările din populație de exemple de instruire pe măsură ce trece timpul.
De exemplu, Google își antrenează clasificatorul de e-mail spam, adaptându-se astfel
clasificatorul pentru spam ca noi tipuri de e-mail spam sunt trimise de către spammeri
și indicat a fi spam de către destinatari.
O îmbunătățire a învățării on-line, potrivită în unele cazuri, este învățarea activă
ing. Aici, clasificatorul poate primi câteva exemple de instruire, dar în primul rând
primește date neclasificate, pe care trebuie să le clasifice. Dacă clasificatorul nu este sigur de
clasificare (de exemplu, exemplul nou sosit este foarte aproape de limita-
ary), atunci clasificatorul poate cere adevărul terenului la un cost semnificativ. Pentru
de exemplu, ar putea să trimită exemplul către Mechanical Turk și să adune opinii despre
oameni adevărați. În acest fel, exemplele de lângă graniță devin exemple de instruire
și poate fi folosit pentru a modifica clasificatorul.
Selecția caracteristicii
Uneori, cea mai grea parte a proiectării unui model bun sau a unui clasificator este să ne gândim
aflați ce caracteristici să utilizați ca intrare în algoritmul de învățare. Să ne reconsiderăm
Exemplul 12.3, unde am sugerat să putem clasifica e-mailurile ca spam sau nu
spam prin examinarea cuvintelor conținute în e-mail. De fapt, explorăm în
detaliați un astfel de clasificator în Exemplul 12.4. După cum sa discutat în Exemplul 12.3, se poate
are sens să te concentrezi pe anumite cuvinte și nu pe altele; de exemplu, ar trebui să eliminăm
oprește cuvintele.
Dar ar trebui, de asemenea, să ne întrebăm dacă există alte informații disponibile
ne-ar ajuta să luăm o decizie mai bună cu privire la spam. De exemplu, spam-ul este adesea
generate de anumite gazde, fie cele care aparțin spammerilor, fie gazde
care au fost cooptate într-o „botnet” în scopul generării de spam.
Astfel, incluzând gazda de origine sau adresa de e-mail de origine în funcție
vectorul care descrie un e-mail ne-ar putea permite să proiectăm un clasificator mai bun și mai mic
rata de eroare.
Crearea unui set de antrenament
Este rezonabil să ne întrebăm unde informațiile de pe etichetă care transformă datele într-un tren-
setul provine din. Metoda evidentă este de a crea etichetele manual, având
o examinare expertă a fiecărui vector de caracteristică și clasificarea corectă. Recent, mulțimea
tehnicile de aprovizionare au fost folosite pentru etichetarea datelor. De exemplu, în multe aplicații
Este posibil să utilizați Mechanical Turk pentru etichetarea datelor. Din moment ce „turcii”
nu sunt neapărat fiabile, este înțelept să folosiți un sistem care permite întrebarea
să li se ceară mai multor persoane diferite, până când o majoritate clară este în favoarea uneia
eticheta.
Deseori se pot găsi date pe Web care sunt etichetate implicit. De exemplu,
Open Directory (DMOZ) are milioane de pagini etichetate după subiect. Aceste date,
folosit ca set de antrenament, poate permite clasificarea altor pagini sau documente
în funcție de subiectul lor, pe baza frecvenței apariției cuvintelor. Un alt
abordarea clasificării în funcție de subiect este de a privi pagina Wikipedia pentru un subiect și

---

Pagina 491

vezi la ce pagini se leagă. Se poate presupune că aceste pagini sunt relevante pentru
subiectul dat.
În unele aplicații putem folosi stelele pe care oamenii le folosesc pentru a evalua produsele
sau servicii de pe site-uri precum Amazon sau Yelp. De exemplu, am putea dori

estimează numărul de stele care ar fi atribuite recenziilor sau tweet-urilor despre un produs, chiar dacă acele recenzii sau tweet-uri nu au în sine evaluări de stele. Dacă folosim recenzii etichetate cu stele ca set de antrenament, putem deduce cuvintele care sunt cel mai frecvent asociat cu recenzii pozitive și negative (numite sentiment analiză). Prezența acestor cuvinte în alte recenzii ne poate spune sentimentul dintre aceste recenzii.

12.1.5 Exerciții pentru secțiunea 12.1

Exercițiul 12.1.1: Refaceți Exemplul 12.2 pentru următoarele forme diferite de f (x).
(a) Solicitați f (x) = ax; adică o linie dreaptă prin origine. Este linia
y = $\frac{14}{15}$x despre care am discutat în exemplul optim?
(b) Permiteți lui f (x) să fie pătratic, adică f (x) = $ax^2 + bx + c$.

12.2 Perceptroni

Un perceptron este un clasificator binar liniar. Intrarea sa este un vector x = [x_1, x_2, ..., x_d] cu componente cu valoare reală. Asociat cu perceptronul este un vector al greutăți w = [w_1, w_2, ..., w_d], de asemenea, cu componente cu valoare reală. Fiecare percepție tron are un prag θ.Ieșirea perceptronului este +1 dacă wx> θși ieșirea este -1 dacă wx <θ.Cazul special în care wx = θva fi întotdeauna privit ca „greșit”, în sensul pe care îl vom descrie în detaliu atunci când ajungem la secțiunea 12.2.1.
Vectorul de greutate w definește un hiperplan cu dimensiunea d - 1 - setul de toate punctele x astfel încât wx = θ,așa cum este sugerat în Fig. 12.4. Puncte pe poziție latura activă a hiperplanului sunt clasificate +1 și cele de pe partea negativă sunt clasificate −1. Un clasificator perceptron funcționează numai pentru date care sunt liniare separabil, în sensul că există un hiperplan care separă toate puncte itive din toate punctele negative. Dacă există multe astfel de hiperplane, perceptronul va converge la unul dintre ei și, prin urmare, va clasifica corect pe toți punctele de antrenament. Dacă nu există un astfel de hiperplan, atunci perceptronul nu poate converge la orice hiperplan. În secțiunea următoare, vom discuta despre vectorul suport mașini, care nu au această limitare; ei vor converge la unele separări tor care, deși nu este un clasificator perfect, se va descurca cât mai bine sub metrică care trebuie descrisă în secțiunea 12.3.

12.2.1 Antrenarea unui Perceptron cu Prag zero

Pentru a antrena un perceptron, examinăm setul de antrenament și încercăm să găsim o greutate vectorul w și pragul θastfel încât toți vectorii caracteristici cu y = +1 (

Pagina 492

w
wx = θ
Figura 12.4: Un perceptron împarte un spațiu cu un hiperplan în două jumătăți de spațiu exemple pozitive) sunt pe partea pozitivă a hiperplanului și a tuturor celor cu y = −1 (exemplele negative) sunt pe partea negativă. Poate fi sau nu este posibil să se facă acest lucru, deoarece nu există nicio garanție că orice hiperplan le separă pe toate exemplele pozitive și negative din setul de antrenament.
Începem prin a presupune că pragul este 0; simpla augmentare necesară gestionarea unui prag necunoscut este discutat în secțiunea 12.2.4. Următorul metoda va converge la un hiperplan care separă pozitivul și exemple negative, cu condiția să existe unul.
1. Inițializați vectorul de greutate w la toate 0.
2. Alegeți un parametru de rată de învățare η,care este un număr real mic, pozitiv. Alegerea lui ηafectează convergența perceptronului. Dacă ηeste prea mic, atunci convergența este lentă; dacă este prea mare, atunci hotărârea de decizie va fi ,,Dansează” și din nou va converge încet, dacă este deloc.
3. Luați în calcul fiecare exemplu de antrenament t = (x, y).
(a) Fie y′ = wx
(b) Dacă y′ și y au același semn, atunci nu faceți nimic; t este corect clasi- fied.
(c) Cu toate acestea, dacă y′ și y au semne diferite, sau y′ = 0, înlocuiți w cu w + ηyx. Adică, reglați ușor w în direcția lui x.
Cazul bidimensional al acestei transformări pe w este sugerat în Fig. 12.5. Observați cum mișcarea w în direcția lui x mută hiperplanul care este perpendicular pe w într-o astfel de direcție încât face mai probabil ca x să fie să fie pe partea corectă a hiperplanului, deși nu garantează că x vor fi apoi corect clasificate.

X
1
X
1
wx = 0
w
w '
w'.x = 0
η
Figura 12.5: Un punct clasificat greșit x_1 mută vectorul w
Exemplul 12.4: Să luăm în considerare instruirea unui perceptron pentru a recunoaște e-mailul spam. Setul de antrenament este format din perechi (x, y) unde x este un vector de 0 și 1, cu fiecare componentă x_i corespunzătoare prezenței (x_i = 1) sau absenței (x_i = 0) unui anumit cuvânt din e-mail. Valoarea lui este +1 dacă e-mailul este cunoscut să fie spam și −1 dacă se știe că nu este spam. În timp ce numărul de cuvinte găsit în setul de instruire al e-mailurilor este foarte mare, vom folosi un exemplu simplificat unde există doar cinci cuvinte: „și”, „viagra”, „cea”, „a” și „nigeria”.
Figura 12.6 prezintă setul de instruire a șase vectori și clasele lor corespunzătoare.
și viagra nigeria
y
A
1
1
0
1
1
+1
b
0
0
1
1
0
−1
c
0
1
1
0
0
+1
d
1
0
0
1
0
−1
e
1
0
1
0
1
+1
f
1
0
1
1
0
−1
Figura 12.6: Date de instruire pentru e-mailurile spam
În acest exemplu, vom folosi rata de învățare η= 1/2 și vom vizita fiecare exemplu de antrenament o dată, în ordinea prezentată în Fig. 12.6. Începem cu w = [0, 0, 0, 0, 0] și calculează wa = 0. Deoarece 0 nu este pozitiv, mutăm w în

direcția lui a executând w: = w + (1/2) (+ 1) a. Noua valoare a lui w
este deci
w = [0, 0, 0, 0, 0] + [
1
2
,
1
2
, 0,
1
2
,
1
2
] = [
1
2
,
1
2
, 0,
1
2
,
1
2
]
Apoi, ia în considerare bwb = [1
2 , 1
2 , 0, 1
2 , 1
2]. [0, 0, 1, 1, 0] = 1
2 . Din moment ce asociate

Pragmatica instruirii pe e-mailuri

Când reprezentăm e-mailurile sau alte documente mari ca exemple de instruire, nu am vrea cu adevărat să construim vectorul 0 și 1 cu a componentă pentru fiecare cuvânt care apare chiar o dată în colecția de e-mailuri. A face acest lucru ne-ar oferi de obicei vectori rare cu milioane de componente. Mai degrabă, creați un tabel în care toate cuvintele care apar în e-mailurilor li se atribuie numere întregi 1, 2, ..., indicând componenta acestora. Cand noi procesați un e-mail în setul de instruire, faceți o listă a componentelor în care vectorul are 1; adică, utilizați reprezentarea standard rar pentru vector.
Dacă eliminăm cuvintele stop din reprezentare sau chiar eliminăm cuvintele cu un scor TF.IDF scăzut, atunci facem vectorii care reprezintă e-mailurile mai rar și astfel comprimă și mai mult datele. Doar vectorul w trebuie să aibă toate componentele sale listate, deoarece nu va fi rar după ce un număr mic de exemple de instruire au fost procesate.
y pentru b este -1, b este clasificat greșit. Astfel atribuim
w: = w + (1/2) (- 1) b = [
1
2
,
1
2
, 0,
1
2
,
1
2] - [0, 0,
1
2
,
1

2
, 0] = [
1
2
,
1
2, -
1
2
, 0,
1
2
]
Exemplul de instruire c este următorul. Calculăm
wc = [
1
2
,
1
2, -
1
2
, 0,
1
2
]. [0, 1, 1, 0, 0] = 0
Deoarece y asociat pentru c este +1, c este, de asemenea, clasificat greșit. Astfel atribuim
w: = w + (1/2) (+ 1) c = [
1
2
,
1
2, -
1
2
, 0,
1
2
] + [0,
1
2
,
1
2
, 0, 0] = [
1
2
, 1, 0, 0,
1
2
]
Următorul exemplu de instruire d este următorul:
wd = [
1
2
, 1, 0, 0,
1
2
]. [1, 0, 0, 1, 0] = 1
Deoarece y asociat pentru d este -1, d este, de asemenea, greșit clasificat. Astfel atribuim
w: = w + (1/2) (- 1) d = [
1
2
, 1, 0, 0,
1
2] - [
1
2
, 0, 0,
1

2, 0] = [0, 1, 0, -
1
2
,
1
2
]
Pentru exemplul de instruire e calculăm we = [0, 1, 0, - 1
2
, 1
2
]. [1, 0, 1, 0, 1] = 1
2
.
Deoarece y asociat pentru e este +1, e este clasificat corect și nu se schimbă în w
se face. În mod similar, pentru f calculăm
wf = [0, 1, 0, -
1
2
,
1
2]. [1, 0, 1, 1, 0] = -
1
2

---

deci f este corect clasificat. Dacă verificăm de la a până la d, descoperim că acest lucru
le clasifică în mod rect și pe acestea. Astfel, am convergut către un perceptron care
clasifică corect toate exemplele setului de antrenament. De asemenea, face o anumită sumă
de sens: spune că „viagra” și „nigeria” sunt indicative de spam, în timp ce „de”
este indicativ pentru nonspam. Consideră „și” și „neutrul”, deși noi
ar prefera probabil să acorde „și”, „din” și „aceloraşi” aceeaşi pondere. ✷

12.2.2 Convergența perceptronelor

După cum am menționat la începutul acestei secțiuni, dacă punctele de date sunt liniare
separabil, atunci algoritmul perceptron va converge într-un separator. In orice caz,
dacă datele nu sunt separabile liniar, atunci algoritmul va repeta în cele din urmă a
vector de greutate și buclă infinită. Din păcate, este adesea greu de spus, în timpul
rularea algoritmului, care dintre aceste două cazuri se aplică. Când datele
este mare, nu este fezabil să ne amintim de toți vectorii de greutate anteriori pentru a vedea dacă
repetăm un vector și chiar dacă am putea, perioada de repetare ar fi
cel mai probabil este atât de mare încât am vrea să terminăm algoritmul mult timp
înainte să repetăm.
O a doua problemă cu privire la reziliere este aceea, chiar dacă datele de instruire sunt
separabil liniar, întregul set de date ar putea să nu fie separabil liniar. The
consecința este că s-ar putea să nu existe nicio valoare în rularea algoritmului
pentru un număr foarte mare de runde, în speranța de a converge la un separator.
Prin urmare, avem nevoie de o strategie pentru a decide când să terminăm perceptronul
algoritm, presupunând că convergența nu a avut loc. Iată câteva obișnuite
teste de reziliere.
1. Termină după un număr fix de runde.
2. Încheiați când numărul de puncte de antrenament clasificate greșit se oprește
ing.
3. Reținețiun set de test din datele de antrenament și, după fiecare rundă, rulați
perceptron pe datele testului. Terminați algoritmul atunci când numărul
de erori pe setul de testare nu se mai schimbă.
O altă tehnică care va ajuta convergența este de a reduce rata de antrenament ca
numărul de runde crește. De exemplu, am putea permite rata de antrenament
ηpentru a începe de la o anumită η_0 și a coborî la η_0 / (1 + ct) după a t-a rundă,
unde c este o mică constantă.

12.2.3 Algoritmul Winnow

Există multe alte reguli pe care le-ai putea folosi pentru a ajusta greutățile pentru un perceptron. Nu
se garantează că toți algoritmii posibili vor converge, chiar dacă există un hiperplan
separând exemple pozitive și negative. Unul care converge se numește
Winnow și această regulă va fi descrisă aici. Winnow presupune că această caracteristică

Pagina 496

vectorii constau din 0 și 1, iar etichetele sunt +1 sau −1. Spre deosebire de elementele de bază
algoritm perceptron, care poate produce componente pozitive sau negative în
vectorul de greutate w, Winnow produce doar greutăți pozitive.
Algoritmul general Winnow permite să existe o varietate de parametri
selectate și vom lua în considerare doar o variantă simplă. Cu toate acestea, toate variantele
au în comun ideea că există un prag pozitiv θ.Dacă w este
vectorul de greutate curent, și x este vectorul caracteristică din setul de antrenament care suntem
luând în considerare în prezent, calculăm wx și îl comparăm cu θ.Dacă wx $\leq \theta$și
clasa pentru x este +1, atunci trebuie să ridicăm greutățile lui w din acele componente
unde x are 1. Înmulțim aceste greutăți cu un număr mai mare decât 1. The
cu cât este mai mare acest număr, cu atât rata de antrenament este mai mare, așa că dorim să alegem un număr
care nu este prea aproape de 1 (sau convergența va fi prea lentă), dar nici prea mare
(sau vectorul de greutate poate oscila). În mod similar, dacă wx $\geq \theta$,dar clasa lui x este
−1, atunci vrem să coborâm greutățile lui w în acele componente în care x este 1.
Înmulțim acele greutăți cu un număr mai mare de 0 dar mai mic de 1. Din nou,
vrem să alegem un număr care nu este prea aproape de 1, dar nici prea mic, pentru
evita convergența lentă sau oscilația.
Vom da detaliile algoritmului folosind factorii 2 și 1/2, pentru
cazurile în care dorim să ridicăm greutăți și, respectiv, să coborâm greutăți. start
algoritmul Winnow cu un vector de greutate w = [w_1, w_2, ..., w_d] ale cărui
componentele sunt 1 și lăsați pragul θegal d, numărul de dimensiuni ale
vectorii din exemplele de instruire. Fie (x, y) următorul exemplu de antrenament
de luat în considerare, unde x = [x_1, x_2, ..., x_d].
1. Dacă wx> θși y = +1, sau wx <θ și y = −1, atunci exemplul este
clasificate corect, deci nu se face nicio modificare la w.
2. Dacă wx $\leq \theta$,dar y = +1, atunci ponderile pentru componentele unde x are
1 sunt prea mici ca grup. Dublați fiecare dintre componentele corespunzătoare
de w. Adică, dacă x_i = 1, atunci setați $w_i := 2w_i$.
3. Dacă wx $\geq \theta$,dar y = −1, atunci ponderile pentru componentele unde x are
1 sunt prea mari ca grup. Înjumătățiți fiecare dintre componentele corespunzătoare
de w. Adică, dacă x_i = 1, atunci setați $w_i := w_i / 2$.
Exemplul 12.5: Să reconsiderăm datele de antrenament din Fig. 12.6. Inițializați
w = [1, 1, 1, 1, 1] și să θ= 5. Mai întâi, luați în considerare vectorul caracteristică a = [1, 1, 0, 1, 1].
wa = 4, care este mai mic decât θ.Deoarece eticheta asociată pentru a este +1, aceasta
exemplu este clasificat greșit. Când un exemplu etichetat +1 este clasificat greșit, trebuie
dublați toate componentele în care exemplul are 1; în acest caz, toate cu excepția
a treia componentă a a este 1. Astfel, noua valoare a lui w este [2, 2, 1, 2, 2].
În continuare, luăm în considerare exemplul de instruire b = [0, 0, 1, 1, 0]. wb = 3, care este mai mic
decât θ.Cu toate acestea, eticheta asociată pentru b este -1, deci nu este necesară nicio modificare la w.
Pentru c = [0, 1, 1, 0, 0] găsim wc = 3 <θ,în timp ce eticheta asociată este +1.
Astfel, dublăm componentele lui w unde componentele corespunzătoare ale lui
c sunt 1. Aceste componente sunt a doua și a treia, deci noua valoare a lui w este
[2, 4, 2, 2, 2].

Pagina 497

Următoarele două exemple de instruire, d și e nu necesită nicio schimbare, deoarece acestea sunt
corect clasificate. Cu toate acestea, există o problemă cu f = [1, 0, 1, 1, 0], deoarece
wf = 6> θ,în timp ce eticheta asociată pentru f este -1. Astfel, trebuie să împărțim
prima, a treia și a patra componentă a w cu 2, deoarece acestea sunt componentele
unde f are 1. Noua valoare a lui w este [1, 4, 1, 1, 2].
X
y
wx OK? și viagra nigeria
1
1
1
1
1
un +1
4
Nu
2

2
1
2
2
b −1
3
da
c +1
3
Nu
2
4
2
2
2
d −1
4
da
e +1
6
da
f
−1
6
Nu
1
4
1
1
2
un +1
8
da
b −1
2
da
c +1
5
Nu
1
8
2
1
2
d −1
2
da
e +1
5
Nu
2
8
4
1
4
f
−1
7
Nu
1
8
2
1
2
4

Figura 12.7: Secvența actualizărilor la w efectuate de algoritmul Winnow pe setul de antrenament din Fig. 12.6

Încă nu am convergut. Se pare că trebuie să luăm în considerare fiecare dintre exemple de instruire de la a până la f din nou. La sfârșitul acestui proces, algoritmul a converg la un vector de greutate w = [1, 8, 2, 1

$_2$, 4], care cu pragul $\theta = 5$
clasifică corect toate exemplele de instruire din Fig. 12.6. Detaliile despre
cei 12 pași către convergență sunt prezentați în Fig. 12.7. Această cifră oferă
eticheta asociată y și produsul punct calculat al lui w și caracteristica dată
vector. Ultimele cinci coloane sunt cele cinci componente ale w după procesare
fiecare exemplu de antrenament. ✷

12.2.4 Permiterea pragului să varieze

Să presupunem acum că alegerea pragului 0, ca în secțiunea 12.2.1 sau pragul d,
ca în Secțiunea 12.2.3 nu este de dorit sau că nu cunoaștem cel mai bun prag
a folosi. Cu prețul adăugării unei alte dimensiuni vectorilor de caracteristici, putem
tratați θca una dintre componentele vectorului de greutate w. Acesta este:
1. Înlocuiți vectorul greutăților $w = [w_1, w_2, ..., w_d]$ cu
$w' = [w_1, w_2, ..., w_d, \theta]$
2. Înlocuiți fiecare vector de caracteristici $x = [x_1, x_2, ..., x_d]$ cu
$x' = [x_1, x_2, ..., x_d, -1]$

Pagina 498

Apoi, pentru noul set de antrenament și vectorul de greutate, putem trata pragul
ca 0 și utilizați algoritmul Secțiunii 12.2.1. Justificarea este că $w'.x' > 0$
este echivalent cu $\sum_{i=1}^{d} w_i x_i + \theta \times -1 = wx - \theta > 0$, care la rândul său este echivalent cu
$wx > \theta$.Aceasta din urmă este condiția unui răspuns pozitiv de la un perceptron
cu prag θ.
De asemenea, putem aplica algoritmul Winnow la datele modificate. Vântura
necesită ca toți vectorii de caracteristici să aibă 0 și 1, ca componente. Cu toate acestea, putem
permiteți un −1 în componenta vectorului caracteristică pentru θdacă o tratăm în felul acesta
opus modului în care tratăm componentele care sunt 1. Adică, dacă instruirea
exemplul este pozitiv și trebuie să creștem celelalte greutăți, ci ne împărțim
componenta pragului cu 2. Și dacă exemplul de instruire este negativ,
și trebuie să micșorăm celelalte greutăți, înmulțim componenta pragului
de 2.
și viagra nigeria
θ
y
A
1
1
0
1
1
−1 +1
b
0
0
1
1
0
−1 −1
c
0
1
1
0
0
−1 +1
d
1
0
0
1
0
−1 −1
e
1

0
1
0
1
−1 +1
f
1
0
1
1
0
−1 −1
Figura 12.8: Date de instruire pentru e-mailurile spam, reprezentând o a șasea componentă negativul pragului

Exemplul 12.6: Să modificăm setul de antrenament din Fig. 12.6 pentru a încorpora un al șaselea „cuvânt” care reprezintă negativul $-\theta$ al pragului. Noile date este prezentat în Fig. 12.8.

X
y
wx
BINE? și viagra nigeria θ
1
1
1
1
1
1
un +1
3 da
b
−1
1 nr
1
1
1
2
1
2
1
2
c +1
- 1
2
Nu
1
2
1
1
2
1
1
d
−1
1
2
Nu
1
2
2
1
1
4
1
2
Figura 12.9: Secvența actualizărilor la w efectuate de algoritmul Winnow pe setul de antrenament din Fig. 12.8

Începem cu un vector de greutate w cu șase 1, așa cum se arată în prima linie din Fig. 12.9. Când calculăm wa = 3, folosind primul vector caracteristică a, noi sunt fericiți, deoarece exemplul de formare este pozitiv, la fel și produsul dot. Cu toate acestea, pentru al doilea exemplu de antrenament, calculăm wb = 1. Deoarece

exemplu este negativ, iar produsul punct este pozitiv, trebuie să ajustăm greutățile. Deoarece b are 1 în componentele a treia și a patra, 1 în cele corespunzătoare componentele lui w sunt înlocuite cu 1/2. Ultima componentă, corespunzătoare lui θ, trebuie dublat. Aceste ajustări dau noul vector de greutate [1, 1, $\frac{1}{2}$, $\frac{1}{2}$, 1, 2] prezentat în a treia linie din Fig. 12.9.

Vectorul caracteristică c este un exemplu pozitiv, dar wc = - $\frac{1}{2}$. Astfel, trebuie dubla a doua și a treia componentă a lui w, deoarece c are 1 în cor-componente care răspund și trebuie să înjumătățim ultima componentă a lui w, care corespunde lui θ.Rezultatul w = [1, 2, 1, $\frac{1}{2}$, 1, 1] este afișat în a patra linie din Fig. 12.9. În continuare, d este un exemplu negativ. Deoarece wd = $\frac{1}{2}$, trebuie din nou reglați greutățile. Înjumătățim greutățile din prima și a patra componentă și dublați ultima componentă, obținând w = [$\frac{1}{2}$, 2, 1, $\frac{1}{4}$, 1, 2]. Acum, toate ex-pozitive probele au un produs punct pozitiv cu vectorul de greutate și toate negative exemplele au un produs punct negativ, deci nu există alte modificări la greutăți.

Perceptronul proiectat are un prag de 2. Are greutățile 2 și 1 pentru ,,Viagra” și „nigeria” și greutăți mai mici pentru „și” și „de”. De asemenea are greutatea 1 pentru „the”, ceea ce sugerează că „the” este la fel de indicativ pentru spam ca „nige-ria ”, ceva ce ne îndoim este adevărat. Cu toate acestea, acest perceptron clasifică toate exemple corect. ✷

12.2.5 Perceptroni multiclase

Există mai multe moduri în care ideea de bază a perceptronului poate fi ex-îngrijit. Vom discuta despre transformări care permit hiperavioanelor să servească ca limite neliniare în secțiunea următoare. Aici, ne uităm la modul în care perceptronii pot să fie folosit pentru a clasifica datele în mai multe clase.

Să presupunem că ni se oferă un set de antrenament cu etichete în k clase diferite. start prin instruirea unui perceptron pentru fiecare clasă; aceste perceptroni ar trebui să aibă fiecare același prag θ.Adică, pentru clasa i tratează un exemplu de antrenament (x, i) ca fiind pozitiv exemplu și toate exemplele (x, j), unde j = i, ca exemplu negativ. Presupune că vectorul de greutate al perceptronului pentru clasa i este determinat să fie w_i după Instruire.

Având în vedere un nou vector x pentru a clasifica, calculăm w_i .x pentru toate i = 1, 2, ..., k. Noi luați clasa x pentru a fi valoarea lui i pentru care w_i .x este maxim, cu condiția acea valoare este cel puțin θ.În caz contrar, se presupune că x nu aparține niciunui k clase.

De exemplu, să presupunem că vrem să clasificăm paginile Web într-o serie de subiecte, precum sport, politică, medicină și așa mai departe. Putem reprezenta pagini Web prin a vector cu 1 pentru fiecare cuvânt prezent în pagină și 0 pentru cuvintele care nu sunt prezente (de desigur, am vizualiza paginile doar în acest fel; nu am construi vectori în realitate). Fiecare subiect are anumite cuvinte care tind să indice acel subiect. De exemplu, paginile de sport ar fi pline de cuvinte precum „câștigă”, „obiectiv”, „jucat” și așa mai departe. Vectorul de greutate pentru acest subiect ar da greutăți mai mari pentru cuvinte care caracterizează acel subiect.

O nouă pagină ar putea fi clasificată ca aparținând subiectului care oferă cel mai mare scor atunci când produsul punct al vectorului paginii și vectorilor de greutate pentru subiectele sunt calculate. O interpretare alternativă a situației este pentru a clasifica o pagină ca aparținând tuturor acelor subiecte pentru care este produsul dot peste un anumit prag.

12.2.6 Transformarea setului de antrenament

În timp ce un perceptron trebuie să utilizeze o funcție liniară pentru a separa două clase, este

posibilă transformarea vectorilor unui set de antrenament înainte de aplicarea unui perceptron-
algoritm bazat pe separarea claselor. În principiu, este întotdeauna posibil să
găsim o astfel de transformare, atâta timp cât suntem dispuși să ne transformăm într-o
spațiul dimensional. Dar dacă înțelegem suficient despre datele noastre, putem face adesea
găsiți o transformare simplă care funcționează. Un exemplu ar trebui să ofere elementul de bază
idee.
Figura 12.10: Transformarea de la coordonatele dreptunghiulare la cele polare transformă acest lucru
antrenament stabilit într-unul cu un hiperplan de separare
Exemplul 12.7: În Fig. 12.10 vedem o parcelă de locuri de vizitat de acasă.
Coordonatele orizontale și verticale reprezintă latitudinea și longitudinea
locuri. Unele dintre locuri au fost clasificate în „excursii de o zi” - locuri apropiate
suficient pentru a vizita într-o zi - și „excursii”, care necesită mai mult de o zi
a vizita. Acestea sunt cercurile și respectiv pătratele. Evident, nu există

---

linie dreaptă care separă excursiile de o zi de excursii. Cu toate acestea, dacă înlocuim
coordonatele carteziene prin coordonate polare, apoi în spațiul transformat al
coordonate polare, cercul punctat prezentat în Fig. 12.10 devine un hiperplan.
În mod formal, transformăm vectorul $x = [x_1, x_2]$ în $[\sqrt{x_1^2 + x_2^2}, \arctan(x_2 / x_1)]$.
De fapt, putem face și reducerea dimensionalității datelor. Unghiul de
punctul este irelevant și doar raza $\sqrt{x_1^2 + x_2^2}$ chestiuni. Astfel, putem
transformă vectorii punctuali în vectori cu o singură componentă dând distanța dintre
punct de la origine. Asociată cu distanțele mici va fi clasa
etichetați „excursie de o zi”, în timp ce distanțele mai mari vor fi asociate cu eticheta
"excursie." Antrenarea perceptronului este extrem de ușoară. ✷

12.2.7 Probleme cu perceptronii

În ciuda extensiilor discutate mai sus, există unele limitări ale capacității
de perceptroni pentru a clasifica unele date. Cea mai mare problemă este că uneori
datele nu sunt inerent separate de un hiperplan. Un exemplu este prezentat în
Fig. 12.11. În acest exemplu, punctele din cele două clase se amestecă aproape de limită astfel
că orice linie prin puncte va avea puncte din ambele clase pe cel puțin una
a laturilor.
Figura 12.11: Un set de antrenament poate să nu permită existența unei separări
hiperplan
După cum sa menționat în secțiunea 12.2.6, este, în principiu, posibil să se găsească unele
funcționează pe punctele care le transformă într-un alt spațiu în care se află
separabile liniar. Cu toate acestea, a face acest lucru ar putea duce foarte bine la supra-amenajarea situației
unde clasificatorul funcționează foarte bine pe setul de antrenament, pentru că a fost
proiectate cu grijă pentru a gestiona corect fiecare exemplu de antrenament. Cu toate acestea, pentru că
clasificatorul exploatează detalii despre setul de instruire care nu se aplică altora

---

exemple care trebuie clasificate în viitor, clasificatorul nu va efectua
bine pe date noi.
?
Figura 12.12: În general, mai mult de un hiperplan poate separa clasele dacă
pot fi deloc separate
O altă problemă este ilustrată în Fig. 12.12. De obicei, dacă clasele pot fi separate
arate de un hiperplan, atunci există multe hiperplane diferite care vor
separați punctele. Cu toate acestea, nu toate hiperavioanele sunt la fel de bune. Pentru
poziție, dacă alegem hiperplanul care este cel mai îndepărtat în sensul acelor de ceasornic, atunci punctul
indicat de "?" va fi clasificat ca un cerc, chiar dacă intuitiv îl vedem ca.
mai aproape de pătrate. Când întâlnim mașini vector-suport în secțiunea 12.3,
vom vedea că există o modalitate de a insista ca hiperplanul ales să fie acela
care într-un sens împarte spațiul cel mai corect.
O altă problemă este ilustrată în Fig. 12.13. Cele mai multe reguli pentru antrenament
un perceptron se oprește imediat ce nu există puncte clasificate greșit. Ca rezultat,
hiperplanul ales va fi unul care doar reușește să clasifice o parte din

punctează corect. De exemplu, linia superioară din Fig. 12.13 tocmai a reușit
pentru a găzdui două dintre pătrate, iar linia inferioară tocmai a reușit
găzdui două dintre cercuri. Dacă oricare dintre aceste linii reprezintă finalul
vector de greutate, atunci greutățile sunt orientate spre una dintre clase. Acea
adică clasifică corect punctele din setul de antrenament, dar linia superioară
ar clasifica noi pătrate care sunt chiar sub el ca cercuri, în timp ce linia inferioară
ar clasifica cercurile chiar deasupra acestuia ca pătrate. Din nou, o alegere mai echitabilă
de separare a hiperplanului va fi prezentată în secțiunea 12.3.

12.2.8 Implementarea paralelă a perceptronelor

Instruirea unui perceptron este un proces inerent secvențial. Dacă numărul
O serie de dimensiuni ale vectorilor implicați este uriașă, atunci am putea obține unele

---

Pagina 503

Figura 12.13: Perceptronii converg de îndată ce ajunge la hiperplanul de separare
regiunea dintre clase

paralelism prin calcularea produselor punct în paralel. Cu toate acestea, așa cum am discutat în
conexiune cu Exemplul 12.4, vectorii de înaltă dimensiune sunt probabil să fie rare
și pot fi reprezentate mai succint decât s-ar aștepta de la ei
lungime.

Pentru a obține paralelism semnificativ, trebuie să modificăm perceptronul
algoritm ușor, astfel încât multe exemple de instruire (un „lot”) sunt utilizate cu
același vector de greutate estimat w. Această modificare a algoritmului se modifică ușor
ce face algoritmul, în comparație cu implementarea secvențială
unde ne schimbăm w după fiecare exemplu de instruire greșit. Cu toate acestea, dacă
rata de învățare este mică, așa cum este în mod normal, apoi reutilizând un singur w pentru multe antrenamente
exemplele fac o mică diferență în valoarea rezultată a w după lotul de
sunt luate în considerare exemple de instruire. De exemplu, să formulăm paralela
algoritm ca job MapReduce.

Funcția Hărți: Fiecărei sarcini Harta i se oferă o bucată de exemple de instruire,
și fiecare sarcină de hartă cunoaște vectorul de greutate actual w. Sarcina Hartă calculează
wx pentru fiecare vector de caracteristică $x = [x_1, x_2, ..., x_k]$ în bucata sa și o compară
punctează produsul cu eticheta y, care este +1 sau −1, asociat cu x. Dacă semnele
de acord, nu sunt produse perechi cheie-valoare pentru acest exemplu de instruire. Cu toate acestea, dacă
semnele nu sunt de acord, atunci pentru fiecare nenul x component $_i$ a x perechea cheie-valoare
(i, ηyx_i) este produs; aici, ηeste constanta de învățare utilizată pentru a instrui acest lucru
perceptron. Observați că ηyx_i este creșterea pe care am dori să o adăugăm la
actuala componentă a lui w, iar dacă $x_i = 0$, atunci nu este nevoie să se producă
o pereche cheie-valoare. În interesul paralelismului, amânăm această schimbare până când noi
poate acumula multe modificări în faza Reduce.

Funcția Reduce: pentru fiecare tastă i, sarcina Reduce care gestionează tasta i
adaugă toate creșterile asociate și apoi adaugă acea sumă componentei ith

---

Pagina 504

Perceptronii privind fluxul de date

Deși am văzut setul de instruire ca date stocate, disponibile pentru repetare
utilizată pe orice număr de treceri, perceptronii pot fi folosiți și într-un flux
setare. Adică, putem presupune că există o secvență infinită de antrenament
exemple, dar că fiecare poate fi folosit o singură dată. Detectarea spamului e-mail este
un bun exemplu de flux de antrenament. Utilizatorii raportează e-mailuri spam și, de asemenea
raportați e-mailurile care au fost clasificate ca spam, dar nu sunt. Fiecare e-mail, așa cum este
ajunge, este tratat ca un exemplu de antrenament și modifică greutatea actuală
vector, probabil cu o cantitate foarte mică.

Dacă setul de antrenament este un flux, nu convergem niciodată cu adevărat, și de fapt
este posibil ca punctele de date să nu fie separabile liniar. Cu toate acestea, în orice moment,
avem o aproximare la cel mai bun separator posibil. Mai mult, dacă
exemplele din flux evoluează în timp, așa cum ar fi cazul e-mailului
spam, atunci avem o aproximare care apreciază mai mult exemplele recente
decât exemple din trecutul îndepărtat, la fel ca în decădere exponențială
tehnica ferestrelor din secțiunea 4.7.
de w.

Probabil, aceste schimbări nu vor fi suficiente pentru a antrena perceptronul. Dacă există
au loc modificări în w, atunci trebuie să începem un nou job MapReduce care să facă

același lucru, poate cu bucăți diferite de setul de antrenament. Cu toate acestea, chiar dacă întregul set de antrenament a fost folosit în prima rundă, acesta poate fi folosit din nou, deoarece efectul său asupra w va fi diferit dacă w s-a schimbat.

12.2.9 Exerciții pentru secțiunea 12.2

Exercițiul 12.2.1: Modificați setul de antrenament din Fig. 12.6 astfel încât și exemplul b include cuvântul „nigeria” (totuși rămâne un exemplu negativ - poate cineva povestind despre călătoria lor în Nigeria). Găsiți un vector de greutate care separă exemple pozitive și negative, folosind:

(a) Metoda de formare de bază din secțiunea 12.2.1.

(b) Metoda Winnow din secțiunea 12.2.3.

(c) Metoda de bază cu un prag variabil, așa cum este sugerat în secțiunea 12.2.4.

(d) Metoda Winnow cu un prag variabil, așa cum este sugerat în secțiunea 12.2.4.

! Exercițiul 12.2.2: Pentru următorul set de antrenament:

([1, 2], +1) ([2, 1], +1)

([2, 3], −1) ([3, 2], −1)

Un truc cheie pentru obținerea paralelismului

Metoda pe care am folosit-o în secțiunea 12.2.8 pentru a transforma un proces serial inerent într-unul paralel apare frecvent atunci când avem de-a face cu seturi de date mari. În versiunea serială a algoritmului, există o stare care se schimbă la fiecare Etapa. În acest caz, starea este vectorul de greutate w. Atâta timp cât se schimbă în starea tinde să fie mică la fiecare pas, putem repara starea și calcula, în paralel, modificările la această stare exactă care ar fi cauzate de fiecare a etapelor seriale. După pasul paralel, combinați modificările pentru a face un starea nouă și repetați pasul paralel până la convergență.

descrieți toți vectorii w și pragurile θastfel încât hiperplanul (într-adevăr a linie) definită de wx - θ= 0 separă corect punctele.

! Exercițiul 12.2.3: Să presupunem că următoarele patru exemple constituie un antrenament a stabilit:

([1, 2], −1) ([2, 3], +1)

([2, 1], +1) ([3, 2], −1)

(a) Ce se întâmplă atunci când încercați să instruiți un perceptron pentru a le clasifica puncte folosind 0 ca prag?

!! (b) Este posibil să se schimbe pragul și să se obțină un perceptron care clasifică în mod corect aceste puncte?

(c) Sugerați o transformare folosind polinoame pătratice care se vor transforma aceste puncte astfel încât să devină liniar separabile.

12.3 Suport-Mașini Vector

Putem vizualiza o mașină vector-suport, sau SVM, ca o îmbunătățire a perceptron care este conceput pentru a aborda problemele menționate în secțiunea 12.2.7. Un SVM selectează un anumit hiperplan care nu doar separă punctele din cele două clase, dar o face într-un mod care maximizează marja - distanța între hiperplan și cele mai apropiate puncte ale setului de antrenament.

În această secțiune, începem cu o discuție despre SVM pentru puncte de antrenament care sunt separabile și arătăm cum să maximizăm marja într-un astfel de caz. Atunci considerăm o problemă mai complexă, în care punctele nu sunt liniare separabil. În acest caz, scopul nostru este diferit. Trebuie să găsim un hiperplan care face tot ce poate în separarea celor două clase. Dar „cel mai bun” este un lucru complicat noțiune. Vom dezvolta o funcție de pierdere care penalizează punctele clasificate greșit, dar de asemenea, unul care penalizează într-o măsură mai mică punctele care sunt corect clasificate, dar sunt prea aproape de hiperplanul de separare. Această chestiune va fi abordată în Secțiunea 12.3.3.

12.3.1 Mecanica unui SVM

Scopul unui SVM este de a selecta un hiperplan wx + b = 0 care maximizează distanța γîntre hiperplan și orice punct al setului de antrenament. 1 The ideea este sugerată de Fig. 12.14. Acolo, vedem punctele a două clase și a hiperplanul împărțindu-i.

A sustine
vectori
γ
γ
wx + b = 0
Figura 12.14: Un SVM selectează hiperplanul cu cea mai mare marjă posibilă
γîntre hiperplan și punctele de antrenament
Intuitiv, suntem mai siguri de clasa de puncte care sunt departe de
separând hiperplanul decât suntem de puncte apropiate de acel hiperplan. Astfel, ea
este de dorit ca toate punctele de antrenament să fie cât mai departe de hiperplan
(dar pe partea corectă a acelui hiperplan, desigur). Un avantaj suplimentar
de a alege hiperplanul de separare pentru a avea o marjă cât mai mare este
că pot exista puncte mai apropiate de hiperplan în setul complet de date, dar nu
în setul de antrenament. Dacă da, avem șanse mai mari ca aceste puncte să fie
clasificate corespunzător decât dacă am alege un hiperplan care separă antrenamentul
puncte, dar a permis ca unele puncte să fie foarte aproape de hiperplanul în sine. In aceea
caz, există o șansă justă ca un nou punct care să fie aproape de un punct de antrenament care
era, de asemenea, în apropierea hiperplanului care ar fi clasificat greșit. Această problemă a fost discutată
în secțiunea 12.2.7 în legătură cu Fig. 12.13.
Vedem și în Fig. 12.14 două hiperplane paralele la distanța γde
hiperplan central wx + b = 0, iar fiecare atinge unul sau mai multe dintre
vectori de sprijin. Acestea din urmă sunt punctele care constrânge de fapt divizarea
hiperplan, în sensul că sunt toate la distanță γde hiperplan.
În majoritatea cazurilor, un set de puncte d-dimensional are d + 1 vectori de susținere, așa cum este
cazul din Fig. 12.14. Cu toate acestea, pot exista mai mulți vectori de suport dacă sunt prea mulți
punctele se întind pe hiperplanele paralele. Vom vedea un exemplu bazat
[1] Constanta b din această formulare a unui hiperplan este aceeași cu negativul pragului
θîn tratamentul perceptronilor din secțiunea 12.2.

---

pe punctele din Fig. 11.1, unde se dovedește că toate cele patru puncte sunt suport
vectori, chiar dacă datele bidimensionale au în mod normal trei.
O declarație provizorie a obiectivului nostru este:

- Având un set de antrenament $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, maximizați γ(prin variație
w și b) sub rezerva constrângerii că pentru toate i = 1, 2, ..., n,
$y_i (wx_i + b) \geq \gamma$

Observați că y_i, care trebuie să fie +1 sau −1, determină ce parte a hiperplanului
punctul x_i trebuie să fie activat, deci relația ≥ cu γeste întotdeauna corectă. Cu toate acestea, ea
poate fi mai ușor să exprimăm această condiție ca două cazuri: dacă y = +1, atunci w.x + b ≥ γ,
iar dacă y = −1, atunci wx + b ≤ $-\gamma$.
Din păcate, această formulare nu funcționează corect. Problema
este că, mărind w și b, putem permite întotdeauna o valoare mai mare a γ.Pentru
de exemplu, să presupunem că w și b satisfac constrângerea de mai sus. Dacă înlocuim w
cu 2w și b cu 2b, observăm că pentru toate i, $y_i ((2w) .x_i + 2b) \geq 2\gamma$. Astfel, 2w
iar 2b este întotdeauna o alegere mai bună decât w și b, deci nu există cea mai bună alegere și
fără γmaxim.

12.3.2 Normalizarea hiperavionului

Soluția la problema pe care am descris-o intuitiv mai sus este normalizarea
vectorul de greutate w. Adică unitatea de măsură perpendiculară pe separată
Hiperplanul de evaluare este vectorul unitar g / g. Amintiți-vă că w este Frobenius
norma sau rădăcina pătrată a sumei pătratelor componentelor lui w. Noi
va cere ca w să fie astfel încât hiperplanele paralele care ating doar
vectorii suport sunt descrișI prin ecuațiile w.x + b = +1 și w.x + b = −1,
după cum sugerează Fig. 12.15. Ne vom referi la hiperavioanele definite de acestea
două ecuații ca hiperplanele superioare și inferioare, respectiv.
Se pare că am setat γ= 1. Dar, din moment ce folosim w ca vector unitate,
marja γeste numărul de „unități”, adică pași în direcția w necesară
a merge între hiperplanul de separare și hiperplanele paralele. Scopul nostru
devine să maximizeze γ,care este acum multiplul vectorului unitar g / g
între hiperplanul de separare și hiperplanele superioare și inferioare.
Primul nostru pas este de a demonstra că maximizarea γeste la fel ca minimizarea
w. Luați în considerare unul dintre vectorii de susținere, să zicem x_2 prezentat în Fig. 12.15. Fie x_1 BE
proiecția lui x_2 pe hiperplanul superior, de asemenea, așa cum este sugerat în Fig. 12.15.
Rețineți că x_1 nu trebuie să fie un vector de sprijin sau chiar un punct al setului de antrenament.
Distanța de la x_2 la x_1 în unități de greutate / greutate este 2γ. Acesta este,
$x_1 = x_2 + 2\gamma$

w
w
(12.1)
Deoarece x_1 se află pe hiperplanul definit de wx + b = +1, știm asta
$wx_1 + b = 1$. Dacă substituim x_1 folosind ecuația 12.1, obținem
$w. (x_2 + 2\gamma$
w
w)
+ b = 1

Pagina 508

wx + b = +1
γ
γ
wx + b = 0
wx
+ b = 0
+ b = −1
w
||
/ || **w**
1
2
X
X
Figura 12.15: Normalizarea vectorului de greutate pentru un SVM
Regrupăm termenii, vedem
$wx_2 + b + 2\gamma$
ww
w
= 1
(12.2)
Dar primii doi termeni ai ecuației 12.2, $wx_2 + b$, însumează la -1, deoarece știm
că x_2 se află pe hiperplanul inferior, wx + b = −1. Dacă deplasăm acest −1 de la stânga
la dreapta în ecuația 12.2 și apoi împărțim la 2, concluzionăm că
γ
ww
w
= 1
(12.3)
Observați, de asemenea, că ww este suma pătratelor componentelor lui w.
Adică, $ww = w_2$. Concluzionăm din ecuația 12.3 că γ= 1 / w.
Această echivalență ne oferă o modalitate de reformulare a problemei de optimizare
menționat inițial în secțiunea 12.3.1. În loc să maximizăm γ,vrem să
mize w, care este inversul lui γ.Acesta este:

- Având un set de antrenament (x_1, y_1), (x_2, y_2), ..., (x_n, y_n), minimizați w (în funcție de
ing w și b) sub rezerva constrângerii că pentru toate i = 1, 2, ..., n,
$y_i (wx_i + b) \geq 1$

Exemplul 12.8: Să luăm în considerare cele patru puncte din Fig. 11.1, presupunând că
alternează ca exemple pozitive și negative. Adică setul de antrenament
este format din
([1, 2], +1) ([2, 1], −1)
([3, 4], +1) ([4, 3], −1)

Pagina 509

Fie w = [u, v]. Scopul nostru este de a minimiza $\sqrt{u_2 + v_2}$ sub rezerva constrângerilor
derivăm din cele patru exemple de instruire. Pentru prima, unde x_1 = [1, 2] și
y_1 = +1, constrângerea este (+1) (u + 2v + b) = u + 2v + b ≥ 1. Pentru a doua,
unde x_2 = [2, 1] și y_2 = −1, constrângerea este (−1) (2u + v + b) ≥ 1 sau
2u + v + b ≤ −1. Ultimele două puncte sunt tratate în mod analog, iar cele patru
constrângerile pe care le derivăm sunt:
u + 2v + b ≥ 1
2u + v + b ≤ −1

3u + 4v + b ≥ 1 4u + 3v + b ≤ −1
Vom acoperi în detaliu subiectul modului în care se optimizează în condiții de constrângere; subiectul este larg și multe pachete sunt disponibile pentru a fi utilizate. Secțiunea 12.3.4 discută o metodă - coborârea în gradient - în legătură cu a aplicație mai generală a SVM, unde nu există hiperplan de separare. Un ilustrarea modului în care funcționează această metodă va apărea în Exemplul 12.9. În acest exemplu simplu, soluția este ușor de văzut: b = 0 și w = [u, v] = [−1, +1]. Se întâmplă ca toate cele patru constrângeri să fie îndeplinite exact; adică fiecare din cele patru puncte este un vector suport. Acest caz este neobișnuit, de când datele sunt bidimensionale, ne așteptăm doar la trei vectori de susținere. Însă faptul că exemplele pozitive și negative se află pe linii paralele permite tuturor celor patru constrângeri care trebuie îndeplinite exact. ✷

12.3.3 Găsirea separatorilor aproximativi optimi

Vom lua în considerare acum găsirea unui hiperplan optim în cazul mai general, unde, indiferent de hiperplanul pe care îl alegem, vor exista câteva puncte pe parte greșită și poate câteva puncte care sunt pe partea corectă, dar prea aproape la hiperplanul de separare în sine, deci cerința de marjă nu este îndeplinită. A situația tipică este prezentată în Fig. 12.16. Vedem două puncte care sunt clasificate greșit; se află pe partea greșită a hiperplanului de separare wx + b = 0. De asemenea, noi vezi două puncte care, deși sunt clasificate corect, sunt prea aproape de hiperplanul de separare. Vom numi toate aceste puncte puncte rele.
Fiecare punct rău atrage o penalizare atunci când evaluăm un posibil hiperplan. Valoarea penalității, în unități care urmează să fie determinată ca parte a optimizării proces, este arătat de săgeata care duce la punctul rău din hiperplan pe partea greșită a căreia se află punctul rău. Adică săgețile măsoară distanța față de hiperplanul wx + b = 1 sau wx + b = −1. Primul este linia de bază pentru exemple de instruire care ar trebui să fie deasupra separării hiperplan (deoarece eticheta y este +1), iar acesta din urmă este linia de bază pentru puncte care ar trebui să fie mai jos (pentru că y = −1).
Avem multe opțiuni cu privire la formula exactă pe care dorim să o minimize. Intuitiv, dorim ca w să fie cât mai mic, așa cum am discutat în Secțiunea 12.3.2. Dar vrem și penalizările asociate cu punctele proaste să fie cât mai mic posibil. Cea mai comună formă de compromis este exprimată printr - o formulă care implică termenul w cu $_2$ /2 și un alt termen care presupune ori constante suma penalizărilor.

Pagina 510

wx + b = −1
wx + b = +1
wx + b = 0
Misclasificat
Prea aproape
la hotar
Figura 12.16: Punctele care sunt clasificate greșit sau sunt prea aproape de separare hiperavionul suportă o penalizare; valoarea penalizării este proporțională cu lungimea săgeții care duce la acel punct
Pentru a vedea de ce minimizând termenul w cu $_2$ /2 închide sens, rețineți că minimizare w este la fel ca minimizarea oricărei funcții monotone a lui w, deci este cel puțin un opțiunea de a alege o formulă în care vom încerca să reducă la minimum w cu $_2$ /2. Această expresie este de dorit deoarece derivatul său în raport cu orice componentă a lui w este că componentă. Adică, dacă w = [w $_1$, w $_2$, ..., w $_d$], apoi w $_2$ /2 este $\frac{1}{2}\sum_{i=1}^{n} w_{eu}^2$, deci e derivată parțială ∂/ ∂w$_i$ este w $_i$. Această situație are sens pentru că, așa cum vom face noi vezi, derivatul termenului de penalizare față de w $_i$ este un timp constant fiecare x $_i$, componenta corespunzătoare a fiecărui vector de caracteristică al cărui exemplu de instruire atrage o pedeapsă. La rândul său, asta înseamnă că vectorul w și vectorii lui setul de instruire este proporțional cu unitățile componentelor lor.
Astfel, vom lua în considerare cum să minimalizăm funcția specială
f (w, b) =
$$\frac{1}{2}\sum_{j=1}^{d} w_j^2$$

$_j + C$
$\sum_{i=1}^{n}$ maxim $\{0, 1 - y_i ($
$\sum_{j=1}^{d}$
$w_j x_{ij} + b)\}$
(12,4)
Primul termen încurajează w mic, în timp ce al doilea termen, care implică
C constantă care trebuie aleasă corect, reprezintă pedeapsa pentru punctele proaste
într-un mod care trebuie explicat mai jos. Presupunem că există n exemple de instruire
(x_i , y_i) pentru i = 1, 2, ..., n și $x_i = [x_{i1} , x_{i2} , ..., x_{id}]$. De asemenea, ca și înainte, w =
$[w_1 , w_2 , ..., w_d]$. Rețineți că cele două însumări $\sum_{j=1}^{d}$ exprimă produsul punct
de vectori.
C constantă, numită parametru de regularizare, reflectă cât de importantă este
clasificarea greșită este. Alegeți un C mare dacă chiar nu doriți să clasificați greșit

---

puncte, dar ați accepta o marjă restrânsă. Alegeți un C mic dacă sunteți bine
cu câteva puncte clasificate greșit, dar doresc ca majoritatea punctelor să fie departe de
limita (adică marja este mare).
Trebuie să explicăm funcția de penalizare (al doilea termen) în ecuația 12.4. The
însumarea peste i are un termen
$L (x_i , y_i) = \max \{0, 1 - y_i ($
$\sum_{j=1}^{d}$
$w_j x_{ij} + b)\}$
pentru fiecare exemplu de antrenament x_i . L este funcția de articulație, sugerată în Fig. 12.17,
și numim valoarea sa pierderea articulației. Fie $z_i = y_i (\sum_{j=1}^{d} w_j x_{ij} + b)$. Când z_i este 1
sau mai mult, valoarea lui L este 0. Dar pentru valori mai mici ale lui z_i , L crește liniar ca z_i
scade.
−2
−1
0
1
2
3
0
1
2
max {0, 1− *z* }
z = y
eu
w .
eu
X
(
+ *b*)
Figura 12.17: Funcția balama scade liniar pentru $z \leq 1$ și apoi rămâne
0
De vreme ce va trebui să luăm derivata cu privire la fiecare w_j de
$L (x_i , y_i)$, rețineți că derivatul funcției balama este discontinuu. Este
$-y_i x_{ij}$ pentru $z_i <1$ și 0 pentru $z_i > 1$. Adică, dacă $y_i = +1$ (adică, a
exemplu este pozitiv), atunci
∂L
∂w_j
= dacă
$\sum_{j=1}^{d}$
$w_j x_{ij} + b \geq 1$ apoi 0 altul - x_{ij}
Mai mult, dacă $y_i = -1$ (adică exemplul de antrenament ith este negativ), atunci

∂L

∂w_j

= dacă

$\sum_{j=1}^{d}$

$w_j x_{ij} + b \leq -1$ apoi 0 altul x_{ij}

Cele două cazuri pot fi rezumate ca unul singur, dacă implicăm valoarea lui $_i$, ca:

∂L

∂w_j

= dacă y_i (

$\sum_{j=1}^{d}$

$w_j x_{ij} + b) \geq 1$ apoi 0 altul - $y_i x_{ij}$

(12,5)

12.3.4 Soluții SVM după coborârea gradientului

O abordare comună pentru rezolvarea ecuației 12.4 este utilizarea programării pătratice.
Pentru datele la scară largă, o altă abordare, coborârea în gradient are un avantaj. Noi
poate permite ca datele să rămână pe disc, mai degrabă decât să le păstreze pe toate în memorie, care
este necesar în mod normal pentru rezolvatorii pătratici. Pentru a implementa coborârea în gradient, noi
calculați derivata ecuației față de b și fiecare componentă
w_j al vectorului w. De vreme ce dorim să minimalizăm f (w, b), mutăm b și
componentele w_j în direcția opusă direcției gradientului. The
cantitatea pe care o mutăm fiecare componentă este proporțională cu derivata cu respect
la acea componentă.
Primul nostru pas este să folosim trucul Secțiunii 12.2.4 pentru a face parte b din
vector greutate w. Observați că b este într-adevăr negativul unui prag pe punct
produs wx, astfel încât să putem adăuga o componentă (d + 1) st la w și să adăugăm o
componentă suplimentară cu valoare +1 pentru fiecare vector de caracteristici din setul de antrenament (nu
−1 așa cum am făcut în secțiunea 12.2.4).
Trebuie să alegem o constantă ηpentru a fi fracțiunea gradientului pe care îl mutăm
w în fiecare rundă. Adică atribuim

$w_j := w_j - \eta$

∂f

∂w_j

pentru toate j = 1, 2, ..., d + 1.
Derivatul ∂

∂w_j

din primul termen din ecuația 12.4, $\frac{1}{2}\sum_{j=1}^{d} w_i^2$, este ușor;
este w_j . Cu toate acestea, al doilea termen implică funcția balama, deci este mai greu
a exprima. Vom folosi o expresie if-then pentru a descrie aceste derivate, ca
în ecuația 12.5. Acesta este:

∂f

∂w_j

= w_j + C

$\sum_{i=1}^{n}$ (dacă y_i (

$\sum_{j=1}^{d}$

$w_j x_{ij} + b) \geq 1$ apoi 0 altul - $y_i x_{ij}$)

(12,6)

Rețineți că această formulă ne oferă o derivată parțială cu privire la fiecare compoziție
nentul lui w, inclusiv w_{d+1} , care este b, precum și la greutățile w_1 , w_2 , ..., w_d .
Continuăm să folosim b în loc de echivalentul w_{d+1} în condiția if-then to
amintește-ne de forma în care este descris hiperplanul dorit.
Pentru a executa algoritmul descendent-descendent pe un set de antrenament, alegem:

1. Valori pentru parametrii C și η.
2. Valorile inițiale pentru w, inclusiv componenta (d + 1) a b.
Apoi, în mod repetat:
(a) Calculați derivatele parțiale ale lui f (w, b) față de w_j 's.
(b) Ajustați valorile lui w scăzând $\eta \partial$
∂w_j
din fiecare w_j .

Pagina 513

Exemplul 12.9: Figura 12.18 prezintă șase puncte, trei pozitive și trei negative
tive. Ne așteptăm ca cea mai bună linie de separare să fie orizontală și singura
întrebarea este dacă hiperplanul de separare și scara lui w vor fi sau nu
face ca punctul (2, 2) să fie clasificat greșit sau să stea prea aproape de limită.
Inițial, vom alege w = [0, 1], un vector vertical cu o scară de 1, și noi
va alege b = −2. Ca rezultat, vedem în Fig. 12.18 că punctul (2, 2) se află
pe hiperplanul inițial și cele trei puncte negative sunt chiar la margine.
Valorile parametrilor pe care le vom alege pentru coborârea în gradient sunt C = 0,1 și
η= 0,2.
w
hiperplan
Marja
Marja
Inițială
(1,4)
(2,2)
(1,1)
(2,1)
(3,1)
(3,4)
Figura 12.18: Șase puncte pentru un exemplu de gradient-coborâre
Începem prin a încorpora b ca a treia componentă a lui w și pentru notațional
comoditate, vom folosi u și v ca primele două componente, mai degrabă decât
obișnuit w_1 și w_2 . Adică luăm w = [u, v, b]. De asemenea, extindem
punctele dimensionale ale setului de antrenament cu o a treia componentă care este întotdeauna 1.
Adică setul de antrenament devine
([1, 4, 1], +1) ([2, 2, 1], +1) ([3, 4, 1], +1)
([1, 1, 1], −1) ([2, 1, 1], −1) ([3, 1, 1], −1)
În Fig. 12.19 vom tabela condițiile if-then și contribuțiile rezultate
mențiuni la însumările peste i în ecuația 12.6. Suma trebuie să fie
multiplicat cu C și adăugat la u, v sau b, după caz, pentru a implementa
secțiunea 12.6.
Adevărul sau falsitatea fiecăreia dintre cele șase condiții din Fig. 12.19 determină
contribuția termenilor din însumările peste i în ecuația 12.6. Noi
va reprezenta starea fiecărei condiții printr-o succesiune de x și o, cu x
reprezentând o condiție care nu deține și o reprezentând una care o face.
Primele câteva iterații de coborâre în gradient sunt prezentate în Fig. 12.20.

Pagina 514

pentru u pentru v
pentru b
dacă
u + 4v + b ≥ +1 atunci
0 altceva
−1
−4
−1
dacă 2u + 2v + b ≥ +1 atunci
0 altceva
−2
−2
−1

dacă 3u + 4v + b ≥ +1 atunci
0 altceva
−3
−4
−1
dacă
u + v + b ≤ −1 atunci
0 altceva
+1
+1
+1
dacă
2u + v + b ≤ −1 atunci
0 altceva
+2
+1
+1
dacă
3u + v + b ≤ −1 atunci
0 altceva
+3
+1
+1
Figura 12.19: Sumați fiecare dintre acești termeni și multiplicați cu C pentru a obține contribuția de puncte negative la derivatele lui f față de u, v și b
w = [u, v]
b
Rău
∂ / ∂u
∂ / ∂v
∂ / ∂b
(1)
[0.000, 1.000] −2.000 oxoooo
−0.200
0,800 −2,100
(2)
[0,040, 0,840] -1,580 oxoxxx
0,440
0,940 -1,380
(3) [-0,048, 0,652] -1,304 oxoxxx
0,352
0,752 -1,104
(4) [−0.118, 0.502] −1.083 xxxxxx −0.118 −0.198 −1.083
(5) [-0.094, 0.542] -0.866 oxoxxx
0,306
0,642 −0,666
(6) [-0.155, 0.414] -0.733 xxxxxx
Figura 12.20: Începutul procesului de coborâre în gradient
Luați în considerare linia (1). Se arată valoarea inițială a w = [0, 1], așa cum am sugerat în Fig. 12.18 a fost valoarea inițială a lui w. Reamintim că folosim u și v pentru componente ale lui w, deci u = 0 și v = 1. Vedem și valoarea inițială a lui b = −2, care este valoarea adecvată pentru hiperplanul inițial prezentat în Fig. 12.18.
Trebuie să folosim aceste valori pentru u, v și b pentru a evalua condițiile din Fig. 12.19.
Prima dintre condițiile din Fig. 12.19 este u + 4v + b ≥ +1. Partea stângă este 0 + 4 + (- 2) = 2, deci condiția este îndeplinită. Cu toate acestea, a doua condiție, 2u + 2v + b ≥ +1 eșuează. Partea stângă este 0 + 2 + (- 2) = 0. Faptul că suma este 0 înseamnă că al doilea punct (2, 2) se află exact pe hiperplanul de separare și nu în afara marginii. A treia condiție este îndeplinită, deoarece 0 + 4 + (- 2) = 2 ≥ +1.
Ultimele trei condiții sunt, de asemenea, îndeplinite și, de fapt, sunt îndeplinite exact. Pentru de exemplu, a patra condiție este u + v + b ≤ −1. Partea stângă este 0 + 1 + (- 2) = −1.
Astfel, modelul oxoooo reprezintă rezultatul acestor șase condiții, ca și noi vezi în prima linie din Fig. 12.20.
Folosim aceste condiții pentru a calcula derivatele parțiale. Pentru ∂f/ ∂u, noi folosiți u în locul lui w_j în ecuația 12.6. Această expresie devine astfel
u + C (0 + (- 2) + 0 + 0 + 0 + 0) = 0 +
1
10 (−2) = −0,2
Suma care înmulțește C poate fi explicată în acest fel. Pentru fiecare dintre cele șase condiții

din Fig. 12.19, luați 0 dacă condiția este îndeplinită și luați valoarea în
coloana etichetată „pentru u” dacă nu este satisfăcută. În mod similar, pentru v în locul lui w j we

Pagina 515

obține ∂f/ ∂v= 1+ 1
10 (0 + (- 2) + 0 + 0 + 0 + 0) = 0,8. În cele din urmă, pentru b obținem
∂f / ∂b= −2 + 1
10 (0 + (- 1) + 0 + 0 + 0 + 0) = −2.1.
Acum putem calcula noile w și b care apar pe linia (2) din Fig. 12.20.
Deoarece am ales η= 0,2, noua valoare a lui u este 0 - 1
5 (-0,2) = -0,04, noul
valoarea lui v este 1− 1
5 (0,8) = 0,84, iar noua valoare a lui b este −2− 1
5 (−2,1) = −1,58.
Pentru a calcula derivatele prezentate în linia (2) din Fig. 12.20 trebuie mai întâi să verificăm
condițiile din Fig. 12.19. În timp ce rezultatele primelor trei condiții
nu s-au schimbat, ultimii trei nu mai sunt mulțumiți. De exemplu,
a patra condiție este u + v + b ≤ −1, dar 0,04 + 0,84 + (−1,58) = −0,7, care este
nu mai puțin de -1. Astfel, modelul punctelor rele devine oxoxxx. Noi acum
au mai mulți termeni diferiți de zero în expresiile derivatelor. De exemplu
∂f / ∂u= 0,04 + 1
10 (0 + (- 2) + 0 + 1 + 2 + 3) = 0,44.
Valorile lui w și b în linia (3) sunt calculate din derivatele lui
linia (2) în același mod în care au fost calculate în linia (2). Noile valori
nu schimbați modelul punctelor rele; este încă oxoxxx. Cu toate acestea, atunci când noi
repetați procesul pentru linia (4), constatăm că toate cele șase condiții sunt nesatisfăcute.
De exemplu, prima condiție, u + 4v + b ≥ +1 nu este îndeplinită, deoarece
(−0.118 + 4 × 0.502 + (−1.083) = 0.807, care este mai mic decât 1. De fapt,
primul punct a devenit prea aproape de hiperplanul de separare, chiar dacă este
clasificate corespunzător.
Putem vedea că în linia (5) din Fig. 12.20, problemele cu prima și
al treilea punct este corectat și ne întoarcem la modelul oxoxxx de puncte rele.
Cu toate acestea, la linia (6), punctele au devenit din nou prea apropiate de separare
hiperplan, deci ne întoarcem la modelul xxxxxx al punctelor negative. Esti invitat
pentru a continua succesiunea actualizărilor la w și b pentru mai multe iterații.
Ne-am putea întreba de ce procesul gradient-coborâre pare să fie convergent
pe o soluție în care cel puțin unele dintre puncte sunt în interiorul marginii, când
există un hiperplan evident (orizontal, la înălțimea 1,5) cu o margine de 1/2,
care separă punctele pozitive și negative. Motivul este că atunci când noi
ales C = 0,1 spuneam că nu ne pasă prea mult dacă
există puncte în interiorul marginilor sau chiar dacă punctele sunt clasificate greșit. Eram
spunând, de asemenea, că ceea ce era important era o marjă mare (care corespunde
un mic w), chiar dacă unele puncte au încălcat aceeași marjă. ✷

12.3.5 Coborârea în gradient stochastic

Algoritmul de coborâre în gradient descris în secțiunea 12.3.4 este adesea numit lot
coborâre în gradient, deoarece la fiecare rundă sunt luate în considerare toate exemplele de antrenament
considerat ca un „lot”. Deși este eficient pentru seturile de date mici, poate fi prea mult timp
consumând pentru a executa pe un set de date mare, unde trebuie să vizităm fiecare antrenament
exemplu, de multe ori înainte de convergență.
O alternativă, numită coborâre de gradient stochastic, consideră un antrenament ex-
amplu sau câteva exemple de instruire la un moment dat și ajustează estimarea curentă a
funcția de eroare (w în exemplul SVM) în direcția indicată numai de
micul set de exemple de instruire luate în considerare. Sunt posibile runde suplimentare,

Pagina 516

utilizarea altor seturi de exemple de instruire; acestea pot fi selectate aleatoriu sau
conform unor strategii fixe. Rețineți că este normal ca unii membri ai
setul de instruire nu este folosit niciodată într-un algoritm de descindere a gradientului stocastic.
Exemplul 12.10: Reamintim algoritmul de descompunere UV discutat în sec.
secțiunea 9.4.3. Acest algoritm a fost descris ca un exemplu de degradare a lotului
aromă. Putem privi fiecare dintre intrările neblane din matricea M care suntem

încercând să aproximăm după produs UV ca exemplu de instruire și eroare
funcția este eroarea rădăcină-medie-pătrat între produsul ma-
trices U și V și matricea M, luând în considerare numai acele elemente în care M este
nonblank.
Cu toate acestea, dacă M are un număr foarte mare de intrări non-blank, așa cum ar fi
cazul dacă M a reprezentat, să zicem, achiziții de articole de către clienții Amazon sau filme
pe care clienții Netflix l-au evaluat, atunci nu este practic să se facă repetate
trece peste întregul set de intrări neblane ale lui M la ajustarea intrărilor
în U și V. O implementare descendentă stocastică a gradientului ar privi o
o intrare neblankă a lui M și calculați schimbarea la fiecare element al lui U și
V care ar face produsul UV să fie de acord cu acel element al lui M. Am fi
nu faceți schimbarea completă a elementelor lui U și V, ci alegeți mai degrabă
o anumită rată de învățare ηmai mică de 1 și schimbă fiecare element al lui U și V cu
fracțiunea ηdin cantitatea care ar fi necesară pentru ca UV să fie egal cu M în
intrarea aleasă. ✷
Există un compromis între descrierea în gradient a lotului și stochastic numit
coborâre în gradient de minibatch. În versiunea minibatch, partiționăm întregul
antrenament setat în „minibatch-uri”, de o anumită dimensiune, de exemplu, 1000 de examene de
ples. Lucrăm la un minibatch la un moment dat, calculând modificările la w folosind
Ecuația 12.4, dar însumând doar peste exemplele de antrenament selectate.

12.3.6 Implementarea paralelă a SVM

Prima observație este că descendența de gradient stochastic este inerent serială,
întrucât starea sistemului - vectorul w și constanta b - se schimbă cu
fiecare exemplu de instruire luat în considerare. Pe de altă parte, coborârea în gradient a lotului
este paralelizat cel mai ușor, deoarece folosește fiecare exemplu de instruire începând de la
aceeași stare și combină efectele acestor exemple de antrenament doar la sfârșit
a unei rotunde.
Astfel, putem paralela SVM folosind descendența gradientului într-un mod analog
la ceea ce am sugerat pentru perceptroni în secțiunea 12.2.8. Puteți începe cu
curent w și b și împărțiți exemplele de antrenament în minibatch-uri, creând unul
sarcină pentru fiecare minibatch. Sarcinile aplică fiecare ecuația 12.4 minibatch-ului lor,
iar modificările aduse stării, w și b, sunt însumate după o rundă paralelă.
Noua stare este calculată prin însumarea tuturor modificărilor, iar procesul poate
repetați cu noua stare distribuită tuturor sarcinilor.

12.3.7 Exerciții pentru secțiunea 12.3

Exercițiul 12.3.1: Continuați iterațiile din Fig. 12.20 pentru încă trei iterații.
tiuni.
Exercițiul 12.3.2: Următorul set de antrenament respectă regula conform căreia pozitivul
toate exemplele au vectori ale căror componente însumează 10 sau mai multe, în timp ce suma
este mai mic de 10 pentru exemplele negative.
([3, 4, 5], +1) ([2, 7, 2], +1) ([5, 5, 5], +1)
([1, 2, 3], −1) ([3, 3, 2], −1) ([2, 4, 1], −1)
(a) Care dintre acești șase vectori sunt vectori suport?
! (b) Sugerați un vector w și o constantă b astfel încât hiperplanul definit de
wx + b = 0 este un bun separator pentru exemplele pozitive și negative.
Asigurați-vă că scara lui w este astfel încât toate punctele să fie în afara marginii;
adică, pentru fiecare exemplu de antrenament (x, y), aveți y (wx + b) ≥ +1.
! (c) Începând cu răspunsul dvs. la partea (b), utilizați coborârea în gradient pentru a găsi
w și b optime. Rețineți că, dacă începeți cu un hiperplan de separare,
și scalați corect w, atunci al doilea termen al ecuației 12.4 va fi
fii întotdeauna 0, ceea ce îți simplifică considerabil munca.
! Exercițiul 12.3.3: Următorul set de antrenament respectă regula conform căreia pozitivul
exemple au toate vectori ale căror componente au o sumă impar, în timp ce suma
este chiar pentru exemplele negative.
([1, 2], +1) ([3, 4], +1) ([5, 2], +1)
([2, 4], −1) ([3, 1], −1) ([7, 3], −1)
(a) Sugerați un vector de pornire w și o constantă b care clasifică cel puțin trei dintre
punctele corect.
!! (b) Începând cu răspunsul dvs. la (a), utilizați coborârea în gradient pentru a găsi optimul
w și b.

12.4 Învățarea de la vecinii cei mai apropiați

În această secțiune luăm în considerare câteva exemple de „învățare” în care întregul
setul de antrenament este stocat, probabil preprocesat într-un mod util, și apoi utilizat

pentru a clasifica exemple viitoare sau pentru a calcula valoarea etichetei care este cea mai mare probabil asociat cu exemplul. Vectorul caracteristic al fiecărui exemplu de antrenament este tratat ca un punct de date într-un anumit spațiu. Când ajunge un punct nou și trebuie fie clasificate, găsim exemplul sau exemplele de instruire care sunt cele mai apropiate de punct nou, în funcție de măsura distanței pentru acel spațiu. Estimat eticheta este apoi calculată prin combinarea exemplelor cele mai apropiate într-un fel.

Pagina 518

12.4.1 Cadrul pentru calculele cele mai apropiate

Setul de antrenament este mai întâi preprocesat și stocat. Deciziile au loc când ajunge un nou exemplu, numit exemplu de interogare și trebuie clasificat. Există mai multe decizii pe care trebuie să le luăm pentru a proiecta un algoritm bazat pe vecin care va clasifica exemple de interogare. Enumerăm ei aici.

1. Ce măsură de distanță folosim?
2. La câți dintre cei mai apropiați vecini ne uităm?
3. Cum ponderăm cei mai apropiați vecini? În mod normal, oferim o funcție (funcția kernel) a distanței dintre exemplul de interogare și cei mai apropiați vecini din setul de antrenament și utilizați această funcție pentru a cântări vecini. Dacă nu există ponderare, atunci funcția kernel nu trebuie să fie specificat.
4. Cum definim eticheta de asociat cu interogarea? Această etichetă este o funcție a etichetelor celor mai apropiați vecini, poate ponderată cu funcția kernel sau poate nu.

12.4.2 Învățarea cu cel mai apropiat vecin

Cele mai simple cazuri de învățare cu cel mai apropiat vecin sunt atunci când alegem doar un vecin care este cel mai apropiat de exemplul de interogare. În acest caz, nu are rost pentru ponderarea vecinilor, deci funcția kernel este omisă. De asemenea este si de obicei o singură alegere posibilă pentru funcția de etichetare: luați eticheta interogarea să fie aceeași cu eticheta celui mai apropiat vecin.

Exemplul 12.11: Figura 12.21 prezintă câteva exemple de câini care durează a apărut în Fig. 12.1. Am renunțat la majoritatea exemplelor pentru simplitate, rămânând doar trei chihuahua, doi teckel și doi beagle. Din moment ce vectorii înălțime-greutate care descriu câinii sunt bidimensionali, există un simplu și mod eficient de a construi o diagramă Voronoi pentru punctele, în care bisectoarele perpendiculare ale liniilor dintre fiecare pereche de puncte este construită. Fiecare punct primește o regiune în jurul său, conținând toate punctele în care se află cel mai apropiat. Aceste regiuni sunt întotdeauna convexe, deși pot fi deschise infinitul într-o singură direcție. [2] Este, de asemenea, un fapt surprinzător că, chiar dacă există O (n^2) bisectoare perpendiculare pentru n puncte, diagrama Voronoi poate fi găsită în timp O (n log n).

În Fig. 12.21 vedem diagrama Voronoi pentru cele șapte puncte. Limita-berbecii care separă câinii de rase diferite sunt prezentate solide, în timp ce limitele

[2] În timp ce regiunea care aparține oricărui punct este convexă, uniunea regiunilor pentru două sau s-ar putea ca mai multe puncte să nu fie convexe. Astfel, în Fig. 12.21 vedem că regiunea pentru toate Dachshunds iar regiunea pentru toate Beagles nu este convexă. Adică, există puncte p_1 și p_2 care sunt ambele clasificate Dachshunds, dar punctul de mijloc al liniei dintre p_1 și p_2 este clasificat ca a Beagle și invers.

Pagina 519

Beagles
Teckelii
Chihuahua

Figura 12.21: Diagrama Voronoi pentru cele trei rase de câini

între câinii din aceeași rasă sunt afișați în linii. Să presupunem un exemplu de interogare q este furnizat. Rețineți că q este un punct din spațiul din Fig. 12.21. Găsim regiunea în care se încadrează q și dați q eticheta exemplului de antrenament căruia acea regiune aparține. Rețineți că nu este prea greu să găsiți regiunea q. Noi trebuie să stabilim în ce parte a anumitor linii se încadrează q. Acest proces este același așa cum am folosit în secțiunile 12.2 și 12.3 pentru a compara un vector x cu un hiperplan perpendicular pe un vector w. De fapt, dacă liniile care formează de fapt părți ale Diagrama Voronoi este preprocesată corespunzător, putem face determinarea în O (log n) comparații; nu este necesar să comparați q cu tot O (n log n) liniile care fac parte din diagramă. ✷

12.4.3 Învățarea funcțiilor unidimensionale

Un alt caz simplu și util de învățare cu cel mai apropiat vecin este unidimensional date. În această situație, exemplele de instruire sunt de forma ([x], y), iar noi le va scrie ca (x, y), identificând un vector unidimensional cu singurul său componentă. De fapt, setul de instruire este o colecție de eșantioane ale valorii a unei funcții y = f (x) pentru anumite valori ale lui x și trebuie să interpolăm funcția f în toate punctele. Există multe reguli care ar putea fi folosite și vom face asta descrie doar câteva dintre abordările populare. După cum sa discutat în secțiunea 12.4.1, abordările variază în numărul de vecini pe care îi folosesc, indiferent dacă sunt sau nu

---

Pagina 520

vecinii sunt ponderati și, dacă da, cum variază greutatea în funcție de distanță.
Să presupunem că folosim o metodă cu k vecinii cei mai apropiați și x este punctul de interogare.
Fie x_1 , x_2 , ..., x_k cel mai apropiat k vecini ai lui x și să fie greutatea asociată cu punct de antrenament (x_i , y_i) be w_i . Atunci estimarea etichetei y pentru x este

$$\sum_{i=1}^{k} w_i y_i / \sum_{i=1}^{k} w_i$$

. Rețineți că această expresie oferă media ponderată de etichetele celor mai apropiați vecini.

Exemplul 12.12: Vom ilustra patru reguli simple, folosind setul de instruire (1, 1), (2, 2), (3, 4), (4, 8), (5, 4), (6, 2) și (7, 1). Aceste puncte reprezintă a funcție care are un vârf la x = 4 și se descompune exponențial pe ambele părți.
Rețineți că acest set de antrenament are valori de x care sunt uniform distanțate. Nu este cerința ca punctele să fie distanțate uniform sau să aibă orice alt model regulat.
Unele modalități posibile de interpolare a valorilor sunt:

1
2
3
4
5
6
7
1
2
3
4
5
6
7

(a) Un vecin cel mai apropiat
(b) Media a doi vecini cei mai apropiați

Figura 12.22: Rezultatele aplicării primelor două reguli din Exemplul 12.12

1. Cel mai apropiat vecin. Folosiți doar cel mai apropiat vecin. Nu e nevoie pentru o ponderare. Ia doar valoarea oricărui f (x) pentru a fi eticheta y asociată ated cu punctul de set de antrenament cel mai apropiat de punctul de interogare x. Rezultatul folosind această regulă pe exemplul de antrenament descris mai sus este prezentat în Fig. 12.22 (a).
2. Media celor doi vecini cei mai apropiați. Alegeți 2 ca număr cel mai apropiat vecinii de folosit. Greutățile acestor două sunt fiecare 1/2, indiferent de modul în care sunt departe de punctul de interogare x. Rezultatul acestei reguli pe exemplu setul de antrenament este în Fig. 12.22 (b).
3. Media ponderată a celor doi vecini cei mai apropiați. Alegem din nou două vecinii cei mai apropiați, dar îi ponderăm în proporție inversă cu dis- din punctul de interogare. Să presupunem că cei doi vecini sunt cei mai apropiați de interogare

---

Pagina 521

punctul x sunt x_1 și x_2 . Să presupunem mai întâi că $x_1 < x < x_2$. Apoi greutatea din x_1 , inversul distanței sale față de x, este 1 / (x - x_1), iar greutatea lui x_2 este 1 / (x_2 - x). Media ponderată a etichetelor este

(y 1
x - x_1
+
y 2
x_2 - x) / (
1
x - x_1

+
1
x_2 - x)
care, atunci când înmulțim numărătorul și numitorul cu $(x - x_1)(x_2 - x)$,
simplifică la
$y_1 (x_2 - x) + y_2 (x - x_1)$
$x_2 - x_1$
Această expresie este interpolare liniară a celor mai apropiați doi vecini, ca
prezentat în Fig. 12.23 (a). Când ambii vecini cei mai apropiați sunt de aceeași parte
din interogarea x, aceleași greutăți au sens, iar estimarea rezultată este
o extrapolare. Vedem extrapolarea în Fig. 12.23 (a) în intervalul x = 0
la x = 1. În general, când punctele sunt distanțate neuniform, putem găsi interogare
puncte din interior unde ambii vecini sunt pe o parte.
4. Media celor trei apropiați vecini. Putem media orice număr de
cei mai apropiați vecini pentru a estima eticheta unui punct de interogare. Figura 12.23 (b)
arată ce se întâmplă pe exemplul nostru de antrenament atunci când cele mai apropiate trei
se folosesc vecini, fără ponderare. Ponderarea celor trei vecini,
cum ar fi în proporție inversă cu distanțele lor până la punctul în cauză,
este o altă opțiune.
✷
1
2
3
4
5
6
7
1
2
3
4
5
6
7
(a) Media ponderată a doi vecini
(b) Media a trei vecini
Figura 12.23: Rezultatele aplicării ultimelor două reguli din Exemplul 12.12

12.4.4 Regresia kernelului

O modalitate de a construi o funcție continuă care reprezintă datele unui antrenament
stabilit bine este să ia în considerare toate punctele din setul de antrenament, dar să pondereze punctele
folosind o funcție de nucleu care se descompune odată cu distanța. O alegere populară este de a utiliza
o distribuție normală (sau „curbă clopot”), deci greutatea unui punct de antrenament x
când interogarea este q este $e^{-(x-q)^2/\sigma^2}$. Aici σ este abaterea standard a
distribuție (un parametru pe care îl selectați) și interogarea q este media. Aproximativ,
punctele aflate la distanța σ de q sunt puternic ponderate, iar cele aflate mai departe au
greutate mică. Există un avantaj în utilizarea unei funcții kernel, cum ar fi
distribuție normală, care este continuă și definită pentru toate punctele antrenamentului
a stabilit; acest lucru asigură că funcția rezultată învățată din date este ea însăși
continuu. (A se vedea exercițiul 12.4.6 pentru o discuție a problemei atunci când este mai simplu
se utilizează ponderarea.)
Exemplul 12.13: Să folosim cele șapte exemple de instruire din Exemplul 12.12.
Pentru a simplifica calculul, nu vom folosi distribuția normală ca
funcția kernel, ci mai degrabă o altă funcție continuă a distanței, și anume
$w = 1 / (x - q)^2$. Adică, greutățile se descompun ca pătratul distanței. Presupune
interogarea q este 3,5. Greutățile w_1, w_2, ... w_7 din cele șapte exemple de antrenament
$(x_i, y_i) = (i, 8/2^{|i-4|})$ pentru i = 1, 2, ..., 7 sunt prezentate în Fig. 12.24.
(1) x_i
1
2
3
4
5
6
7
(2) y_i
1
2
4
8

4
2
1
(3) w_i
4/25 4/9
4
4
4/9
25/04/49
(4) $w_i y_i$
4/25 8/9 16 32 16/9 8/25 4/49
Figura 12.24: Ponderile punctelor atunci când interogarea este q = 3.5
Liniile (1) și (2) din Fig. 12.24 dau cele șapte puncte de antrenament. Greutatea din fiecare când interogarea este q = 3.5 este dată în linia (3). De exemplu, pentru $x_1 = 1$, greutatea $w_1 = 1 / (1 - 3,5)^2 = 1 / (- 2,5)^2 = 4/25$. Apoi, linia (4) arată fiecare y_i ponderat de greutatea de la linia (3). De exemplu, coloana pentru x_2 are valoare 8/9 deoarece $w_2 y_2 = 2 \times (4/9)$.
Pentru a calcula eticheta pentru interogarea q = 3,5 însumăm valorile ponderate a etichetelor din setul de antrenament, așa cum este dat de linia (4) din Fig. 12.24; această sumă este 51,23. Împărțim apoi la suma greutăților din linia (3). Această sumă este 9,29, deci raportul este 51,23 / 9,29 = 5,51. Această estimare a valorii etichetei pentru q = 3.5 pare rezonabil din punct de vedere intuitiv, deoarece q se află la jumătatea distanței dintre două puncte cu etichetele 4 și 8. ✷

12.4.5 Tratarea datelor euclidiene de înaltă dimensiune

Am văzut în secțiunea 12.4.2 că cazul bidimensional al datelor euclidiene este destul de usor. Există mai multe structuri de date la scară largă care au fost dezvoltate

---

Pagina 523

Probleme în limita pentru exemplul 12.13

Să presupunem că q este exact egal cu unul dintre exemplele de antrenament x. Dacă folosim distribuția normală ca funcție de nucleu, nu există nicio problemă cu greutatea lui x; este 1. Cu toate acestea, cu funcția kernel discutată în Exemplul 12.13, greutatea lui x este $1 / (x - q)^2 = \infty$. Din fericire, această greutate apare atât în numărătorul, cât și în numitorul expresiei care estimează eticheta q. Se poate arăta că în limită pe măsură ce se apropie q x, eticheta lui x domină toți ceilalți termeni atât în numărător, cât și numitor, deci eticheta estimată a lui q este aceeași cu eticheta lui x.
Acest lucru are un sens excelent, deoarece q = x în limită.
a găsit vecini apropiați când numărul dimensiunilor crește și setul de antrenament este mare. Nu vom acoperi aici aceste structuri, deoarece subject ar putea umple singură o carte și există multe locuri disponibile pentru a afla mai multe aceste tehnici, denumite colectiv structuri de indici multidimensionali. Vezi note bibliografice pentru acest capitol pentru informații despre structuri precum kd-Trees, R-Trees și Quad Tree.
Din păcate, pentru datele de înaltă dimensiune, este puțin posibil de făcut evitați să căutați o mare parte din date. Acest fapt este o altă manifestare a „blestemului dimensionalității” din secțiunea 7.1.3. Două modalități de a face față ,,Blestem” sunt următoarele:
1. Fișiere VA. Deoarece oricum trebuie să analizăm o mare parte din date pentru a găsi cei mai apropiați vecini ai unui punct de interogare, am putea evita o structura datelor complexă cu totul. Acceptați că trebuie să scanăm întregul fișier, dar faceți acest lucru în două etape. În primul rând, este prezentat un rezumat al fișierului creat, folosind doar un număr mic de biți care aproximează valorile fiecare componentă a fiecărui vector de antrenament. De exemplu, dacă folosim doar de ordin înalt (1/4) al biților din componentele numerice, atunci putem crea un fișier care este (1/4) mărimea setului de date complet. Cu toate acestea, prin scanare acest fișier putem construi o listă de candidați care ar putea fi printre k cei mai apropiați vecini ai interogării q, iar această listă poate fi o mică parte din întregul set de date. Căutăm apoi doar pe acești candidați în întregime fișier, pentru a determina care k sunt cele mai apropiate de q.
2. Reducerea dimensiunii. Putem trata vectorii setului de antrenament ca matrice, unde rândurile sunt vectorii exemplului de antrenament și coloanele corespund componentelor acestor vectori. Aplicați una dintre tehnicile de reducere a dimensionalității din capitolul 11, pentru a comprima vectori la un număr mic de dimensiuni, suficient de mici încât tehnicile

pentru indexarea multidimensională se poate folosi. Desigur, la procesare
un vector de interogare q, aceeași transformare trebuie aplicată la q înainte

---

Pagina 524

căutând cei mai apropiați vecini ai q.

12.4.6 Tratarea distanțelor neeuclidiene

Până în acest moment, am presupus că măsura distanței este euclidiană. Cum-
oricând, majoritatea tehnicilor pot fi adaptate în mod natural la o distanță arbitrară
funcția d. De exemplu, în secțiunea 12.4.4 am vorbit despre utilizarea unui disc normal
tributarea ca funcție de nucleu. De vreme ce ne gândeam la o dimensiune unidimensională
antrenat într-un spațiu euclidian, am scris exponentul ca - $(x - q)^2$. In orice caz,
pentru orice funcție de distanță d, putem folosi ca greutate a unui punct x la distanță
d (x, q) din punctul de interogare q valoarea lui

$$e^{-(d(x-q))^2/\sigma^2}$$

Rețineți că această expresie are sens dacă datele sunt în unele dimensiuni ridicate
Spațiul euclidian și d este distanța euclidiană obișnuită sau distanța Manhattan sau
orice altă distanță discutată în secțiunea 3.5.2. De asemenea, are sens dacă d este Jaccard
distanță sau orice altă măsură de distanță.
Cu toate acestea, pentru distanța Jaccard și celelalte măsuri de distanță pe care le considerăm
prezentat în secțiunea 3.5 avem, de asemenea, opțiunea de a utiliza hashing sensibil la localitate,
subiectul Capitolului 3. Reamintim că aceste metode sunt doar aproximative și ele
ar putea produce negative negative - exemple de instruire care erau aproape de vecinii unui
interogare, dar care nu apar într-o căutare.
Dacă suntem dispuși să acceptăm astfel de erori ocazional, putem construi gălețile
pentru setul de antrenament și păstrați-le ca reprezentare a setului de antrenament.
Aceste găleți sunt concepute astfel încât să le putem prelua pe toate (sau aproape pe toate, deoarece există
să fie false negative) punctele de pregătire care au o similaritate minimă cu a
interogare dată q. În mod echivalent, va apărea una dintre gălețile la care interogarea hashează
conține toate acele puncte la o distanță maximă de q. Sperăm că așa
mulți dintre cei mai apropiați vecini ai q, așa cum necesită metoda noastră, vor fi găsiți printre aceștia
găleți.
Cu toate acestea, dacă interogările diferite au distanțe radical diferite de cele mai apropiate
vecini, nu totul se pierde. Putem alege mai multe distanțe $d_1 < d_2 < d_3 < \cdots$.
Construiți găleți pentru hashing sensibil la localitate utilizând fiecare dintre aceste distanțe.
Pentru o interogare q, începeți cu gălețile pentru distanța d_1 . Dacă vom găsi destul de aproape
vecini, am terminat. În caz contrar, repetați căutarea folosind cupele pentru d_2 ,
și așa mai departe, până când se găsesc suficienți vecini apropiați.

12.4.7 Exerciții pentru secțiunea 12.4

Exercițiul 12.4.1: Să presupunem că am modificat Exemplul 12.11 pentru a ne uita la cele două
cei mai apropiați vecini ai unui punct de interogare q. Clasificați q cu eticheta comună dacă acestea
doi vecini au aceeași etichetă și lasă q neclasificat dacă etichetele din
vecinii sunt diferiți.
(a) Schițați limitele regiunilor pentru cele trei rase de câini din Fig. 12.21.

Pagina 525

! (b) Limitele ar consta întotdeauna din segmente de linie dreaptă pentru oricare
date de instruire?
Exercițiul 12.4.2: Să presupunem că avem următorul set de antrenament
([1, 2], +1) ([2, 1], −1)
([3, 4], −1) ([4, 3], +1)
care este setul de instruire utilizat în Exemplul 12.9. Dacă folosim cel mai apropiat-vecin
învățând cu cel mai apropiat vecin ca estimare a etichetei unei interogări
punct, ce puncte de interogare sunt etichetate cu +1?
Exercițiul 12.4.3: Luați în considerare setul de antrenament unidimensional
(1, 1), (2, 2), (4, 3), (8, 4), (16, 5), (32, 6)
Descrieți funcția f (q), eticheta returnată ca răspuns la interogare
q, când interpolarea utilizată este:
(a) Eticheta celui mai apropiat vecin.
(b) Media etichetelor celor mai apropiați doi vecini.
! (c) Media, ponderată în funcție de distanță, a celor mai apropiați doi vecini.
(d) Media (neponderată) a celor mai apropiați trei vecini.

! Exercițiul 12.4.4: Aplicați funcția kernel din Exemplul 12.13 la datele din
Exercițiul 12.4.3. Pentru interogările q din intervalul 2 <q <4, care este eticheta lui q?
Exercițiul 12.4.5: Care este funcția care estimează eticheta punctelor de interogare
folosind datele din Exemplul 12.12 și media celor mai apropiați patru vecini?
!! Exercițiul 12.4.6: Funcții simple de ponderare precum cele din Exemplul 12.12
nu trebuie să definiți o funcție continuă. Putem vedea că funcțiile construite
în Fig. 12.22 și Fig. 12.23 (b) nu sunt continue, dar Fig. 12.23 (a) este. Are
media ponderată a celor mai apropiați doi vecini oferă întotdeauna o funcție continuă?

12.5 Arbori de decizie

Un arbore de decizie este un program de ramificare care utilizează proprietățile unui vector de caracteristică pentru
produce clasa căreia îi aparține acea intrare. În general, arătăm deciziile
realizat sub forma unui copac. În această secțiune, vom discuta cum să proiectăm
arbori care clasifică corect datele de antrenament. Într-un arbore de decizie, fiecare nod nonleaf
reprezintă un test pe intrare. Copiii săi sunt fie teste (pe care le denotăm prin
elipse) sau o frunză care este o concluzie despre ieșire (notată printr-un dreptunghi).
Copiii unui nod de testare sunt etichetați după rezultatul testului. De obicei,

---

Pagina 526

există doar două rezultate - adevărat sau fals (da sau nu) - dar ar putea exista
numărul de rezultate ale unui test.
De asemenea, în această secțiune, vom examina modalitățile de a exploata paralelismul în timp ce căutăm
pentru cel mai eficient copac. Întrucât dotarea excesivă este o problemă obișnuită, vom face și noi
vorbiți despre cum să simplificați arborele prin eliminarea nodurilor, pentru a reduce supraadaptarea
fără a pierde prea multă acuratețe.

12.5.1 Utilizarea unui arbore de decizie

Să începem cu un exemplu de date de antrenament și un arbore care ar putea
să fie construite din aceste date. Tabelul din Fig. 12.25 prezintă 12 țări
populația (în milioane), continentul lor și sportul lor preferat. Vom trata
Populația și continentul ca trăsături ale vectorului de intrare și ale sportului preferat
ca clasă sau ieșire. Presupunem că țările în sine nu sunt cunoscute și
vrem să prezicem sportul preferat doar de pe continent și populație.
În special, vrem să prezicem sportul preferat al altor țări decât acestea
12, doar de pe continentul și populația lor.
Țară
Populația continentului
Sport
Argentina
SA
44
Fotbal
Australia
Aus
34
Cricket
Brazilia
SA
211
Fotbal
Canada
N / A
36
Hochei
Cuba
N / A
11
Baseball
Germania
Euro
80
Fotbal
India
Asia
1342
Cricket
Italia

Euro
59
Fotbal
Rusia
Asia
143
Hochei
Spania
Euro
46
Fotbal
Regatul Unit
Euro
65
Cricket
Statele Unite
N / A
326
Baseball
Figura 12.25: Sporturile preferate ale țărilor
Exemplul 12.14: În Fig. 12.26 este un arbore care clasifică corect cei doisprezece
țările din Fig. 12.25. Pentru a clasifica o țară, având în vedere populația și continentul său,
începem de la rădăcină și aplicăm testul la rădăcină. Mergem la copilul din stânga dacă
rezultatul testului este adevărat și pentru copilul potrivit dacă nu. Ori de câte ori suntem
la un nod fără frunze, aplicăm testul la acel nod și ne mutăm din nou spre stânga sau
copil potrivit, în funcție de testul satisfăcut sau nu. Când ajungem la o
frunze, vom scoate clasa la frunza respectivă.
Puteți verifica dacă fiecare dintre cele douăsprezece țări este corect clasificată prin aceasta
copac. De exemplu, luați în considerare Spania. Continentul Spaniei este Europa, deci
testul la rădăcină este satisfăcut. Astfel, mergem la copilul stâng al rădăcinii, care

---

Continent =
60 <Pop <70
35 <Pop <200
Continent =
N / A
Cricket
Fotbal
Hochei
Cricket
Baseball
Da
Da
Da
Da
N
N
N
N
SA sau Eur
Figura 12.26: Un arbore de decizie pentru sporturile preferate din țări
ne cere să testăm dacă populația Spaniei este între 60 și 70 de milioane.
Nu este, așa că mergem la copilul potrivit, care este o frunză care ne spune corect că
Sportul preferat al Spaniei este fotbalul.
Cu toate acestea, există multe țări care nu se află în tabelul din Fig. 12.25 și
arborele decizional nu merge prea bine pe acestea. De exemplu, luați în considerare Pakistanul, a
țară din Asia cu o populație de 182 milioane. Începând de la rădăcină, găsim
condiția de acolo nu este îndeplinită, deoarece Pakistanul nu este un
Țară americană. Astfel, mergem la copilul potrivit al rădăcinii. Din moment ce pop-
din Pakistan este cuprinsă între 35 și 200 de milioane, ne deplasăm la stânga,
după care ni se spune că sportul preferat în Pakistan este hocheiul.
Problema cu care ne confruntăm este supra-dotarea. Adică, testul la rădăcină are sens;
Fotbalul este cel mai popular în America de Sud și Europa. Dar teste pe populație
sunt probabil inutile, deoarece este puțin probabil ca dimensiunea unei țări să aibă ceva
să facă cu ce sport îi place oamenii săi. În acest exemplu, am folosit pur și simplu
populația să distingă între numărul mic de țări din Fig. 12.25
care ajung la un nod al copacului. Dar, spre deosebire de rădăcină, testele de la al doilea nivel

nu spune cu adevărat nimic care să se aplice pentru ansamblul mai mare, nevăzut de țări. ✷

12.5.2 Măsuri de impuritate

Pentru a proiecta un arbore de decizie, trebuie să alegem teste bune la diferite nonleaf nodurile arborelui. Ne-ar plăcea să folosim cât mai puține niveluri, astfel încât noi punctele de date pot fi clasificate rapid și avem șansa de a evita suprasolicitare pe care am văzut-o în Exemplul 12.14. În mod ideal, ne-ar plăcea toate intrările care ajung la un anumit nod pentru a avea aceeași clasă, de atunci putem face asta nodul unei frunze și clasificarea corectă a tuturor exemplelor de antrenament care ajung la nod. Putem formaliza proprietatea pe care am dori-o pentru un nod prin noțiunea de impuritate. Există multe măsuri de impuritate pe care le-am putea folosi, dar toate le au

---

proprietatea că sunt 0 pentru un nod la care se ajunge doar prin exemple de instruire cu o singură clasă. Iată trei dintre cele mai comune măsuri de impuritate. Fiecare se aplică unui nod atins prin exemple de instruire cu n clase, cu p_i fiind fracțiune din acele exemple de instruire care aparțin clasei i, pentru i = 1, 2, ..., n.

1. Acuratețe: fracțiunea de intrări care sunt clasificate corect, sau $1 - \max(p_1, p_2, ..., p_n)$.
2. Impuritate GINI: $1 - \sum_{i=1}^{n} (p_i)^2$.
3. Entropie: $\sum_{i=1}^{n} p_i \log_2(1/p_i)$.

Exemplul 12.15: Luați în considerare impuritatea rădăcinii din Fig. 12.26. Sunt patru clase. Fotbalul este sportul preferat din 5/12 din țările care participă la antrenament date. Baseball și Hochei sunt fiecare clasa a 1/6 din exemplele de antrenament, în timp ce Cricket este clasa a 1/4 din exemplele de instruire. Impuritatea rădăcina conform măsurii de precizie este deci 1 - 5/12 = 7/12 = .583. Impuritatea GINI a rădăcinii este $1 - (1/6)^2 - (1/6)^2 - (1/4)^2 - (5/12)^2 =$ 103/144 = 0,715. Entropia rădăcinii este

$$\frac{1}{6}\log_2(6) + \frac{1}{6}\log_2(6) + \frac{1}{4}\log_2(4) + \frac{5}{12}\log_2(12/5) = 1{,}875$$

Rețineți faptul că aceste măsuri de impuritate au valori destul de diferite este neuniform portant. Aceste măsuri au diferite game de valori posibile; vezi Exercise 12.5.2.

Copilul stâng al rădăcinii este mult mai puțin impur. Este atins de cei șase țări din America de Sud și Europa, dintre care cinci le place fotbalul și unul îi place Cricket. Impuritatea acestui nod conform măsurii de precizie este astfel 1 - 5/6 = 1/6 = .167, în timp ce impuritatea GINI este $1 - (1/6)^2 - (5/6)^2 = 5/18 =$ .278. Entropia copilului stâng al rădăcinii este $\frac{1}{6}\log_2(6) + \frac{5}{6}\log_2(6/5) = .643$.

✷

12.5.3 Proiectarea unui nod arbore de decizie

Scopul proiectării nodurilor este de a produce copii a căror impuritate medie ponderată este cât mai mic posibil, unde ponderea copiilor este proporțională cu numărul de exemple de instruire care ajung la nod. În principiu, testul la un nod ar putea fi orice funcție a intrării. Așa cum este acest set de posibilități în esență infinit, trebuie să ne restrângem la teste simple la fiecare nod. În cele ce urmează, vom limita testele posibile la decizii binare bazate pe unul dintre cei doi factori:

1. Compararea unei caracteristici numerice a vectorului de intrare cu o constant.
2. Un test dacă o caracteristică categorică a vectorului de intrare se află într-un set de valorile posibile.

Exemplul 12.16: Observați că testele la copiii rădăcinii din Fig. 12.26 nu îndeplinesc condiția (1) pentru caracteristicile numerice. De exemplu, dreptul copilul rădăcinii este ȘI logic al a două comparații, populația> 55 și populație <200. Totuși, am putea folosi ambele condiții dacă le-am înlocui nodul unic cu două noduri, fiecare testând una dintre condiții. Radacina este un test pentru apartenența la valoarea caracteristicii Continent din setul de două continentele SA și Eur, deci îndeplinește condiția (2). ✷

Să presupunem că ni se dă un nod al arborelui decizional la care este atins un subset a exemplelor de instruire. Dacă nodul este pur, adică toate aceste exemple de instruire au aceeași ieșire, apoi facem nodul o frunză cu ieșirea respectivă ca valoare. Cu toate acestea, dacă impuritatea este mai mare decât zero, vrem să găsim testul care dă cea mai mare reducere a impurității. Când selectăm acest test, suntem liberi pentru a alege orice caracteristică a vectorului de intrare. Dacă alegem o caracteristică numerică, putem alege orice constantă pentru a împărți exemplele de instruire în două seturi, unul mergând la copilul stâng și celălalt mergând la copilul potrivit. Alternativ, dacă alegem o caracteristică categorică, apoi putem alege orice set de valori pentru test de membru. Vom lua în considerare fiecare dintre aceste cazuri pe rând.

12.5.4 Selectarea unui test folosind o caracteristică numerică

Să presupunem că vrem să împărțim un set de exemple de instruire pe baza unui numeric caracteristica A. În exemplul nostru curent, A ar putea fi doar Populație, una dintre cele două caracteristici - Populație și continent - care sunt componente ale vectorilor de caracteristici (reamintim presupunem că nu vedem numele țării; acel nume este folosit doar pentru identificați fiecare exemplu de instruire). Pentru a selecta cel mai bun punct de întrerupere, noi

1. Ordonați exemplele de instruire în funcție de valoarea lor de A. Lăsați valorile din A în această ordine să fie un_1, un_2, ..., un_n.
2. Pentru j = 1, 2, ..., n, calculați numărul de exemple de antrenament din fiecare clasă dintre un_1, un_2, ..., un_j. Rețineți că aceste numărări se pot face incremental, întrucât numărul pentru o clasă după al doilea exemplu este fie același cu numărați pentru j - 1 (dacă al doilea exemplu din această clasă) sau unul mai mult de numărul pentru j - 1 (dacă exemplul al j-lea este în această clasă).
3. Din numărul calculat în pasul anterior, calculați ponderat-impuritatea medie presupunând că testul trimite primele j exemple de antrenament la copilul din stânga și exemplele n - j rămase la nodul din dreapta.

Aici, trebuie să presupunem că impuritatea poate fi calculată din calculele pentru fiecare clasă. Acesta este cu siguranță cazul celor trei măsuri de impuritate - precizie, GINI și entropie - despre care am discutat în secțiunea 12.5.2.

4. Selectați acea valoare a lui j care minimizează impuritatea medie ponderată. Cum-vreodată, rețineți că nu orice valoare posibilă a lui j poate fi utilizată la acest pas, deoarece este posibil ca $a_j = a_{j+1}$. Trebuie să ne restrângem alegerea j la acestea

valori astfel încât $a_j < a_{j+1}$, astfel încât să putem folosi $A < (a_j + a_{j+1}) / 2$ ca comparație.

Exemplul 12.17: Să presupunem că folosim comparația rădăcinii din Fig. 12.26, care trimite cele șase țări din Europa și America de Sud către copilul stâng și celelalte șase spre dreapta. Acum, trebuie să împărțim cele șase țări care ajung copilul stâng în așa fel încât impuritatea ponderată a celor doi copii ai copilul stâng al rădăcinii este cât se poate de mic. Am putea folosi fie continentul sau Populație caracteristică pentru a face împărțirea și trebuie să le luăm în considerare pe ambele în ordine pentru a găsi scindarea care minimizează impuritatea. Aici, vom lua în considerare numai Populația, deoarece aceasta este singura caracteristică numerică. În Fig. 12.27 vedem șase țări care ajung la copilul stâng al rădăcinii, ordonate de populația lor.

Ctry.
Pop.
Sp.
n_s
n_c
$p_s \leq$
$p_c \leq$
$p_s >$
$p_c >$

Im≤
Sunt>
Wtd.
Arg.
44
S
1
0
1
0
4/5
1/5
0
25.08
15.04
Spania
46
S
2
0
1
0
3/4
1/4
0
3/8
1/4
Italia
59
S
3
0
1
0
2/3
1/3
0
4/9
2/9
Regatul Unit
65
C
3
1
3/4
1/4
1
0
3/8
0
1/4
Ger.
80
S
4
1
4/5
1/5
1
0
25.08
0
15.04
Sutien.
211
S
5
1
5/6
1/6
-
-
-
-
-
Figura 12.27: Calcularea indicelui GINI pentru fiecare posibilă separare

Coloanele explică calculul după cum urmează. În primul rând, coloana etichetată
„Sp” este sportul preferat, care pentru acest set de țări este fie Fotbal (S), fie
Cricket (C). Coloanele n_S și n_C sunt numărările cumulative ale numărului
de rânduri cu fotbal sportiv, respectiv Cricket. De exemplu, pe rând
pentru Germania, vedem $n_S = 4$, deoarece patru din cele cinci rânduri din Argentina în jos
în Germania au fotbal sportiv. În acel rând, avem și $n_C = 1$, doar din moment ce
unul dintre primele cinci rânduri are sport Cricket.
Următoarele două coloane, etichetate $p_{S\leq}$ și $p_{C\leq}$ sunt fracțiile rândurilor
până la și inclusiv rândul în cauză, care au fotbal sportiv și cricket,
respectiv. De exemplu, în rândul pentru Germania, patru din cele cinci rânduri din
Argentina către Germania au fotbal sportiv, deci $p_{S\leq} = 4/5$. Cele două coloane
după aceea, etichetate $p_{S>}$ și $p_{C>}$, sunt aceleași fracții, dar pentru rânduri
care se află sub rândul în cauză. De exemplu, în rândul pentru Spania vedem
$p_{S>} = 3/4$, deoarece trei din cele patru rânduri din Italia până în Brazilia au sport
Fotbal, iar unul dintre aceste rânduri are cricket sportiv. Rețineți că nu avem
pentru a face un număr cumulativ lucrând rândurile. Putem obține numărul 3 până la
scăderea $n_S = 2$ pentru Spania din valoarea inferioară a n_S (cea pentru Brazilia),
care este 5. În mod similar, știm că există un rând sub Spania cu sport Cricket
deoarece 1 este diferența dintre n_C pentru Brazilia și Spania.
Apoi, vedem coloanele pentru impurități, $Im_{\leq}$ și $Im_{>}$. Acestea sunt GINI
impuritățile copiilor din stânga și din dreapta nodului pe care le proiectăm, presupunând
că folosim o comparație „populație <c”, pentru unele c care se află între

---

populațiile pentru rândul în cauză și rândul următor. Astfel, în fiecare rând, noi
au $Im_{\leq} = 1 - (p_{S\leq})^2 - (p_{C\leq})^2$ și $Im_{>} = 1 - (p_{S>})^2 - (p_{C>})^2$. Pentru
de exemplu, în rândul pentru Spania, avem $Im_{\leq} = 1 - 1^2 - 0^2 = 0$ și
$Im_{>} = 1 - (3/4)^2 - (1/4)^2 = 3/8$
În cele din urmă, ultima coloană este impuritatea GINI ponderată pentru cei doi copii,
ponderat de numărul țărilor care ajung la fiecare dintre acești copii. Pentru
de exemplu, în rândul pentru Spania, dacă împărțim între populațiile din Spania și
următoarea țară mai populată, Italia, apoi două țări merg la copilul stâng și
patru la copilul potrivit. Astfel, impuritatea ponderată este (2/6) 0+ (4/6) (3/8) = 1/4.
Dintre cele cinci rânduri după care este posibilă o împărțire, îl găsim pe cel mai mic ponderat
impuritatea să fie 2/9, pe care o obținem prin împărțirea după Italia. Adică am folosi
un test ca „Populația <60” pentru a trimite Argentina, Spania și Italia, toate
ca fotbalul, în stânga. Celelalte trei țări, dintre care două le place fotbalul și
un Cricket, sunt trimise copilului potrivit. Deoarece copilul stâng este pur, este o frunză
declarând că fotbalul este sportul, în timp ce copilul potrivit va trebui împărțit din nou.
Din punct de vedere tehnic, trebuie să considerăm că testul la copilul stâng al
rădăcina nu implică deloc populația, ci mai degrabă este un test pe continent.
Cu toate acestea, nu putem face mai bine împărțind Europa de America de Sud; acea
oferă o impuritate GINI ponderată de 1/4. ✷

12.5.5 Selectarea unui test folosind o caracteristică categorică

Acum ia în considerare modul în care am putea împărți exemplele de instruire care ajung la un nod, folosind
valoarea unei caracteristici categorice A. Pentru a evita să ia în considerare tot ce este posibil
subseturi ale valorilor lui A, vom lua în considerare doar cazul în care există
două clase. În exemplul nostru de execuție, am putea presupune că cele două clase
sunt Fotbal (S) și orice altceva decât fotbal (N). Când sunt două clase, noi
poate ordona valorile lui A după fracția de exemple de antrenament cu acea valoare
aparținând clasei I.
Deoarece există doar două clase, partiția valorilor cu cea mai mică
impuritatea se va baza cu siguranță pe un set de valori care este un prefix al acestui ordin.
Adică, acest set constă din valorile lui A ale căror fracțiuni din prima clasă
sunt peste un prag. Astfel, exemplele care merg la copilul stâng vor avea
multe exemple în prima clasă, în timp ce exemplele care merg bine vor avea multe
în clasa a doua.
Procesul de găsire a divizării în lista ordonată de valori care minimizează
impuritatea ponderată este atunci în esență aceeași ca și pentru caracteristicile numerice,
pe care le-am explicat în secțiunea 12.5.4. Diferența este că acum mergem
în jos o listă de valori A, mai degrabă decât o listă de exemple de instruire. Vom
lucrează un exemplu, bazat pe fetura categorică Continent din alergarea noastră
exemplu.
Exemplul 12.18: Să vedem cum am putea proiecta rădăcina din Fig. 12.26, ca-
însumând că am recunoscut doar două clase: Fotbal (S) și alte sporturi (N).

La fel ca înainte, vom folosi GINI ca măsură a impurității pentru noduri. Figura 12.28 rezumă calculul nostru. În coloanele etichetate S și N, vedem numărul 12.25 pentru care sportul preferat este fotbalul și altceva. Continentele sunt ordonate în funcție de fracțiunea țărilor lor din clasa de Fotbal. Astfel, America de Sud este prima, deoarece 100% din pregătirea sa exemplele sunt în clasa S. Apoi vine Europa, unde 75% din pregătirea sa exemplele sunt în clasa S. Celelalte trei continente au 0% în fotbal și ar fi putut fi comandat în orice fel.
Cont.
S
N
n_S
n_N
$p_S \leq$
$p_N \leq$
$p_S >$
$p_N >$
Im≤
Sunt>
Wtd.
SA
2
0
2
0
1
0
3/10
7/10
0
21/50
7/20
Euro
3
1
5
1
5/6
1/6
0
1
18.05
0
5/36
N / A
0
3
5
4
5/9
4/9
0
1
40/81
0
27/10
Asia
0
2
5
6
5/11
6/11
0
1
60/121
0
5/11
Aus
0
1
5

7
5/12
7/12
-
-
-
-
-
Figura 12.28: Calcularea indicelui GINI pentru seturi de continente
Urmatoarele coloane din Fig. 12.28 sunt n_S și n_N . Acestea sunt cumulative
sumele numărului de exemple din clasa S și nu din clasa S, respectiv,
numărând de sus. De exemplu, rândul pentru America de Nord are $n_S = 5$
și $n_N = 4$, deoarece printre primele trei rânduri există cinci exemple în
clasa S și patru în clasa N. Reținești că, în ceea ce privește caracteristicile numerice, putem
calculați suma cumulativă printr-o singură trecere de sus în jos. De asemenea,
putem obține numerele din fiecare clasă dintre toate rândurile de mai jos prin scăderea
n_S sau n_N din valoarea sa corespunzătoare din rândul de jos.
Următoarele două coloane sunt fracțiile din fiecare clasă pentru acel rând și mai sus.
Adică, $p_S \leq = n_S / (n_S + n_N)$ și $p_N \leq = n_N / (n_S + n_N)$. După ce vin
cele două coloane $p_S >$ amd $p_N >$ care dau fracțiile din clasele S și
N dintre toate rândurile de mai jos. De exemplu, în rândul pentru NA, sunt zero
membrii clasei S de mai jos, pe care îi putem vedea deoarece $n_S = 5$ în ambele rânduri
pentru NA și rândul de jos. De asemenea, sunt 3 membri ai clasei N de mai jos,
deoarece valoarea lui n_N pentru rândul NA este 4, în timp ce valoarea lui n_N în rândul de jos
este 7.
Urmează coloanele Im≤ și Im>. La fel ca în Exemplul 12.17, acestea sunt GINI
impuritățile copiilor din stânga și din dreapta, asumând rândul în cauză și toate
rândurile de mai sus sunt trimise către copilul din stânga și toate rândurile de mai jos sunt trimise către copilul din dreapta.
Apoi, ultima coloană oferă impuritatea GINI ponderată a copiilor. Pentru
de exemplu, luați în considerare intrările din rândul pentru NA. Dacă folosim testul pentru a verifica dacă
Continent este unul dintre SA, Eur și NA pentru a trimite exemple copilului stâng, apoi nouă
dintre exemplele de instruire vor merge la stânga, iar restul de trei vor merge la dreapta. Prin urmare,
impuritatea GINI ponderată pentru această împărțire este (9/12) (40/81) + (3/12) 0 = 10/27.
De departe cea mai bună divizare vine după Eur, cu o impuritate ponderată de 5/36.
Aceasta a fost împărțirea pe care am folosit-o în Fig. 12.26. ✷

12.5.6 Proiectarea în paralel a arborilor de decizie

Proiectarea arborilor de decizie utilizând metodele descrise tocmai implică o problemă serioasă
cantitatea de calcul. Procedura trebuie aplicată fiecărui nod al
copac. Mai mult, în timp ce am descris ce să facem cu o caracteristică a
vector de intrare, același lucru trebuie făcut pentru fiecare caracteristică, după care alegem
cel mai bine împărțit între toate caracteristicile. Dacă caracteristica este numerică, trebuie să sortăm toate
exemple de instruire care ajung la nod. Dacă caracteristica este categorică, avem nevoie
pentru a grupa mai întâi exemplele de instruire după valoarea caracteristicii și apoi a sorta
valorile în funcție de fracțiunea de exemple care aparțin primei clase.
Mai mult, dacă există într-adevăr mai mult de două clase, trebuie să luăm în considerare toate modurile
să împartă clasele în două grupuri; aceste grupuri joacă apoi rolul celor două
cursuri în discuția secțiunii 12.5.5.
Cu toate acestea, pentru a accelera procesul, există o mulțime de paralelism ușor disponibil.

- La un nod, putem găsi cele mai bune împărțiri pentru toate caracteristicile în paralel.
- Toate nodurile de la un nivel pot fi proiectate în paralel. Mai mult, fiecare
exemplul de instruire atinge cel mult un nod la orice nivel. Un examen de instruire-
ple va ajunge exact la un nod la un nivel dat, cu excepția cazului în care ajunge la o frunză la
un nivel superior. Astfel, chiar și fără paralelism, ne așteptăm la munca totală
la fiecare nivel să fie aproximativ la fel, mai degrabă decât să crească cu numărul
de noduri.
- Gruparea exemplelor de instruire după valoarea unei caracteristici poate fi realizată eficient
în mod paralel. De exemplu, am discutat despre cum să faceți această sarcină folosind
MapReduce în secțiunea 2.3.8.
- Paralelismul poate accelera considerabil sortarea. În timp ce nu vom discuta
aici, există algoritmi cunoscuți care pot sorta n elemente în paralel în
$O(\log_2 n)$ trepte paralele în cel mai rău caz, sau $O(\log n)$ trepte paralele pornite
in medie.

Să presupunem acum că lucrăm la proiectarea unui nod folosind unul

caracteristica A, poate în paralel cu multe alte perechi nod-caracteristică în paralel.
După grupare (dacă A este nenumerică) și sortare, trebuie să calculăm mai multe
sume acumulate. De exemplu, dacă A este numeric, atunci trebuie să calculăm pentru
fiecare exemplu de antrenament și fiecare clasă numărul de exemple de antrenament cu acesta
clasă, printre acest exemplu de instruire și toate exemplele anterioare din lista sortată.
Calculul sumelor acumulate pare a fi inerent serial, dar ca
vom vedea, poate fi paralelizat destul de bine. Odată ce am acumulat
sumele, putem calcula fracțiile necesare și valorile impurității asociate
fiecare membru al listei, în paralel. Observați că calculele rând cu rând
sugerate în Exemplele 12.17 și 12.18 sunt toate independente și atât de deschise către a
implementare paralelă.
Pentru a completa povestea paralelizării pentru arborii de decizie, să vedem cum
pentru a calcula sumele acumulate în paralel. În mod formal, să presupunem că ni se dă o

---

Pagina 534

lista numerelor $a_1, a_2, ..., a_n$ și dorim să calculăm $x_i = \sum_{j=1}^{i} a_j$ pentru toți
i = 1, 2, ..., n. Ar putea părea că avem nevoie de n pași pentru a calcula toate x_i-urile,
și asta ar putea consuma timp când n este mare; adică există o pregătire amplă
a stabilit. Cu toate acestea, există un algoritm de împărțire și cucerire pentru a calcula toate x-urile
în pași paraleli O (log n), după cum urmează:
BAZĂ : Dacă n = 1, atunci $x_1 = a_1$. Baza necesită un pas paralel.
INDUCERE : Dacă n> 1, împărțiți lista lui într-o jumătate stângă și o jumătate dreaptă,
cât mai uniform posibil. Adică, dacă n este egal, jumătatea stângă este $a_1, a_2, ..., a_{n/2}$ și
jumătatea dreaptă este $a_{n/2+1}, a_{n/2+2}, ..., a_n$. Dacă n este impar, atunci jumătatea stângă se termină cu
$a_{\lfloor n/2 \rfloor}$ și jumătatea din dreapta începe cu un $\lceil n/2 \rceil$.
Jumătate la stânga
Sume acumulate
Sume acumulate
Jumătate dreaptă
Toate sumele acumulate
$n/2$
adăugați la fiecare
X
Figura 12.29: Pas recursiv în calculul paralel al sumelor acumulate
Pasul recursiv este sugerat de Fig. 12.29. În paralel, aplicăm acest lucru
algoritm la jumătățile stânga și dreapta. Odată ce am făcut acest lucru, ultimul
suma mulată a jumătății stângi, prezentată ca $x_{n/2}$, este apoi adăugată în paralel cu fiecare
din sumele acumulate din jumătatea dreaptă. Ca rezultat, dacă rezultatul ith în
jumătatea dreaptă este $\sum_{j=n/2+1}^{i} a_j$, prin adăugarea lui $x_{n/2}$, i-am schimbat valoarea în
corectul $\sum_{j=1}^{n/2+i} a_j$.
De fiecare dată când aplicăm pasul recursiv, avem nevoie de o singură adăugire, făcută în
paralel. Dacă n este o putere de 2, atunci fiecare aplicație a etapei recursive se împarte
dimensiunea listelor la care trebuie să lucrăm cu 2, deci numărul total de paralele
etapele sunt $1 + \log_2 n$. Dacă n nu este o putere de 2, de câte ori efectuăm
pasul recursiv nu este mai mare decât ceea ce am avea nevoie dacă n ar fi următorul mai mare
puterea de 2. Astfel, $1 + \lceil \log_2 n \rceil$ pași paraleli este suficient pentru orice n.

12.5.7 Tunderea nodului

Dacă proiectăm un arbore de decizie folosind cât mai multe niveluri necesare, astfel încât fiecare frunză
este pur, este posibil să avem un costum excesiv pentru datele de antrenament. Dacă putem verifica
proiectare folosind alte exemple - fie un set de testare pe care l-am reținut de la
date de instruire sau un set de date noi - avem posibilitatea de a simplifica
copac și în același timp limitează supra-amenajarea.

Pagina 535

Găsiți un nod N care are doar frunze ca și copii. Construiți un copac nou prin înlocuire
N și copiii săi de o frunză, și oferă acelei frunze clasa sa majoritară ca rezultat.
Apoi, comparați performanța arborilor vechi și noi pe date care nu erau

folosit ca exemple de antrenament atunci când am proiectat arborele. Dacă există puțină diferență
între ratele de eroare ale copacilor vechi și noi, apoi decizia luată la
nodul N a contribuit probabil la supra-amenajarea și nu se adresa unei proprietăți
a întregului set de exemple pentru care a fost destinat arborele decizional. Noi
poate arunca arborele vechi și îl poate înlocui cu arborele nou, mai simplu. Pe de altă parte
mână, dacă arborele nou are o rată de eroare semnificativ mai mare decât vechiul, atunci
decizia luată la nodul N reflectă într-adevăr o proprietate a datelor și avem nevoie
pentru a păstra copacul vechi și a arunca noul. În ambele cazuri, ar trebui să continuăm
privind alte noduri ai căror copii sunt frunze și vezi dacă acestea pot fi înlocuite
prin frunze fără o creștere semnificativă a ratei de eroare.
Exemplul 12.19: Să luăm în considerare nodul din Fig. 12.26 cu testul „Conti-
nent = NA. ” Are două frunze ca copii și astfel poate juca rolul de nod
N deasupra. În datele de antrenament din Fig. 12.25 N este atins de trei antrenamente
exemple, Cuba, SUA și Australia. Pe măsură ce doi dintre acești trei merg la frunză
etichetat „Baseball”, vom lua în considerare înlocuirea lui N cu o frunză etichetată „Baseball”.
Luați în considerare ce se întâmplă dacă copacii vechi și noi sunt aplicați întregului
ansamblu de țări ale lumii. În primul rând, rețineți că pentru a ajunge la nodul N, o țară
trebuie să fie pe unul dintre continentele Americii de Nord, Asia, Australia sau Africa.
Mai mult, trebuie să fie fie o țară mică (mai puțin de 35 de milioane de locuitori)
sau o țară populată (peste 200 de milioane). Există multe astfel de țări,
cum ar fi țările mici din Africa sau țările mari precum China sau Indonezia, niciuna
dintre care au cricketul sau baseballul ca sport preferat. Dacă luăm în considerare
țările din Caraibe și America Centrală ca aparținând Americii de Nord,
atunci N este atins de multe alte țări, dintre care doar câteva au baseball
ca sportul lor preferat.
Concluzia este că indiferent dacă N este înlocuit sau nu cu o frunză „Baseball”,
arborele va avea o rată de eroare semnificativă în aceste țări și, prin urmare,
rata de eroare a celor doi copaci aplicați tuturor țărilor va fi aproximativ aceeași. Noi
concluzionează că acest nod N reflectă doar artefacte ale setului mic și atipic
din douăsprezece exemple de instruire pe care le-am ales. Astfel, N poate fi înlocuit în siguranță cu
o frunză fără prea mare creștere a ratei de eroare pe întregul set de țări. ✷

12.5.8 Păduri de decizie

Deoarece un singur arbore de decizie cu mai multe niveluri are probabil multe noduri la
nivelurile inferioare care reprezintă supraadaptarea, există o altă abordare a utilizării
de arbori de decizie care s-au dovedit destul de utili în practică. Este obișnuit să se utilizeze
păduri de decizie de mulți copaci care votează clasa către care se referă un anumit dat
apartine. Fiecare copac din pădure este proiectat folosind aleatoriu sau sistematic
caracteristicile alese și este limitată doar la un număr mic de niveluri, de multe ori unul sau
Două. Astfel, fiecare copac are o impuritate ridicată la fiecare dintre frunzele sale, dar colectiv

---

Metode de ansamblu

Pădurile de decizie nu sunt decât un exemplu de strategie importantă de luare
decizii bune. Putem folosi mai mulți algoritmi diferiți pentru a face același lucru
decizie. Apoi, combinăm opiniile diferiților algoritmi în
într-un fel, poate învățând cele mai bune greutăți așa cum am sugerat în secțiunea
secțiunea 12.5.8. În cazul unei păduri de decizie, toți algoritmii care contribuie
sunt de același tip: un arbore de decizie. Cu toate acestea, algoritmii de decizie
poate fi, de asemenea, de diferite feluri. De exemplu, soluția câștigătoare la
Provocarea Netflix (a se vedea secțiunea 9.5) a fost de acest tip, unde mai multe
Au fost aplicate fiecare tehnici de învățare automată și evaluarea filmelor
estimările combinate pentru a lua o decizie mai bună.
de multe ori se descurcă mult mai bine la datele testelor decât oricare arborele, oricât de multe niveluri ar fi
are. Mai mult, putem proiecta fiecare dintre copacii din pădure în paralel, așa că
poate fi chiar mai rapid pentru a proiecta o colecție mare de copaci puțin adânci decât pentru a proiecta
un copac adânc.
Modul evident de a combina rezultatele tuturor copacilor într-o decizie
pădurea trebuie să ia un vot majoritar. Dacă există mai mult de două clase, atunci noi
așteptați doar o pluralitate de voturi pentru una dintre clase, în timp ce nici o clasă nu poate fi cea
alegerea a mai mult de jumătate din copaci. Cu toate acestea, modalități mai complexe de combinare
rezultatele copacilor pot da un răspuns mai precis decât votul simplu.
De multe ori putem „învăța” greutățile adecvate pentru a le aplica opiniilor fiecărui copac.
De exemplu, să presupunem că avem un set de antrenament $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$
și o colecție de arbori de decizie $T_1, T_2, ..., T_k$ care constituie decizia noastră
pădure. Când aplicăm pădurea de decizie la unul dintre exemplele de instruire
(x_i, y_i), obținem un vector de clase $c_i = [c_{i1}, c_{i2}, ..., c_{ik}]$, unde c_{ij} este rezultatul

a arborelui T_j aplicat intrării x_i. Știm clasa corectă pentru intrarea x_i; este y_i. Astfel, avem un nou set de antrenament $(c_1, y_1), (c_2, y_2), ..., (c_n, y_n)$, pe care îl putem folosiți pentru a prezice clasa adevărată din opiniile tuturor copacilor din pădure. Pentru de exemplu, am putea folosi acest set de instruire pentru a instrui un perceptron sau SVM. Facand deci, punem greutățile potrivite pe părerea fiecăruia dintre copaci, pentru a combină opiniile lor în mod optim.

12.5.9 Exerciții pentru secțiunea 12.5

Exercițiul 12.5.1: Să presupunem că un set de antrenament are exemple în patru clase și fracțiile de exemple din aceste clase sunt 1/2, 1/3, 1/8 și 1/24. Ce este impuritatea rădăcinii unui arbore de decizie conceput pentru acest set de instruire, dacă măsura impurității este (a) Precizia (b) GINI (c) Entropia?

Exercițiul 12.5.2: Dacă un set de date constă din exemple aparținând lui n diferite clase, care este impuritatea maximă dacă măsura impurității este (a)

---

racy (b) GINI (c) Entropie?

! Exercițiul 12.5.3: O proprietate importantă a unei funcții f este convexitatea, adică că dacă x <z <y, atunci

f (z)> z - x
y - x
f (x) + y - z
y - x
f (y)

Mai puțin formal, curba lui f între x și y se află deasupra liniei drepte între punctele (x, f (x)) și (y, f (y)). În cele ce urmează, presupuneți că există două clasele, și f (x) este impuritatea când x este fracția de exemple din prima clasă.

(a) Dovediți că impuritatea GINI este convexă.

(b) Dovediți că măsura Entropiei impurității este convexă.

(c) Dați un exemplu pentru a arăta că măsura de precizie a impurității nu este mereu convex. Sugestie: Rețineți că convexitatea necesită inegalități stricte; A linia dreaptă nu este convexă.

Exercițiul 12.5.4: Pentru a vedea de ce este importantă convexitatea, repetați calculul lui Fig. 12.27, dar folosind Precizia ca măsură de impuritate. Ce merge prost?

Exercițiul 12.5.5: Exemplul continuu 12.19, să presupunem că am înlocuit nod etichetat „Continent = NA” de o frunză etichetată „Baseball”. Care alta nodurile interioare credeți că pot fi înlocuite cu frunze fără un semn semnificativ crește la rata de eroare atunci când arborele este aplicat tuturor țărilor lumii?

12.6 Compararea metodelor de învățare

Fiecare dintre metodele discutate în acest capitol și în alte părți are avantajele sale. În această secțiune de închidere, vom lua în considerare:

- Metoda se referă la caracteristici categorice sau numai la numerice Caracteristici?
- Metoda se ocupă în mod eficient de vectori de caracteristici de înaltă dimensiune?
- Este ușor de înțeles modelul pe care îl construiește metoda?

Perceptroni și mașini vector-suport: Aceste metode pot fi manipulate milioane de caracteristici, dar au sens numai dacă caracteristicile sunt numerice. Sunt eficiente numai dacă există un separator liniar sau cel puțin un hiperplan care separă aproximativ clasele. Cu toate acestea, putem separa punctele prin o delimitare neliniară dacă mai întâi transformăm punctele pentru a face separatorul fie liniar. Modelul este exprimat printr-un vector, normal la separare

---

hiperplan. Deoarece acest vector are adesea o dimensiune foarte mare, poate fi foarte greu de interpretat modelul.

Clasificarea și regresia celui mai apropiat-vecin: Aici, modelul este instruirea sa stabilit, așa că ne așteptăm să fie ușor de înțeles. Ap-proach se poate ocupa de date multidimensionale, deși cu cât este mai mare numărul de dimensiuni, cu atât setul de antrenament va fi mai rar și, prin urmare, cu atât este mai puțin probabil este că vom găsi un punct de antrenament foarte aproape de punctul pe care trebuie să îl clasificăm. Adică, „blestemul dimensionalității” face ca metodele cele mai apropiate să fie

reglabil în dimensiuni ridicate. Aceste metode sunt cu adevărat utile numai pentru numerice
, deși s-ar putea permite caracteristici categorice cu un număr mic de
valori. De exemplu, o caracteristică categorică binară, cum ar fi {masculin, feminin}, ar putea
au valorile înlocuite cu 0 și 1, deci nu a existat distanță în această dimensiune
între indivizi de același gen și distanța 1 între alte perechi de
indivizi. Cu toate acestea, trei sau mai multe valori nu pot fi atribuite numerelor care sunt
echidistant. În cele din urmă, metodele cu cel mai apropiat vecin au mulți parametri de setat,
inclusiv măsura distanței pe care o folosim (de exemplu, cosinus sau euclidian), numărul de
vecinii să aleagă și funcția kernel de utilizat. Diferite alegeri rezultă
clasificare diferită și, în multe cazuri, nu este evident ce alegeri produc
cele mai bune rezultate.
Copaci de decizie: spre deosebire de alte metode discutate în acest capitol, Decizie
copacii sunt utili atât pentru caracteristicile categorice, cât și pentru cele numerice. Modelele pro-
sunt în general destul de ușor de înțeles, deoarece fiecare decizie este reprezentată de
un nod al copacului. Cu toate acestea, această abordare este cea mai utilă pentru dimensiunile reduse
vectori de caracteristici. După cum sa discutat în secțiunea 12.5.7, construirea arborilor de decizie cu
multe niveluri duc adesea la supra-dotare. Dar dacă un arbore de decizie are puține niveluri, atunci
nici măcar nu poate menționa mai mult decât un număr mic de caracteristici. Ca urmare,
Cea mai bună utilizare a copacilor de decizie este adesea crearea unei păduri de decizie de multe adâncimi mici
copaci și combină decizia lor într-un fel.

12.7 Rezumatul capitolului 12

♦ Seturi de antrenament: un set de antrenament constă dintr-un vector de caracteristici, fiecare componentă
dintre care este o caracteristică și o etichetă care indică clasa la care obiectul
reprezentat de vectorul de caracteristică aparține. Funcțiile pot fi categorice -
aparținând unei liste enumerate de valori - sau numerice.

♦ Seturi de testare și supra-montare: atunci când antrenezi un clasificator pe un set de antrenament,
este util să eliminați o parte din setul de antrenament și să utilizați datele eliminate
ca set de testare. După producerea unui model sau clasificator fără a utiliza testul
set, putem rula clasificatorul pe setul de testare pentru a vedea cât de bine merge. Dacă
clasificatorul nu funcționează la fel de bine pe setul de testare ca pe setul de antrenament
folosit, atunci ne-am îmbrăcat în setul de antrenament conform cu particularitățile
a datelor setului de instruire care nu sunt prezente în datele în ansamblu.

♦ Învățarea on-line prin loturi: În învățarea prin loturi, setul de instruire este
disponibil oricând și poate fi utilizat în treceri repetate. Învățare on-line
folosește un flux de exemple de instruire, fiecare dintre acestea putând fi utilizat o singură dată.

♦ Perceptroni: Această metodă de învățare automată presupune că setul de antrenament are
doar două etichete de clasă, pozitive și negative. Perceptronii funcționează atunci când sunt acolo
este un hiperplan care separă vectorii caracteristici ale exemplelor pozitive
din cele ale exemplelor negative. Convergem la acel hiperplan prin
ajustarea estimării noastre pentru hiperplan cu o fracțiune - rata de învățare -
a direcției care este media punctelor clasificate greșit în prezent.

♦ Algoritmul Winnow: Acest algoritm este o variantă a perceptronului
algoritm care necesită ca componentele vectorilor de caracteristici să fie 0 sau 1.
Exemplele de instruire sunt examinate într-un mod round-robin și dacă
clasificarea chiriei unui exemplu de instruire este incorectă, componentele
separatorul estimat în care vectorul caracteristică are 1 este ajustat în sus sau
în jos, în direcția care va face mai probabil acest exemplu de antrenament
este corect clasificat în runda următoare.

♦ Separatoare neliniare: Când punctele de antrenament nu au o funcție liniară
care separă două clase, este încă posibil să se utilizeze un perceptron
pentru a le clasifica. Trebuie să găsim o funcție pe care o putem folosi pentru a transforma
puncte astfel încât în spațiul transformat, separatorul să fie un hiperplan.

♦ Suport-mașini vectoriale: SVM îmbunătățește perceptronii prin găsirea-
un hiperplan de separare care nu doar separă pozitivul și negativul
puncte tive, dar o face într-un mod care maximizează marja - distanța
perpendicular pe hiperplan spre cele mai apropiate puncte. Punctele care
se află exact la această distanță minimă sunt vectorii de susținere. Alterna-
tiv, SVM poate fi proiectat pentru a permite puncte prea apropiate de
hiperplan sau chiar pe partea greșită a hiperplanului, dar minimizați
eroarea datorată unor astfel de puncte deplasate.

♦ Rezolvarea ecuațiilor SVM: Putem seta o funcție a vectorului care

este normal pentru hiperplan, lungimea vectorului (care determină marja) și penalizarea pentru puncte de pe partea greșită a marginilor. Parametrul de regularizare determină importanța relativă a unui larg marjă și o mică penalizare. Ecuațiile pot fi rezolvate prin mai multe metode, inclusiv coborârea în gradient și programarea pătratică.

♦ Coborâre în gradient: Aceasta este o metodă pentru minimizarea unei funcții de pierdere care depinde de multe variabile și de un exemplu de antrenament, prin găsirea repetată (derivatul) funcției de pierdere față de fiecare variabilă atunci când dat un exemplu de antrenament și mutarea valorii fiecărei variabile în direcție care scade pierderea. Modificarea variabilelor poate fi fie acumularea modificărilor sugerate de fiecare exemplu de antrenament (lot descendenta gradientului), rezultatul unui singur exemplu de antrenament (gradient stochastic

---

descendență) sau rezultatul utilizării unui subset mic de exemple de instruire (coborârea gradientului minibatch).

♦ Învățarea cea mai apropiată-vecină: În această abordare a învățării automate, ca model este folosit întregul set de antrenament. Pentru fiecare punct („interogare”) să fie clasificate, căutăm cei mai apropiați vecini ai săi din setul de antrenament. The clasificarea punctului de interogare este o funcție a etichetelor acestor k vecini. Cel mai simplu caz este atunci când k = 1, caz în care putem lua eticheta punctului de interogare să fie eticheta celui mai apropiat vecin.

♦ Regresie: un caz obișnuit de învățare cu cel mai apropiat vecin, numit regres-siune, apare atunci când există un singur vector de caracteristică și acesta, precum și eticheta, sunt numere reale; adică datele definesc o funcție cu valoare reală a o variabilă. Pentru a estima eticheta, adică valoarea funcției, pentru un punct de date neetichetat, putem efectua unele calcule care implică k cei mai apropiați vecini. Exemple includ medierea vecinilor sau luarea unui medie ponderată, unde greutatea unui vecin este o funcție descrescătoare ziunea distanței sale față de punctul a cărui etichetă încercăm să o determinăm.

♦ Arbori de decizie: Această metodă de învățare construiește un copac în care fiecare inte-nodul rior susține un test despre intrare și ne trimite la unul dintre copiii săi în funcție de rezultatul testului. Fiecare frunză oferă o decizie despre clasa căreia îi aparține intrarea.

♦ Măsuri de impuritate: Pentru a ajuta la proiectarea unui arbore de decizie, avem nevoie de o măsură despre cât de pur, adică, aproape de o singură clasă, este setul de exemple de instruire care ajung la un anumit nod al arborelui decizional. Posibile măsuri de impuritatea include Acuratețea (fracțiunea de exemple de instruire cu greșeală clasă), GINI (1 minus pătratele fracțiilor de exemple din fiecare dintre clasele) și Entropy (suma fracțiilor de exemple de instruire din fiecare clasă de ori logaritmul inversului acelei fracții).

♦ Proiectarea unui nod arbore de decizie: trebuie să luăm în considerare fiecare caracteristică posibilă de testat la un nod și trebuie să rupem setul de examene de formare-plesuri care ajung la nod într-un mod care minimizează impuritatea medie a copiii săi. Pentru o caracteristică numerică, putem comanda exemplele traning prin valoarea acelei caracteristici și utilizați un test care rupe într-un fel această listă care minimizează impuritatea medie. Pentru o caracteristică categorică comandăm valorile acelei caracteristici prin fracțiunea de exemple de instruire cu acea valoare care aparțin unei anumite clase și rupe lista pentru a minimiza media impuritate.

12.8 Referințe pentru capitolul 12

Perceptronul a fost introdus în [14]. [8] introduce ideea de maximizare a marginea în jurul hiperplanului de separare. O carte cunoscută pe această temă este [12].

Algoritmul Winnow este de la [11]. A se vedea, de asemenea, analiza în [2]. Mașinile de suport vector au apărut în [7]. [6] și [5] sunt sondaje utile. [10] vorbește despre un algoritm mai eficient pentru cazul caracteristicilor rare (majoritatea elementele vectorilor caracteristici sunt zero). Utilizarea metodelor gradient-coborâre

se găsește în [3, 4].
Pentru informații despre structuri de indici cu dimensiuni ridicate pentru cel mai apropiat vecin
învățare, vezi capitolul 14 din [9].
Lucrarea inițială privind construcția arborilor de decizie este [13]. [1] este popular
lucrare care descrie metodologia utilizată în acest capitol.

1. H. Blockeel și L. De Raedt, „Inducerea de sus în jos a logicii de ordinul întâi
copaci de decizie, ”Inteligență artificială 101: 1-2 (1998), pp. 285-297.
2. A. Blum, „Suport empiric pentru ceva winnow și majoritar ponderat-
rithms: rezultate pe un domeniu de planificare calendaristică, ”Machine Learning 26
(1997), pp. 5-23.
3. L. Bottou, „Învățarea automată pe scară largă cu gradient stocastic de-
parfum, ”Proc. 19 Intl. Conf. on Computational Statistics (2010), pp. 177–
187, Springer.
4. L. Bottou, „Trucuri de gradient stochastic, rețele neuronale”, în Tricks of the
Trade, Reloaded, pp. 430-445, editat de G. Montavon, GB Orr și K.-
R. Mueller, Note de curs în informatică (LNCS 7700), Springer,
2012.
5. CJC Burges, „Un tutorial privind mașinile vectoriale de suport pentru recunoașterea modelelor
nition, ”Data Mining and Knowledge Discovery 2 (1998), pp. 121–167.
6. N. Cristianini și J. Shawe-Taylor, An Introduction to Support Vector
Mașini și alte metode de învățare bazate pe kernel, Cambridge University-
sity Press, 2000.
7. C. Cortes și VN Vapnik, „Rețele vector-suport”, Machine Learn-
ing 20 (1995), pp. 273–297.
8. Y. Freund și RE Schapire, „Clasificare marjă mare folosind per-
algoritm ceptron, ”Machine Learning 37 (1999), pp. 277–296.
9. H. Garcia-Molina, JD Ullman și J. Widom, Sisteme de baze de date:
Carte completă, Prentice Hall, Upper Saddle River NJ, 2009.
10. T. Joachims, „Antrenarea SVM-urilor liniare în timp liniar”. Proc. Al 12-lea ACM
SIGKDD (2006), pp. 217-226.
11. N. Littlestone, „Învățarea rapidă atunci când atributele irelevante abundă: a
nou algoritm cu prag liniar, ”Machine Learning 2 (1988), pp. 285–
318.

Pagina 542

12. M. Minsky și S. Papert, Perceptrons: Introducere în calcul
Geometrie (ediția a II-a), MIT Press, Cambridge MA, 1972.
13. JR Quinlan, „Inducerea arborilor de decizie”, Machine Learning 1 (1986),
pp. 81–106.
14. F. Rosenblatt, „Perceptronul: un model probabilistic pentru informații
stocare și organizare în creier ", Psychological Review 65: 6 (1958),
pp. 386-408.

Pagina 543

Capitolul 13

Plase neuronale și adâncime Învățare

În secțiunile 12.2 și 12.3 am discutat despre proiectarea „neuronilor” unici (percepție)
troni). Acestea iau o colecție de intrări și, pe baza ponderilor asociate
aceste intrări, calculează un număr pe care, în comparație cu un prag, îl determină
dacă se afișează „da” sau „nu”. Aceste metode ne permit să separăm intrările
în două clase, atâta timp cât clasele sunt separabile liniar. Cu toate acestea, majoritatea
problemele de interes și importanță nu sunt separabile liniar. In acest capitol,
vom lua în considerare proiectarea rețelelor neuronale, care sunt colecții de perceptroni,
sau noduri, unde ieșirile unui rang (sau strat de noduri) devin intrări
la noduri de la următorul strat. Ultimul strat de noduri produce ieșirile de
întreaga rețea neuronală. Antrenarea rețelelor neuronale cu multe straturi necesită
un număr enorm de exemple de instruire, dar s-a dovedit a fi extrem
tehnică puternică, denumită învățare profundă, atunci când poate fi utilizată.

De asemenea, luăm în considerare mai multe forme specializate de plase neuronale care s-au dovedit util pentru tipuri speciale de date. Aceste forme se caracterizează prin faptul că necesită acest lucru anumite seturi de noduri din rețea împart aceleași greutăți. De când am învățat totul greutatea tuturor intrărilor către toate nodurile rețelei este în general a sarcină dificilă și consumatoare de timp, aceste forme speciale de rețea simplifică foarte mult procesul de instruire a rețelei pentru a recunoaște clasa sau clasele dorite de intrări. Vom studia rețelele neuronale convoluționale (CNN), care sunt concepute în mod special pentru a recunoaște clasele de imagini. De asemenea, vom studia necesitățile recurente rețele ral (RNN) și rețele de memorie pe termen scurt (LSTM), care sunt concepute pentru a recunoaște clase de secvențe, cum ar fi propozițiile (secvențe de cuvinte).
523

13.1 Introducere în plasele neuronale

Începem discuția despre rețelele neuronale cu un exemplu extins. După care, introducem planul general al unei rețele neuronale și câteva terminologii importante.

Exemplul 13.1: Problema pe care o discutăm este să învățăm conceptul că „bine” vectorii de biți sunt cei care au două 1 consecutive. Din moment ce vrem să ne ocupăm doar exemple mici, vom presupune că vectorii de biți au lungimea patru. Al nostru exemplele de instruire vor avea astfel forma ([x_1, x_2, x_3, x_4], y), unde fiecare dintre x_i sunt biți, 0 sau 1. Există 16 exemple posibile de instruire și vom face presupunem că ni se oferă niște subseturi ca set de pregătire. Observați că opt dintre vectorii de biți posibili sunt buni - au 1 consecutive, și acolo sunt, de asemenea, opt exemple „rele”. De exemplu, 0111 și 1100 sunt bune; 1001 și 0100 sunt rele. [1]

Pentru început, să ne uităm la o rețea neuronală care rezolvă exact această problemă simplă. Modul în care am putea proiecta această rețea din exemple de instruire este adevăratul subiect discuție, dar această rețea va servi drept exemplu pentru ceea ce ne-am dori obține. Plasa este prezentată în Fig. 13.1.

x 1
x 2
x 3
x 4
x 1
x 2
x 3
x 4
0,5
y
1
1
2.5
1.5
1
1
2
0
0
0
1
1

Figura 13.1: O rețea neuronală care spune dacă un vector bit are 1 consecutive

Plasa are două straturi, primul constând din două noduri, iar al doilea cu un singur nod care produce ieșirea y. Fiecare nod este un perceptron, exact așa cum a fost descris în secțiunea 12.2. În primul strat, primul nod este caracterizat prin vectorul de greutate [w_1, w_2, w_3, w_4] = [1, 2, 1, 0] și pragul 2,5. Deoarece fiecare intrare x_i este fie 0, fie 1, observăm că singura modalitate de a ajunge la o sumă $\sum_{i=1}^{4} x_i w_i$ cât 2,5 este dacă x_2 = 1 și cel puțin unul dintre x_1 și x_3 este, de asemenea, 1. Ieșirea acestui nod este 1 dacă și numai dacă intrarea este una dintre 1100, 1101, 1110, 1111, 0110 sau 0111. Adică recunoaște acei vectori de biți care încep fie cu două 1 sau au două 1 în mijloc. Singurele intrări bune pe care nu le face

[1] Vom arăta vectori de biți ca șiruri de biți în cele ce urmează, astfel încât să putem evita virgulele între componente, fiecare dintre ele fiind 0 sau 1.

---

recunoaște sunt cele care se termină cu 11, dar nu au 11 în altă parte. acestea sunt 0011 și 1011.

Din fericire, al doilea nod din primul strat, cu greutăți [0, 0, 1, 1] și pragul 1.5 dă ieșirea 1 ori de câte ori $x_3 = x_4 = 1$, și nu altfel. Acest nodul recunoaște astfel intrările 0011 și 1011, precum și alte bunuri intrări care sunt recunoscute și de primul nod.

Acum, să ne întoarcem la al doilea strat, cu un singur nod; acel nod are greutățile [1, 1] și pragul 0,5. Se comportă astfel ca o „poartă SAU". Da ieșire y = 1 ori de câte ori unul sau ambele noduri din primul strat au ieșire 1, dar dă ieșirea y = 0 dacă ambele noduri din primul strat dau ieșirea 0. Astfel, rețeaua neuronală din Fig. 13.1 oferă ieșirea 1 pentru toate intrările bune, dar nici una intrările proaste. ✷

x_1
x_2
x_3
x_4
x_1
x_2
x_3
x_4
y
1
1
1
2
0
0
0
1
1
1
1
−0,5
1
−2,5
1
−1,5

Figura 13.2: Realizarea pragului 0 pentru toate nodurile

Este util în multe situații să presupunem că nodurile au un prag de 0. Reamintim din secțiunea 12.2.4 că putem converti întotdeauna un perceptron cu un prag diferit de zero la unul cu un prag 0 dacă adăugăm o intrare suplimentară. Această intrare are întotdeauna valoarea 1 și o greutate egală cu −t. De exemplu, putem convertiți rețeaua din Fig. 13.1 la cea din Fig. 13.2.

13.1.1 Rețelele neuronale, în general

Exemplul 13.1 și rețeaua sa din Fig. 13.1 sunt mult mai simple decât orice ar fi să fie o aplicație utilă a rețelelor neuronale. Cazul general este sugerat de Fig. 13.3. Primul, sau stratul de intrare, este intrarea, care se presupune a fi un vector al unora lungime n. Fiecare componentă a vectorului $[x_1, x_2, ..., x_n]$ este o intrare în rețea. Există unul sau mai multe straturi ascunse și, în cele din urmă, la final, un strat de ieșire, care dă rezultatul netului. Fiecare dintre straturi poate avea un număr diferit de noduri și, de fapt, alegerea numărului potrivit de noduri la fiecare strat este un

o parte importantă a procesului de proiectare pentru plasele neuronale. Mai ales, rețineți că stratul de ieșire poate avea mai multe noduri. De exemplu, rețeaua neuronală ar putea clasifica intrări în mai multe clase diferite, cu un nod de ieșire pentru fiecare clasă.

.
.
.
.
.
.
.
.

.
. . .
X
Straturi ascunse
Ieșire
strat
X
1
2
.
.
.
x_n
y
y
y_1
2
k
Intrare
strat
Figura 13.3: Cazul general al unei rețele neuronale
Fiecare strat, cu excepția stratului de intrare, este format din unul sau mai multe noduri, care aranjăm în coloana care reprezintă acel strat. Ne putem gândi la fiecare nod ca perceptron. Intrările către un nod sunt ieșiri ale unora sau ale tuturor nodurilor din stratul anterior. Astfel încât să putem presupune că pragul pentru fiecare nod este zero, putem permite, de asemenea, unui nod să aibă o intrare care este o constantă, de obicei 1, ca am sugerat în Fig. 13.2. O greutate este asociată cu fiecare intrare pentru fiecare nod. Ieșirea nodului depinde de $\Sigma x_i w_i$, unde suma este peste toate intrările x_i și w_i este greutatea acelei intrări. Uneori, ieșirea este fie 0, fie 1; ieșirea este 1 dacă suma respectivă este pozitivă și 0 în caz contrar. Totuși, așa cum vom face vezi în secțiunea 13.2, este adesea convenabil atunci când încerci să înveți ponderile pentru o rețea neuronală care rezolvă unele probleme, pentru a avea ieșiri care sunt aproape întotdeauna aproape de 0 sau 1, dar poate fi ușor diferit. Motivul, intuitiv, este acela atunci este posibil ca ieșirea unui nod să fie o funcție continuă a acestuia intrări. Putem folosi apoi coborârea în gradient pentru a converge la valorile ideale ale tuturor greutățile din plasă.

Pagina 547

13.1.2 Interconectări între noduri

Rețelele neuronale pot diferi în ceea ce privește modul în care nodurile dintr-un strat sunt conectate la noduri la stratul din dreapta lui. Cel mai general caz este când fiecare nod primește ca intră ieșirile fiecărui nod al stratului anterior. Un strat care primește se spune că toate ieșirile din stratul anterior sunt complet conectate. Altele opțiunile pentru alegerea interconectărilor sunt:

1. Aleatoriu. Pentru unii m, alegem pentru fiecare nod m noduri din precedent strat și face ca acestea, și numai acelea, să fie intrări în acest nod.
2. Împreună. Împărțiți nodurile unui strat într-un anumit număr de clustere. În următorul strat, care se numește strat de pooling, există un nod pentru fiecare cluster, iar acest nod are tot și numai membrul clusterului său ca intrări.
3. Convolutional. Această abordare a interconectării, pe care o discutăm în mai multe detalii în secțiunea următoare și secțiunea 13.4, vizualizează nodurile fiecăruia strat așezat într-o grilă, de obicei bidimensională. Într-un convolutional strat, un nod corespunzător coordonatelor (i, j) primesc ca intrări noduri ale stratului anterior care au coordonate într-o regiune mică în jurul (i, j). De exemplu, nodul (i, j) la un strat convoluțional poate au ca intrări acele noduri din stratul anterior care corespund coordonate (p, q), unde $i \le p \le i + 2$ și $j \le q \le j + 2$ (adică pătratul a laturii 3 al cărei colț stânga jos este punctul (i, j).

13.1.3 Rețele neuronale convoluționale

O rețea neuronală convoluțională sau CNN conține una sau mai multe convoluționale straturi. De asemenea, pot exista straturi neconvenționale, cum ar fi straturile complet conectate și straturi grupate. Cu toate acestea, există o constrângere suplimentară importantă: greutățile la intrări trebuie să fie aceleași pentru toate nodurile unei singure convoluționale strat. Mai exact, să presupunem că fiecare nod (i, j) dintr-un strat convoluțional primește (i + u, j + v) ca una dintre intrările sale, unde u și v sunt constante mici. Apoi, există o greutate w asociată cu u și v (dar nu cu i și j). Pentru orice i și j, greutatea de pe intrarea în nod pentru (i, j) provenind de la ieșirea nodului i + u, j + v) din stratul anterior trebuie să fie w.

Această restricție face ca formarea unui CNN să fie mult mai eficientă decât antrenarea unui
rețea neuronală generală. Motivul este că există mult mai puțini parametri la fiecare
strat și, prin urmare, pot fi utilizate mai puține exemple de instruire decât dacă fiecare
nod sau fiecare strat are propriile greutăți pe care le poate descoperi procesul de instruire.
CNN s-au găsit extrem de utile pentru sarcini precum recunoașterea imaginii.
De fapt, CNN se inspiră din modul în care procesează ochiul uman
imagini. Neuronii ochiului sunt aranjați în straturi, în mod similar cu straturile
a unei plase neuronale. Primul strat preia intrări care sunt în esență pixeli ai
imagine, fiecare pixel rezultatul unui senzor din retină. Nodurile primului
stratul recunoaște caracteristici foarte simple, cum ar fi marginile dintre lumină și întuneric.
Observați că un pătrat mic de pixeli, să zicem 3 × 3, ar putea prezenta o margine la a

---

Pagina 548

unghi special, de exemplu, dacă colțul din stânga sus este ușor și ceilalți opt pixeli
întuneric. Mai mult, algoritmul pentru recunoașterea unei margini de un anumit tip este
la fel, indiferent unde în câmpul vizual apare acest mic pătrat. Acea
observația justifică constrângerea CNN că toate nodurile unui strat folosesc
aceleași greutăți. În ochi, straturile suplimentare combină rezultatele din precedent
straturi pentru a recunoaște structuri din ce în ce mai complexe: limite lungi, regiuni
de culoare asemănătoare și în cele din urmă fețele și toate obiectele familiare pe care le vedem zilnic.
Vom avea mai multe de spus despre rețelele neuronale convoluționale din Secu-
secțiunea 13.4. Mai mult, CNN-urile sunt doar un exemplu de rețea neuronală
unde anumite familii de noduri sunt constrânse să aibă aceleași greutăți. Pentru
de exemplu, în secțiunea 13.5, vom lua în considerare rețelele neuronale recurente și lungi
rețele de memorie pe termen scurt, care sunt special adaptate pentru a recunoaște
proprietăți de secvențe, cum ar fi propoziții (secvențe de cuvinte).

13.1.4 Probleme de proiectare pentru plasele neuronale

Construirea unei rețele neuronale pentru rezolvarea unei probleme date este parțial artă și parțial
știință. Înainte să putem începe să pregătim o plasă găsind greutățile de pe intrări
care ne servesc cel mai bine obiectivelor, trebuie să luăm o serie de decizii de proiectare. Aceste
include răspunsul la următoarele întrebări:
1. Câte straturi ascunse ar trebui să folosim?
2. Câte noduri vor fi în fiecare dintre straturile ascunse?
3. În ce mod vom interconecta ieșirile unui strat la intrări
al următorului strat?
Mai mult, în secțiunile ulterioare vom vedea că există și alte decizii care au nevoie
de făcut atunci când antrenăm rețeaua neuronală. Acestea includ:
4. Ce funcție de cost ar trebui să minimizăm pentru a exprima care sunt cele mai bune greutăți?
5. Cum ar trebui să calculăm ieșirile fiecărei porți în funcție de
intrări? Am sugerat că modul normal de a calcula ieșirea este să
luați o sumă ponderată de intrări și comparați-o cu 0. Dar există și alte
calcule care servesc mai bine în circumstanțe comune.
6. Ce algoritm folosim pentru a exploata exemplele de instruire în scopul de a
optimizați greutățile?

13.1.5 Exerciții pentru secțiunea 13.1

!! Exercițiul 13.1.1: Dovediți că problema din Exemplul 13.1 nu poate fi rezolvată
de un perceptron; adică punctele bune și rele nu sunt separabile liniar.

Pagina 549

! Exercițiul 13.1.2: Luați în considerare problema generală a identificării vectorilor de biți ai
lungimea n având două 1 consecutive. Să presupunem un singur strat ascuns cu unele
numărul de porți. Care este cel mai mic număr de porți pe care le puteți avea în
strat ascuns dacă (a) n = 5 (b) n = 6?
! Exercițiul 13.1.3: Proiectați o rețea neuronală care funcționează ca o poartă exclusivă sau
adică ia două intrări și dă ieșirea 1 dacă exact una dintre intrări este
1 dă ieșirea 0 în caz contrar. Sugestie: amintiți-vă că atât greutățile, cât și pragurile
poate fi negativ.
! Exercițiul 13.1.4: Dovediți că nu există un perceptron unic care să se comporte ca.
o poartă exclusivă.
! Exercițiul 13.1.5: Proiectați o rețea neuronală pentru a calcula exclusivitatea - sau a trei
intrări; adică ieșirea 1 dacă un număr impar al celor trei intrări este 1 și ieșirea
0 dacă un număr par de intrări este 1.

13.2 Rețele Feedforward dense

În secțiunea anterioară, am expus pur și simplu o rețea neuronală care a funcționat pentru Problema „1 consecutiv”. Cu toate acestea, adevărata valoare a rețelelor neuronale provine abilitatea noastră de a le proiecta date date de instruire. Pentru a proiecta o rețea, există multe alegeri care trebuie făcute, cum ar fi numărul de straturi și numărul de noduri pentru fiecare strat, așa cum sa discutat în secțiunea 13.1.4. Aceste alegeri pot fii mai multă artă decât știință. Partea de calcul a instruirii, care este mai mult știința decât arta, este în primul rând alegerea greutăților pentru intrările în fiecare nod. Tehnicile de selectare a greutăților implică de obicei convergența utilizând coborâre dient. Dar coborârea în gradient necesită o funcție de cost, care trebuie să fie a funcția continuă a greutăților. Plasele discutate în secțiunea 13.1.4, pe pe de altă parte, utilizați perceptroni a căror ieșire este fie 0, fie 1, deci ieșirile sunt nu sunt în mod normal funcții continue ale intrărilor. În această secțiune, vom cuss diferite moduri în care se poate modifica comportamentul nodurilor într-o rețea, astfel încât ieșirile devin funcții continue ale intrărilor și, prin urmare, sunt rezonabile funcția de cost aplicată rezultatelor va fi, de asemenea, continuă.

13.2.1 Notare algebră liniară

Putem descrie succint rețeaua neuronală pe care am folosit-o consecutiv Problema lui 1 folosind notația algebrei liniare. Nodurile de intrare formează un vector [2] $x = [x_1, x_2, x_3, x_4]$, în timp ce nodurile ascunse formează un vector $h = [h_1, h_2]$. The 4 muchii care leagă intrarea de nodul ascuns 1 formează un vector de greutate $w_1 = [w_{11}, w_{12}, w_{13}, w_{14}]$ și, în mod similar, avem vectorul de greutate w_2 pentru nodul ascuns 2.

[2] Presupunem că toți vectorii sunt în mod implicit vectori de coloană. Cu toate acestea, este adesea mai convenabil pentru a scrie vectori de rând și vom face acest lucru în text. Dar în formule, vom folosi transpunerea operator când vrem de fapt să folosim vectorul mai degrabă ca rând decât ca coloană.

De ce să folosiți algebra liniară?

Concizia notationala este unul dintre motivele pentru care se foloseste notatia algebrelor liniare pentru neurețele ral. O alta este performanta. Se pare că grafica pro-
unitățile de încetare (GPU-uri) au circuite care permit paralelizare ridicată operații liniar-algebrice. Mutiplicarea unei matrice și a unui vector folosind un sin-
gle linear algebra operator este mult mai rapid decât codarea aceluiași operator folosind bucle imbricate. Cadrele moderne de învățare profundă (de exemplu, TensorFlow, PyTorch, Caffe) valorifică puterea GPU-urilor pentru a accelera dramatic calcule de rețea neuronală.

Pragul intră în nodurile stratului ascuns (de exemplu, negativele vechi) formează un vector 2 $b = [b_1, b_2]$, numit adesea vectorul de părtinire. Perceptronul aplică funcția de pas neliniar pentru a produce ieșirea sa, definită ca:

$$\text{pas}(z) = \begin{cases} 1 & \text{când } z > 0 \\ 0 & \text{in caz contrar} \end{cases}$$

Fiecare nod ascuns h_i poate fi acum descris folosind expresia:

$$h_i = \text{pas}(w_i^T x + b_i) \text{ pentru } i = 1, 2$$

Am putea organiza vectorii de greutate w_1 și w_2 într-o matrice de greutate 2×4 W, unde al doilea rând al lui W este w_{eu}^T. Astfel, nodurile ascunse pot fi descrise folosind expresia:

$$h = \text{pas}(Wx + b)$$

În cazul unei intrări vectoriale, funcția pas operează doar în funcție de element vectorul pentru a produce un vector de ieșire de aceeași lungime. Putem folosi un similar aranjament pentru a descrie transformarea care produce rezultatul final din stratul ascuns. În acest caz, ieșirea finală este un y scalar, deci în loc de o greutate matricea W avem nevoie doar de un vector de greutate $u = [u_1, u_2]$ și de o singură polarizare c. Noi astfel ai:

$$y = \text{pas}(u^T h + c)$$

Notarea liniar-algebră funcționează la fel de bine atunci când avem intrări mai mari și mai multe noduri într-un singur strat ascuns. Trebuie doar să scalăm matricea de greutate și vectorul de polarizare în mod corespunzător. Adică matricea W are câte un rând pentru fiecare nod în strat și o coloană pentru fiecare ieșire din stratul anterior (sau pentru fiecare intrare, dacă acesta este primul strat); vectorul de polarizare are o componentă pentru fiecare nod. De asemenea, este ușor de tratat cazul în care există mai multe nod din stratul de ieșire. De exemplu, într-o problemă de clasificare multiclass, noi

ar putea avea un nod de ieșire y_i corespunzător clasei țintă i. Pentru o intrare dată, ieșirile specifică probabilitatea ca intrarea să aparțină corespunzătoare

---

clasă. Această aranjare are ca rezultat un vector de ieșire $y = [y_1, y_2, ..., y_n]$, unde n este numărul de clase. Rețeaua simplă din secțiunea anterioară avea un boolean ieșire, corespunzătoare a două clase de ieșire (adevărat și fals), așa că am putea avea a modelat ieșirea la fel de bine ca un vector 2. În cazul unei ieșiri vectoriale, conectăm straturile ascunse și de ieșire printr-o matrice de greutate adecvată dimensiuni în locul vectorului de greutate pe care l-am folosit pentru exemplu. Perceptronii din exemplul nostru au folosit o funcție de pas neliniar. Mai gen- în general, putem folosi orice altă funcție neliniară, numită funcție de activare, în urma transformării liniare. Descriem activarea utilizată în mod obișnuit funcții începând cu secțiunea 13.2.2.

Exemplul nostru simplu a folosit un singur strat ascuns de noduri între intrare și straturile de ieșire. În general, putem avea multe straturi ascunse, așa cum a fost sugerat în Fig. 13.3. Fiecare strat ascuns introduce o matrice suplimentară de greutăți și vector de prejudecăți, precum și propria funcție de activare. Acest tip de rețea este numită rețea feedforward, deoarece toate marginile sunt orientate „înainte”, de la intrare la ieșire, fără cicluri.

Să presupunem că există layers straturi ascunse și un strat de ieșire suplimentar, numerotat $\ell + 1$. Fie matricea de greutate pentru stratul ith să fie W_i și să fie vectorul de polarizare pentru acel strat be b_i. Greutățile $W_1, W_2, ..., W_{l+1}$ și prejudecăți $b_1, b_2, ..., b_{l+1}$ constituie parametrii modelului. Obiectivul nostru este să învățăm cele mai bune valori pentru ca acești parametri să realizeze sarcina la îndemână. În curând vom descrie cum continuați să învățați parametrii modelului.

13.2.2 Funcții de activare

Un nod (perceptron) într-o rețea neuronală este conceput pentru a da un 0 sau 1 (da sau nu) ieșire. Adesea, dorim să modificăm că ieșirea este una din mai multe moduri, așa că aplicăm o funcție de activare la ieșirea unui nod. În unele cazuri, activarea funcția preia toate ieșirile unui strat și le modifică ca grup. the motivul pentru care avem nevoie de o funcție de activare este după cum urmează. Abordarea pe care o vom folosi pentru a afla valori bune ale parametrilor pentru rețea este descendența în gradient. Astfel, noi au nevoie de funcții de activare care „joacă bine” cu coborâre în gradient. În special, căutăm funcții de activare cu următoarele proprietăți:

1. Funcția este continuă și diferențiată peste tot (sau aproape oriunde).
2. Derivatul funcției nu se satură (adică devine foarte mic, tendind spre zero) peste intervalul de intrare așteptat. Deriva foarte mică tivele tind să oprească procesul de învățare.
3. Derivatul nu explodează (adică devine foarte mare, tindând spre infinit), deoarece acest lucru ar duce la probleme de instabilitate numerică.

Funcția pas nu îndeplinește condițiile (2) și (3). Derivatul său explodează la 0 și este 0 peste tot. Astfel, funcția pas nu se joacă bine cu coborâre în gradient și nu este o alegere bună pentru o rețea neuronală profundă.

13.2.3 Sigmoidul

Având în vedere că nu putem folosi funcția pas, căutăm alternative în clasa de funcții sigmoide - așa-numita datorită curbei în formă de S pe care acestea funcțiile prezintă. Cea mai frecvent utilizată funcție sigmoidă este logistica sigmoid:

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^{x}}{1 + e^{x}}$$

Observați că sigmoidul are valoarea 1/2 la x = 0. Pentru x mare, sigmoidul se apropie de 1, iar pentru x negativ mare, sigmoidul se apropie de 0. Sigmoidul logistic, la fel ca toate funcțiile pe care le vom discuta, se aplică vectori elementar, deci dacă $x = [x_1, x_2, ..., x_n]$ atunci

$\sigma(x) = [\sigma(x_1), \sigma(x_2), ..., \sigma(x_n)]$
Sigmoidul logistic are mai multe avantaje față de funcția de pas ca modalitate
pentru a defini ieșirea unui perceptron. Sigmoidul logistic este continuu și diferit
ferentabil, deci ne permite să folosim coborârea în gradient pentru a descoperi cele mai bune greutăți.
Deoarece valoarea sa este în intervalul [0, 1], este posibil să se interpreteze ieșirile din
rețea ca probabilitate. Cu toate acestea, sigmoidul logistic satura foarte repede
pe măsură ce ne îndepărtăm de „regiunea critică" în jurul valorii de 0. Deci derivatul merge
spre zero și învățarea bazată pe gradient se poate opri. Adică greutăți aproape
nu mai schimba, odată ce scapă de 0.
În secțiunea 13.3.3, atunci când descriem algoritmul de propagare înapoi, vom face acest lucru
vezi că avem nevoie de derivatele funcțiilor de activare și ale funcțiilor de pierdere. La fel de
un exercițiu, puteți verifica că dacă $y = \sigma(x)$, atunci
$\frac{dy}{dx} = y(1 - y)$

13.2.4 Tangenta hiperbolică

Strâns legată de sigmoid este funcția tangentă hiperbolică, definită de:
$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Simpla manipulare algebrică produce:
$\tanh(x) = 2\sigma(2x) - 1$
Deci, tangenta hiperbolică este doar o versiune scalată și deplasată a sigmoidului.
Are două proprietăți dorite care îl fac atractiv în anumite situații:
ieșirea este în intervalul [−1, 1] și este simetrică în jurul valorii de 0. De asemenea, împărtășește
proprietăți bune și problema de saturație a sigmoidului. Puteți arăta asta
dacă $y = \tanh(x)$ atunci
$\frac{dy}{dx} = 1 - y^2$

6
4
2
0
2
4
6
0,5
1.0
3
2
1
1
2
3
1.0
0,5
0,5
1.0
(A)
(b)
Figura 13.4: Funcțiile sigmoide logistice (a) și tangente hiperbolice (b)
Figura 13.4 prezintă funcțiile tangente sigmoide și hiperbolice logistice.
Observați diferența de scară de-a lungul axei x între cele două diagrame. Este ușor
pentru a vedea că funcțiile sunt identice după schimbare și redimensionare.

13.2.5 Softmax

Funcția softmax diferă de funcțiile sigmoide prin faptul că nu funcționează
elementar pe un vector. Mai degrabă, funcția softmax se aplică unui întreg
vector. Dacă $x = [x_1, x_2, ..., x_n]$, atunci softmax $\mu(x) = [\mu(x_1), \mu(x_2), ..., \mu(x_n)]$
Unde
$\mu(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$
Softmax împinge cea mai mare componentă a vectorului către 1 în timp ce împinge toate
celelalte componente spre zero. De asemenea, toate ieșirile însumează 1, indiferent
a sumei componentelor vectorului de intrare. Astfel, ieșirea fișierului
funcția softmax poate fi interpretată ca o distribuție de probabilitate.
O aplicație obișnuită este utilizarea softmax în stratul de ieșire pentru o clasă
problemă de ficare. Vectorul de ieșire are o componentă corespunzătoare fiecăruia
clasa țintă, iar ieșirea softmax este interpretată ca probabilitatea
intrare aparținând clasei corespunzătoare.
Softmax are aceeași problemă de saturație ca și funcția sigmoidă, de când una
componenta devine mai mare decât toate celelalte. Există o soluție simplă la acest lucru

problemă, totuși, când softmax este utilizat la nivelul de ieșire. În acest caz, este obișnuit să alegeți entropia încrucișată ca funcție de pierdere, care anulează exponențierea în definiția softmax și evită saturația. Entropia încrucișată este explicată în

---

Pagina 554

Precizia calculului Softmax

Numitorul funcției softmax implică calcularea unei sume a forma $\Sigma_j e^{x_j}$. Când x_j -urile iau o gamă largă de valori, exponenții lor e^{x_j} iau o gamă și mai largă de valori - unele mici și altele foarte mari. Adăugarea unor numere în virgulă mobilă foarte mari și foarte mici duce la numerice probleme de inexactitate în reprezentări în virgulă mobilă cu lățime fixă (cum ar fi 32- sau 64 de biți). Din fericire, există un truc pentru a evita această problemă. Noi observă că

$$\mu(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} = \frac{e^{x_i - c}}{\sum_j e^{x_j - c}}$$

pentru orice constantă c. Alegem $c = \max_j x_j$, astfel încât $x_j - c \leq 0$ pentru toate j. Acest lucru asigură că $e^{x_j - c}$ este întotdeauna între 0 și 1 și duce la un alt calcul precis. Cele mai multe cadre de învățare profundă vor avea grijă calculați softmax în acest mod.

Secțiunea 13.2.9. Abordăm problema diferențierii funcției softmax în secțiunea 13.3.3.

13.2.6 Unitate liniară certificată

Unitatea liniară rectificată sau ReLU este definită ca:

$$f(x) = \max(0, x) = \begin{cases} x, & \text{pentru } x \geq 0 \\ 0, & \text{pentru } x < 0 \end{cases}$$

Denumirea acestei funcții derivă din analogia cu rectificarea pe jumătate de undă în Inginerie Electrică. Funcția nu este diferențiată la 0, dar este diferențiată oriunde altundeva, inclusiv în puncte arbitrare apropiate de 0. În practică, „setăm” derivatul la 0 să fie fie 0 (derivatul din stânga), fie 1 (derivatul din dreapta).

În rețelele neuronale moderne, o versiune a ReLU a înlocuit sigmoidul ca de-alegerea defecțiunii funcției de activare. Popularitatea ReLU derivă din două proprietăți:

1. Gradientul ReLU rămâne constant și nu se satură niciodată pentru pozitiv x, accelerarea antrenamentului. S-a constatat în practică că rețelele care utilizați ReLU oferă o accelerare semnificativă în antrenament în comparație cu sigmoid activare.
2. Atât funcția, cât și derivata ei pot fi calculate folosind elementare și operații matematice eficiente (fără exponențiere).

ReLU suferă de o problemă legată de saturația derivatului său când $x < 0$. Odată ce valorile de intrare ale unui nod devin negative, este posibil ca

---

Pagina 555

3
2
1
1
2
3
1
1
2
3
3
2
1
1
2
3
1
1
2
3
(A)
(b)

Figura 13.5: ReLU (a) și ELU (b), cu funcții $\alpha = 1$

ieșirea nodului se „blochează” la 0 prin restul antrenamentului. Aceasta se numește problema ReLU pe moarte.
Leaky ReLU încearcă să remedieze această problemă definind activarea funcționează după cum urmează:
f (x) = {
X,
pentru x ≥ 0
αx, pentru x <0
unde αeste de obicei o valoare pozitivă mică, cum ar fi 0,01. ReLU parametric (PReLU) face din αun parametru care trebuie optimizat ca parte a procesului de învățare.
O îmbunătățire atât a funcțiilor ReLU originale, cât și a celor scurse este Expo-unitatea liniară esențială sau ELU. Această funcție este definită ca:
f (x) = {
X,
pentru x ≥ 0
α(e x - 1), pentru x <0
unde α≥ 0 este un hiperparametru. Adică, αeste menținut fix în timpul învățării proces, dar putem repeta procesul de învățare cu valori diferite de la αla găsiți cea mai bună valoare pentru problema noastră. Valoarea nodului satura la −α pentru valori negative mari ale lui x și o alegere tipică este α= 1. Unitatea ELU este activarea medie a nodurilor către zero, ceea ce pare să accelereze învățarea proces comparativ cu alte variante ReLU.

13.2.7 Funcții de pierdere

O funcție de pierdere cuantifică diferența dintre predicțiile unui model și valorile de ieșire observate în lumea reală (adică în setul de antrenament). Presupune

---

ieșirea observată corespunzătoare intrării x este y iar ieșirea prezisă este y. Apoi, o pierdere L (y, y) cuantifică eroarea de predicție pentru acest single intrare. De obicei, luăm în considerare pierderea unui set mare de observații, de exemplu ca întregul set de antrenament. În acest caz, de obicei pierdem media pe toate exemple de instruire.
Vom lua în considerare separat două cazuri. În primul caz, există un singur nod de ieșire și produce o valoare reală. În acest caz, studiem „regresia pierderi." În al doilea caz, există mai multe noduri de ieșire, fiecare dintre ele indicând că intrarea este un membru al unei anumite clase; studiem această chestiune sub ,,Pierdere din clasificare” în secțiunea 13.2.9.

13.2.8 Pierderea de regresie

Să presupunem că modelul are o singură ieșire cu valoare continuă și (x, y) este un antrenament exemplu. Pentru aceeași intrare x, să presupunem ieșirea prezisă a rețelei neuronale este y. Atunci pierderea de eroare pătrată L (y, y) a acestei predicții este:
L (y, y) = (y - y) 2
În general, calculăm pierderea pentru un set de predicții. Să presupunem că observatul (adică setul de antrenament) perechile de intrare-ieșire sunt T = {(x 1 , ˆy 1), (x 2 , ˆy 2), ..., (x n , ˆy n)}, în timp ce perechile de intrare-ieșire corespunzătoare prezise de model sunt P = {(x 1 , y 1), (x 2 , y 2), ..., (x n , y n)}. Eroarea medie pătrată (MSE) pentru set este:
L (P, T) =
1
n
n
∑
i = 1
(y i - ˆy i) 2
Rețineți că eroarea medie pătrată este doar pătrată a RMSE. Este convenabil a omite rădăcina pătrată pentru a simplifica derivata funcției, pe care o vom face utilizarea în timpul antrenamentului. În orice caz, atunci când minimizăm MSE, de asemenea, suntem automat minimizați RMSE.
O problemă cu MSE este că este foarte sensibilă la valori aberante datorită pătratului termen. Câteva valori aberante pot contribui foarte mult la pierdere și pot înmulți efectul altor puncte, făcând procesul de antrenament susceptibil de leagăne sălbatice.
O modalitate de a rezolva această problemă este de a utiliza Huber Loss. Să presupunem z = y - y, iar δeste o constantă. Pierderea Huber L δeste dată de:
L δ(z) = {
z 2
dacă | z | ≤ δ
2δ (| z | - 1

$2\ \delta$)altfel
Figura 13.6 contrastează eroarea pătrată și funcțiile de pierdere Huber.
În cazul în care avem un vector y de ieșiri, mai degrabă decât o singură ieșire,
folosim y - y în loc de | y - y | în definițiile erorii pătrate medii și
Pierderea lui Huber.

Pagina 557

5
4
3
2
1
1
2
3
4
5
5
10
15
20
25
Figura 13.6: Pierderea Huber (linie continuă, δ= 1) și Eroare pătrată (linie punctată) ca
funcțiile lui z = y - y

13.2.9 Pierderea clasificării

Luați în considerare o problemă de clasificare multiclasă cu clasele țintă C_1, C_2, ..., C_n.
Să presupunem că fiecare punct din setul de antrenament este de forma (x, p) unde x este intrarea
iar p = [p_1, p_2, ..., p_n] este ieșirea. Aici p_i oferă probabilitatea ca
intrarea x aparține clasei C_i, cu $\sum_i p_i = 1$. În multe cazuri, suntem siguri că
o intrare aparține unei anumite clase C_i; în acest caz $p_i = 1$ și $p_j = 0$ pentru
i = j. În general, putem interpreta p_i ca nivelul nostru de certitudine că intrarea x
aparține clasei C_i și p ca distribuție de probabilitate peste clasele țintă.
Ne proiectăm rețeaua neuronală pentru a produce ca ieșire un vector
q = [q_1, q_2, ..., q_n]
de probabilități, cu $\sum_i q_i = 1$. Ca și mai înainte, interpretăm q ca o probabilitate
distribuție pe clasele țintă, cu q_i denotând probabilitatea modelului
acea intrare x aparține clasei țintă C_i. În secțiunea 13.2.5 am descris o sim-
pentru a produce un astfel de vector de probabilitate ca ieșire: utilizați softmax
funcție de activare în stratul de ieșire al rețelei.
Deoarece atât ieșirea etichetată, cât și ieșirea modelului sunt probabilități de
contribuții, este firesc să căutăm o funcție de pierdere care să cuantifice distanța
între două distribuții de probabilitate. Reamintim definiția entropiei din
Secțiunea 12.5.2. Adică H (p), entropia unei distribuții discrete de probabilitate
p este:

$$H(p) = -\sum_{i=1}^{n} p_i \log p_i$$

Imaginați-vă un alfabet de n simboluri și mesaje folosind aceste simboluri. Cina-
poza la fiecare locație din mesaj, probabilitatea ca simbolul i să apară este

---

Pagina 558

p_i. Apoi, un rezultat cheie din teoria informației este că dacă codificăm mesaje
folosind un cod binar optim, numărul mediu de biți pe simbol necesar
codarea mesajelor este H (p).
Să presupunem că nu am știut distribuția probabilității simbolului p când noi
proiectați schema de codare. În schimb, credem că simbolurile apar în continuare
o distribuție de probabilitate diferită q. Ne-am putea întreba care este numărul mediu
de biți pe simbol va fi dacă folosim această schemă de codificare suboptimală. Un bine-
rezultatul cunoscut din teoria informației afirmă că numărul mediu de biți din
acest caz este entropia încrucișată H (p, q), definită ca:

$$H(p, q) = -\sum_{i=1}^{n} p_i \log q_i$$

Rețineți că H (p, p) = H (p) și, în general, H (p, q) ≥ H (p). Diferența
între entropia încrucișată și entropia este numărul mediu de suplimentare

biți necesari pentru fiecare simbol. Este o măsură rezonabilă a distanței dintre distribuțiile p și q, numite divergența Kullblack-Liebler (divergența KL) și notat D (pq):

$$D(pq) = H(p, q) - H(p) = \sum_{i=1}^{n} p_i \log \frac{p_i}{q_i}$$

Chiar dacă divergența KL este adesea privită ca o distanță, nu este cu adevărat o măsură de distanță pentru că nu este comutativă. Cu toate acestea, este perfect adecvată ca funcție de pierdere pentru scopurile noastre, deoarece există de fapt o inerentă asimetrie în situație: p este adevărul la sol, în timp ce q este cel prezis ieșire. Observați că minimizarea pierderii KL-divergență a unui model este echivalentă la minimizarea pierderii de entropie încrucișată, deoarece termenul H (p) depinde doar de de intrare și este independent de modelul care este învățat.

În practică, entropia încrucișată este cea mai frecvent utilizată funcție de pierdere pentru clasa probleme de clasificare. Rețelele proiectate pentru clasificare folosesc adesea un softmax funcție de activare în stratul de ieșire. Aceste alegeri sunt atât de comune încât multe implementări, cum ar fi TensorFlow, oferă o singură funcție care bines softmax cu entropie încrucișată. În plus față de comoditate, un motiv de făcut la fel este că funcția combinată este mai stabilă numeric, iar derivata ei ia, de asemenea, o formă simplă, așa cum arătăm în secțiunea 13.3.3.

13.2.10 Exerciții pentru secțiunea 13.2

Exercițiul 13.2.1: Arătați că pentru sigmoidul logistic σ,dacă $y = \sigma(x)$, atunci

$$\frac{dy}{dx} = y(1 - y)$$

Exercițiul 13.2.2: Arătați că dacă $y = \tanh(x)$ atunci

$$\frac{dy}{dx} = 1 - y^2$$

Exercițiul 13.2.3: Arată că $\tanh(x) = 2\sigma(2x) - 1$.

Exercițiul 13.2.4: Arătați că $\sigma(x) = 1 - \sigma(-x)$.

Exercițiul 13.2.5: Arătați că pentru orice vector $[v_1, v_2, ..., v_k]$, $\sum_{i=1}^{k} u(v_i) = 1$.

1
y
x_1
x_2
x_3
x_4
y
3
y
2
1
= 1
= 0
= 1
= 0
1
2
3
−4
1
−1
2
3
−2
−2
1
−2
1
1

1
4
1
1
2
2
3
Figura 13.7: Rețea neuronală pentru exercițiul 13.2.6
Exercițiul 13.2.6: În Fig. 13.7 este o rețea neuronală cu valori particulare arătate pentru toate greutățile și intrările. Să presupunem că folosim funcția sigmoidă pentru calculează ieșirile nodurilor la primul strat și folosim softmax pentru a calcula ieșirile nodurilor din stratul de ieșire.
(a) Calculați valorile ieșirilor pentru fiecare dintre cele cinci noduri.
! (b) Presupunând că fiecare dintre greutăți și fiecare dintre x_i este o variabilă, expresă ieșirea primului nod de ieșire (de sus) în termeni de greutăți și x_{eu} sunt,
! (c) Găsiți derivata funcției dvs. din partea (b) cu privire la greutatea primului (de sus) intrare la primul nod (de sus) din primul strat.

13.3 Propagarea înapoi și coborârea în gradient

Trecem acum la problema formării unei rețele profunde. Instruirea unei rețele înseamnă găsirea unor valori bune pentru parametrii (greutăți și praguri) ale rețea. De obicei, vom avea acces la un set de instruire cu intrări / ieșiri etichetate perechi. Procesul de instruire încearcă să găsească valori ale parametrilor care să minimizeze pierderea medie pe setul de antrenament. Speranța este că setul de formare este reprezentat date privind modelul pe care îl va întâlni în viitor și, prin urmare, pierderea medie pe setul de antrenament este o măsură bună a erorii medii la toate intrări posibile. Cu toate acestea, trebuie să fim atenți; întrucât rețelele profunde au multe

---

Pagina 560

parametrii, este posibil să se găsească parametri care produc pierderi reduse de antrenament, dar cu toate acestea, performanța slabă în lumea reală. Acest fenomen se numește potrivire, o problemă pe care am menționat-o de mai multe ori, începând cu secțiunea 9.4.4. Pentru moment, presupunem că obiectivul nostru este de a găsi parametri care să minimizeze pierderea așteptată pe setul de antrenament. Acest obiectiv este atins prin coborârea în gradient. Există un algoritm elegant numit backpropagation care ne permite să calculăm acești gradienți în mod eficient. Înainte de a descrie propagarea înapoi, avem nevoie de câteva preliminarii.

13.3.1 Calculați graficele

Un grafic de calcul captează fluxul de date al unei rețele profunde. În mod formal, un calcul grafic este un grafic aciclic direcționat (DAG). Fiecare nod din graficul de calcul are un operand și, opțional, un operator. Operandul poate fi un scalar, un vector, sau o matrice. Operatorul este un operator de algebră liniară (cum ar fi + sau ×), un funcție de activare (cum ar fi σ)sau o funcție de pierdere (cum ar fi MSE). Când un nod are atât un operand, cât și un operator, operatorul este scris deasupra operandului. Când un nod are doar un operand, ieșirea acestuia este valoarea asociată cu operandul. Ieșirea unui nod cu un operator este rezultatul aplicării operatorul său predecesorilor săi imediați din grafic și apoi atribuirea rezultatul operandului. În general, operatorul poate fi o expresie arbitrară care își folosește intrările pentru a produce o ieșire. [3]
Exemplul 13.2: Figura 13.8 prezintă graficul de calcul pentru un dens cu un singur strat rețea descrisă de $y = \sigma(Wx + b)$ unde x este intrarea și y este ieșirea.
Apoi calculăm o pierdere MSE în raport cu ieșirea setată de antrenament y. Adică noi au un singur strat de n noduri. Vectorul y are lungimea n și reprezintă ieșirile fiecăruia dintre aceste noduri. Există k intrări și (x, y) reprezintă una exemplu de instruire. Matricea W reprezintă greutățile pe intrările nodurilor; adică W_{ij} este greutatea pentru intrarea j la nodul ith. În cele din urmă, b reprezintă n părtiniri, deci elementul său i este negativul pragului nodului i.
tu
v
y
y
X
W
b
L
MSE
σ

Figura 13.8: Calculați graficul pentru o rețea densă cu un singur strat
Pentru graficul din Fig. 13.8, avem:
[3] Uneori contează ordinea operanzilor pentru operator. Vom ignora acest detaliu aici și presupunem că este înțeles din context.

$u = Wx$
$v = u + b$
$y = \sigma(v)$
$L = MSE(y, y)$

Fiecare dintre acești pași corespunde unuia dintre cele patru noduri din rândul din mijloc, în ordine din stânga. Primul pas corespunde nodului cu operand u și operator ×. Iată un exemplu în care trebuie înțeles că nodul etichetat W este primul argument. Dacă este necesar, am putea eticheta fiecare intrare marginea cu un număr pentru a indica ordinea acestuia, dar în acest caz ordinea ar trebui să fie evident, deoarece un vector de coloană x nu ar putea multiplica o matrice W decât dacă matricea a avut un singur rând. Al doilea pas corespunde nod cu operator + și operand v. Aici, ordinea argumentelor nu materie, deoarece + pe vectori este comutativ. ✷

13.3.2 Gradienți, iacobieni și regula lanțului

Scopul algoritmului de propagare înapoi este de a calcula gradientul funcția de pierdere în raport cu parametrii rețelei. Atunci, putem reglați ușor parametrii în direcțiile care vor reduce pierderea și repetați procesul până când ajungem la o selecție de valori ale parametrilor pentru care puțin este posibilă îmbunătățirea pierderii. Reamintim definiția gradientului: dat a funcția $f: R^N \to R$ de la un vector cu valoare reală la un scalar, dacă $x = [x_1, x_2, ..., x_n]$ și $y = f(x)$ atunci este dat gradientul lui y față de x, notat cu $\nabla_x y$ de

$$\nabla_x y = [\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, ..., \frac{\partial y}{\partial x_n}]$$

Exemplul 13.3: Am putea lăsa funcția f să fie funcția de pierdere, de ex. pierderea de eroare, pe care o denotăm L. Această pierdere este o funcție scalară a ieșirii y:

$$L(y) = y - y^2 = \sum_{i=1}^{n} (y_i - y_i)^2$$

Deci putem nota cu ușurință gradientul lui L față de y:

$$\nabla_y L = [2(y_1 - y_1), (y_2 - y_2), ..., 2(y_n - y_n)] = 2(y - y)$$

✷

Generalizarea gradientului către funcții cu valoare vectorială se numește Jacobian. Să presupunem că avem o funcție $f: R^m \to R^n$ și $y = f(x)$. The Jacobian $J_x(y)$ este dat de: [4]

[4] Jacobianul este uneori definit ca transpunerea definiției noastre. Formulările sunt echivalente. Reamintim că presupunem că toți vectorii sunt vectori de coloană, cu excepția cazului în care sunt transpuși, dar le arătăm ca vectori de rând, astfel încât să poată fi scrise în linie.

$J_x(y) =$

$$\begin{bmatrix} \frac{\partial y_1}{\partial x_1} \end{bmatrix}$$

...
∂y_n
∂x_1
...
...
...
$1y_1$
∂x_m
...
∂y_n
∂x_m
□
□
□
□
□
Vom folosi regula lanțului pentru derivate din calcul. Dacă y = g (x)
și z = f (y) = f (g (x)), atunci regula lanțului spune:
dz
dx
=
dz
dy
dy
dx
De asemenea, dacă z = f (u, v) unde u = g (x) și v = h (x), atunci
dz
dx
=
∂z
∂u
du
dx
+
∂z
∂v
dv
dx
Pentru funcțiile vectorilor, putem reformula regula lanțului în termeni de gradienți și
Iacobeni. Să presupunem că y = g (x) și z = f (y) = f (g (x)) atunci:
$\nabla_x z = J_x(y) \nabla_y z$
Dacă z = f (u, v) unde u = g (x) și v = h (x), atunci
$\nabla_x z = J_x(u) \nabla_u z + J_x(v) \nabla_v z$

13.3.3 Algoritmul Backpropagation

Scopul algoritmului de propagare înapoi este de a calcula gradientul pierderii
funcționează în raport cu parametrii rețelei. Luați în considerare calculul
grafic din Fig. 13.8. Aici funcția de pierdere L este funcția MSE. Vom
utilizați notația g (z) pentru a reprezenta ∇_z (L), adică gradientul pierderii
funcția L față de un vector z. Știm deja gradientul lui L
în ceea ce privește ieșirea y:
$g(y) = \nabla_y (L) = 2 (y - y)$
Lucrăm înapoi prin graficul de calcul, aplicând regula lanțului la
fiecare etapă. În fiecare moment, alegem un nod pe care toți succesorii îl au deja
a fost procesat. Să presupunem că a este un astfel de nod și să presupunem că are doar unul imediat
succesorul b din grafic (rețineți că în graficul de calcul simplu din Fig. 13.8,
fiecare nod are doar un succesor imediat). Deoarece am procesat deja
nodul b, am calculat deja g (b). Acum putem calcula g (a) folosind
regula lantului:
$g(a) = J_a(b)\, g(b)$
În cazul în care nodul a are mai mult de un nod succesor, vom folosi mai mult
versiunea sumă generală a regulii lanțului. Adică g (a) ar fi suma
termenii de mai sus pentru fiecare succesor b al lui a.

Deoarece va trebui să calculăm aceste gradiente de mai multe ori, o dată pentru fiecare iterația coborârii în gradient, putem evita calculele repetate prin adăugare noduri suplimentare la graficul de calcul pentru propagarea înapoi: un nod pentru fiecare calcul de gradient. În general, iacianul J_a (b) este o funcție a ambelor a și b, și astfel nodul pentru g (a) va avea arcuri de la nodurile pentru a, b și g (b). Cadrele populare pentru învățarea profundă (de exemplu, TensorFlow) știu cum se calculează expresia funcțională pentru iacobieni și gradienți ai operatori folosiți în mod obișnuit, cum ar fi cei care apar în Fig. 13.8. În acest caz, dezvoltatorul trebuie doar să furnizeze graficul de calcul și cadrul va adăuga noile noduri pentru propagarea înapoi. Figura 13.9 arată rezultatul calculați graficul cu noduri de gradient adăugate.

g (*W*)
g (**v**)
g (**y**)
tu
v
y
σ
L
MSE
y
(**b**)
g
g (**u**)
W
X
b

Figura 13.9: Calculați graficul cu noduri de gradient

Exemplul 13.4: Vom elabora expresiile funcționale pentru gradienți a tuturor nodurilor din Fig. 13.8. Știm deja g (y). Deci următorul nod pe care îl alegem a procesa este v.

g (v) = ∇_v (L) = J_v (y) ∇_y (L) = J_v (y) g (y)

Știm că y = σ(v). Deoarece σ este un operator înțelept, Jacobianul J_v (y) ia o formă deosebit de simplă. Folosind derivatul pentru logistică funcția sigmoidă din secțiunea 13.2.2, vedem că

Iy $_i$
∂v $_j$ = {
y $_i$ (1 - y $_i$) dacă i = j
0
in caz contrar

Prin urmare, Jacobianul este o matrice diagonală:

J $_v$ (y) =
□
y $_1$ (1 - y $_1$)
0
...
0
0
y $_2$ (1 - y $_2$) ...
0
...
...
...
...
0
0
... y $_n$ (1 - y $_n$)
□
□
□
□
□

Să presupunem că $s = [s_1, s_2, ..., s_n]$ este un vector definit de $s_i = y_i (1 - y_i)$ (adică, diagonala matricei iacobiene). Putem exprima g (v) simplu ca

$g(v) = s \circ g(y)$

unde $a \circ b$ este vectorul rezultat din produsul elementar al lui a și b. [5]

Acum că avem g (v), putem calcula g (b) și g (u). Avem g (b) =

$J_b(v)\, g(v)$ și $g(u) = J_u(v)\, g(v)$. De cand

$v = u + b$

este simplu să verificăm asta

$J_b(v) = J_u(v) = I_n$

unde I_n este matricea de identitate $n \times n$. Deci avem

$g(b) = g(u) = g(v)$

Am ajuns în cele din urmă la matricea W. Amintim că u = Wx. Există un potențial problema aici, pentru că toate utilajele pe care le-am instalat funcționează pentru vectori, în timp ce W este o matrice. Reamintim însă din secțiunea 13.2.1 că am asamblat matricea W dintr-un set de vectori $w_1, w_2, ..., w_n$, unde w_i^T este al doilea rând al lui W. Trucul este să ia în considerare fiecare dintre acești vectori separat și să-și calculeze gradientul folosind formula obișnuită.

$g(w_i) = J_{w_i}(u)\, g(u)$

Știm că $u_i = w_i^T x$ și niciunul dintre celelalte u_j nu au nicio dependență de w_i pentru i = j. Prin urmare, $J_{w_i}(u)$ iacobian este zero peste tot cu excepția ith coloană, care este egală cu x. Astfel avem

$g(w_i) = g(u_i)\, x$

✷

Exemplul 13.5: După cum am menționat, rețelele neuronale pentru clasificare se folosesc adesea o activare softmax în stratul final urmată de o pierdere de entropie încrucișată. Vom acum calculați gradientul operatorului combinat.

Să presupunem că intrarea la operatorul combinat este y; să fie $q = \mu(y)$ și să $l = H(p, q)$, unde p reprezintă adevăratul vector de probabilitate pentru corespunzător exemplu de instruire. Noi avem:

$$\log(q_i) = \log\left(\frac{e^{y_i}}{\sum_j e^{y_j}}\right) = y_i - \log\left(\sum_j e^{y_j}\right)$$

[5] Această operație numită uneori produsul Hadamard, pentru a nu-l confunda cu produsul obișnuit cu puncte, care este suma componentelor produsului Hadamard.

Prin urmare, observând că $\sum_i p_i = 1$, avem:

$$l = H(p, q) = -\sum_i p_i \log q_i = -\sum_i p_i \left(y_i - \log\left(\sum_j e^{y_j}\right)\right) = -\sum_i p_i y_i - \log\left(\sum_j e^{y_j}\right) \sum_i p_i = -\sum_i p_i y_i - \log\left(\sum_j e^{y_j}\right)$$

Diferențiat, obținem:

$$\frac{\partial l}{\partial y_k} = -p_k + \frac{e^{y_k}}{\sum_j e^{y_j}}$$

$= -p_k + \mu(y_k)$
$= q_k - p_k$
Prin urmare, ajungem la rezultatul destul de îngrijit:
$\nabla_y l = q - p$
Acest gradient combinat nu saturează sau explodează și duce la o bună învățare
comportament. Această observație explică de ce funcționează pierderea softmax și a entropiei încrucișate
atât de bine împreună în practică. ✷

13.3.4 Coborârea gradientului iterant

Având în vedere un set de exemple de antrenament, rulăm graficul de calcul în ambele direcții
pentru fiecare exemplu: înainte (pentru a calcula pierderea) și înapoi (pentru a calcula
gradienții). Punem în medie pierderile și gradienții din setul de instruire
calculați pierderea medie și gradientul mediu pentru fiecare vector parametru.
La fiecare iterație actualizăm fiecare vector parametru în direcția opusă
până la gradientul său, deci pierderea va tinde să scadă. Să presupunem că z este un parametru
vector. Noi am stabilit:
$z \leftarrow z - \eta g(z)$
Aici ηeste un hiperparametru, rata de învățare. Ne oprim fie coborârea în gradient
atunci când pierderea dintre iterațiile succesive se modifică doar cu o cantitate mică (de exemplu,
am atins un minim local) sau după un număr fix de iterații.
Este important să alegeți cu atenție rata de învățare. O valoare prea mică înseamnă
coborârea în gradient ar putea necesita un număr foarte mare de iterații pentru a converge. De asemenea
o valoare mare poate provoca oscilații mari în valorile parametrilor și niciodată

---

conduc la convergență. De obicei, alegerea ratei de învățare potrivite este o chestiune de încercare
și eroare. De asemenea, este posibil și obișnuit să variați rata de învățare. Începe cu
o rată de învățare inițială η_0 . Apoi, la fiecare iterație, înmulțiți rata de învățare
cu un factor $\beta(0 < \beta < 1)$ până când rata de învățare atinge o valoare suficient de mică.
Când avem un set mare de antrenament, este posibil să nu dorim să folosim întregul
set de instruire pentru fiecare iterație, deoarece ar putea consuma prea mult timp. Deci pentru fiecare
iterație eșantionăm aleatoriu un „minibatch” de exemple de antrenament. Această variantă
se numește descendență în gradient stocastic ”, așa cum sa discutat în secțiunea 12.3.5, deoarece
estimăm gradienții folosind un eșantion diferit al setului de antrenament la fiecare
repetare.
Am lăsat deschisă întrebarea despre cum sunt inițializate valorile parametrilor
înainte de a începe coborârea în gradient. Abordarea obișnuită este să le alegeți la
Aleatoriu. Abordările populare includ eșantionarea fiecărei intrări în mod uniform
dom în [−1, 1] sau alegerea aleatorie folosind o distribuție normală. Observa asta
inițializarea tuturor greutăților la aceeași valoare ar cauza toate nodurile dintr-un strat
să ne comportăm în același mod și astfel nu am profita niciodată de avantajul de a avea
diferite noduri dintr-un strat recunosc diferite caracteristici ale intrării.

13.3.5 Tensori

Anterior, ne-am imaginat că intrările într-o rețea neuronală sunt unidimensionale
vectori. Dar nu există niciun motiv pentru care nu putem vedea intrarea ca având o valoare mai mare
dimensiune.
Exemplul 13.6: O fotografie la scară de gri ar putea fi reprezentată printr-o imagine bidimensională
matrice de numere reale, corespunzătoare intensității fiecărui pixel. Fiecare pixel
pentru o imagine color necesită de obicei 3 dimensiuni. Adică fiecare pixel în sine este
un vector cu trei componente, să zicem pentru intensitatea pixelului în roșu,
culori verzi și albastre. Un mod util de a vizualiza o imagine color ca intrare într-un
rețeaua neuronală este să gândim fiecare exemplu de antrenament ca o matrice bidimensională de
pixeli, unde valoarea fiecărui pixel nu este un număr real, așa cum avem până acum
imaginat, dar un vector cu trei dimensiuni, câte una pentru fiecare dintre cele trei culori.
✷
În mod similar, am văzut fiecare strat al unei rețele neuronale ca o coloană de noduri.
Dar nu există niciun motiv pentru care nu ne putem imagina nodurile dintr-un strat care urmează să fie organizate
ca o matrice bidimensională sau chiar o matrice de dimensiuni mai mare de două.
În cele din urmă, am văzut valorile de intrare ca numere reale și vizualizate în mod similar
valorile produse de fiecare nod ca număr real. Dar ne-am putea gândi și la asta
valorile atașate fiecărei intrări sau ieșiri a unui nod ca vector sau ca
structura dimensionala. Generalizarea naturală a vectorilor și matricelor este
tensorul, care este o matrice n-dimensională de scalari.
Din păcate, algoritmul de propagare inversă pe care l-am descris funcționează pentru vecini
tori, nu tensori de dimensiuni superioare. În astfel de cazuri, apelăm la același truc
am folosit în secțiunea 13.3.3, unde am derulat matricea W în colecția de

vectori. Așa cum considerăm o matrice m × n ca un set de m n-vectori, putem re-
gard un tensor tridimensional de dimensionalitate l × m × n ca un set de lm n-vectori,
și în mod similar pentru tensorii de dimensiuni superioare.

Exemplul 13.7: Acest exemplu se bazează pe setul de date MNIST. [6] Acest set de date
este format din 28 × 28 de imagini monocrome, fiecare reprezentată printr-o dimensiune bidimensională
matrice de biți pătrate ale căror laturi au lungimea de 28. Scopul nostru este să construim o rețea neuronală
care determină dacă o imagine corespunde unei cifre scrise de mână (0-9) și
dacă da, care. Luați în considerare o singură imagine X, care este o matrice de 28 × 28. Presupune
primul strat al rețelei noastre este un strat dens [7] format din 49 de noduri ascunse,
pe care îl vom imagina este aranjat într-o matrice 7 × 7. Modelăm stratul ascuns
ca matrice 7 × 7 H, unde ieșirea nodului din rândul i și coloana j este h_{ij} .

Putem modela greutățile pentru fiecare dintre intrările de 28 × 28 și pentru fiecare dintre
7 × 7 noduri ca tensor de greutate W cu dimensiunile 7 × 7 × 28 × 28. Adică W_{ijkl}
reprezintă greutatea pixelului de intrare în rândul i și coloana j a imaginii la
nodul a cărui poziție în matricea de noduri este rândul k și coloana l. Apoi:

$$h_{ij} = \sum_{k=1}^{28} \sum_{l=1}^{28} w_{ijkl} x_{kl} \text{ pentru } 1 \leq i, j \leq 7$$

unde omitem termenul de părtinire pentru simplitate (adică presupunem toate pragurile tuturor
nodurile sunt 0.

Un mod echivalent de a gândi la această structură pentru a aplata intrarea X în
un vector x de lungime 784 (din 28 × 28 = 784) și stratul ascuns H într-un
vectorul h de lungime 49. Aplatizăm tensorul de greutate W după cum urmează: ultimele două
dimensiunile sunt aplatizate într-o singură dimensiune pentru a se potrivi cu x și cu primele două
dimensiunile sunt aplatizate într-o singură dimensiune pentru a se potrivi cu vectorul ascuns h,
rezultând o matrice de greutate de 49 × 784. Să presupunem că, ca în secțiunea 13.2.1, avem
w_i^T indica rândul i - lea al acestei noi matrice de greutate. Acum putem scrie:

$$h_i = w_i^T x \text{ pentru } 1 \leq i \leq 49$$

Este simplu să vezi că există o mapare 1-la-1 între ascuns
noduri în vechile și noile aranjamente. Mai mult, ieșirea fiecărui ascuns
nodul este determinat de un produs punct, la fel ca în secțiunea 13.2.1. Astfel tensorul
notația este doar o modalitate convenabilă de grupare a vectorilor. ✷

Astfel tensorii utilizați în rețelele neuronale au puține în comun cu
tensori folosiți în Fizică și alte științe matematice. Un tensor în con-
textul este doar o colecție imbricată de vectori. Singura operație tensorică pe care o vom face
necesitatea este aplatizarea unui tensor prin îmbinarea dimensiunilor ca în Exemplul 13.7. Noi
poate utiliza algoritmul de propagare înapoi descris în secțiunea 13.3.3 pentru tensori
odată ce i-am aplatizat în mod corespunzător.

[6] A se vedea yann.lecun.com/exdb/mnist/.

[7] În realitate, primul strat de rețea pentru această problemă este probabil un strat convoluțional.
A se vedea secțiunea 13.4

13.3.6 Exerciții pentru secțiunea 13.3

Exercițiul 13.3.1: Acest exercițiu folosește rețeaua neuronală din Fig. 13.7. In orice caz,
presupunem că ponderile tuturor intrărilor sunt mai degrabă variabile decât constante
afișate. Rețineți, totuși, că unele intrări nu alimentează unul dintre nodurile din
primul strat, deci aceste greutăți sunt fixate la 0. Să presupunem că vectorul de intrare este x,
vectorul de ieșire este y, iar ieșirea celor două noduri din stratul ascuns este
vectorul z. De asemenea, să fie matricea și vectorul de polarizare care conectează x la z să fie W_1 și
b_1 , în timp ce matricea și vectorul de polarizare care leagă z de y sunt W_2 și b_2 . Să presupunem
că funcția de activare a stratului ascuns este tangenta hiperbolică și
funcția de activare la ieșire este funcția de identitate (adică, nicio modificare la
se realizează ieșirile). În cele din urmă, presupuneți că funcția de pierdere este o eroare pătrată medie,
unde y este adevăratul vector de ieșire pentru un anumit vector de intrare x. Desenați calculul

grafic care arată cum se calculează y din x.
Exercițiul 13.3.2: Pentru rețeaua descrisă în exercițiul 13.3.1:
(a) Ce este $J_y(z)$?
(b) Ce este $J_z(x)$?
(c) Exprimați g (x) în termeni de g (tanh (z)).
(d) Exprimați g (x) în funcție de funcția de pierdere.
(e) Desenați graficul de calcul cu calcul de gradient pentru întreaga rețea
muncă.

13.4 Rețele neuronale convoluționale

Luați în considerare un strat de rețea complet conectat pentru procesarea imaginilor de 224 × 224, cu fiecare pixel codificat folosind 3 valori de culoare (adesea numite canale - valorile de intensitate pentru roșu, verde și albastru)). [8] Numărul de parametri de greutate din stratul de conexiune pentru fiecare nod de ieșire este 224 × 224 × 3 = 150, 528. Să presupunem am avut 224 de noduri în stratul de ieșire. Am ajunge cu un total de peste 33 de milioane de parametri! Având în vedere că setul nostru de formare de imagini este de obicei al ordinea a zeci sau sute de mii de imagini, numărul imens de parametri ar duce rapid la supradezionare chiar și folosind doar un singur strat.
O rețea neuronală convolutională (CNN) reduce considerabil numărul de pași rametere profitând de proprietățile datelor de imagine. CNN prezintă două noi varietăți de straturi de rețea: straturi convoluționale și straturi de grupare.
[8] Aceasta este dimensiunea majorității imaginilor de pe ImageNet, de exemplu.

13.4.1 Straturi convoluționale

Straturile convoluționale utilizează faptul că caracteristicile imaginii sunt adesea descrise de mici zone adiacente din imagine. De exemplu, la prima convoluțională strat, am putea recunoaște mici secțiuni de margini în imagine. La straturile ulterioare, structuri mai complexe, cum ar fi caracteristici care arată ca petale de flori sau ochi ar putea fi recunoscut. Ideea care simplifică calculul este faptul că recunoașterea caracteristicilor precum marginile nu depinde de locul din imagine marginea este. Astfel, putem antrena un singur nod pentru a recunoaște o mică secțiune a margine, să spunem o margine printr-o regiune de pixeli 5 × 5. Această idee ne avantajează în două căi.
1. Nodul în cauză are nevoie doar de intrări de la 25 pixeli corespunzătoare orice pătrat 5 × 5, nu intrări din toți 224 × 224 pixeli. Asta ne salvează mult în reprezentarea CNN instruit.
2. Numărul de greutăți pe care trebuie să le învățăm în procesul de instruire este redus mult. Pentru fiecare nod din strat, avem nevoie de o singură greutate pentru fiecare intrare în acel nod - spuneți 75 de greutăți dacă un pixel este reprezentat de Valorile RGB, nu cele 150.528 de greutăți pe care le-am sugerat mai sus ar fi necesare pentru un strat obișnuit, complet conectat.
Ne vom gândi la nodurile dintr-un strat convoluțional ca la filtre pentru învățare Caracteristici. Un filtru examinează o mică zonă spațială contiguă a imaginii - în mod tradițional, o zonă pătrată mică, cum ar fi un pătrat de 5 × 5 de pixeli. În plus, deoarece multe caracteristici de interes pot apărea oriunde în imaginea de intrare (și posibil în mai multe locații), aplicăm același filtru în multe locații pe intrare.
Pentru a simplifica lucrurile, să presupunem că intrarea constă din imagini monocromatice, adică, imagini la scară de gri ale căror pixeli au fiecare o singură valoare. Fiecare imagine este astfel codificat de o matrice bidimensională de pixeli cu dimensiunea 224 × 224. Un filtru 5 × 5 F este codificat de o matrice de greutate 5 × 5 W și un parametru b de polarizare simplă. Cand filtrul este aplicat pe o regiune pătrată de dimensiuni similare a imaginii de intrare X cu colțul din stânga sus al filtrului aliniat cu imaginea pixel x_{ij}, răspunsul filtrul la această poziție, notat r_{ij}, este dat de:

$$r_{ij} = \sum_{k=0}^{4} \sum_{l=0}^{4} x_{i+k,\, j+l}\, w_{kl} + b \tag{13.1}$$

Acum glisăm filtrul de-a lungul lungimii și lățimii imaginii de intrare, aplicând filtrul la fiecare poziție, astfel încât să captăm fiecare regiune pătrată 5 × 5 posibilă de pixeli în imagine. Observați că putem aplica filtrul în locațiile de intrare

$1 \le i \le 220$ și $1 \le j \le 220$, deși nu se „potrivește” la pozițiile cu a
mai mare i sau j. Setul de răspunsuri rezultate r_{ij} este apoi trecut printr-un
funcția de activare pentru a forma harta de activare R a filtrului. În cele mai multe cazuri,
funcția de activare este ReLU sau una dintre variantele sale. Când este instruit, adică

Pagina 570

greutățile w_{ij} ale filtrului sunt determinate, filtrul va recunoaște o caracteristică a
imaginea și harta de activare indică dacă (sau în ce măsură) acest caracter
este prezent la fiecare poziție a imaginii.
Exemplul 13.8: În Fig. 13.10 (b), vedem un filtru 2 × 2, care urmează să fie aplicat
la imaginea 4 × 4 din Fig. 13.10 (b). Pentru a face acest lucru, așezăm filtrul peste toate cele nouă
cele 2 × 2 pătrate ale imaginii. În figură, sugerăm plasarea filtrului
peste pătratul 2 × 2 din colțul din dreapta sus. După suprapunerea filtrului, noi
înmulțiți fiecare dintre elementele de filtrare cu elementul de imagine corespunzător și
apoi ia suma produselor. În principiu, atunci trebuie să adăugăm o părtinire
termen, dar în acest exemplu, vom presupune că părtinirea este 0.
1 0 1 0
0 1 0 1
1 1 0 0
0 0 0 1
1 0
0 −1
0 0 0
1 1 −1
−1 1 0
(a) 4 4 imagine
(b) 2 2 filtru
(c) 3 3 răspuns
Figura 13.10: Aplicarea unui filtru unei imagini
O altă modalitate de a privi acest proces este că transformăm filtrul într-un vector
prin concatenarea rândurilor sale, în ordine, și facem același lucru cu pătratul din
imagine. Apoi, luăm produsul punct al vectorilor. De exemplu, filtrul
poate fi gândit ca vectorul [1, 0, 0, −1] și pătratul din stânga sus
colțul imaginii poate fi considerat vectorul [1, 0, 0, 1]. Produsul dot
dintre acești vectori este 1 × 1 + 0 × 0 + 0 × 0 + (- 1) × 1 = 0. Astfel, rezultatul, prezentat
în Fig. 13.10 (c), are un 0 pentru intrarea sa din stânga sus.
Pentru un alt exemplu, dacă glisăm filtrul în jos pe un rând, produsul punct al
filtrul ca vector și vectorul format din primele două elemente ale
al doilea și al treilea rând al imaginii este 1 × 0 + 0 × 1 + 0 × 1 + (- 1) × 1 = −1.
Astfel, primul element al celui de-al doilea rând al convoluției este -1. ✷
Când avem de-a face cu imagini color, intrarea are trei canale. Adică fiecare
pixelul este reprezentat de trei valori, una pentru fiecare culoare. Să presupunem că avem o culoare
imagine de dimensiune 224 × 224 × 3. Ieșirea filtrului va avea, de asemenea, trei canale,
și astfel filtrul este acum codificat de o matrice de greutate 5 × 5 × 3 W și simplă
parametru de părtinire b. Harta de activare R rămâne încă 5 × 5, cu fiecare răspuns
dat de:

$$r_{ij} = \sum_{k=0}^{4} \sum_{l=0}^{4} \sum_{d=1}^{3} x_{i+k,\, j+l,\, d}\, w_{kld} + b \tag{13.2}$$

În exemplul nostru, ne-am imaginat un filtru de dimensiunea 5. În general, dimensiunea filtrului este
un hiperparametru al stratului convoluțional. Filtrele de dimensiunea 3, 4 sau 5 sunt cele mai multe

---

Pagina 571

deseori folosit. Rețineți că dimensiunea filtrului specifică doar lățimea și înălțimea
filtrul; numărul de canale al filtrului se potrivește întotdeauna cu numărul de

canale de intrare.
Harta de activare din exemplul nostru este puțin mai mică decât intrarea. În
în multe cazuri, este convenabil ca harta de activare să aibă aceeași dimensiune ca
intrare. Putem extinde repsonse folosind zero padding: adăugare suplimentară
rânduri și coloane de zerouri pentru tamponarea intrării. O umplere zero de p corespunde
pentru a adăuga p rânduri de zerouri fiecare în partea de sus și în jos și p coloane în stânga
și dreapta, mărind dimensionalitatea intrării cu 2p de-a lungul ambelor lățimi
și înălțime. O umplere zero de 2 în exemplul nostru ar mări dimensiunea de intrare
până la 228 × 228 și rezultă într-o hartă de activare cu dimensiunea 224 × 224, aceeași dimensiune ca
imaginea de intrare originală.
Al treilea hiperparametru de interes este pasul. În exemplul nostru, am presupus
că aplicăm filtrul în orice punct posibil al imaginii de intrare. Am putea
gândiți-vă, în loc să alunecați filtrul câte un pixel, de-a lungul lățimii și înălțimii
din intrare, corespunzând unui pas s = 1. În schimb, am putea glisa
filtrați de-a lungul lățimii și înălțimii imaginii cu doi sau trei pixeli odată,
corespunzând unui pas s de 2 sau 3. Cu cât pasul este mai mare, cu atât este mai mic
harta de activare comparativ cu intrarea.
Să presupunem că intrarea este un m × m pătrat de pixeli, ieșirea un n × n pătrat,
dimensiunea filtrului este f, pasul este s și umplerea zero este p. Se vede ușor că:
$n = (m - f + 2p) / s + 1$
(13,3)
În special, trebuie să fim atenți să alegem hiperparametrii astfel încât să fie uniform
împarte m - f + 2p; altfel am avea un aranjament nevalid pentru convorbiri
stratul lutional și majoritatea implementărilor software ar face excepție.
Ne putem gândi intuitiv la un filtru căutând o caracteristică de imagine, cum ar fi
o pată de culoare sau o margine. Clasificarea unei imagini necesită de obicei identificarea
multe caracteristici. Prin urmare, folosim multe filtre, ideal unul pentru fiecare caracteristică utilă.
În timpul instruirii CNN, sperăm că fiecare filtru va învăța să se identifice
una dintre aceste caracteristici. Să presupunem că folosim k filtre; pentru a păstra lucrurile simple, noi
constrângeți toate filtrele pentru a utiliza aceeași dimensiune, pas și umplutură zero. Apoi ieșirea
conține k hărți de activare. Prin urmare, dimensionalitatea stratului de ieșire este
n × n × k, unde n este dat de ecuația 13.3.
Setul de k filtre împreună constituie un strat convoluțional. Date de intrare
cu d canale, un filtru de dimensiunea f necesită parametri $df^2 + 1$ (parametrii greutății df^2)
eteri și 1 parametru de polarizare). Prin urmare, un strat convoluțional de k astfel de filtre
folosește parametri $k (df^2 + 1)$.
Exemplul 13.9: Continuând exemplul ImageNet, să presupunem că intrarea constă
de 224 × 224 × 3 imagini și folosim un strat convoluțional de 64 de filtre, fiecare de dimensiune
5, stride 1 și zero padding 2. Dimensiunea stratului de ieșire este 224 × 224 × 64.
Fiecare filtru are nevoie de 3 × 5 × 5 + 1 = 76 de parametri (inclusiv unul pentru prag)
iar stratul convoluțional conține 64 × 76 = 4864 parametri - ordine de
magnitudine mai mică decât numărul de parametri pentru un strat complet conectat
cu aceleași dimensiuni de intrare și ieșire. ✷

Pagina 572

13.4.2 Convoluție și corelație încrucișată

Această subsecțiune este un scurt ocol pentru a explica de ce rețelele neuronale convoluționale
sunt numite astfel. Nu este o condiție prealabilă pentru niciunul dintre celelalte materiale din acest document
capitol.
Stratul convoluțional este numit datorită asemănării pe care o are cu
operație de convoluție din analiza funcțională, care este adesea utilizată în semnal
prelucrarea și teoria probabilității. Funcțiile date f și g, definite de obicei
în domeniul timpului, convoluția lor (f ∗ g) (t) este definită ca:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)\, g(t - \tau)\, d\tau = \int_{-\infty}^{\infty} f(t - \tau)\, g(\tau)\, d\tau$$

Aici suntem interesați de versiunea discretă a convoluției, unde f și g sunt
definit peste numerele întregi:

$$(f * g)(i) = \sum_{k=-\infty}^{\infty}$$

$$f(k)\, g(i-k) = \sum_{k=-\infty}^{\infty} f(i-k)\, g(k)$$

Adesea convoluția este privită ca folosind funcția g pentru a transforma funcția f. În în acest context, funcția g este numită uneori nucleu. Când nucleul este finit, deci g (k) este definit doar pentru k = 0, 1, ..., m − 1, definiția se simplifică la:

$$(f * g)(i) = \sum_{k=0}^{m-1} f(i-k)\, g(k)$$

Putem extinde definiția la funcții bidimensionale:

$$(f * g)(i, j) = \sum_{k=0}^{m-1} \sum_{l=0}^{m-1} f(i-k, j-l)\, g(k, l)$$

Să definim h pentru a fi nucleul obținut prin răsucirea g, adică h (i, j) = g (−i, −j) pentru i, j ∈ {0, ..., m− 1}. Se poate verifica că convoluția f ∗ h a f cu nucleul răsturnat h este dat de:

$$(f * h)(i, j) = \sum_{k=0}^{m-1} \sum_{l=0}^{m-1} f(i+k, j+l)\, g(k, l) \tag{13,4}$$

Observați similaritatea ecuației 13.4 cu ecuația 13.1, ignorând termenul de părtinire b. Funcționarea stratului convoluțional poate fi gândită ca o convoluție a intrarea cu un nucleu răsturnat. Această similitudine este motivul pentru care convoluțional straturile sunt denumite astfel, iar filtrele sunt uneori numite nuclee.

Corelația încrucișată f⋆g este definită de (f⋆g) (x, y) = (f ∗ h) (x, y) unde h este versiunea răsturnată a g. Astfel, funcționarea stratului convoluțional poate de asemenea, poate fi văzută ca o corelație încrucișată a intrării cu filtrul.

Pagina 573

13.4.3 Straturile de punere în comun

Un strat de grupare ia ca intrare ieșirea unui strat convoluțional și produce o ieșire cu întindere spațială mai mică. Reducerea dimensiunii se realizează prin folosind o funcție de pooling pentru a calcula agregate pe regiuni contigue mici a intrării. De exemplu, am putea folosi pooling-ul maxim peste non-suprapunere 2 × 2 regiunile de intrare; în acest caz există un nod de ieșire corespunzător fiecare regiune 2 × 2 care nu se suprapune a intrării, valoarea ieșirii fiind valoarea maximă dintre cele 4 intrări din regiune. Agregarea funcționează independent pe fiecare canal al intrării. Stratul de ieșire rezultat este de 25% dimensiunea stratului de intrare. Există trei elemente în definirea unui strat de pooling:

1. Funcția de pooling, care este cel mai frecvent funcția max, dar ar putea în teorie să fie orice funcție agregată, cum ar fi media.
2. Dimensiunea f a fiecărui pool, care specifică faptul că fiecare pool folosește un pătrat f × f de intrări.
3. Pasul dintre piscine, analog pasului folosit în convolu-
stratul tional.

Cele mai frecvente cazuri de utilizare în practică sunt f = 2 și s = 2, care specifică 2x2 regiuni care nu se suprapun și f = 3, s = 2, care specifică 3 × 3 regiuni cu unele suprapuneri. Valorile mai mari ale lui f duc la pierderea prea mare a informațiilor în practică. Rețineți că operațiunea de punere în comun micșorează înălțimea și lățimea strat de intrare, dar păstrează numărul de canale. Funcționează independent pe fiecare canal al intrării sale. Rețineți că, spre deosebire de stratul de convoluție, nu este practică obișnuită de a utiliza umplutura zero pentru stratul maxim de grupare. Ponderea este adecvată dacă credem că caracteristicile sunt aproximativ in-

variantă la traduceri mici. De exemplu, s-ar putea să ne pese de rudă
locațiile caracteristicilor, cum ar fi picioarele sau aripile, și nu locațiile exacte ale acestora. În
astfel de cazuri de punere în comun pot reduce foarte mult dimensiunea stratului ascuns care se formează
intrarea în straturile ulterioare ale rețelei.
Exemplul 13.10: Să presupunem că aplicăm gruparea maximă cu dimensiunea = 2 și stride =
2 la ieșirea 224 × 224 × 64 a stratului convoluțional din Exemplul 13.9.
Rezultatul rezultat este de dimensiunea 112 × 112 × 64. ✷

13.4.4 Arhitectura CNN

Acum că am văzut elementele de bază ale rețelelor neuronale convoluționale,
le putem pune împreună pentru a construi rețele adânci. Un CNN tipic alternează
straturi convoluționale și de strângere, care se termină cu unul sau mai multe straturi complet conectate
care produc rezultatul final.
Exemplul 13.11: De exemplu, Fig. 13.11 este o arhitectură de rețea simplă pentru
clasificarea imaginilor ImageNet într-una din cele 1000 de clase de imagini, inspirate în mod liber de

---

Pagina 574

VGGnet. 9 Această rețea simplă alternează strict convoluțional și pooling
straturi. În practică, rețelele performante sunt bine adaptate sarcinii lor,
și poate stiva straturi convoluționale direct unul peste altul cu
strat de pooling ocazional între. Mai mult, există adesea mai mult de unul
strat complet conectat înainte de ieșirea finală. Intrarea în primul strat este
224 × 224, imagine cu 3 canale. Intrarea în fiecare strat ulterior este ieșirea
a stratului anterior.

Tipul stratului	Mărime	Număr filtru	tampon	Mărime ieșire
Convoluție	3	1	1	64	224 × 224 × 64
Max Pool	2	2	0	64	112 × 112 × 64
Convoluție	3	1	1	128	112 × 112 × 128
Max Pool	2	2	0	128	56 × 56 × 128
Convoluție	3	1	1	256	56 × 56 × 256
Max Pool	2	2	0	256	28 × 28 × 256
Convoluție	3	1	1	512

14 × 14 × 512
Max Pool
2
2
0
512
14 × 14 × 512
Convoluție
3
1
1
1024
14 × 14 × 1024
Max Pool
2
2
0
1024
7 × 7 × 1024
Complet conectat
1 × 1 × 1000
Figura 13.11: Straturi ale unei rețele neuronale convoluționale
Primul strat este un strat convoluțional, format din 64 de filtre, fiecare cu trei canale, cum ar fi cazul oricărui procesor de imagine color. Filtrele sunt 3 × 3, iar pasul este 1, deci fiecare 3 × 3 pătrat al imaginii este o intrare pentru filtrul. Există o unitate de zero-padding, deci numărul de ieșiri al acestuia stratul este egal cu numărul de intrări. Mai mult, observați că putem vizualiza rezultatul ca o altă matrice 224 × 224. Fiecare element al acestei matrice este format din 64 de filtre, fiecare dintre ele este un pixel cu 3 canale.
Ieșirea primului strat este alimentată către un strat de pool maxim, în care ne împărțim matricea 224 × 224 în 2 × 2 pătrate (deoarece atât dimensiunea cât și pasul sunt 2). Astfel, matricea 224 × 224 a devenit matrice 112 × 112 și există încă aceleași 64 de filtre.
La al treilea strat, care este din nou convoluțional, luăm 112 × 112 matrice de pixeli din al doilea strat ca intrare. Acest strat are mai multe filtre decât primul strat - 128 mai exact. Motivul intuitiv al creșterii este că primul strat recunoaște structuri foarte simple, precum marginile, și există nu prea multe structuri simple diferite. Cu toate acestea, al treilea strat ar trebui să fie recunoscând caracteristici ceva mai complexe, iar aceste caracteristici pot implica o 6 × 6 pătrat al imaginii de intrare, datorită grupării efectuate la al doilea strat. În mod similar, fiecare strat convoluțional ulterior ia intrări din precedent
9 K. Simonyan și A. Zussman, „Rețele convoluționale foarte profunde pentru imagini la scară largă Recunoaștere ”, arXiv: 1409–1556v6.

Câte noduri dintr-un strat convoluțional?

Ne-am referit la un nod al unui strat convoluțional ca „filtru”. Acea filtrul poate fi un singur nod sau, ca în Exemplul 13.11, un set de mai multe noduri, câte unul pentru fiecare canal. Când antrenăm CNN, stabilim greutățile pentru fiecare filtru, deci există relativ puține noduri. De exemplu, în primul strat în Exemplul 13.11, există 192 de noduri, trei pentru fiecare dintre cele 64 de filtre. Cu toate acestea, atunci când aplicăm CNN instruit la o intrare, aplicăm fiecare filtrați la fiecare pixel din intrare. Astfel, în Exemplul 13.11, fiecare dintre cele 64 filtrele primului strat se aplică la 224 × 224 = 50, 176 pixeli. Cheia de reținut este faptul că, deși CNN are o reprezentare succintă, aplicarea rețelei neuronale reprezentate la date necesită o semnificativă cantitatea de calcul. La fel, apropo, se poate spune și pentru celălalt forme specializate de rețea neuronală pe care le discutăm în secțiunea 13.5. stratul de grupare și filtrele sale pot reprezenta structuri de dimensiuni progresiv mai mari și complexități. Astfel, numărul de filtre a fost ales pentru a se dubla la fiecare stratul de convoluție.
În cele din urmă, al unsprezecelea și ultimul strat este un strat complet conectat. Are 1000 noduri, corespunzătoare celor 1000 de clase de imagini pe care încercăm să le învățăm recunoaște. Fiind complet conectat, fiecare dintre aceste 1000 de noduri ia toate 7 × 7 × 3 = 147 ieșiri din al 10-lea strat; factorul 3 provine din faptul că toate filtrele de straturile anterioare au trei canale. ✷

Proiectarea CNN și a altor arhitecturi de rețea profundă este încă mai multă artă decât
știință. Cu toate acestea, în ultimii ani, au apărut câteva reguli generale
care merită reținute:
1. Rețele profunde care utilizează multe straturi convoluționale, fiecare folosind multe mici
filtrele sunt mai bune decât rețelele superficiale care utilizează filtre mari.
2. Un model simplu este de a folosi straturi convoluționale care păstrează spațiul
extinderea intrării lor cu umplutură zero și reducerea dimensiunii efectuată
clusiv prin strângerea straturilor.
3. Pașii mai mici funcționează mai bine în practică decât pașii mai mari.
4. Este foarte util să aveți dimensiunea de intrare uniform divizibilă de 2 de multe ori.

13.4.5 Implementare și instruire

O examinare a ecuației 13.1 (generalizată, astfel încât filtrul este mai degrabă f × f decât
5 × 5) sugerează că putem exprima fiecare intrare în ieșirea convoluției ca
produsul punct al vectorilor, urmat de o sumă scalară. Pentru a face acest lucru, trebuie să ne aplatizăm

---

Pagina 576

filtrul F și regiunea corespunzătoare din intrarea în vectori. Considera
convoluția unui tensor m × m × 1 X (adică X este de fapt o matrice m × m)
cu un filtru f × f F și polarizare b, pentru a produce ca ieșire matricea n × n Z.
Acum explicăm cum să implementăm operația de convoluție folosind o singură
multiplicare vector-matrice.
Mai întâi aplatizăm filtrul F într- un vector g ^{2} × 1. Apoi creăm matrice
Y din X după cum urmează: fiecare regiune pătrată f × f a lui X este aplatizată în af ^{2} × 1
vector și toți acești vectori sunt aliniați ca coloane pentru a forma un singur f ^{2} × n ^{2}
matricea Y. Construiți vectorul n ^{2} × 1 b astfel încât toate intrările sale să fie egale cu
părtinire b. Apoi
z = Y ^{T} g + b
produce un vector ^{2} × 1 z. Mai mult, fiecare element al z este un singur element în
convoluție. Prin urmare, putem rearanja intrările din z într-o matrice n × n
Z care dă ieșirea convoluției.
Observați că matricea Y este mai mare decât intrarea X (aproximativ cu a
factorul f ^{2}), deoarece fiecare intrare a lui X se repetă de multe ori în Y. Prin urmare,
această implementare folosește multă memorie. Cu toate acestea, înmulțind o matrice și
un vector este extrem de rapid pe hardware-ul modern, cum ar fi Procesarea grafică
Unități (GPU-uri), și așa este metoda utilizată în practică.
Această abordare a circumvoluțiilor de calcul poate fi ușor extinsă la caz
de intrări cu mai multe canale. Mai mult, putem face față cazului
unde avem mai degrabă k filtre decât doar 1. Trebuie apoi să înlocuim vectorul
g cu o matrice df ^{2} × k G și utilizați o matrice Y mai mare (df ^{2} × n ^{2}). De asemenea, avem nevoie
pentru a utiliza o matrice de polarizare n ^{2} × k B, unde fiecare coloană repetă termenul de polarizare a
filtrul corespunzător. În cele din urmă, ieșirea convoluției este exprimată prin
o n ^{2} × k matrice C, cu o coloană pentru ieșirea fiecărui filtru, unde:
C = Y ^{T} F + B
Am explicat cum se efectuează trecerea înainte prin convolu-
stratul tional. În timpul antrenamentului, va trebui să ne propagăm înapoi prin strat.
Deoarece fiecare intrare în ieșirea convoluției este un produs punctat de vectori urmăriți
cu o sumă, putem folosi tehnicile din secțiunea 13.3.3 pentru a calcula derivatele
tives. Se pare că derivata unei convoluții poate fi exprimată și ca
o convoluție, dar nu vom intra în detalii aici.

13.4.6 Exerciții pentru secțiunea 13.4

Exercițiul 13.4.1: Să presupunem că imaginile sunt 512 × 512 și că vom folosi un filtru care este
3 × 3.
(a) Câte răspunsuri vor fi calculate pentru acest strat al unei CNN?
(b) Câtă umplutură zero este necesară pentru a produce o ieșire de dimensiuni egale
la intrare?

Pagina 577

(c) Să presupunem că nu facem nicio umplutură zero. Dacă ieșirea unui strat este
de intrare în următorul strat, după câte straturi nu va exista ieșire la
toate?
Exercițiul 13.4.2: Repetați exercițiul 13.4.1 (a) și (c) pentru cazul în care există
un pas de trei.

Exercițiul 13.4.3: Să presupunem că avem ieșirea unui m × m convoluțional
strat cu k filtre, fiecare având d canale. Aceste ieșiri sunt alimentate către un pooling
strat cu mărimea f și stride s. Câte valori de ieșire are stratul de pooling
legume și fructe?
Exercițiul 13.4.4: Pentru acest exercițiu, presupuneți că intrările sunt biți simpli 0 (alb)
și 1 (negru). Luați în considerare un filtru 3 × 3, ale cărui greutăți sunt w_{ij} , pentru $0 \le i \le 2$ și
$0 \le j \le 2$ și a cărui părtinire este b. Sugerează valori și părtiniri, astfel încât rezultatul
acest filtru ar detecta următoarele caracteristici simple:
(a) O delimitare verticală, unde coloana din stânga este 0 și celelalte două coloane
sunt 1.
(b) O margine diagonală, în care doar triunghiul de trei pixeli în partea superioară
colțul din dreapta sunt 1.
(c) un colț, în care pătratul 2 × 2 din dreapta jos este 0 și celălalt
pixeli sunt 1.

13.5 Rețele neuronale recurente

Așa cum CNN este o familie specializată de rețele neuronale pentru procesarea 2-
date de imagine dimensionale, rețelele neuronale recurente (RNN) sunt rețele specializate
conceput în mod special pentru prelucrarea datelor secvențiale. Datele secvențiale apar în mod natural
în multe setări: o propoziție este o succesiune de cuvinte; un videoclip este o secvență de
imagini; un ticker bursier este o succesiune de prețuri.
Luați în considerare un exemplu simplu din domeniul procesării limbajului. Fiecare intrare
este o propoziție, modelată ca o succesiune de cuvinte. După procesarea fiecărui prefix al
secvența, am dori să prezicem următorul cuvânt din propoziție; ieșirea
la fiecare pas este un vector de probabilitate între cuvinte. Exemplul nostru sugerează două
considerente cheie de proiectare:
1. Rezultatul în fiecare punct depinde de întregul prefix al propoziției
până în acel moment și nu doar ultimul cuvânt. Rețeaua trebuie să se mențină
ceva „amintire” a trecutului.
2. Modelul lingvistic de bază nu se schimbă între pozițiile din
secvență, deci ar trebui să folosim aceiași parametri (greutăți pentru fiecare dintre
noduri) la fiecare poziție.

---

s
X
y
V
U
s
X
y
V
U
s
X
y
s
s
X
y
V
U
W
n
n
n
W
V
U
W
W
W
(b) RNN derulat de lungime
(a) Unitatea de bază a unui RNN.
n.
...
0
1
1
1

2
2
2
Figura 13.12: Arhitectura RNN
Aceste considerații conduc în mod natural la un model de rețea recurent, în care noi efectuați aceeași operație la fiecare pas, cu intrarea la fiecare pas depinde de ieșirea din pașii anteriori. Figura 13.12 prezintă structura a unei rețele neuronale recurente tipice. Intrarea este o secvență $x_1, x_2, ..., x_n$, iar ieșirea este, de asemenea, o secvență $y_1, y_2, ..., y_n$. În exemplul nostru, fiecare intrare x_i reprezintă un cuvânt, iar fiecare ieșire y_i este un vector de probabilitate pentru următorul cuvânt în secvență. Intrarea x_i este de obicei codificată ca un vector 1 fierbinte - un vector de lungime egal cu numărul de cuvinte posibile, [10] cu un 1 în poziția corespunzătoare cuvântului introdus și un 0 în toate celelalte poziții. Acolo sunt două diferențe importante între o rețea neuronală generală și RNN:
1. RNN are intrări la toate (sau aproape toate) straturile și nu doar la primul strat.
2. Greutățile la fiecare dintre primele n straturi sunt constrânse să fie aceleași; aceste greutăți sunt matricile U și W din ecuația 13.5 de mai jos. Prin urmare, fiecare dintre primele n straturi are același set de noduri și corespunde nodurile din fiecare dintre straturi împart greutăți (și sunt astfel într-adevăr același nod), la fel ca și nodurile unui CNN care reprezintă locații diferite greutăți și sunt astfel într-adevăr același nod.
La fiecare pas t, avem un vector de stare ascuns s_t care funcționează ca memorie în care rețeaua codifică informații despre prefixul secvenței sale a văzut. Starea ascunsă la momentul t este o funcție a intrării la momentul t și starea ascunsă la momentul t - 1:
$s_t = f(Ux_t + Ws_{t-1} + b)$
(13,5)
[10] Deoarece ar putea exista, în principiu, un număr infinit de cuvinte, în practică am putea dedica componente ale vectorului doar la cele mai comune cuvinte sau la cele mai importante atât în cererea în cauză. Alte cuvinte ar fi reprezentate de un singur supliment componentă a vectorului.

Aici f este o funcție de activare neliniară, cum ar fi tanh sau sigmoid. U și W sunt matrici de greutăți, și b este un vector de părtiniri. Definim s_0 ca un vector al toate zerourile. Ieșirea la momentul t este o funcție a stării ascunse la momentul t, după fiind transformat de o matrice de parametri V și o funcție de activare g:
$y_t = g(Vs_t + c)$
În exemplul nostru, g ar putea fi funcția softmax pentru a se asigura că ieșirea este a vector de probabilitate valid.
RNN din Figura 13.12 are o ieșire la fiecare pas de timp. În unele aplicații cationi, cum ar fi traducerea automată, avem nevoie de o singură ieșire la sfârșitul anului fiecare propozitie. În astfel de cazuri, ieșirea unică a RNN este procesată în continuare de unul sau mai multe straturi complet conectate pentru a genera ieșirea finală.
Este cel mai simplu să presupunem că intrările RNN noastre sunt secvențe de lungime fixă de lungime n. În acest caz, derulăm pur și simplu RNN pentru a conține n pași de timp.
În practică, multe aplicații trebuie să se ocupe de secvențe de lungime variabilă, de exemplu, propoziții de diferite lungimi. Există două abordări pentru a face față acestei situații ziune:
1. Căptușeală zero. Remediați n pentru a fi cea mai lungă secvență pe care o procesăm și tamponați secvențe mai scurte pentru a avea lungimea n.
2. Bucketing. Grupați secvențele în funcție de lungime și construiți o separare RNN pentru fiecare lungime.
Se utilizează o combinație a acestor două abordări. Putem crea o găleată pentru un un număr mic de lungimi diferite și atribuiți o secvență cupei de cea mai scurtă lungime care este cel puțin la fel de lungă ca secvența. Apoi, într-o găleată folosim padding pentru secvențe care sunt mai scurte decât lungimea maximă pentru galeata aia.

13.5.1 Instruirea RNN-urilor

Folosim propagarea înapoi pentru a antrena un RNN, la fel cum am face orice rețea neuronală. Haideți să trecem printr-un exemplu. Să presupunem că intrarea noastră constă din secvențe de lungime n. Rețeaua noastră folosește funcția de activare tanh pentru actualizarea stării, softmax pentru ieșire, iar funcția de pierdere este entropia încrucișată. Din moment ce rețeaua are mai multe ieșiri, una la fiecare pas de timp, căutăm să minimalizăm eroarea totală e, definit ca suma pierderilor e_i la fiecare etapă i:
e =
n

$\sum_{i=1}$
e_i
Pentru a simplifica notarea, folosim următoarele convenții. Să presupunem că x și y sunt vectori, iar z este un scalar. Definim:
$\frac{dz}{dx} = \nabla_x z$
$\frac{dy}{dx} = J_x(y)$

Mai mult, să presupunem că W este o matrice și w este vectorul obținut prin concatenare rândurile lui W. Apoi:
$\frac{dz}{dW} = \frac{dz}{dw}\frac{dy}{dW} = \frac{dy}{dw}$
Aceste convenții se extind în mod natural și la derivatele parțiale.
Folosim backpropagation pentru a calcula gradienții erorii cu respect la parametrii de rețea. Ne concentrăm pe $\frac{de}{dW}$; gradienții pentru U și V sunt similar, și lăsat ca exerciții pentru cititor. Este clar că:
$\frac{de}{dW} = \sum_{t=1}^{n} \frac{de_t}{dW}$
Concentrându-ne pe pasul t, avem:
$\frac{de_t}{dW} = \frac{ds_t}{dW}\frac{de_t}{ds_t}$
Lăsăm ca exercițiu să verificăm că:
$\frac{de_t}{ds_t} = V^T(y_t - y_t)$
(13,6)
Setarea $R_t = \frac{ds_t}{dW}$, observăm că $s_t = \tanh(z_t)$, unde $z_t = Ws_{t-1} + Ux_t + b$, noi avem:
$R_t = \frac{ds_t}{dz_t}\frac{dz_t}{dW}$
Este simplu să verificăm dacă $\frac{ds_t}{dz_t}$ este matricea diagonală A definită de:
$a_{ij} = \begin{cases} 1 - s_{t_i}^2 & \text{când } i = j \\ 0 & \end{cases}$

in caz contrar
Ar trebui să fim atenți să observăm că z_t este o funcție a lui W atât direct, cât și
indirect, deoarece s_{t-1} depinde și de W. Deci trebuie să exprimăm $\frac{dz_t}{dW}$ ca sumă de
doi termeni:

$$\frac{dz_t}{dW} = \frac{\partial z_t}{\partial W} + \frac{\partial z_t}{\partial s_{t-1}} \frac{ds_{t-1}}{dW}$$

Se verifică cu ușurință că:

$$\frac{\partial z_t}{\partial s_{t-1}} = W^T$$

Forma lui $\frac{\partial s_t}{\partial W}$
este un pic mai complicat. Este o matrice B cu majoritatea zero intrări,
elementele nenule fiind intrări din s_{t-1}. Lăsăm calculul lui B

---

ca exercițiu pentru cititor. Acum, observând asta
$\frac{ds_{t-1}}{dW}$
este doar R_{t-1}, avem
reapariție:

$$R_t = A(B + W^T R_{t-1})$$

Setând $P_t = AB$ și $Q_t = AW^T$, vom termina cu:

$$R_t = P_t + Q_t R_{t-1} \tag{13,7}$$

Putem folosi această recurență pentru a stabili o evaluare iterativă a R_t și de acolo
de
$\frac{}{dW}$. Inițializăm iterația setând R_0 la matrice cu toate zerourile. Acest
metoda iterativă pentru calcularea gradienților unui RNN se numește Backpropa-
gation Through Time (BPTT), deoarece dezvăluie efectele pașilor anteriori anteriori
pe etapele de timp ulterioare.

13.5.2 Gradiente de dispariție și explozie

RNN-urile sunt un model simplu și atrăgător pentru învățarea din secvențe și
algoritmul BPTT este ușor de implementat. Din păcate, RNN
au un defect fatal care limitează utilizarea lor în multe aplicații practice. ei
sunt eficiente numai la învățarea conexiunilor pe termen scurt între elementele din apropiere
în ordine și ineficient la învățarea conexiunilor la distanță. Lung-
conexiunile la distanță sunt cruciale în multe aplicații; de exemplu, se poate
fie un număr arbitrar de cuvinte sau clauze care separă un verb sau un pronume de
subiectul cu care este asociat.
Pentru a înțelege cauza acestei limitări, haideți să derulăm Equa-
secțiunea 13.7:

$$\begin{aligned} R_t &= P_t + Q_t R_{t-1} \\ &= P_t + Q_t (P_{t-1} + Q_{t-1} R_{t-2}) \\ &= P_t + Q_t P_{t-1} + Q_t Q_{t-1} R_{t-2} \\ &\ldots \end{aligned}$$

Rezultă în cele din urmă:

$$R_t = P_t + \sum_{j=0}^{t-1} P_j \prod_{k=j+1}^{t}$$

Q_k
(13,8)
Din ecuația 13.8, este clar că contribuția pasului i la R_t este dată de:
R_i
$_t = P_i$
$_t$
$\prod$
$_{k=i+1}$
Q_k
(13,9)
Ecuația 13.9 include produsul mai multor matrice care arată ca diagonală
matrice A. Fiecare intrare în A este strict mai mică de 1. La fel ca produsul multora
numere, fiecare strict mai mică de 1, se apropie de zero pe măsură ce adăugăm mai multe multiplicări
bomboane, termenul $\prod$
$_t$
$_{k=i+1}$ Q_k se apropie de zero pentru $i \ll t$. Cu alte cuvinte,
gradientul la pasul t este determinat în întregime de câțiva pași de timp anteriori, cu

---

foarte puțină contribuție din pașii de timp mult mai anteriori. Acest fenomen se numește
problema degradării gradienților.
Ecuația 13.9 are ca rezultat gradienții de dispariție deoarece am folosit acțiunea tanh
funcția de activare pentru actualizarea stării. Dacă în schimb folosim alte funcții de activare
cum ar fi ReLU, ajungem la produsul multor matrice cu intrări mari,
rezultând problema exploziei gradienților. Gradienții care explodează sunt mai ușori
de manipulat decât gradienții care dispar, deoarece putem decupa gradientul la fiecare
pas pentru a se afla într-un interval fix. Cu toate acestea, RNN-urile rezultate încă au probleme
învățarea asociațiilor la distanță.

13.5.3 Memorie pe termen scurt (LSTM)

Modelul LSTM este un rafinament al modelului RNN de bază pentru a aborda problemele
lema învățării asociațiilor la distanță. În ultimii ani, LSTM a făcut-o
a devenit popular ca model de învățare secvențială de facto și a fost folosit
cu succes în multe aplicații. Să înțelegem intuiția din spatele
Modelul LSTM înainte de al descrie formal. Principalele elemente ale LSTM
modelul sunt:
1. Capacitatea de a uita informațiile prin curățarea lor din memorie. De exemplu,
atunci când analizăm un text, am putea dori să aruncăm informații despre un
propoziție când se termină. Sau atunci când se analizează secvența de cadre dintr-un
film, s-ar putea să dorim să uităm de locația unei scene atunci când
următoarea scenă începe.
2. Capacitatea de a salva informațiile selectate în memorie. De exemplu, când
procesăm recenziile produselor, s-ar putea să dorim să salvăm doar cuvinte care exprimă
opiniile (de exemplu, excelente, teribile) și ignoră alte cuvinte.
3. Capacitatea de a se concentra doar asupra aspectelor memoriei care sunt imediat
relevante. De exemplu, concentrați-vă doar pe informații despre personajele
scena curentă a filmului sau doar la subiectul propoziției în prezent
fiind analizat. Putem implementa acest accent folosind o arhivă pe două niveluri
tectură: o memorie pe termen lung care reține informații despre întreg
prefixul procesat al secvenței și o memorie de lucru restricționată
la elementele de relevanță imediată.
Modelul RNN are un singur vector de stare ascuns s_t la momentul t. LSTM
modelul adaugă un vector de stare suplimentar c_t, numit starea celulei, de fiecare dată t.
Intuitiv, starea ascunsă corespunde memoriei de lucru și stării celulei
corespunde memoriei pe termen lung. Ambii vectori de stare au aceeași lungime,
și ambele au intrări în intervalul [-1, 1]. Ne putem imagina memoria de lucru
având cele mai multe intrări aproape de zero, cu doar intrările relevante activate.
Ingredientul arhitectural care permite abilitatea de a uita, a salva și a
focalizarea este poarta. O poartă g este doar un vector de aceeași lungime ca un vector de stare

---

s; fiecare dintre intrările porții este între 0 și 1. Produsul Hadamard $_{11}$ $s \circ g$
ne permite să trecem selectiv prin anumite părți ale stării în timp ce filtrăm
alții. De obicei, un vector poartă este creat printr-o combinație liniară a ascunsului

starea și intrarea curentă. Apoi aplicăm o funcție sigmoidă pentru a-i „squash”
intrările se situează între 0 și 1. În general, un LSTM poate utiliza mai multe
tipuri de vectori de poartă, fiecare pentru un scop diferit. La momentul t, putem crea un
poarta g după cum urmează:

$$g = \sigma(Ws_{t-1} + Ux_t + b)$$

Aici W și U sunt matrici de greutate și b este un vector de polarizare.
La momentul t, calculăm mai întâi un vector de actualizare a stării candidat h_t bazat pe
starea ascunsă anterioară și intrarea curentă:

$$h_t = \tanh(W_h s_{t-1} + U_h x_t + b_h)$$

(13.10)

Rețineți că W, U și b cu indicele h sunt două matrice de greutate și o polarizare
vector pe care îl învățăm și îl folosim doar pentru a calcula h_t pentru fiecare t.
De asemenea, calculăm două porți, poarta de uitare f_t și poarta de intrare i_t. The
uita poarta determină ce aspecte ale memoriei pe termen lung păstrăm. The
poarta de intrare determină ce părți ale actualizării stării candidate trebuie salvate în
memorie pe termen lung. Aceste porți sunt calculate folosind diferite matrice de greutate
și vectori de părtinire, care, de asemenea, trebuie învățați. Indicăm aceste matrice și
vector cu indicele f și respectiv i.

$$f_t = \sigma(W_f s_{t-1} + U_f x_t + b_f)$$

(13.11)

$$eu_{nu} = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

(13.12)

Actualizăm memoria pe termen lung folosind porțile și actualizarea candidatului
vector după cum urmează: [12]

$$c_t = c_{t-1} \circ f_t + h_t \circ i_t$$

(13.13)

Acum că am actualizat memoria pe termen lung, trebuie să actualizăm
memorie. Facem acest lucru în doi pași. Primul pas este crearea unei porți de ieșire
o_t. Al doilea pas este de a aplica această poartă pe memoria pe termen lung, urmată de
o activare tanh: [13]

$$o_t = \sigma(W_u s_{t-1} + U_u x_t + b_u)$$

(13.14)

$$s_t = \tanh(c_t \circ o_t)$$

(13.15)

[11] Produsul Hadamard al vectorilor $[x_1, x_2, ..., x_n]$ și $[y_1, y_2, ..., y_n]$ este vectorul a cărui
componentele sunt produsele componentelor corespunzătoare ale celor doi vectori de argumente,
adică $[x_1 y_1, x_2 y_2, ..., x_n y_n]$. Aceeași operație poate fi aplicată oricăror matrice care au
aceleași dimensiuni.

[12] Din punct de vedere tehnic, o intrare de 1 în poarta de uitare are ca rezultat păstrarea memoriei corespunzătoare
intrare; deci poarta uitării ar trebui să fie numită cu adevărat poarta amintirii. În mod similar, poarta de intrare
s-ar putea numi mai bine poarta de salvare. Aici urmăm convenția de denumire folosită în mod obișnuit
in literatura.

[13] Încă o dată, poarta de ieșire ar putea fi denumită mai bine poarta de focalizare, deoarece focalizează
memoria de lucru pe anumite aspecte ale memoriei pe termen lung.

Pagina 584

Aici, folosim indicele u pentru a indica două matrici de greutate suplimentare și o polarizare
vector - cele referitoare la rezultat - care trebuie învățat.
În cele din urmă, ieșirea la momentul t este calculată exact în același mod ca
Ieșire RNN:

$$y_t = g(V s_t + d)$$

(13,16)

unde g este o funcție de activare, V este o matrice de greutate și d este un vector de polarizare.
Ecuațiile 13.10 până la 13.16 descriu operațiunile de actualizare a stării la a
pasul t de timp unic al unui LSTM. Ne putem gândi la RNN simplu ca la un caz special
din LSTM. Când setăm poarta uitării la toate 0 (așa că le aruncăm pe toate dinainte
memorie pe termen lung) și poarta de intrare către toate 1 (salvează întreaga stare candidată
actualizare), și poarta de ieșire către toate 1 (memoria de lucru este aceeași ca pe termen lung
memorie), obținem ceva care arată foarte aproape de un RNN, singura diferență
fiind un factor tanh suplimentar.
Abilitatea de a uita selectiv permite LSTM-urilor să evite dispariția-
problemă de gradient în detrimentul introducerii multor mai mulți parametri decât
vanilie RNN's. Deși nu vom oferi o dovadă riguroasă aici, observăm că
cheia pentru evitarea gradienților de dispariție este actualizarea pe termen lung a memoriei în
secțiunea 13.13. Au fost propuse mai multe variante ale modelului LSTM de bază;

cea mai obișnuită variantă este modelul de unitate recurentă închisă (GRU), care folosește un singur vector de stare în loc de doi vectori de stare (pe termen lung și pe termen scurt). Un GRU are mai puțini parametri decât un LSTM și ar putea fi potrivit în unele situații cu seturi de date mai mici.

13.5.4 Exerciții pentru secțiunea 13.5

Exercițiul 13.5.1: În acest exercițiu, vi se cere să proiectați greutățile de intrare pentru unul sau mai multe noduri ale stării ascunse a unui RNN. Intrarea este o secvență de biți, doar 0 sau 1. [14] Rețineți că puteți utiliza alte noduri pentru a ajuta la nod solicitat. De asemenea, rețineți că puteți aplica o transformare la ieșirea din nod, astfel încât un răspuns „da" are o valoare și un răspuns „nu" are o altă valoare.

(a) Un nod pentru a semnaliza când intrarea este 1 și intrarea anterioară este 0.

! (b) Un nod pentru a semnaliza când ultimele trei intrări au fost toate 1.

!! (c) Un nod pentru a semnaliza când intrarea este aceeași cu intrarea anterioară.

! Exercițiul 13.5.2: Verificați ecuația 13.6.

! Exercițiul 13.5.3: Dați formulele pentru gradienții de
dU
și de
dV
pentru gen-
RNN eral din Fig.13.12.

[14] Ar trebui să înțelegem că RNN, ca orice rețea neuronală, trebuie învățat din date, nu a fost conceput așa cum vă sugerăm să faceți aici.

13.6 Regularizare

Până în prezent, ne-am prezentat obiectivul ca unul de minimizare a pierderilor (adică predicție eroare) pe setul de antrenament. Coborâre în gradient și coborâre în gradient stocastic ajută-ne să atingem acest obiectiv. În practică, obiectivul real al antrenamentului este pentru a minimiza pierderile de intrări noi și nevăzute până acum. Speranța noastră este că setul nostru de formare este reprezentativ pentru intrările viitoare necunoscute, deci o pierdere mică pe setul de instruire se traduce prin performanțe bune la intrări noi. Din păcate, modelul instruit uneori învață idiosincrasia datelor de instruire care permit are pierderi reduse de antrenament, dar nu generalizează bine la intrări noi - familiare problema suprasolicitării.

Cum ne putem da seama dacă un model are costum? În general, împărțim disponibilul date într-un set de antrenament și un set de testare. Antrenăm modelul folosind doar date de set de antrenament, reținând setul de testare. Apoi evaluăm performanța a modelului de pe setul de testare. Dacă modelul se comportă mult mai prost pe setul de testare decât pe setul de antrenament, știm că modelul are overfit. Presupunând puncte de date sunt independenți unul de celălalt, putem alege o fracțiune din datele disponibile puncte la întâmplare pentru a forma setul de testare. Un raport comun pentru antrenament: test împărțirea este 80:20. adică 80% din datele pentru antrenament și 20% pentru test. Noi trebuie sa fiți atenți, totuși: în problemele de învățare secvențială (de exemplu, modelarea seriilor temporale), starea secvenței în orice moment al timpului codifică informații despre trecut. În astfel de cazuri, piesa finală a secvenței este un set de testare mai bun.

Suprapunerea este o problemă generală care afectează toate modelele de învățare automată. Cu toate acestea, rețelele neuronale profunde sunt deosebit de susceptibile la supraadaptare, deoarece utilizează mult mai mulți parametri (greutăți și părtiniri) decât alte tipuri de modele. Au fost dezvoltate mai multe tehnici pentru a reduce supraadaptarea în adâncime rețelelor, de obicei prin tranzacționarea unei erori de formare superioară pentru o mai bună generalizare. The procesul este denumit regularizarea modelului. În această secțiune descriem câteva dintre cele mai importante metode de regularizare pentru învățarea profundă.

13.6.1 Penalizări ale normelor

Coborârea în gradient nu este garantată pentru a învăța parametrii (greutăți și părtiniri) care reduc pierderile de antrenament la un minim absolut. În practică, procedure învață parametri care corespund unui minim local în cadrul formării pierderi. De obicei, există multe minime locale, iar unele ar putea duce la o mai bună generare eralizare decât altele. În practică, s-a observat că soluțiile unde greutățile învățate au valori absolute scăzute tind să se generalizeze mai bine decât soluții cu greutăți mari.

Putem forța coborârea în gradient pentru a favoriza soluțiile cu valori de greutate reduse până la adăugând un termen funcției de pierdere. Să presupunem că w este vectorul întregii greutăți valorile din model, iar L_0 este funcția de pierdere utilizată de modelul nostru. Definim a funcție nouă de pierdere L după cum urmează:

$$L = L_0 + \alpha w^2 \quad (13,17)$$

Funcția de pierdere L penalizează valorile mari ale greutății. Aici αeste un hiperparametru care se schimbă între minimizarea funcției originale de pierdere L_0 și penalizare asociat cu L_2 -normul greutăților. În loc de L_2 -normă, am putea penalizați L_1 -norma greutăților:

$$L = L_0 + \alpha \sum_i |w_i| \quad (13.18)$$

În practică, se observă că pedeapsa normală L_2 funcționează cel mai bine pentru majoritatea aplicații. Pedeapsa normală L_1 este utilă în unele situații care necesită model compresie, deoarece tinde să producă modele în care sunt multe dintre greutăți zero.

13.6.2 abandon

Abandonul este o tehnică care reduce supraadaptarea prin modificări aleatorii rețeaua neuronală profundă subiacentă. Amintiți-vă că atunci când ne antrenăm folosind stochastic coborâre în gradient, la fiecare pas, eșantionăm la întâmplare un minibatch de intrări la proces. Când folosim abandonul, selectăm, de asemenea, la întâmplare o anumită fracțiune (să spunem jumătate) din toate nodurile ascunse din rețea și ștergeți-le, împreună cu orice margini conectate la acestea. Apoi efectuăm propagare înainte și înapoi propagarea pentru minibatch folosind această rețea modificată și actualizați greutăți și prejudecăți. După procesarea minibatch-ului, restaurăm toate cele șterse noduri și margini. Când eșantionăm următorul minibatch, ștergem altul subset aleatoriu de noduri și repetați procesul de instruire.

Fracția de noduri ascunse șterse de fiecare dată este un hiperparametru numit rata abandonului. Când antrenamentul este complet și folosim de fapt întregul rețea, trebuie să luăm în considerare faptul că întreaga rețea conține o rețea mai mare numărul de noduri ascunse decât rețelele utilizate pentru instruire. Prin urmare, avem nevoie pentru a scala greutatea pe fiecare margine de ieșire dintr-un nod ascuns de abandon rată.

De ce abandonul excesiv reduce supraadaptarea? Au fost puse mai multe ipoteze înainte, dar poate cel mai convingător argument este că abandonul permite o rețea neuronală unică pentru a se comporta eficient ca o colecție de rețele neuronale. Imaginați-vă că avem o colecție de rețele neuronale, fiecare cu o altă topologie de rețea. Să presupunem că am instruit fiecare rețea în mod independent folosind date de instruire și a folosit un fel de schemă de votare sau de calculare a mediilor pentru a crea un model de nivel superior. O astfel de schemă ar funcționa mai bine decât oricare dintre rețele viduale. Tehnica abandonului simulează această configurare fără a fi explicită crearea unei colecții de rețele neuronale.

13.6.3 Oprirea timpurie

În secțiunea 13.6.1 am sugerat iterarea prin exemple de instruire (sau mini-loturi) până când atingem un minim local în funcția de pierdere. În practică, acest lucru abordarea duce la supra-dotare. S-a observat că în timp ce pierderea pe

setul de antrenament (pierderea antrenamentului) scade prin procesul de antrenament, pierderea pe setul de testare (pierderea testului) se comportă adesea diferit. Pierderea testului scade pe durata parte inițială a antrenamentului și apoi mulți au atins un minim și de fapt crește după un număr mare de iterații de antrenament, chiar dacă pierderea antrenamentului continuă să cadă.

Intuitiv, punctul în care pierderea testului începe să crească este punctul în pe care procesul de instruire a început să învețe idiosincrazii ale instruirii mai degrabă decât un model generalizabil. O abordare simplă pentru a evita acest lucru problema este oprirea antrenamentului atunci când pierderea testului încetează să scadă. Există, cum vreodată, o problemă subtilă cu această abordare: s-ar putea să ne potrivim din greșeală datele de testare (mai degrabă decât datele de antrenament) prin oprirea antrenamentului la punctul respectiv a pierderii minime de testare. Prin urmare, eroarea de test nu mai este o măsură de încredere a adevăratei performanțe a modelului pe intrări nevăzute până acum.

Soluția obișnuită este utilizarea unui al treilea subset de intrări, setul de validare, la stabiliți punctul în care oprim antrenamentul. Împărțim datele nu doar în seturi de instruire și testare, dar în trei grupe: instruire, validare și testare.

Atât seturile de validare, cât și cele de testare sunt reținute din procesul de instruire. Cand

pierderea din setul de validare încetează să scadă, oprim procesul de instruire.
Deoarece setul de testare nu a jucat niciun rol în procesul de instruire, eroarea de testare
rămâne un indicator fiabil al adevăratei performanțe a modelului.

13.6.4 Augmentarea setului de date

Precizia majorității modelelor de învățare automată crește atunci când furnizăm
date de pregătire condiționată. De obicei, seturile de antrenament mai mari conduc, de asemenea, la o supra-dotare mai mică.
Când datele reale de formare disponibile sunt limitate, putem crea adesea
exemple sintetice de antrenament sintetice prin aplicarea de transformări sau adăugarea de zgomot.
De exemplu, luați în considerare problema de clasificare a cifrelor în care am întâlnit
Exemplul 13.7. Este clar că dacă rotim o imagine corespunzătoare unei cifre
cu câteva grade, rămâne în continuare o imagine cu aceeași cifră. Putem mări
datele de instruire prin aplicarea sistematică a transformărilor de acest fel. unu
modalitatea de a gândi acest proces este ca o modalitate de a codifica cunoștințe suplimentare de domeniu
(de exemplu, o imagine ușor distorsionată a unei pisici este încă o imagine a unei pisici).

13.7 Rezumatul capitolului 13

♦ Rețele neuronale: o rețea neuronală este o colecție de perceptroni (noduri), de obicei
organizate în straturi, unde ieșirile dintr-un strat oferă intrări către
straturile următoare. Primul strat (inout) ia intrări externe, iar
ultimul strat (ieșire) indică clasa intrării. Alte straturi din
mijlocul se numește straturi ascunse și, în general, sunt instruiți să recunoască
concepte intermediare necesare pentru determinarea rezultatului.

♦ Tipuri de straturi: Multe straturi sunt complet conectate, ceea ce înseamnă că fiecare nod
în strat are toate nodurile stratului anterior ca intrări. Alte straturi

---

sunt grupate, ceea ce înseamnă că nodurile stratului anterior sunt partiționate,
iar fiecare nod al acestui strat ia ca intrare doar nodurile unui bloc
a partiției. Straturile convoluționale sunt, de asemenea, utilizate, în special în imagine
procesarea cererilor.

♦ Straturi convoluționale: straturile convoluționale pot fi vizualizate ca și cum ar fi nodurile lor
au fost organizate într-o matrice bidimensională de pixeli, cu fiecare pixel
reprezentată de aceeași colecție de noduri. Greutățile corespunzătoare
nodurile din pixeli diferiți trebuie să fie aceleași, deci sunt efectiv aceleași
nod, și trebuie să învățăm doar un set de greutăți pentru fiecare familie de noduri,
câte unul din fiecare pixel.

♦ Funcții de activare: este determinată ieșirea unui nod într-o rețea neuronală
luând mai întâi suma ponderată a intrărilor sale, folosind greutățile care sunt
învățat în timpul procesului de formare a rețelei. O funcție de activare
se aplică apoi acestei sume. Funcțiile comune de activare includ
funcția sigmoidă, tangenta hiperbolică, softmax și diverse forme de
funcții liniare recificate ale unității.

♦ Funcții de pierdere: acestea măsoară diferența dintre rezultatul
net și ieșirea corectă în funcție de setul de antrenament. Deseori folosit
funcțiile de pierdere includ pierderea de eroare pătrată, pierderea Huber, pierderea clasificării,
și pierderea entropiei încrucișate.

♦ Antrenarea unei rețele neuronale: instruim o rețea neuronală calculând în mod repetat
ieșirea rețelei pe exemple de instruire și calcularea mediei
pierderea exemplelor de instruire. Greutățile pe noduri sunt apoi ajustate cu
propagarea pierderii înapoi prin net, folosind propagarea înapoi
algoritm.

♦ Propagare înapoi: alegând funcțiile noastre de activare și funcțiile de pierdere
pentru a fi diferențiat, putem calcula o derivație a funcției de pierdere cu
respectă fiecare greutate din rețea. Astfel, putem determina
direcția în care să reglați fiecare greutate pentru a reduce pierderea. Folosind lanțul
regula, aceste direcții pot fi calculate strat cu strat, de la ieșire la
Intrarea.

♦ Rețele neuronale convoluționale: acestea constau de obicei dintr-un număr mare
ber de straturi convoluționale, împreună cu straturi reunite și complet conectate
straturi. Sunt potrivite pentru prelucrarea imaginilor, unde primul convolu-
straturile operaționale recunosc caracteristici simple, cum ar fi marginile și straturile ulterioare
recunoaște trăsături progresiv mai complexe.

♦ Rețele neuronale recurente: acestea sunt concepute pentru a recunoaște secvențe,

precum propoziții (secvențe de cuvinte). Există un strat pentru fiecare poziție
în secvență, iar nodurile sunt împărțite în familii, pe care fiecare le are
câte un nod la fiecare strat. Nodurile unei familii sunt constrânse să aibă

---

aceleași greutăți, deci procesul de instruire trebuie, prin urmare, să se ocupe de un
număr relativ mic de greutăți.

♦ Rețele de memorie pe termen scurt: acestea se îmbunătățesc pe RNN prin adăugare
un al doilea vector de stare - starea celulei - pentru a permite unele informații despre
secvența care trebuie păstrată, în timp ce majoritatea informațiilor sunt uitate după un
in timp ce. În plus, învățăm vectori de poartă care controlează ce sunt informațiile
reținute de la intrare, stare și ieșire.

♦ Evitarea supra-dotării: există o serie de tehnici specializate
semnat pentru a evita dotarea excesivă a unei rețele profunde. Acestea includ penalizarea celor mari
greutăți, scăpând aleatoriu unele noduri de fiecare dată când aplicăm un pas de
coborârea în gradient și utilizarea unui set de validare pentru a ne permite să oprim antrenamentul
când pierderea din validare a scăzut.

13.8 Referințe pentru capitolul 13

Pentru informații despre TensorFlow, consultați [12]. Puteți afla despre PyTorch la [10]
și despre Caffe la [1]. Baza de date MNIST este descrisă în [9].
Propagarea înapoi ca modalitate de a antrena plasele neuronale profunde este de la [11].
Ideea rețelelor neuronale convoluționale începe cu [2], care a definit
straturi convoluționale și straturi de grupare. Cu toate acestea, a fost introdus [13]
ideea de a necesita noduri într-un singur strat convoluțional pentru a împărți greutăți. The
aplicarea acestor rețele la recunoașterea caracterelor și la alte sarcini importante
este din [8] și [7]. De asemenea, consultați [6] pentru aplicația la ImageNet a CNN's.
ImageNet este disponibil de la [5].
Rețelele neuronale recurente au apărut pentru prima dată în [4]. Memorie pe termen scurt
este din [3].

1. http://caffe2.ai
2. Fukushima, K., „Neocognitron, un model de rețea neuronală auto-organizat
pentru un mecanism de recunoaștere a modelelor neafectat de schimbarea poziției "
Cibernetică biologică 36: 1 (1980), pp. 193–202.
3. Hochreiter, S. și J. Schmidhuber, „Long Short-Term Memory”, Neural
Calculul 9: 8 (1997), pp. 1735-1780.
4. JJ Hopfield, „Rețele neuronale și sisteme fizice cu colere emergente
abilități de calcul selectiv, ”Proceedings of the National Academy of
Științe 79: 8 (1982), pp. 2554-2558.
5. http://www.image-net.org.
6. Krizhevsky, A, I. Sutskever și GE Hinton, „Clasificarea imaginilor cu
rețele neuronale convoluționale profunde, ”Advances in Neural Information
Sisteme de procesare, pp. 1097-1105, 2012.

---

7. LeCun, Y. și Y. Bengio, „Rețele convoluționale pentru imagini, vorbire,
și serii cronologice, ”The Handbook of Brain Theory and Neural Networks
(M. Arbib, ed.) 3361: 10 (1995).
8. LeCun, Y., B. Boser, JS Denker, D. Henderson, RE Howard și
W. Hubbard, „Propagarea înapoi aplicată recunoașterii codului poștal scris de mână
tion, ”Neural Computation 1: 4 (1989) pp. 541-551.
9. LeCun, Y., C. Cortes și CJC Burges, „Baza de date MNIST
cifre scrise ”, http: // http: //yann.lecun.com/exdb/mnist/
10. http://pytorch.org
11. Rumelhart, DE, GE Hinton și RJ Williams, „reprezentanți ai învățării”
de erori de propagare înapoi, Nature 323 (1986), pp. 533-536.
12. http://www.tensorflow.org
13. Waibel, A., T. Hanazawa, GE Hinton, K. Shikano și KJ Lang,
,,Recunoașterea fonemelor folosind rețele neuronale cu întârziere în timp”, IEEE Trans-
acțiuni asupra acusticii, vorbirii și procesării semnalului 37: 3 (1989), pp. 328–
339.

Index

Algoritmul A-Priori, 224, 225, 231
Pagina accesibilă, 199
Suma acumulată, 513
Impuritate de precizie, 508
Acțiune, 44
Funcția de activare, 531
Harta de activare, 549
Învățare activă, 470
Interogare ad-hoc, 146
Matricea de adiacență, 375
Adomavicius, G., 352
Publicitate, 17, 128, 216, 293
AdWords, 302
Model de grafic de afiliere, 383
Afrati, FN, 77, 78, 426
Grupare aglomerativă, vezi Ierar-
grupare chicală
Agregare, 34, 38
Agrawal, R., 250
Alexandrov, A., 78
Algoritm, 1
Problema tuturor perechilor, 64, 68
Alon, N., 174
Algoritmul Alon-Matias-Szegedy, 158
Amplificare, 113
Interogare analitică, 60
ȘI-construcție, 113
Anderson, C., 352, 353
Andoni, A., 141
ANF, vezi Cartierul aproximativ
funcție
Algoritmul ANF, 419
Apache, 24, 25, 78, 79
Funcția de vecinătate aproximativă, 419
Arbib, M ,, 569
Arc, 405
Arhivă, 144
Întrebați, 204
Regula de asociere, 217, 219
Asociativitate, 27, 45
Atribut, 33
Licitație, 305
Augmentarea datelor, 567
Austern, MH, 79
Autoritate, 204
În medie, 156
Arborele B, 292
Babcock, B., 174, 292
Babu, S., 174
Propagarea înapoi, 539
Propagarea înapoi în timp, 561
Backstrom, L., 426
Punct rău, 489
Geanta, 40, 85
Algoritm de echilibru, 305
Balazinska, M., 78
Trupa, 100
Lățime de bandă, 22
Coș, vezi Coș de piață, 214, 216,
217, 246
495. Coborârea în gradient a lotului, 495
Învățarea în serie, 469
Plasa Bayes, 5
Algoritm BDMO, 283
218. Bere și scutece

Bell, R., 353
Haosul pragmatic al lui Bellkor, 322
Bengio, Y., 569
Berges, CJC, 570
Bergmann, R., 78
Berkhin, P., 212
571

Pagina 592

572
INDEX
Berrar, DP, 461
Betweenness, 363
Algoritmul BFR, 266, 269
BFS, consultați prima căutare în lățime
Bi-clica, 370
Vector de părtinire, 530
Bickson, D., 79
Licitație, 303, 305, 312, 313
Big data, 1
BigTable, 77
Bik, AJC, 79
Clasificare binară, 464
Biomarker, 217
Grafic bipartit, 299, 359, 369, 370
Algoritmul BIRCH, 292
Birrell, A., 79
Bitmap, 232, 233
Bloc, 13, 21, 192
Blockeel, H., 521
Blocarea proprietății, 43, 49
Blog, 200
Filtru Bloom, 152, 230
Bloom, BH, 174
Blum, A., 521
Bohannon, P., 78
Boldi, P., 426
Corecția Bonferroni, 6
Principiul lui Bonferroni, 5, 6
Marcaj, 198
Boral, H., 427
Borkar, V., 77, 78
Boser, B., 570
Bottou, L., 521
BPTT, consultați Propagarea înapoi
timp
Bradley, PS, 292
Prima căutare în lățime, 363
Breiman, L., 20
Comerciant cu amănuntul de cărămidă, 216, 320, 321
Brin, S., 212
Potrivire largă, 305
Broder, AZ, 20, 141, 212
Bu, Y., 78
Găleată, 10, 149, 164, 168, 230, 283
Buget, 304, 311
Budiu, M., 79
Sistem sincron în vrac, 51
Burges, CJC, 521
Burrows, M., 78
559
Caffe, 530, 569
Set de articole candidate, 227, 240
Pereche de candidați, 81, 100, 231, 234
Carey, M., 77, 78
Caracteristică categorică, 464, 511, 517
Starea celulei, 562
Centroid, 255, 258, 264, 267, 271

Chabbert, M., 353
Regula lanțului, 542
Chandra, T., 78
Chang, F., 78
Canal, 548
Matricea caracteristică, 89
Charikar, MS, 141
Chaudhuri, S., 141
Punct de control, 52
Chen, M.-S., 250
Copil, 364
Holeră, 4
Chowdhury, M., 79
Model de date cronice, 173
Chunk, 24, 240, 270
CineMatch, 349
Clasa, 464
Pierderea din clasificare, 537
Clasificator, 330, 463
Faceți clic pe flux, 145
Rata de clic, 297, 303
Clique, 369
Cloud computing, 17
Cluster computing, 21, 22
Arborele cluster, 278, 279
Clustera, 77
Clustering, 4, 17, 253, 337, 355, 361,
463
Clustroid, 258, 264
CNN, vezi Rețeaua neuronală convoluțională
Rețea de colaborare, 358

Filtrare colaborativă, 5, 18, 84, 293,
319, 333, 359
Colos, 24
Matricea ortonormală coloană, 443
Combiner, 27, 189, 191
Costul comunicării, 22, 54, 403
Comunitate, 18, 355, 366, 368, 399
Comutativitate, 27, 45
Raport competitiv, 17, 298, 301, 306
Grafic complet, 369, 370
Set comprimat, 270
Calculați graficul, 540
Calculează nodul, 21, 22
Joc de computer, 327
Concept, 444
Spațiu conceptual, 449
397. Conductanță, 397
Încredere, 218, 219
Recomandare bazată pe conținut, 319,
324
Convergență, 475
Convexitate, 517
Stratul convoluțional, 549, 551
Rețea neuronală convoluțională, 18, 523,
527, 548
Cooper, BF, 78
Coordonate, 254
Cortes, C., 521, 570
Distanta cosinus, 107, 117, 325, 330,
450
Numărătoare, 162, 283
Acoperind o ieșire, 66
Lista lui Craig, 294

Craswell, N., 317
Credit, 364
Cristianini, N., 521
Entropie încrucișată, 533, 538
Validare încrucișată, 469
Crowdsourcing, 470
CUR-descompunere, 429, 452
Algoritmul CURE, 274, 278
Currey, J., 79
Blestemul dimensionalității, 256, 280, 503, 518
Cut, 374, 375
Permutarea ciclică, 99
Cilindru, 13
Czajkowski, G., 79
DAG, vezi Graficul aciclic direcționat
Darts, 152
Das Sarma, A., 78
Das, T., 79
Dasgupta, A., 426
Exploatarea datelor, 1
Știința datelor, 1
Flux de date, 17, 244, 282, 296, 484
Sistem de gestionare a fluxului de date, 144
Baza de date, 17
Datar, M., 142, 174, 292
Algoritmul Datar-Gionis-Indyk-Motwani, 163
Dave, A., 79 de ani
De Raedt, L., 521
Punct mort, 179, 182, 183, 205
Dean, J., 78
Fereastra în descompunere, 169, 246
Pădure de decizie, 515, 518
Arborele decizional, 18, 330, 464, 467, 468, 505, 518
Învățare profundă, 3, 18, 523
Deerwester, S., 460
Grad, 371, 400
Matricea de grade, 376
Dehnert, JC, 79
del.icio.us, 326, 359
Ștergere, 108
Denker, JS, 570
Matricea densă, 31, 452
Densitate, 263, 265
Densitatea muchiilor, 396
Căutare în profunzime, 417
Determinant, 431
Set de dezvoltare, 468
DeWitt, DJ, 78 de ani
DFS, consultați Sistem de fișiere distribuit
Matricea diagonală, 443
Diametru, 263, 265, 406
Scutece și bere, 216
Diferență, 34, 37, 41

Pagina 594

Tabel de dimensiuni, 60
Reducerea dimensiunii, 18, 340, 429, 503
Grafic aciclic direcționat, 364
Grafic direcționat, 405
Aruncați setul, 270
Disc, 13, 221, 255, 278
Bloc disc, consultați Blocare

Anunț grafic, 294, 295
Măsura distanței, 105, 253, 361, 538
Elemente distincte, 154, 157
Sistem de fișiere distribuit, 21, 24, 214,
221
DMOZ, consultați Open directory
Document, 82, 86, 217, 254, 313, 325,
326, 466
Frecvența documentului, consultați Documente inverse
frecvența umentului
Domeniu, 202
Produs dot, 107, 544
Drineas, P., 460
Abandon, 566
Dryad, 77
DryadLINQ, 77
Construcție duală, 360
Dubitzky, W., 461
Dumais, ST, 460
Sarcină Dup-elimin, 49
535. DL ReLU, 535
e, 13
Editați distanța, 108, 110
Eigenpair, 431
Valoare proprie, 179, 377, 430, 440, 441
Eigenvector, 179, 377, 430, 435, 440
ELU, 535
E-mail, 358
Energie, 448
Ansamblu, 331
Metode de ansamblu, 516
Rezoluția entității, 122, 123
Entropie, 508, 537
Equijoin, 34
Erlingsson, I., 79
Ernst, M., 78
Ethernet, 21, 22
Distanța euclidiană, 105, 119, 502
Spațiul euclidian, 105, 109, 254, 255,
258, 274
Ewen, S., 78
Exclusiv-sau, 529
Explicabilitate, 3
Derivat exploziv, 531
Gradient exploziv, 562
Unitate liniară exponențială, vezi ELU
Fereastra în descompunere exponențială, vezi De-
fereastră caying
Extrapolare, 501
Facebook, 18, 198, 356
Tabel de informații, 60
Eșec, 23, 30, 43, 51
Faloutsos, C., 427, 461
Fals negativ, 100, 111, 239
Fals pozitiv, 100, 111, 152, 239
Familia de funcții, 112
Fang, M., 250
Fayyad, UM, 292
Feature, 278, 324-326
Selecție de funcții, 470
Caracteristică vector, 464, 517
Rețea Feedforward, 531
Pe deplin, D., 79
Fikes, A., 78
Dosar, 23, 24, 221, 239
Filtru, 44, 549
Filtrare, 151
Amprentă digitală, 125
Licitație la primul preț, 305
Punct fix, 114, 204

Flajolet, P., 174
Algoritmul Flajolet-Martin, 155, 419
Flatmap, 44
Turtire, 547
Flink, 77, 78
Grafic de flux, 42
Uitați poarta, 563
Fortunato, S., 426
Fotakis, D., 426
Franklin, MJ, 79

INDEX
575
Franceză, JC, 292
Găleată frecventă, 231, 233
Set de articole frecvente, 5, 214, 224, 226, 370,
463
Perechi frecvente, 225
Tabel cu articole frecvente, 226
Freund, Y., 521
Freytag, J.-C., 78
Prieteni, 356
Relația prietenilor, 59
Frieze, AM, 141
Norma Frobenius, 433, 447
Fukushima, K., 569
Strat complet conectat, 527
Furnas, GW, 460
Gaber, MM, 20
Ganti, V., 141, 292
Garcia-Molina, H., 20, 212, 250, 292,
427
Garofalakis, M., 174
Poarta, 562
Unitate recurentă închisă, 564
Eliminare gaussiană, 180
Gehrke, J., 174, 292
Generalizare, 469
Subgraf generat, 370
Gen, 324, 336, 350
GFS, consultați sistemul de fișiere Google
Ghemawat, S., 78
Gibbons, PB, 174, 427
Impuritatea GINI, 508
Gionis, A., 142, 174
Giraph, 77, 78
Girvan, M., 426
Algoritmul Girvan-Newman, 363
Minim global, 342
Algoritmul GN, vezi Girvan-Newman Al-
goritm
Gobioff, H., 78
Golub, GH, 460
Gonzalez, J., 79
Google, 176, 187, 302
Sistem de fișiere Google, 24
Google+, 356
GPU, consultați Unitatea de procesare grafică, 556
Coborâre în gradient, 18, 48, 348, 386, 492,
526, 529, 531, 539
Granzow, M., 461
Grafic, 51, 64, 356, 399, 406
Unitate de procesare grafică, 530
GraphLab, 77
GraphX, 79
Algoritm lacom, 296, 297, 300, 304
Algoritm GRGPF, 278

GroupByKey, 45 de ani
Grupare, 26, 34, 38
Atribut de grupare, 34
Groupon, 359
Grover, R., 78
GRU, vezi Unitate recurentă închisă
Gruber, RE, 78
Guestrin, C., 79
Guha, S., 292
Gunda, PK, 79
Gyongi, Z., 212
Produs Hadamard, 544, 563
Hadoop, 25, 79
Sistem de fișiere distribuite Hadoop, 24
HaLoop, 51, 77
Distanță de lovire, 74, 109, 117
Hanazawa, T., 570
Harris, M., 350
Harshman, R., 460
Funcția Hash, 10, 87, 92, 100, 149,
152, 155
Hash join, 37
Tasta Hash, 10, 312
Masă Hash, 10, 12, 13, 223, 230, 233,
234, 312, 314, 400
Haveliwala, TH, 212
HDFS, consultați sistemele de fișiere distribuite Hadoop-
tem
Cap, 413
Hitter greu, 400
Heise, A., 78 de ani
Hellerstein, JM, 79
Henderson, D., 570
Henzinger, M., 142

Pagina 596

Stratul ascuns, 525
Starea ascunsă, 558
Gruparea ierarhică, 255, 257, 275,
338, 361
Pierderea balamalei, 491
Hinton, GE, 569, 570
HITS, 204
Hive, 77, 79
Hochreiter, S., 569
Hoger, M., 78
417. Hopcroft, JE, 417
Hopfield, JJ, 569
Horn, H., 79
Howard, RE, 570
Howe, B., 78 de ani
Hsieh, WC, 78
Hub, 204
Hubbard, W, 570
Pierderea lui Huber, 536
Hueske, F., 78
Tangentă hiperbolică, 532, 562
Căutare subiect indusă de hyperlink, consultați HITS
Hiperparametru, 535
Hiperavion, 486
Hyracks, 77
Documente identice, 130
Matricea de identitate, 431
IDF, consultați Frecvența inversă a documentului
Imagine, 145, 325, 326
ImageNet, 548, 553, 569

IMDB, consultați Baza de date Internet Movie
Imielinski, T., 250
Subset imediat, 242
Immorlica, N., 142
Pagina importantă, 176
Impresie, 294
Impuritate, 507
În componentă, 181
Pagina inaccesibilă, 199
Rânduri sau coloane independente, 443
Index, 11, 400
Indyk, P., 141, 142, 174
Integrarea informațiilor, 6
Inițializați grupurile, 267
Intrare, 64, 464
Poarta de intrare, 563
Stratul de intrare, 525
Inserare, 108
Învățarea bazată pe instanțe, 467
Interes, 218
Internet Movie Database, 324, 350
Interpolare, 501
Intersecție, 34, 36, 40, 85
În aer subțire, 323
Frecvența inversă a documentului, 9, vezi TF.IDF
Indicele inversat, 176, 294
Ioannidis, YE, 426
Pachet IP, 145
Isard, M., 79
Componentă izolată, 182
Postul 214, 216, 217, 320, 336, 337
Profilul articolului, 324, 327
Set de articole, 214, 222, 224
Distanță Jaccard, 104, 106, 112, 325,
504
Similitudine Jaccard, 82, 91, 104, 199
Jacobian, 541
Jacobsen, H.-A., 78
Jagadish, HV, 174
Jahrer, M., 353
Jeh, G., 426
Joachims, T., 521
Alătură-te, vezi Alăturare naturală, 45, vezi Multi-
way join, vezi Star join, 401
Alătură-te sarcinii, 49
K-înseamnă, 266
Grafic K-partit, 359
Kahan, W., 460
Kalyanasundaram, B., 318
Kamm, D., 350
Kang, U., 427
Kannan, R., 460
Kao, O., 78
Karlin, A., 298
Kaushik, R., 141
Kautz, WH, 174
Kernel, 552

INDEX
577
Funcția kernel, 498, 502
Componenta cheie, 149
Pereche cheie-valoare, 25, 27
Cuvânt cheie, 303, 331
KL-divergență, 538
Kleinberg, JM, 212
Knuth, DE, 20

Koren, Y., 353
Krioukov, A., 78
Krizhevsky, A., 569
Kumar, R., 20, 79, 212, 426
Kumar, V., 20
Kyrola, A., 79
Etichetă, 356, 464
Multiplicatori lagrangeni, 58
Landauer, TK, 460
Lang, KJ, 426, 570
Matricea laplaciană, 376
Stratul, 523
Evaluare leneșă, 46
LCS, a se vedea cea mai lungă subsecvență comună
Frunza, 365
Leaky ReLU, 535
Rata de învățare, 472, 545
LeCun, Y., 569, 570
Leich, M., 78
Leiser, N, 79
Lungime, 158, 405
Indexarea lungimii, 131
Leser, U., 78
Leskovec, J., 426, 427
Leung, S.-T., 78
Li, P., 142
Probabilitate, 381
Lin, S., 142
Linden, G., 353
Liniage, 47
Ecuații liniare, 180
Separabilitate liniară, 471, 475
Închidere tranzitivă liniară, 410, 415
Link, 33, 176, 190
Matrice de legături de pe web, 205
Link spam, 195, 199
Littlestone, N., 521
Livny, M., 292
Minim local, 342
Localitate, 356
Familie sensibilă la localitate, 116
Funcție sensibilă la localitate, 112
Hash sensibil la localitate, 1, 81, 100,
111, 326, 504
Probabilitatea jurnalului, 387
Logaritm, 13
Sigmoid logistic, 532
Rețea de memorie pe termen scurt, 18,
523, 562
Coadă lungă, 216, 320, 321
Cea mai lungă subsecvență comună, 108
Scăzut, Y., 79
Limita inferioară, 68
Hiperplan inferior, 487
LSH, consultați Hashing sensibil la localitate
LSTM, consultați Memoria pe termen scurt
rețea
Ma, J., 79
Învățarea automată, 2, 18, 330, 463
Maggioni, M., 460
Maghoul, F., 20, 212
Distanța Mahalanobis, 273
Mahoney, MW, 426, 460
Memorie principală, 221, 222, 230, 255
Malewicz, G, 79
Malik, J., 427
Manber, U., 142
Distanța Manhattan, 106
Manning, CP, 20
Multe-multe potriviri, 126

Relație mulți-mulți, 64, 214
Multe-una potrivită, 126
Harta, 44
Sarcină hartă, 25, 27
Lucrător pe hartă, 28, 30
Schema de mapare, 65
MapReduce, 21, 25, 30, 189, 191, 241,
287, 401, 408, 483
Marja, 485
Coș de piață, 5, 17, 213, 214, 221
Markl, V., 78

Pagina 598

Procesul Markov, 179, 182, 390
Martin, GN, 174
Controler principal, 25, 27, 28
Nodul principal, 24
Potrivire, 299
Matias, Y., 174
Matrix, 31, vezi Matricea de tranziție a
web, vezi Mașina stochastică
trix, vezi Matricea substocastică,
189, 204, vezi Matricea utilitară,
340, vezi Matricea adiacenței, vezi
Matrice de grade, vezi Laplacian
matrice, vezi Matricea simetrică
Multiplicarea matricei, 38, 39, 48, 69
Matricea distanțelor, 441
Efectul Matei, 16
Ponderea maximă, 553
Set de articole maxime, 224
Potrivire maximă, 299
Estimarea probabilității maxime, 381
McAuley, J., 427
McCauley, M., 79
Mediu, vezi Media
Eroare medie pătrată, 536
Mecanic turc, 470
Mediană, 156
Mehta, A., 318
Melnik, S., 427
Fuzionarea grupurilor, 258, 261, 272, 276,
281, 285
Merton, P., 20
Miller, GL, 427
Minhashing, 82, 90, 103, 106, 113, 326
496
Minicluster, 270
Minsky, M., 522
Minutiile, 125
Mirrokni, VS, 142
Pagina oglindă, 83
Mitzenmacher, M., 141
ML, vezi Învățarea automată
MLE, vezi Estimarea maximă a probabilității
ziune
Set de date MNIST, 547
Model, 381
Modelare, 1
Momente, 157
Monotonicitate, 224
Montavon, G., 521
Moore-Penrose pseudoinvers, 453
Cele mai comune elemente, 169
Motwani, R., 142, 174, 250, 292
Mueller, K.-R., 521

Clasificare multiclass, 464, 479
Indicele multidimensional, 503
Algoritm Multihash, 234
Multiplicare, 31, vezi Matrix multi-
plication, 189, 204
Multiset, vezi Bag
Algoritmul cu mai multe etape, 232
Multiple join, 56, 402
Mumick, IS, 174
Mutație, 111
Nume nume, consultați Nodul principal
Alăturare naturală, 34, 37, 38, 55
Naughton, JF, 78
Naumann, F., 78
Navathe, SB, 250
Căutare vecinătate, vezi Locality-sens-
itive hashing
Cel mai apropiat vecin, 18, 464, 468, 497,
518
Frontieră negativă, 242
Exemplu negativ, 472
Vecin, 390
Vecinătate, 406, 418
Profil de vecinătate, 406
Provocare Netflix, 2, 322, 349, 516
Rețea, consultați Rețea socială
Rețea neuronală, 18, 467, 523, 524
Neuron, 527
Newman, MEJ, 426
Articole de ziar, 128, 313, 322
Nod, 508, 523
Tunderea nodului, 514
Distanță non-euclidiană, 264, vezi Co-
distanță sinusoidală, consultați Editare distanță,
vezi distanța Hamming, vezi Jac-
distanța cardului

INDEX
579
Spațiul non-euclidian, 278, 280
Normă, 105, 106
Pedeapsa normei, 565
Distribuție normală, 269
Normalizare, 333, 335, 346
Tăiere normalizată, 375
Problema NP-completă, 369
Caracteristică numerică, 464, 509, 517
O'Callaghan, L., 292
Algoritm off-line, 296
Olston, C., 79
Omiecinski, E., 250
Publicitate on-line, consultați Publicitate
Algoritm on-line, 17, 296
Învățare on-line, 469
Distribuitor on-line, 216, 294, 320, 321
Onose, N., 78
Director deschis, 196, 470
SAU-construcție, 114
Orr, GB, 521
Vectorii ortogonali, 256, 434
Matricea ortonormală, 443, 448
Vectorii ortonormali, 435, 438
Out-component, 181
În afara gradului, 416
Outlier, 255
Ieșire, 64, 464
Poarta de ieșire, 563

Stratul de ieșire, 525
Suprapunere, 331, 348, 467, 468, 481,
506, 514, 518, 565
Comunități suprapuse, 381
Uvertură, 303
Owen, AB, 142
Pagini proprii, 200
Paepcke, A., 142
Pagină, L., 175, 212
PageRank, 4, 17, 31, 32, 49, 175, 177,
189
Perechi, vezi Perechi frecvente
Palmer, CR, 427
Pan, J.-Y., 427
Papert, S., 522
Paralelism, 513
ReLU parametric, 535
Părinte, 364
Park, JS, 250
Partiție, 374
Pass, 222, 225, 233, 238
Calea, 405
Paulson, E., 78
PCA, vezi Analiza componentelor principale
sis
Algoritm PCY, 230, 233, 234
Pedersen, J., 212
Perceptron, 18, 463, 467, 471, 517, 523,
524
Potrivire perfectă, 299
Permutare, 90, 99
Peters, M., 78
Phishing, 2
PORC, 77
Principiul porumbelului, 369
Piotte, M., 353
Condensare pivotantă, 431
Plagiat, 83, 217
Alune, 77
Punct, 253, 283
Alocarea punctelor, 255, 266, 362
Polyzotis, A., 77
Funcția de punere în comun, 553
Stratul de îmbinare, 527, 553
Indexarea poziției, 133, 135
Exemplu pozitiv, 472
Număr întreg pozitiv, 168
Powell, AL, 292
Iterație de putere, 431, 432
Legea puterii, 14
Predicat, 330
Indexarea prefixelor, 132, 133, 135
Pregel, 51, 77
Vector propriu principal, 179, 431
Analiza componentelor principale, 429, 436
Coadă prioritară, 261
Preoți, 383
Confidențialitate, 296
Șir de sondă, 133

Pagina 600

Profil, vezi Profil articol, vezi Profil utilizator
fișier
Proiecție, 34, 36
Pruhs, KR, 318
Pseudoinvers, vezi pseudo-Moore-Penrose

doinvers
Puz, N., 78
PyTorch, 530, 570
Programare quadratică, 492
Interogare, 146, 165, 287
Exemplu de interogare, 498
Quinlan, JR, 522
Arborele R, 292
Rack, 22
Raza, 263, 265, 406
Raghavan, P., 20, 212, 426
Rahm, E., 427
Rajagopalan, S., 20, 212, 426
Ramakrishnan, R., 78, 292
Ramsey, W., 317
Hiperplane aleatorii, 117, 326
Strat de interconectare aleatorie, 527
Surfer aleatoriu, 176, 177, 182, 196, 390
Randomizare, 238
Rang, 442
Cel mai rar-primul ordin, 313
Rastogi, R., 174, 292
Evaluare, 320, 323
RDD, consultați Set de date distribuite rezistent
Accesibilitate, 407, 408, 415
Sistem de recomandare, 18, 319
Unitate liniară rectificată, vezi ReLU
Rețea neuronală recurentă, 18, 523, 557
Recursivitate, 49
Dublare recursivă, 411, 415
Reduceți sarcina, 25, 27
Reduceți lucrătorul, 28, 30
Reductor, 27, 45
Dimensiunea reductorului, 61, 67
Reed, B., 79
Regresie, 464, 502, 518
Pierderea de regresie, 536
Regularizare, 565
Parametru de regularizare, 490
Reichsteiner, A., 461
Reina, C., 292
Relație, 33
Algebra relațională, 33
ReLU, 534, 562
Replicare, 24
Rata de replicare, 61, 68
Reprezentare, 278
Punct reprezentativ, 275
Eșantion reprezentativ, 149
Eșantionarea rezervorului, 174
PageRank rezidual, 393
Set de date distribuit rezistent, 43
Răspuns, 549
Reporniți, 391
Set reținut, 270
Venituri, 304
Rheinlander, A., 78
Sumator de transportare a ondulațiilor, 168
RMSE, consultați Eroare rădăcină-medie-pătrat
RNN, vezi Rețea neuronală recurentă
Robinson, E., 78
Rocha, LM, 461
Eroare rădăcină medie-pătrat, 322, 341, 447,
536
Rosa, M., 426
Rosenblatt, F., 522
Date de rotunjire, 335
Rând, vezi Tuple
Matricea ortonormală de rând, 448
Rowsum, 278

Royalty, J., 78 de ani
Rumelhart, DE, 570
Curba S, 102, 111
Saberi, A., 318
Salihoglu, S., 78
Eșantion, 238, 242, 245, 247, 267, 275,
279
Eșantionare, 148, 162
Saturație, 531
Savasere, A., 250
Sax, MJ, 78

INDEX
581
SCC, vezi Compoziții puternic conectate
nent, vezi Strâns conectat
componentă
Schapire, RE, 521
Schelter, S., 78
Schema, 33
Schmidhuber, J., 569
Schutze, H., 20
Scor, 123
Căutare anunț, 294
Motor de căutare, 187, 203
Interogare de căutare, 145, 176, 198, 294, 312
Licitație la al doilea preț, 305
Stocare secundară, consultați Disk
Selecție, 33, 35
Evaluare seminaivă, 409, 411, 413,
414
Senzor, 145
Analiza sentimentului, 471
Set, 89, 131, vezi Set de articole
Setați diferența, consultați Diferența
Shankar, S., 78
Shawe-Taylor, J., 521
Shenker, S., 79
Shi, J., 427
Shikano, K., 570
Shim, K., 292
Zoster, 86, 103, 128
Shivakumar, N., 250
Coș de cumpărături, 216
Cele mai scurte căi, 52
Siddharth, J., 142
Sigmoid, 532
Semnătura, 82, 89, 91, 103
Matricea semnăturii, 92, 100
Silberschatz, A., 174
Silberstein, A., 78
Similaritate, 5, 17, 82, 213, 326, 334
Similitudine, 62
Simonyan, K., 554
Simrank, 389
Singleton, RC, 174
Valoare singulară, 443, 447, 448
Descompunere cu valoare unică, 340, 429,
442, 452
Șase grade de separare, 408
Schiță, 119
Inclinare, 28
Fereastra culisantă, 146, 162, 169, 283
Închidere tranzitivă inteligentă, 413, 415
Smith, B., 353
SNAP, 426
Grafic social, 356

Rețea socială, 18, 355, 356, 429
Softmax, 533
Algoritmul SON, 240
Sursa, 405
Nod sursă, 389
Spațiu, 105, 253
Spam, vezi Termen spam, vezi Link spam,
358, 470
Fermă de spam, 199, 202
Masa spam, 202, 203
Spark, 41, 43, 51, 77, 79
Matrice rară, 31, 90, 91, 189, 190, 320
Partiționare spectrală, 374
Capcană pentru păianjen, 182, 185, 205
Split, 46
Clusterele despărțitoare, 281
SQL, 22, 33, 77, 79
Eroare pătrată, 536
Pătrate, 404
Srikant, R., 250
Srivastava, U., 78, 79
Abaterea standard, 271, 273
Interogare permanentă, 146
Platforma de analiză a rețelei Stanford,
vezi SNAP
Star join, 60
Stata, R., 20, 212
Modelul statistic, 2
Stare, 313
Steinbach, M., 20
Funcția pas, 530, 531
Coborâre în gradient stochastic, 348, 495
Matricea stochastică, 179, 431
Stoica, I., 79
Opriți gruparea, 259, 263, 265
Opriți cuvintele, 9, 88, 128, 217, 325
Stratosfera, 77

Flux, consultați Flux de date
Puterea de membru, 387
Stride, 551
Șir, 131
Striping, 32, 189, 191
Marginea puternică, 358
Componentă puternic conectată, 181, 417
Grafic puternic conectat, 179, 406
Matrice substocastică, 182
Lungimea sufixului, 135
Rezumare, 4
Suma, 168
Sun, J., 461
Supercomputer, 21
Cod suprapus, vezi filtrul Bloom,
173
Supermarket, 216, 238
Superstep, 52
Învățare supravegheată, 463, 465
Suport, 214, 239, 240, 242, 244
Vector suport, 486
Mașină vector-suport, 18, 463, 468,
485, 517
Pagina de suport, 200
Suri, S., 427
Număr surpriză, 158
Sutskever, I., 569

SVD, vezi Descompunerea valorii singulare
SVM, consultați Mașină vector-suport
Swami, A., 250
Matricea simetrică, 377, 430
Szegedy, M., 174
Etichetă, 326, 359
Coada, 413
Lungime coadă, 155, 419
Tan, P.-N., 20
Țintă, 405
Pagina țintă, 200
Tarjan, RE, 417
Sarcină, 23
Impozitare, 182, 185, 200, 205
Expansiunea Taylor, 14
Taylor, M., 317
Apel telefonic, 358
Set Teleport, 196, 197, 202, 391
Teleportarea, 186
Tendril, 181
Tensor, 48, 546
TensorFlow, 41, 79, 530, 538, 543, 570
Termen, 176
Frecvența termenului, 9, vezi TF.IDF
Termen de spam, 176, 199
Set de testare, 468, 475
TF, vezi Frecvența termenului
TF.IDF, 9, 325, 467
Theobald, M., 142
Thrashing, 191, 230
Prag, 102, 171, 214, 240, 244, 471,
477
TIA, vezi Conștientizarea informațiilor totale
Timestamp, 163, 284
Algoritmul lui Toivonen, 242
Toivonen, H., 250
Tomkins, A., 20, 79, 212, 426
Tong, H., 427
PageRank sensibil la subiect, 195, 202
Toscher, A., 353
Conștientizarea informațiilor totale, 6
Atingerea vidului, 323
Exemplu de instruire, 464
Rata de antrenament, 475
Set de antrenament, 463, 464, 470, 480
Tranzacție, vezi Coș
Transformare, 44
Matricea de tranziție, 391
Matricea de tranziție a Web-ului, 178, 189,
190, 192, 429
Închidere tranzitivă, 49, 407
Reducere tranzitivă, 418
Transpune, 205
Transpunere, 111
Arborele, 260, 278, 279, vezi Arborele deciziei
Triunghi, 399
Inegalitatea triunghiului, 105
Matricea triunghiulară, 223, 232
Grafic tripartit, 359
Metoda triplelor, 223, 232
TrustRank, 202
Pagina de încredere, 202

Tsourakakis, CE, 427
Tub, 182

Tuple, 33
Tuzhilin, A., 352
Twitter, 18, 313, 356
Tzoumas, K., 78
Ullman, JD, 20, 77–79, 250, 292, 426
Grafic nedirectat, vezi Grafic
Uniunea, 34, 36, 40, 85
Vector unitate, 430, 435
Set universal, 131
Învățare fără supraveghere, 463
Hiperplanul superior, 487
Utilizator, 320, 336, 337
Profil utilizator, 328
Matrice utilitară, 320, 323, 340, 429
Descompunerea UV, 340, 350, 429, 496
Fișier VA, 503
Valduriez, P., 427
Set de validare, 468, 567
Van Loan, CF, 460
Gradient de dispariție, 561
Vapnik, VN, 521
Variabilă, 158
Vassilvitskii, S., 427
Vazirani, U., 318
Vazirani, V., 318
Vector, 31, 105, 109, 179, 189, 204,
205, 254
Vernica, R., 78
VGGnet, 553
Vigna, S., 426
Vitter, J., 174
Volum, 397
Volum (dintr-un set de noduri), 375
von Ahn, L., 327, 353
von Luxburg, U., 427
Diagrama Voronoi, 498
Waibel, A., 570
Wall, ME, 461
Ora ceasului de perete, 55
Wallach, DA, 78
Wang, J., 350
Wang, W., 142
Warneke, D., 78
Marginea slabă, 358
Weaver, D., 78
Structura web, 181
Greutate, 471, 526
Weiner, J., 20, 212
Whizbang Labs, 3
Widom, J., 20, 79, 174, 292, 426
Wikipedia, 358, 470
Williams, RJ, 570
Fereastra, vezi Fereastra glisantă, vezi De-
fereastră caying
Windows, 13
Algoritmul Winnow, 475
Cuvânt, 217, 254, 325
Număr de cuvinte, 25, 44
Procesul lucrătorului, 28
Flux de lucru, 42, 49, 54
Magazin de lucru, 144
Xiao, C., 142
Xie, Y., 461
Yahoo, 303, 326
Yang, J., 427
Yerneni, R., 78
York, J., 353
Yu, JX, 142
Yu, PS, 250
Yu, Y., 79

Zaharia, M., 79
Căptușeală zero, 551, 559
Zhang, CH, 142
Zhang, H., 461
Zhang, T., 292
Legea lui Zipf, 15, vezi Legea puterii
Zoeter, O., 317
Zussman, A., 554