Adatbányászat

Bodon Ferenc, Buza Krisztián

Tartalomjegyzék

Előszó 7

1. Bevezetés **13**
1.1. A tudásfeltárás folyamata 16
1.2. Adatbányászati alapfeladatok 22
1.3. Sikeres alkalmazások 27
1.4. Az adatbányászat megközelítései 29
1.5. Adatbányászati algoritmusokkal szembeni alapkövetelmények . . 31
1.6. Szabványok 32
1.7. Adatbányászati rendszer architektúrája 33
1.8. Az adatbányászat feltételei 34

2. Alapfogalmak, jelölések **42**
2.1. Halmazok, relációk, függvények, sorozatok 42
2.2. Valószínűségszámítás 45
2.2.1. Valószínűségi változók feltételes függetlensége 49
2.2.2. Nevezetes eloszlások 49
2.2.3. Egyenlőtlenségek 51
2.2.4. Entrópia 52
2.3. Statisztika 53
2.3.1. Hipotézisvizsgálat 53
2.3.2. Az F-próba 54
2.3.3. A χ^2-próba 54
2.3.4. Függetlenségvizsgálat 55
2.4. Gráfelmélet 57
2.5. Adatstruktúrák 59
2.5.1. Szófák 59
2.5.2. Piros-fekete fák 61
2.5.3. Hash-tábla 62
2.6. Számítógép-architektúrák 62
2.6.1. Többszintű memória, adatlokalitás 62
2.6.2. Csővezetékes feldolgozás, elágazás-előrejelzés 63

3. Előfeldolgozás, távolságfüggvények **65**
3.1. Attribútum típusok 65
3.2. Távolsági függvények 67
3.2.1. Bináris attribútum 68
3.2.2. Kategória típusú attribútum 69
3.2.3. Sorrend típusú attribútum 70
3.2.4. Intervallum típusú attribútum 70
3.2.5. Vegyes attribútumok 72
3.2.6. Speciális esetek 72
3.3. Előfeldolgozás 74
3.3.1. Hiányzó értékek kezelése 74
3.3.2. Attribútumtranszformációk 75
3.3.3. Adatok torzítása 76
3.3.4. Diszkretizálás 77
3.3.5. Normalizálás 79
3.3.6. Mintavételezés 80
3.3.7. Sokdimenziós adatok, dimenziócsökkentés 88
3.3.8. Duplikátumok kiszűrése 101
3.3.9. Aggregáció 103
3.3.10. Monotonizáció 104

4. Osztályozás és regresszió **106**
4.1. Az osztályozás és a regresszió feladata, jelölések 108
4.1.1. Az elméleti regressziós görbe 110
4.1.2. Maximum likelihood osztályozás 110
4.2. k-legközelebbi szomszéd módszer 111
4.2.1. Dimenzióátok és a legközelebbi szomszéd módszere . . . 113
4.2.2. A legközelebbi szomszéd érzékenysége 114
4.2.3. Az osztályozás felgyorsítása 118
4.3. Lineárisan szeparálható osztályok 120
4.3.1. Perceptron tanulási szabály 123
4.3.2. Winnow módszer 124
4.3.3. Rocchio-eljárás 125
4.3.4. Lineáris regresszió 127
4.3.5. Logisztikus regresszió 129
4.4. Mesterséges neurális hálózatok 134
4.5. Döntési szabályok 136
4.5.1. Szabályhalmazok és szabálysorozatok 139
4.5.2. Döntési táblázatok 139
4.5.3. Az 1R algoritmus 140
4.5.4. A Prism módszer 141

4.6. Döntési fák . . . 144
4.6.1. Döntési fák és döntési szabályok . . . 144
4.6.2. A döntési fa előállítása . . . 146
4.6.3. Az ID3 algoritmus . . . 147
4.6.4. Feltételek a csomópontokban . . . 149
4.6.5. Vágási függvények . . . 149
4.6.6. Továbbfejlesztések . . . 152
4.6.7. Súlyozott divergenciafüggvények alapján definiált vágási függvények . . . 153
4.6.8. Döntési fák metszése . . . 154
4.6.9. Döntési fák ábrázolása . . . 155
4.6.10. Regressziós fák és modell fák . . . 155
4.7. Bayes-hálózatok . . . 156
4.7.1. Naív Bayes-hálók . . . 157
4.7.2. Naív Bayes-hálók és a logisztikus regresszió kapcsolata . . . 159
4.7.3. Bayes hihetőségi hálók . . . 161
4.8. Szupport Vektor Gépek (SVM-ek) . . . 162
4.9. Ensemble modellek . . . 163
4.9.1. Dietterich elmélete . . . 164
4.9.2. Boosting . . . 166
4.9.3. Bagging és Stacking . . . 167
4.10. Tanuló algoritmusok értékelése . . . 168
4.10.1. Túltanulás . . . 169
4.10.2. Kiértékelési protokollok . . . 171
4.10.3. Mérőszámok . . . 172
4.10.4. Osztályozók összehasonlítása . . . 177
4.11. További osztályozási protokollok . . . 179
4.11.1. Semi-supervised osztályozás . . . 180
4.11.2. Active Learning . . . 181
4.11.3. Transfer learning . . . 182
4.11.4. Többosztályos és többcímkés osztályozás . . . 182
4.12. Ajánlórendszerek és ritka mátrixok faktorizációja . . . 183
4.12.1. Collaborative filtering . . . 183
4.12.2. Gradiens módszeren alapuló mátrix faktorizáció . . . 184
4.13. További gyakorlati, alkalmazási problémák . . . 186
4.13.1. Többosztályos osztályozási feladatok visszavezetése bináris osztályozásra . . . 186
4.13.2. Kategorikus attribútumok kezelése . . . 187
4.13.3. Feature extraction . . . 188
4.13.4. Paraméterkeresés . . . 189
4.13.5. Mit tegyünk, ha az osztályozónk nem (elég) jó? . . . 190

5. Gyakori mintázatok és asszociációs szabályok 204
5.1. Gyakori elemhalmazok 204
5.1.1. A gyakori elemhalmaz fogalma 204
5.1.2. Az APRIORI algoritmus 209
5.1.3. Az ECLAT algoritmus 227
5.1.4. Az FP-GROWTH algoritmus 230
5.1.5. További technikák 234
5.1.6. Mintavételező algoritmus elemzése 235
5.1.7. Elemhalmazok Galois lezártja 236
5.1.8. Kényszerek kezelése 239
5.1.9. Többszörös támogatottsági küszöb 240
5.2. Asszociációs szabályok 242
5.2.1. Az asszociációs szabály fogalma 242
5.2.2. Érdekességi mutatók 245
5.2.3. Szabályok függetlensége 246
5.2.4. Általánosság, specialitás 257
5.2.5. Asszociációs szabályok általánosítása 258
5.2.6. A korreláció nem jelent ok-okozati kapcsolatot 261
5.2.7. Asszociációs szabályok és az osztályozás 264
5.3. Gyakori minták kinyerése 264
5.3.1. A gyakori minta definíciója 265
5.3.2. További feladatok 268
5.3.3. Az algoritmusok jellemzői 272
5.3.4. Az APRIORI módszer 273
5.3.5. Sorozat típusú bemenet 277
5.4. Gyakori sorozatok, bool formulák és epizódok 291
5.4.1. Gyakori sorozatok kinyerése 291
5.4.2. Gyakori bool formulák 299
5.4.3. Gyakori epizódok 300
5.5. Gyakori fák és feszített részgráfok 303
5.5.1. Az izomorfia problémája 304
5.5.2. A gyakori gráf fogalma 306
5.5.3. Gyakori gyökeres fák 307
5.5.4. A gyakori feszített részgráfok 309
5.5.5. A gyakori részgráfok keresése 312

6. Klaszterezés 322
6.1. Legfontosabb lépések a klaszterezés elméleti alapjainak megértéséhez 323
6.1.1. Kleinberg lehetetlenség-elmélete 324
6.1.2. Stabilitás és 'Klaszterezhetőség' 327

6.2. Hasonlóság mértéke, adatábrázolás . . . 328
6.3. A klaszterek jellemzői . . . 329
6.4. A klaszterezés „jósága” . . . 330
6.4.1. Klasszikus mértékek . . . 331
6.4.2. Konduktancia alapú mérték . . . 333
6.4.3. Referencia-klaszterekhez való viszonyítás . . . 335
6.4.4. Klaszterező algoritmusok feladat-alapú kiértékelése . . . 337
6.5. Klaszterező algoritmusok típusai . . . 337
6.6. Particionáló eljárások . . . 339
6.6.1. Forgy k-közép algoritmusa . . . 339
6.6.2. A k-közép néhány további változata . . . 341
6.6.3. A k-medoid algoritmusok . . . 341
6.7. Hierarchikus eljárások . . . 344
6.7.1. Single-, Complete-, Average Linkage Eljárások . . . 344
6.7.2. Ward módszere . . . 347
6.7.3. A BIRCH algoritmus . . . 347
6.7.4. A CURE algoritmus . . . 348
6.7.5. A Chameleon algoritmus . . . 350
6.8. Sűrűség-alapú módszerek . . . 351
6.8.1. A DBSCAN algoritmus . . . 351

7. Idősorok elemzése **353**
7.1. Idősorok ábrázolása . . . 355
7.1.1. Diszkrét Fourier-transzformáció (DFT) . . . 355
7.1.2. Diszkrét Wavelet Transzformáció . . . 357
7.1.3. Szimbólikus Aggregált Approximáció (SAX) . . . 358
7.2. Idősorok távolsága . . . 359
7.3. Idősorok osztályozása és klaszterezése . . . 361

8. Anomáliák feltárása **364**
8.1. Távolság-alapú anomália-kereső eljárások . . . 365
8.2. Osztályozásra és regresszióra épülő anomália-kereső eljárások . . . 365
8.3. Klaszterezés-alapú anomália-keresés . . . 365
8.4. Statisztikai megközelítésen alapuló anomáliakeresés . . . 366

9. Adatbányászat a gyakorlatban: Weka **367**
9.1. A Weka indítása . . . 367
9.2. Adatok betöltése, az ARFF formátum, attribútumtípusok Weka-ban . . . 368
9.3. Előfeldolgozás Weka-ban . . . 369
9.3.1. Adatok konvertálása . . . 370

9.3.2. Hiányzó értékek kezelése 371
9.3.3. Új attribútumok létrehozása 371
9.3.4. Attribútumok törlése 372
9.3.5. Zajszűrés, hibás bejegyzések eltávolítása 372
9.3.6. Adatok torzítása 373
9.3.7. Diszkretizálás 373
9.3.8. Normalizálás 373
9.3.9. Mintavételezés 373
9.3.10. Dimenziószámcsökkentés 374
9.4. Osztályozó eljárások Weka-ban 374
9.4.1. Legközelebbi szomszéd osztályozó 376
9.4.2. Regressziós eljárások 376
9.4.3. Neurális hálózatok 377
9.4.4. Szabály-alapú osztályozók 379
9.4.5. Döntési fák és regressziós fák 379
9.4.6. Bayes-osztályozók 380
9.5. Asszociációs szabályok bányászata 380
9.6. Klaszterező eljárások Weka-ban 381
9.7. Weka használata függvénykönytárként 382

Előszó

Az adatbányászati algoritmusok, technikák és alkalmazások rohamos fejlődésének köszönhetően egyre nagyobb az igény egy magyar nyelvű, naprakész és lehetőség szerint az adatbányászathoz kapcsolódó témák minél szélesebb körét átfogó jegyzetre. Jelen munkánkkal erre az igényre kívánunk választ adni. *Bodon Ferenc: Adatbányászati algoritmusok* c. jegyzete a magyar nyelű adatbányászati irodalom egyik úttörője volt, a jelen mű nagyban épít erre a tanulmányra, kibővítve és kiegészítve azt. Az átdolgozáshoz, bővítéshez tanulmányoztuk neves külföldi egyetemek és nyári egyetemek kurzusainak tematikáját és az utóbbi néhány évben megjelent adatbányászati témájú könyveket és tudományos cikkeket (lásd még a *Újdonságok a jegyzetben* c. szakaszt és a *Köszönetnyilvánítást*). Célunk az, hogy egy olyan jegyzet szülessen, amely az adatbányászati tárgyak hallgatói, oktatói, a terület kutatói és alkalmazói számára egyaránt hasznos, érdekes. Ezért, a nemzetközi trendeknek megfelelően, az elméleti fejezeteket gyakorlati témákkal egészítettük ki, különös tekintettel az adatbányászati algoritmusok sikeres alkalmazásait elsősegítő technikákra, mint például a hiperparaméter-keresés (4.13. fejezet) vagy túltanulás felismertését és a túltanulás elleni védekezést szolgáló módszerek (4.10.1. fejezet és 4.13.5. fejezet).

Történeti áttekintés

A 90-es években a tárolókapacitások méretének igen erőteljes növekedése, valamint az árak nagymértékű csökkenése[1] miatt az elektronikus eszközök és adatbázisok a hétköznapi életben is mindinkább elterjedtek. Az egyszerű és olcsó tárolási lehetőségek a feldolgozatlan adatok felhalmozását eredményezték. Az így létrejött óriási adatbázisok a legtöbb gyakorlati alkalmazásban a közvetlen visszakeresésen és ellenőrzésen kívül nem sok további haszonnal jártak. A ritkán látogatott adatokból „adat temetők” (data tombs) alakultak

[1] A tárolókapacitás növekedése a kilencvenes években még Moore jóslatát is felülmúlta, lásd: [Porter, 1998]

ki [Han és Kamber, 2006], amelyek tárolása haszon helyett pusztán költséget jelentett. Ekkor még nem álltak rendelkezésre olyan eszközök, amivel az adatokban lévő értékes információt ki tudták volna nyerni. Ezért fontos döntések a döntéshozók megérzésein alapultak, nem pedig az információban gazdag adatokon, az adat → információ → döntés lánc nem működött megfelelően. Jól jellemzi ezt a helyzetet John Naisbitt híres mondása, miszerint „We are drowning in information, but starving for knowledge"[2] (Megfulladunk az információtól, miközben tudásra éhezünk).

Egyre több területen merült fel az igény, hogy az adathalmazokból a hagyományosnál árnyaltabb szerkezetű információkat nyerjenek ki. A hagyományos adatbázis-kezelő rendszerek – a közvetlen keresőkérdéseken kívül, illetve az alapvető statisztikai funkciókon túl (átlag, szórás, maximális és minimális értékek meghatározása) – komplexebb feladatokat egyáltalán nem tudtak megoldani, vagy az eredmény kiszámítása elfogadhatatlanul hosszú időbe telt.

A szükség egy új tudományterületet keltett életre, az *adatbányászatot*, amelynek célja: „hasznos, látens információ kinyerése az adatokból". Az adatbányászati algoritmusokat arra tervezték, hogy képesek legyenek az árnyaltabb információ kinyerésére akár óriási méretű adatbázisok esetén is.

Az adatbányászat, mint önálló tudományterület létezéséről az 1980-as évek végétől beszélhetünk. Kezdetben a különböző heurisztikák, a matematikailag nem elemzett algoritmusok domináltak. A 90-es években megjelent cikkek többségét legfeljebb elhinni lehetett, de semmiképpen sem kétely nélkül meggyőződni az egyes írások helytállóságáról. Az algoritmusok futási idejéről és memóriaigényéről általában felszínes elemzéseket és tesztelési eredményeket olvashattunk. Az igényes olvasóban mindig maradt egy-két kérdés, amire nem talált választ. Bizonyos káosz uralkodott, amiben látszólag mindenre volt megoldás, ám ezek a megoldások többnyire részlegesek voltak. Ennek egyik legszembetűnőbb példája a példányokat hasonlóságuk szerint csoportosító, ún. klaszterező algoritmusok területe (6. fejezet), de több korai osztályozó (4. fejezet) és gyakori mintabányász (5.2. fejezet) algoritmus is elméleti szempontból nem kellően alátámasztott heurisztikákat alkalmazott.

A XXI. századba való belépéssel a kutatók körében egyre nagyobb népszerűségnek kezdett örvendeni az adatbányászat. Ennek két oka van: egyrészt a növekvő versenyhelyzet miatt a piaci élet szereplőinek óriási az igénye az adatbázisokban megbújó hasznos információkra. A növekvő igény növekvő kutatói beruházásokat indukált. Másrészt az adatbányászat a maga multi-diszciplináris voltával attraktív terület számos kutató számára. Sorra születtek meg a színvonalas munkák, elemzések, összehasonlítások és mindinkább tiszta irányvonalak rajzolódtak ki. Az elmúlt két évtizedben kifejlesztett eljárásoknak köszönhe-

[2]Megatrends, 1988

tően rengeteg hasznos információt sikerült kinyerni. A speciális alkalmazások mellett némelyik elemző, felismerő eljárással a mindennapi életünkben is rendszeresen találkozunk: ilyen például a kéretlen elektronikus levelek (spam-ek) automatikus felismerése vagy az online kereskedelemben egyre gyakrabban alkalmazott ajánlórendszerek, amelyek a felhasználó ízlését próbálják feltérképezni és ez alapján személyre szabott reklámokat helyeznek el az online áruház weblapján, amikor egy-egy felhasználó belép az adott oldalra.

Ugyanakkor a különféle szenzorok egyre olcsóbbá válásának köszönhetően minden korábbinál nagyságrendekkel nagyobb adathalmazok gyűltek és gyűlnek össze, az adatok nagy részét csak eltárolják és *soha(!)* nem olvassák ki.[3] A kihívás tehát folyamatos, a megoldatlan, nyitott problémákra továbbra is keressük a választ, így a következő évtizedekben is az adatbányászat dinamikus fejlődése várható.

Újdonságok a jegyzetben

Az adatbányászat elmúlt években tapasztalható dinamikus fejlődése, új témák és területek megjelenése tette szükségessé Bodon Ferenc korábbi jegyzetének átdolgozását, bővítését. A bővítés elsősorban az alábbi témákat érinti:

Mátrix faktorizációs algoritmusok. Népszerűek mind kutatásokban, mind alkalmazásokban a ritka mátrixok faktorizációján alapuló adatbányászati eljárások, ezért a jegyzetet is bővítettük ezzel a témával.

Idősorokkal kapcsolatos adatbányászati feladatok. Az adattábla típusú adatokkal kapcsolatos elemző eljárások egyre alaposabb megértése után a figyelem egyre inkább más módon strukturált adatok felé fordul. Ezek egyik legegyszerűbb esete az idősorok, melyekkel külön fejezetben foglalkozunk.

Osztályozó algoritmusok alkalmazása a gyakorlatban. Habár az osztályozó algoritmusok egyes típusairól (pl. neurális hálók, szupport vektor gépek) külön-külön is teljes könyvek jelentek meg, szükségesnek tartottuk az osztályozó algoritmusokhoz kapcsolódó témák bővítését is. Eközben elsődlegesen nem arra fókuszálunk, hogy a meglévő algoritmusok minél nagyobb számú változatát mutassuk be, hanem arra, hogy az osztályozó algoritmusok sikeres gyakorlati alkalmazásához nyújtsunk segítséget az Olvasónak. Ezért a korábbiaknál részletesebben térünk ki olyan témákra, mint például a hiperparaméter-keresés, többosztályos problémák

[3] IBM Storage Fórum, 2012, Budapest

visszavezetése bináris osztályozási feldatokra vagy a kiegyensúlyozatlan méretű osztályok (imbalanced classes) esete.

Ensemble modellek. Gyakran tapasztaljuk, hogy a különböző modellek kombinációja jobb megoldásra vezet, mint az egyes modellek önmagukban, ezért fontosnak tartottuk, hogy a modellek kombinációjával kapcsolatos legfontosabb elméleti eredményeket és leggyakrabban alkalmazott technikákat is ismertessük.

Klaszterezéssel kapcsolatos új eredmények. Számos kutató kritikusan tekint a klaszterezés témakörére azért, mert más feladatokkal ellentétben kevésbé világos, hogy mikor mondhatjuk, hogy az egyik klaszterező algoritmus jobban teljesít a másiknál. A kritikus hangokat csak erősítette Kleinberg lehetetlenségelmélete [Kleinberg, 2002]. Ugyanakkor az újabb kiértékelési technikák, mint például a feladat-alapú kiértékelés (task-based evaluation) és elméleti eredmények, úgy mint a klaszterezés stabilitásával, valamint a klaszterező algoritmusok konvergencia-sebességével kapcsolatos tanulmányok, könnyen új megvilágításba helyezhetik a klaszterezési.

Csomósodás jelensége. Nemrég figyelték meg, hogy az adatbányászati elemzések hátterében lévő adatbázisok széles körére jellemző a csomósodás jelensége. A csomósodás azt jelenti, hogy az adatbázisban található néhány olyan központi szerepű objektum, amelyek az adatbázis meglepően sok további objektumára hasonlítanak. Ez a jelenség, érdekes módon, összefügg a sokdimenziós terek estében tapasztalható problémákkal (curse of dimensionality), és a immáron számos adatbányászati területen léteznek a csomósodást figyelembe vevő algoritmusok. A csomósodás jelenségének bemutatása mellett a megfelelő helyeken utalunk a csomósodást figyelembe vevő osztályozó és klaszterező algoritmusokra.

Külön fejezet a Wekáról. A Weka elnevezésű adatbányászati szoftverrel kapcsolatos tudnivalókat bővítettük és külön fejezetbe szerkesztettük.

Kinek szól ez a jegyzet?

Ez a jegyzet a jelenlegi adatbányászati problémákról és az azokat megoldó algoritmusokról szól. A területek áttekintése mellett az algoritmusok mélyebb szintű megismerése is a cél. Az írás elsősorban informatikus beállítottságú olvasóknak készült, ugyanakkor szívesen ajánljuk minden érdeklőnek. Az egyes fejezetek mélyebb megértését segíti, ha az olvasó tisztában van algoritmus-

[Rónyai és tsa.,1998] és adatbázis-elméleti [Garcia-Molina és tsa., 2008] alapokkal, továbbá nem ismeretlen a valószínűségszámítás [Feller, 1978, Rényi, 1968] és a lineáris algebra [Rózsa, 1991] sem.

A jegyzet célja, hogy az adatbányászati apparátus olyan megismerését nyújtsa, melynek segítségével az olvasó sikerrel oldja meg az egyre több területen felbukkanó újabb és újabb adatbányászati problémákat.

Örömmel fogadjuk a jegyzettel kapcsolatos visszajelzéseket az alábbi címen:

buza@cs.bme.hu

Ajánlott irodalom

[Han és Kamber, 2006] *Data Mining Concepts and Techniques* című könyve egyike az adatbányászat korai nagy sikerű műveinek, amelynek magyar nyelvű fordítása is megjelent. A magyar nyelvű szakirodalomból kiemeljük *Abonyi János* által szerkesztett *Adatbányászat, a hatékonyság eszköze* című könyvet [Abonyi, 2006]. Az adatbányászat rokonterületéről írt kitűnő könyvet *Tikk Domonkos Szövegbányászat* címmel [Tikk, 2007].

Az angol nyelvű szakirodalom legnépszerűbb művei közül kiemeljük *Tan, Steinbach és Kumar Introduction to Data Mining* című könyvét [Tan és tsa., 2005] valamint az *Eibe Frank* és *Ian H. Witten* által írt *Data Mining: Practical Machine Learning Tools and Techniques* című művet [Witten és tsa., 2011]. Mindkettő egyszerűségre törekszik, ezért nyugodtan ajánljuk minden érdeklődőnek. Eibe Frank a Weka egyik főfejlesztője, ennek megfelelően a könyv egy része a Weka használatát tárgyalja. Komolyabb matematikai felkészültséget feltételez *Trevor Hastie, Robert Tibshirani* és *Jerome Friedman* által írt *The Elements of Statistical Learning: Data Mining, Inference and Prediction* című könyv [Hastie és tsa., 2001], valamint *Christopher M. Bishop Pattern Recognition and Machine Learning* című műve [Bishop, 2006].

Köszönetnyilvánítás

Ezúton szeretnénk köszönetet mondani **Rónyai Lajosnak**, a Budapesti Műszaki és Gazdaságtudományi Egyetem tanárának a jegyzet korábbi változatához nyújtott segítségéért, hasznos ötleteiért, útmutatásaiért. Köszönjük **Molnár-Sáska Gábornak**, **Pintér Mártának**, **Szabó Jácintnak**, **Hum Katalinnak**, **Biro Istvánnak** és **Fekete Zsoltnak** az MTA-SZTAKI dolgozóinak valószínűségszámítással kapcsolatos tanácsaikat.

Köszönetet mondunk **Fogaras Dánielnek**, aki az SVD-ről szóló részt írta.

Külön köszönet illeti **Czibula Veronikát** a jelen jegyzet alapjául szolgáló, korábbi tanulmány többszöri, alapos átnézéséért. **Marx Dániel** rengeteg információval látta el a jegyzet első szerzőjét a LaTeX, emacs, Xfig hatékony használatát illetően, amelyet ezúton is köszönünk.

Friedl Katának, ifjabb Benczúr Andrásnak, Lukács Andrásnak, Maricza Istvánnak, Sarlós Tamásnak és Bereczki Tamásnak köszönjük az értékes észrevételeiket, megjegyzéseiket.

A jegyzet második szerzője köszönetet mond Prof. Dr. Alexandros Nanopoulosnak és Prof. Dr. Lars Schmidt-Thieme-nek, akik révén új megközelítésben ismerkedett meg adatbányászati feladatokkal. A jegyzet egyes bővítéseit Andrew Ng, a Stanfordi Egyetem docensének nagysikerű online kurzusa[4] ihlette.

Értékes észrevételeikért és konstruktív javaslataikért köszönet illeti a BME diákjait, többek között (névsorrendben) Erős Pétert, Fekete Gábort, Hajnács Zoltánt, Lajkó Pétert, Petróczi Attilát, Schlotter Ildikót, Szántó Ádámot, Szőke Mónikát és Varga Dánielt.

Végezetül, de nem utolsó sorban, köszönetünket fejezzük ki **Csató Lehelnek**, a kolozsvári Babes-Bolyai Tudományegyetem oktatójának a jelen jegyzet lekorálásáért, értékes tanácsaiért.

[4] www.coursera.org-n megjelent Machine Learning kurzus

1. fejezet

Bevezetés

A számítógép, korunk egyik legjelentősebb találmánya, rohamléptekkel hódít teret az élet minden területén. Amit nagyszüleink még el sem tudtak képzelni, egy generáció alatt nélkülözhetetlenné vált, mára elválaszthatatlan a munkánktól és szórakozásunktól egyaránt.

Az Internet elterjedésével még hangsúlyosabban érzékelhető a számítógép térhódítása: az egyik legnagyobb problémát, a távolságot hidalta át. Üzleti és magáncélú érintkezések váltak lehetővé rövidebb idő alatt és hatékonyan. Adatok millióit kezelik és szállítják számítógépes rendszerek. Az információkon alapuló döntéshozatal ideje lerövidült, hiszen a hozzáférés könnyebbé és gyorsabbá vált.

Ma a vállalatok léte múlhat az információk gyors és pontos begyűjtésén, elemzésén, a rugalmas fejlődésen, valamint az innováción. Az adatok azonban önmagukban nem hasznosak, hanem a belőlük kinyerhető, a vállalat igényeihez igazodó, azt kielégítő információkra van szükség. Ez egy újabb szükségletet teremt: egy olyan eszköz iránti igényt, ami képes arra, hogy információszerzés céljából elemezze a nyers adatokat. Ez az eszköz az **adatbányászat**.

Adatbányászati (data mining) algoritmusokat az adatbázisból történő tudásfeltárás (knowledge discovery in databases) során alkalmaznak. A tudáskinyerés adatbázisokból egy olyan folyamat, melynek során érvényes, újszerű, lehetőleg hasznos és érthető mintákat fedezünk fel az adatokban. Egy ilyen minta az alábbi:

„*Angol tudósok azt állapították meg, hogy aki sokat jár disco-ba, annak nagyobb valószínűséggel alakul ki asztmája.*”
Forrás: Sláger rádió, 2007. október 2., 8 óra 26 perc

Ilyen és ehhez hasonló mintákat gyakran találhatunk különböző lekérdezések segítségével, azonban ez a megoldás lassú, drága és nem elég átfogó. Jogos

tehát az igény, hogy a legismertebb, leggyakoribb elemzéstípusokhoz speciális módszereket, algoritmusokat fejlesszenek ki, amelyek gyorsan és pontosan szolgáltatnak egy objektív képet az adatbázisokban található „kincsről". Ennek szellemében sokféleképpen definiálták az adatbányászatot, ezek közül sorolunk fel néhányat:

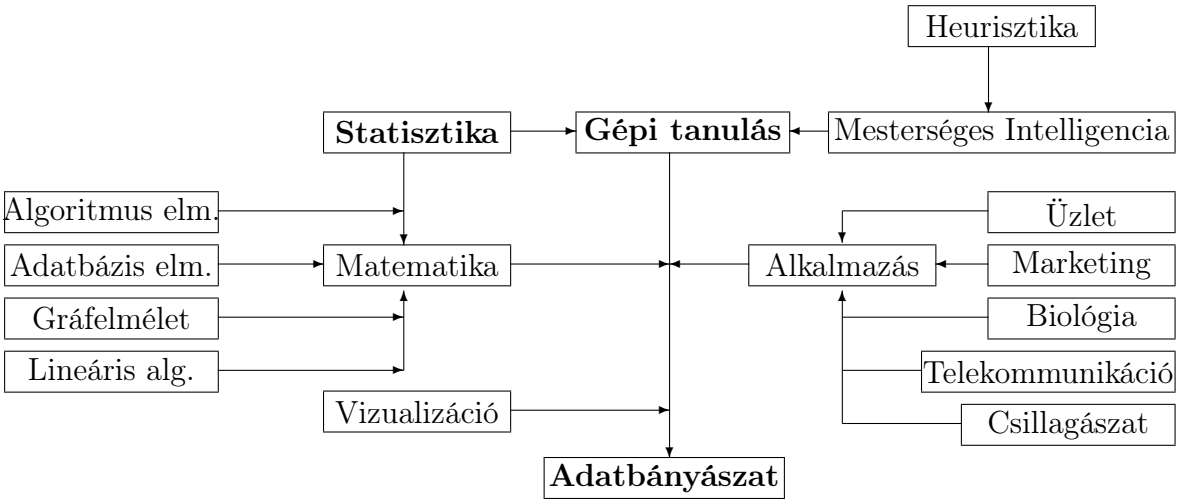
- „The nontrivial *extraction* of implicit, previously unknown, and potentially *useful information* from *data*" (Piatetsky Shapiro)
- „... the automated or convenient *extraction* of *patterns* representing *knowledge* implicitly stored or captured in *large databases, data warehouses, the Web, ... or data streams*." ([Han és Kamber, 2006], xxi oldal)
- „... the process of *discovering patterns* in *data*. The process must be automatic or (more usually) semiautomatic. The *patterns* discovered must be *meaningful*..." ([Witten és tsa., 2011], 5. oldal)
- „... *finding* hidden *information* in a *database*." ([Dunham, 2002], 3. oldal)
- „... the process of employing one or more computer *learning techniques* to automatically *analyze and extract knowledge* from *data contained within a database*." ([Roiger, 2003], 4. oldal)

Egyesek szerint az adatbányászat, mint megnevezés némiképp szerencsétlen [Han és Kamber, 2006]. Ha szénbányászatról beszélünk, a szén bányászására gondolunk. Ezzel ellentétben adatbányászat esetén *nem* adatot bányászunk, hanem — amint a példában is láttuk — a rejtett és számunkra hasznos *tudást* (információt), összefüggéseket keressük egy nagy adathalmazban (szemléletesen: „adathegyben").

Az adatbányászatot az üzleti élet és a marketing keltette életre, ugyanakkor egyre több területen ismerik fel lehetőségeit, melynek eredményeként az alapkutatásoknak is egy fontos eszközévé vált. Adatbányászati eszközöket alkalmaznak többek között az orvosbiológiában, genetikában, távközlésben, vagy a csillagászatban.

Az adatbányászat egy multi-diszciplináris terület, ezt szemlélteti az 1.1 ábra. A rokon területek közül kiemeljük a gépi tanulást és a statisztikát.

A gépi tanulás a klasszikus mesterséges intelligenciából nőtt ki. Míg a mesterséges intelligencia esetében azt hangsúlyozzuk, hogy egy robot, egy számítógépprogram (úgynevezett ágens) önálló döntéseket hoz, kvázi autonóm módon működik, önállóan reagál a környezetéből érkező jelekre, addig a gépi tanulás során általában feltételezzük, hogy korábbi tapasztalatainkat egy (nagy) adatbázissal írjuk le, és azt várjuk, hogy a számítógép ezen adatbázis, ezen tapasztalatok felhasználásával alakítson ki egy döntési mechanizmust, mintegy



1.1. ábra. Az adatbányászat kialakulása

tanuljon a megadott adatokból. Ágensnek tekinthetjünk a környezetével interakcióban lévő, önállóan működő számítógépprogramokat is: például egy olyan programot, amely automatizáltan tölti le weblapok sokaságát, és saját maga dönti el, hogy mely weblapokat töltse le és tárolja el egy adatbázisban.

A gépi tanulást egy, a kötelező gépjármű felelősségbiztosítás területről származó példán keresztül szemléltetjük: gépi tanulást végez egy olyan döntéstámogató rendszer, amely azt elemzi, hogy a múltban milyen tulajdonságú ügyfelek (pl. fiatal vagy öreg; egyedülálló vagy házas; gazdag vagy szegény) hány és mennyire súlyos autóbalesetet okoztak, és a rendszer az elemzés eredménye alapján tesz javaslatot arra, hogy egy-egy új ügyfél mekkora biztosítási díjat fizessen.

Sok kutató a *gépi tanulás* és *adatbányászat* kifejezéseket szinte szinonímaként használja. Ez világosan mutatja a két terület szoros kapcsolatát, azt, hogy számos eljárást, első sorban az osztályozó, regressziós és klaszterező algoritmusokat (például a döntési fákat, neurális hálózatokat, szupport vektor gépeket, centroid-alapú és hierarchikus klaszterezőket, stb.) a gépi tanulás és az adatbányászat is egyaránt magáénak tekinti. Tekinthetjük úgy, hogy a gépi tanulás esetében azt hangsúlyozzuk, hogy a rendszer a korábbi tapasztalatokat elemezve, azokból tanulva, képes következtetéseket levonni, döntési javaslatokat tenni. Egy ajánlórendszer például egy webes áruház eladási adatati alapján termékeket javasol a felhasználóknak. Ezzel szemben, amikor adatbányászatról beszélünk, azt hangsúlyozzuk, hogy a korábbi tapasztalatainkat leíró adatbázis óriási méretű, például a webes áruházak eladási adatai olyan nagy méretűek, hogy „hétköznapi” technikákkal nem tudjuk megfelelően feldolgozni, elemezni az adatokat, nem tudunk az adatokból értelmes összefüggéseket kinyerni, azok

alapján következtetéseket levonni.

Akárcsak a gépi tanulás, az adatbányászat is rengeteg, eredetileg a statisztikából származó eljárást használ. Míg azonban a statisztika egyik alapvető kérdése például, hogy mikor lesz a minta reprezentatív, az adatbányászat során általában abból indulunk ki, hogy egy nagy méretű, a releváns adatokat tartalmazó adathalmaz már rendelkezésre áll, csak nem tudjuk, hogy ezt milyen módszerrel kell elemeznünk ahhoz, hogy értékes tudást nyerjünk ki belőle. Az adatbányászat a statisztikai módszerek alkalmazásakor is a nagy adathalmazokra helyezi a hangsúlyt: egyik alapkérdés, hogy mely eszközök használhatók, és hogyan a nagyon nagy méretű adathalmazok elemzésére.

Összegzésként elmondhatjuk, hogy mára az adatbányászat egy szerteágazó területté nőtte ki magát, miközben több hangsúlyt fektet az algoritmusokra, mint a statisztika, és többet a modellekre, mint a gépi tanulás eszközei (pl. neurális hálózatok).

1.1. A tudásfeltárás folyamata

A tudásfeltárás folyamata [Han és Kamber, 2006, Fayyad, 1996] során hattól tíz fázist szokás elkülöníteni attól függően, hogy mely lépéseket vonjuk össze:

1. **Az alkalmazási terület feltárása és megértése**, fontosabb előzetes ismeretek begyűjtése és felhasználási célok meghatározása.

2. **Adatbázisok kiválasztása.** Kiválasztjuk a használni kívánt adatbázist vagy adatbázisokat, illetve annak számunkra releváns részét, amiből a tudást ki akarjuk nyerni. A kiválasztott adatok akár több, különböző számítógépen elosztva lehetnek jelen, egymástól fizikailag távol. A különböző forrásokból származó adatok kiválasztása során sok problémába ütközhetünk. A különböző adatbázisok különböző módon tárolják adataikat, különböző konvenciókat követnek, különböző mértékegységeket, elsődleges kulcsokat és elnevezést, különböző formátumokat használhatnak és különféle hibák lehetnek jelen. Az integráció egyik kulcsfeladata a duplikátumok kiszűrése: egyazon objektum különböző adatbázisokban lehet jelen, a különböző adatbázisokban kisebb-nagyobb mértékig eltérő formában. Azt szeretnénk ugyanakkor, hogy az integrált, egységes adatbázisban egy objektum pontosan egyszer szerepeljen, lehetőleg hibátlan adatokkal.

3. **Adattisztítás.** Itt olyan alapvető operációkat értünk, mint a téves bejegyzések eltávolítása, hiányos mezők pótlása, zajok szűrése stb. Zajon az adatba épült véletlen hibát értünk. Vannak zajok, amelyeket egyszerű

felfedezni és javítani. Például sztring típusú érték ott, ahol számot várunk, vagy felsorolás típusú attribútumnál ervénytelen érték található.[1] Sajnos a hiba sok esetben észrevétlen marad (például 0.53 helyett 0.35 érték gépelése).

4. **Adatintegráció, adattárházak kialakítása.** Az adattárházak kialakítása során az elemzés számára lényeges adatbázisokat egyesítjük. A harmadik és negyedik lépést együtt gyakran nevezik az adatok előfeldolgozásának. Az egész céget átfogó adatintegráció eredményeként létrejön egy speciális, az elemzést támogató adatbázis, amelyet *adattárháznak* neveznek.

   Példa: A következőkben egy banki rendszer kontextusásban szemléltetjük, hogy egy adattárház mennyiben tér el a hétköznapi működést támogató, úgynevezett *operatív adatbázis*tól. Tegyük fel, hogy tudni szeretnénk egy ügyfél számlaegyenlegét. Az ügyfelet nevezzük Gipsz Jakabnak. Gipsz Jakab számlaegyenlegét az operatív adatbázisból pontosan, gyorsan és naprakészen le tudjuk kérdezni. Gipsz Jakab számlaegyenlegére vonatkozó lekérdezéssel szemben, egy „átfogóbb", elemző jellegű lekérdezés például a következő: „Hogyan alakultak az ügyfelek bankban elhelyezett megtakarításai az elmúlt 12 hónapban?". Ha ezt az operatív adatbázis segítségével szeretnénk megválaszolni, az sok ideig tarthat és túlságosan leterhelheti az operatív adatbázist, és a rendszer terheltsége miatt sok ideig tarthati Gipsz Jakab számlaegyenlegének lekérdezése. Az átfogóbb, sok aggregációt tartalmazó elemző jellegű lekérdezések operatív adatbázison való közvetlen végrehajtása tehát nem praktikus. Az adattárház segítségével azonban épp az ilyen lekérdezéseket tudjuk hatékonyan megválaszolni, támogatva ezáltal a döntéshozatali folyamatokat.

   Az adattárházban szándékosan olyan olyan táblákban tároljuk az adatokat, hogy az elemző jellegű lekérdezések hatékonyan végrehajthatóak legyenek, például elemi adatok helyett aggregátumokat tárolunk, amelyekből a lekérdezésekben szereplő aggregációk gyorsabban kiszámíthatóak, mint az elemi adatokból. Az adattárházhoz intézett, nagyobb ívű átfogóbb lekérdezésekre nem feltétlenül várunk abszolút pontos válaszokat: ha egy adattárházból délután 4-kor kérdezzük le, hogyan alakultak az utóbbi 12 hónapban az ügyfelek megtakarításai, abban még nem biztos, hogy benne lesz Gipsz Jakab aznap lekötött betétje. Az adattárház adatai tehát nem feltétlenül frissek, ugyanakkor nyilván szükséges

[1] Ha például, az adatbázisunk definíciója szerint a *Lakóhely típusa* attribútum a *nagyváros, kisváros, falu* értékeket veheti fel, akkor egy "XI. kerület" bejegyzést hibának tekintünk, amelyet az adattisztítás során javítunk.

az adattárházbeli adatok rendszeres frissítése az operatív adatbázisban tárolt adatok alapján.

Adattárházak alkalmazásakor a trendek, folyamatok elemzése a cél. Az, hogy nem az aktuálisan legfrissebb adatokkal dolgozunk, általában nem okoz gondot, feltéve, hogy a legutóbbi frissítés óta nem következett be radikális változás. Ezzel szemben Gipsz Jakab nyilván nem örülne, ha a betét elhelyezése után este lekérdezve számláját „nem látná" a pénzét, például azért, mert a periodikus frissítés csak hetente egyszer esedékes. Szintén furcsa lenne, ha Gipsz Jakab a számlaegyenlegének lekérdezésekor egy olyan választ kapna a rendszertől, hogy 95 %-os valószínűséggel az egyenlege 100.000 és 200.000 forint közötti.

5. **Adattér csökkentése.** Ebben a lépésben az adatbázisból a cél szempontjából fontos attribútumokat emeljük ki és/vagy dimenziócsökkentést végzünk. Gyakran előfordul, hogy az attribútumok egymással korrelálnak, redundánsak, egy-egy objektum jóval kevesebb attribútummal is leírható, mint az eredeti adatbázisban. Ilyenkor dimenziócsökkentő eljárásokat használhatunk, például PCA-t [Dunteman, 1989, Jolliffe, 2005], MDS-t [Borg és Groenen, 2005], ISOMAP-t [Tenenbaum és tsa., 2000].

6. **Adatbányászati algoritmus típusának kiválasztása.** Eldöntjük, hogy a megoldandó feladat a 1.2. fejezetben bemutatásra kerülő adatbányászati alapfeladatok közül melyikre illeszkedik leginkább.

7. **A megfelelő adatbányászati algoritmus meghatározása.** A feladatot megoldó lehetséges algoritmusok közül kiválasztjuk azt, amelyik a konkrét esetben leginkább célravezető. Megvizsgáljuk az algoritmusok előnyeit, hátrányait, paramétereit, elemezzük a futási idő- és memóriaigényét. Gyakran szükség lehet a meglévő algoritmusok kisebb-nagyobb változtatására, az aktuális feladathoz való adaptációjára.

8. **A választott algoritmus alkalmazása.** Az előkészített adatainkat elemzzük a választott algoritmussal.

9. **A kinyert információ értelmezése, esetleg visszatérés az előző lépésekhez további finomítások céljából.** Megvizsgáljuk, hogy a kinyert (matematikai) összefüggés mit jelent az adott alkalmazási terület kontextusában, mennyiben járul hozzá a terület jobb megértéséhez, egy meglévő termék vagy szolgáltatás javításához, esetleg új termék vagy szolgáltatás létrehozásához.

10. **A megszerzett tudás megerősítése.** Összevetés az elvárásokkal, előzetes ismeretekkel. Eredmények dokumentálása és átadása a végfelhasználónak.

Egy adatbányászati elemzés eredménye akkor nem megfelelő, ha nem sikerül semmilyen új és hasznos összefüggést feltárni. Ennek több oka is lehet, néhányat külön is kiemelünk:

1. Előfordulhat, hogy rosszul választottuk meg az elemzéshez használt algoritmust vagy ennek paramétereit (lásd a 7. és 8. lépést), és egy másik eljárással (vagy más paraméterekkel) találni fogunk valamilyen érdekes összefüggést. Szemléletesen szólva: más oldalról ránézve az adathegyre, lehet, hogy látunk rajta valami érdekeset.

2. Lehetséges, hogy a tudásfeltárási folyamat lépését elrontottuk, olyan transzformációt hajtottunk végre, amely megakadályozta, hogy új összefüggést találjunk. Ha sejtjük, hogy melyik lépést ronottuk el, akkor visszatérünk arra a lépésre és onnantól újrakezdjük a folyamatot.

3. Legrosszabb esetben az is lehetséges, hogy az adatok egyáltalán nem rejtenek semmiféle új, a gyakorlatban hasznosítható összefüggést. Ekkor — sajnos — teljesen elölről kell kezdeni a folyamatot, új adatokat használva.

A sikeres adatbányászati projektekben általában az első öt lépés teszi ki az idő- és pénzráfordítások legalább 80%-át. Ha a célok nem kellőképpen átgondoltak és a bányászandó adatok nem megfelelő minőségűek, akkor könnyen előfordulhat, hogy az adatbányász csak vaktában dolgozik és a kinyert információnak semmi haszna sincs.

A tudásfeltárás során elengedhetetlen, hogy az adatbányász és az alkalmazási terület szakértője szorosan együttműködjön, a projekt minden fázisában ellenőrizzék a betartandó irányvonalakat. Nézzünk erre egy példát: ha adatbányászati eszközökkel sikerül kimutatni, hogy X betegséggel gyakran együtt jár Y betegség is, a kutatóorvos képes eldönteni azt, hogy ez valóban így van-e: megvizsgálhatja, hogy ugyanezen összefüggés más adathalmaz esetén is fennáll-e (esetleg direkt ebből a célból gyűjt adatot). Ha igen, akkor kiderítheti azt, hogy az egyik betegség során keletkezik-e olyan kémiai anyag, vagy elszaporodott-e olyan kórokozó, mely hozzájárul a másik betegség kialakulásához. Ezek alapján azt mondhatjuk, hogy az adatbányász „tippeket” ad a kutatóorvosoknak. Ezen „tippek” jelentősek, ezek óvhatják meg a kutatóorvost attól, hogy — szemléletesen fogalmazva — „rossz helyen tapogatózzon”. Az adatbányászat tehát első sorban új, ígéretes hipotézisek javaslatával járulhat hozzá más területeken zajló kutatásokhoz.

A következő valós példában az életmódra és a megbetegedésekre vonatkozó adatok elemezője jut a következtetésre, hogy a prosztatarák összefügg a szenesedésig sütött hús fogyasztásával. Ezzel „irányt mutat" a kutatóorvosnak, aki a háttérben rejlő kémiai reakciókat és azok biológiai következményeit tárja fel. Ez a konkrét esetben lényegében így is történt: előbb tárták fel a jól átsütött hús fogyasztása és a prosztatarák gyakorisága közötti összefüggést, majd megtalálták a hús sütéskor keletkező PhIP vegyületet és kimutatták, hogy hatására prosztatarák alakulhat ki.[2]

Ez a jegyzet első sorban a 6-8. lépéseket veszi szemügyre. A tudásfeltárási folyamat ezen szakaszát szokták a *szűkebb értelemben vett adatbányászatnak* nevezni. Feltételezzük, hogy rendelkezésünkre áll egy adatbázis, tudjuk, milyen jellegű információra van szükségünk, és az adatbányász feladata, hogy ennek megoldására minél gyorsabb és pontosabb algoritmust adjon.

A tudásfeltárás fentiekben felvázolt folyamatával kapcsolatban megjegyezzük, hogy ez egy *vázlatos séma*, melyet a valós adatbányászati projektek nem feltétlenül követnek teljes mértékben. A folyamat harmadik lépésében említettük például a hiányzó értékek pótlását. Erre azonban nincs feltétlenül szükség, ha később, a nyolcadik lépésben, olyan adatbányászati algoritmust használunk, amely számára nem jelent problémát a hiányzó értékek jelenléte.[3] Sok esetben nem szükséges a teljes folyamatot végrehajtani: egy jól kialakított, megbízható, hibamentes adatokat tartalmazó adattárház rengeteg vezetői döntés támogatására képes lehet. Lehetséges, hogy már önmagában az adattárház is kielégíti a felhasználó igényeit. Esetenként a folyamat lépései akár önmagukban is értékesek lehetnek: például az operatív adatbázison is érdemes lehet elvégezni az adattisztítást, duplikátumok keresését. Végezetül megjegyezzük, hogy hasonló technikákat, algoritmusokat használhatunk több különböző lépés során: például osztályozó és regressziós algoritmusokat nem csak a nyolcadik lépésben használhatunk, hanem duplikátumok keresésére is [Christen, 2008].

Az elemzés célja szerint kétféle adatbányászati tevékenységet különítünk el:

Feltárás: A feltárás során az adatbázisban található mintákat keressük meg. A minták legtöbbször az általános trendeket/szokásokat/jellemzőket írják le, de vannak olyan alkalmazások is (például csalásfelderítés), ahol éppen az általánostól eltérő/nem várt mintákat keressük.

Előrejelzés: Az előrejelzésnél a feltárt minták alapján próbálunk következtetni a jövőre. Például egy elem ismeretlen értékeit próbáljuk előrejelezni az ismert értékek és a feltárt tudás alapján.

[2] Rákkeltő anyagok a McDonaldsban és Burger Kingben, http://index.hu/gazdasag/vilag/mcrak060929

[3] Az osztályozó algoritmusok közül ilyen többek közt a Naive Bayes és nḧány döntési fára építő algoritmus.

Négy fontos elvárásunk van a megszerzett tudással kapcsolatban: (1) legyen könnyen érthető, (2) legyen érvényes, (3) legyen hasznos és (4) legyen újszerű. Az érvényesség eldöntése a terület szakértője mellett az adatbányász (esetleg statisztikus) feladata is. Előfordulhat, hogy helyes modellt adtunk, az algoritmus is jól működött, a kinyert szabály mégsem fedi a valóságot. Bonferroni tétele[4] arra figyelmeztet bennünket, hogy amennyiben a lehetséges következtetések száma túl nagy, akkor egyes következtetések tényleges valóságtartalom nélkül igaznak mutatkoznak, tisztán statisztikai megfontolások alapján.

A helytelen következtetésre az egyik leghíresebb példa az alábbi[5]: Az 50-es években David Rhine parapszichológus diákokat vizsgált meg azzal a céllal, hogy parapszichológiai képességgel rendelkezőket találjon. Minden egyes diáknak 10 lefedett kártya színét kellett megtippelne (piros vagy fekete). A kísérlet eredményeként bejelentette, hogy a diákok 0,1%-a parapszichológiai képességgel rendelkezik (a teljesen véletlenszerűen tippelők között a helyesen tippelők várható száma statisztikailag nagyjából ennyi, hiszen annak valószínűsége, hogy valaki mind a tíz kártyát eltalálja $\frac{1}{2^{10}} = \frac{1}{1024}$). Ezekkel a diákokkal újra elvégezte a kísérletet, ám ezúttal a diákok eredménye teljesen átlagos volt. Rhine következtetése szerint az, aki parapszichológiai képességgel rendelkezik és erről nem tud, elveszti eme képességét, miután tudomást szerez róla.

Egy másik példa a valóságtartalom nélküli szabály kinyerésére az alábbi, megtörtént eset. Amerikában a Dow Jones átlag becsléséhez keresni kezdték azt a terméket, amely árának alakulása leginkább hasonlított a Dow Jones átlag alakulásához. A kapott termék a bangladesi gyapot volt. A bangladesi gyapot ára és a Dow Jones átlagának alakulása közt megfigyelt hasonlóság azonban pusztán a véletlen műve volt.

Rosszabb esetben még az is előfordulhat, hogy az eredményként kapott összefüggés nem csak, hogy nem igaz, abban az értelemben, hogy az összefüggésben szereplő dolgok között a valóságban nincs kapcsolat, hanem épp a kapott összefüggés ellenkezője igaz, lásd a Simpson-paradoxont a 5.2.6. fejezetben.

Az adatok illetve az információk megjelenítésének módja legalább annyira fontos, mint az összefüggések meghatározása. A végfelhasználókat (vezetőket) jobban megragadja egy jól elkészített ábra, mint a matematikai összefüggések nyers tálalása. A megjelenítés tehát fontos része az adatbányászatnak. Ezt igazolja, hogy nagy sikert könyvelnek el az olyan adatbányászati szoftverek, amelyek adatbányászati algoritmusokat nem is futtatnak, pusztán az adatokat jelenítik meg "intelligens" módon, háromdimenziós, színes, forgatható ábrák segítségével. Ezeknél a rendszereknél az összefüggéseket, mintázatokat, közös

[4]http://statpac.com/manual/index.htm?turl=bonferronistheorem.htm
[5]http://infolab.stanford.edu/~ullman/mining/overview.pdf

tulajdonsággal rendelkező csoportokat maguk a felhasználók veszik észre. Az adatbányászati szoftverekről részletesebben 9. fejezetben olvashatunk.

1.2. Adatbányászati alapfeladatok

Nagy adathalmazok elemzésének, a „rejtett" tudás feltárásának igénye sok különböző területen jelentkezett és jelentkezik, úgy mint a marketing, biztosítás, hitelintézetek, orvostudomány vagy mérnöki alkalmazások.

Érdekes módon, a különböző területek szakértői, kutatói — elméleti, „matematikai" szempontból — nagyon hasonló feladatokra jutottak. A feladatok megoldása során alkalmazott eljárások algoritmusok is sokszor egyazon eljárás külöböző változatai. Elsőre talán meglepő lehet, hogy például a kéretlen elektronikus levelek (spam-ek) automatikus felismerésére sok szempontból hasonló modellt használhatunk, mint annak előrejelzésére, hogy egy banki ügyfél vissza fogja-e fizetni a számára folyosított hitelt. Amint látni fogjuk, az osztályozás különböző területeken alkalmazott felismerő és előrejelző rendszerek közös elméleti keretét alkotja. Ehhez hasonlóan az alábbiakban leírt további adatbányászati alapfeladatok is számos alkalmazásban fordulnak elő.

A szakirodalom, lásd pl. [Tan és tsa., 2005], általában négy adatbányászati alapfeladatot határoz meg:

Osztályozás és regresszió. Az osztályozó algoritmusokat és azokkal kapcsolatos ismereteinket különféle felismerési és előrejelzési feladatok közös elméleti hátterének tekinthetjük. Ilyen felismerési feladat többek között a számítógéppel automatikusan végzett kézírásfelismerés, beszédfelismerés vagy jelbeszédi jelek felismerése. Szintén osztályozási feladatnak tekinthető annak előrejelzése, hogy egy bank potenciális ügyfelei közül várhatóan kik fogják késedelem nélkül visszafizetni a hitelüket és kik nem. Hasonló feladat annak becslése, hogy egy biztosítónál gépjárműveiket biztosító ügyfelek közül ki milyen valószínűséggel okoz majd balesetet. Lemorzsolódás-előrejelzési feladatok (azaz mely ügyfelek fogják várhatóan elhagyni az adott szolgáltatót), egy-egy blogbejegyzésre érkező kommentek számának előrejelzése és további „egzotikus" felismerési feladatok is ebbe a körbe tartoznak, mint például a számítógépes felismerése annak, hogy egy adott szöveg szerzője nő-e vagy férfi [Stańczyk, 2011].

Feltehetjük, hogy az adatbázisunk valamilyen példányok (ügyfelek, betegségek, vásárlók, telekommunikációs események, stb.) tulajdonságait írja le. Egy-egy tulajdonság egyszerű esetben egy számmal vagy szimbólummal írható. Ekkor az adatbázis egy nagy táblázat, melynek egyes sorai az egyes példányoknak felelnek meg, oszlopai pedig a tulajdonsá-

goknak (egy-egy oszlop egy-egy tulajdonságnak). Egy ilyen adatbázist szemléltet az 1.2. ábra, amely egy kereskedő ügyfeleinek körében végzett felmérés során gyűtött adatokat tárolja. Az *Életkor* tulajdonság értékei: *fiatal*, *középkorú*, *idős*. A tulajdonság helyett gyakran használjuk majd az attribútum szót[6]. Amikor minden attribútum szám, a példányok egy sokdimenziós tér pontjainak feleltethetők meg, ezért az attribútum helyett a *dimenzió* kifejezést is használhatjuk. A *példányra* más szóval *objektum*, *elem*, *rekord* néven is hivatkozik a szakirodalom.

Ilyen megközelítésben az osztályozás illetve regresszió feladata valamely ismeretlen attribútum becslése illetve előrejelzése. Ezt a kitüntetett attribútumot nevezzük osztályattribútumnak (class attribute, class label). Amennyiben az osztályattribútum értékkészlete diszkrét (az osztályattribútum előre definiált értékek valamelyikét veszi fel), *osztályozási* feladatról beszélünk, ha az osztályattribútum értéke folytonos, akkor *regressziós* feladatról.

Ha például az 1.2. ábrán látható adatbázis esetében néhány ügyfélről nem tudjuk, hogy érdekelni fogja-e őket az akciót, és ezt szeretnénk ügyfelenként előrejelezni, egy osztályozási feladattal van dolgunk. Az osztályattribútum ezesetben az *Érdekli-e az akció* elnevezésű attribútum.

Ha egy biztosítótársaság, a korábbi példák egyikét folytatva, ügyfeleinek különböző tulajdonságait tárolja (életkorukat, jövedelmük nagyságát, az általuk vezetett autó végsebességét, motorjának teljesítményét, stb.) és azt szeretné előrejelezni, hogy egy ügyfél mekkora eséllyel okoz balesetet a következő évben, akkor a baleset valószínűsége lesz az osztályattribútum. Ez az előrejelzési feladat egy regressziós feladat, hiszen az előrejelzendő érték (baleset valószínűsége) folytonos.

Klaszterezés. Osztályozási feladatok esetében, amikor az osztályattribútum értéke néhány, előre definiált érték valamelyike, úgy tekinthetjük, hogy az objektumokat előre definiált csoportok valamelyikébe soroljuk be: egy-egy csoport az osztályattribútum egy-egy értékének felel meg. Ezzel szemben a klaszterezés során a csoportok előre nem ismertek, a feladat a csoportok felfedezése, feltárása, és az egyes objektumok besorolása a megtalált csoportokba. Az objektumokat tehát előre nem definiált csoportokba (klaszterekbe) kell sorolnunk úgy, hogy az egy csoportba tartozó objektumok hasonlóak legyenek, míg a különböző csoportba kerültek különbözzenek egymástól.

[6] A közgazdászok a tulajdonság helyett *ismérvet*, valamely tulajdonság konkrét értéke helyett *ismérv változatot* mondanak.

Tipikus klaszterezési feladat például az ügyfelek szegmentálása, de klaszterező algoritmusokat használhatunk dokumentumok vagy képek csoportosítására, szociális hálózatok és csillagászati adatok elemzésére, valamint nagy teljesítményű szuperszámítógépek komponenseinek elrendezésekor. Klaszterezésre mutat példát az 1.3 ábra első fele. Az adatbázisbeli objektumokat itt a sík pontjainak feleltettük meg. Ez akkor lehetséges, ha azt feltételezzük, hogy az adatbázisbeli objektumok két számmal megadott attribútummal rendelkeznek: az egyik attribútum a vízszintes, a másik pedig a függőleges koordinátatengelynek feleltethető meg. Ilyen ábrázolás mellett a hasonló objektumok egymáshoz közeli pontoknak felelnek meg, az egymástól különböző objektumok pedig távoli pontoknak.

Gyakori minták és asszociációs szabályok keresése. Történetileg kifejezetten érdekes a gyakori mintázatok és asszociációs szabályok keresésének feladata, mert szorosan összefügg az adatbányászat, mint önálló terület kialakulásával. Ezzel szemben az osztályozással, regresszióval, klaszterezéssel már korábban is foglalkoztak.

Asszociációs szabályok alatt olyan jellegű összefüggéseket értünk, mint például az alábbi:

Aki dohányzik és sok alkoholt fogyaszt, sokkal nagyobb eséllyel lesz rákos, mint aki nem.

A gyakori minták keresésének feladatát legtöbbször kereskedelmi példákon keresztül szokták bevezetni. Tételezzük fel, hogy arra vagyunk kíváncsiak, hogy egy bevásárlóközpont által árusított termékek közül melyek azok, amelyeket gyakran vásárolnak egyszerre a vevők. Ebben a kontextusban egy-egy gyakori minta termékek egy halmazát jelöli, olyan termékeket, amelyeket jellegzetesen egyszerre vásárolnak meg. Egy gyakori minta lehet például a

zsemle, tej, szalámi, sajt,

egy másik pedig a

sör és pelenka.

Amint látni fogjuk, a gyakori mintázatok bányászata szorosan összekapcsolódik az asszociációs szabályok bányászatával. Szintén látni fogjuk, hogy a minta típusától függően az alapfeladatnak különböző változatai vannak: kereshetünk gyakori halmazokat, gyakori sorozatokat, gyakori részgráfokat, figyelembe vehetjük azt, hogy a bevásárlóközpont termékei különböző kategóriákba tartoznak, stb.

Anomáliák felismerése. Más szóval: eltéréselemzés, különc pontok keresése, illetve outlier-ek felismerése. Azokat a példányokat, amelyek nem felelnek meg az adatbázis általános jellemzőinek, tulajdonságaik nagy mértékben eltérnek az általánostól, az adatbázisbeli példányok többségétől, *különc* példányoknak nevezzük. Jópár adatbányászati algoritmus az ilyen különc pontoknak nem tulajdonít nagy jelentőséget, zajnak vagy kivételnek kezeli őket. Azonban egyre több területen merül fel az igény, hogy éppen az ilyen különc pontokat találjuk meg. Eltéréselemzés főbb alkalmazási területe a csalások, visszaélések kiszűrése, beleértve a vírusok, hackertámadások, biztosítási csalások, hitelkártyákkal elkövetett illegitim tranzakciók, és a belterjes kereskedés felismerését, mobiltelefon-hálózatok és egészségügyi szolgáltatások jogosulatlan igénybe vételét [Chandola és tsa., 2009]. Különc pontok keresésére mutat példát az 1.3 ábra második fele.

Az anomáliakeresés feladata nagyban összefügg az osztályozással és klaszterezéssel. Sokszor osztályozó algoritmusokat használnak anomáliakeresésre. Ahogy említettük, klaszterezés során az adatbázisbeli objektumokat csoportosítjuk úgy, hogy a hasonlók egy csoportba kerüljenek, különbözők pedig különböző csoportokba. Azok az objektumok, amelyek nem illeszkednek *jól* egyik csoportba sem, különc pontoknak tekinthetők.

Léteznek ugyanakkor az osztályozó és klaszterező algoritmusoktól lényegesen különböző megközelítést követő eltéréselemző algoritmusok, például valószínűségi eloszlásokon, távolság és lokális sűrűség fogalmán alapuló eljárások [Chandola és tsa., 2009]. Ezért tekinthetjük az anomáliakeresést az adatbányászat negyedik alapfeladatának.

A fenti alapfeladatok (osztályozás, klaszterezés, gyakori mintázatok és asszociációs szabályok keresése, anomáliák felismerése) különböző változatai léteznek alkalmazási területtől és ezzel összefüggően az adatok típusától függően. Így külön-külön beszélhetünk például ügyfelek, dokumentumok, röntgenképek osztályozásáról, különböző típusú gyakori minták bányászatáról, stb. Az alapfeladatok különböző változatai mellett az adatbányászat területéhez sorolhatjuk többek között az alábbi, az alapfeladatokhoz lazán kapcsolódó feladatokat, alkalmazásokat is:

Ajánlórendszerek és további, mátrix faktorizáción alapuló eljárások. Az online (webes) kereskedelem utóbbi évtizedben tapasztalható rohamos terjedésével párhuzamosan nőtt az érdeklődés a személyre szabott reklámok, ajánlatok iránt, népszerűvé váltak az *ajánlórendszerekkel* kapcsolatos kutatások. Ha egy webes áruházban, például az Amazon, Netflix vagy Rossmann weblapján, vásárolunk néhány terméket, a webes áruházba való következő bejelentkezésünkkor látható reklámok nem véletlensze-

rűen jelennek meg a képernyőn, hanem korábbi vásárlásaink alapján. A háttérben futó rendszer becsüli, hogy milyen az ízlésünk és, hogy mely további termékekre lehet szükségünk a korábban vásároltakhoz kapcsolódóan, stb. Ehhez hasonlóan a Youtube (és más videomegosztó rendszerek) személyreszabottan ajánl számunkra videókat, a Facebook lehetséges ismerősöket ajánl.

Ajánlórendszernek (recommender system) nevezünk egy olyan rendszert, amely a termékek halmazából a felhasználók számára személyre szabottan ajánl néhányat. Amikor egy ajánlórendszer termékeket ajánl, ezt általában az alapján teszi, hogy a felhasználó által még nem vásárolt termékeket rangsorolja és a rangsorból kiválasztja az első néhányat, amelyek várhatóan leginkább érdeklik őt. Ahhoz, hogy az ajánlatok személyre szabottak legyenek, a rangsorolást minden felhasználóra külön-külön végzi el a rendszer, felhasználónként más rangsorokat generál a korábbi vásárlások figyelembe vételével. A rangsorolás legtöbbször úgy történik, hogy a rendszer az egyes termékekhez kiszámít egy valószínűséget vagy egy folytonos skálán értelmezett pontszámot, amely azt jellemzi, hogy az adott felhasználót az adott termék mennyire érdekli. A rendszer kimenete tehát folytonos érékek becslése, amely alapján az ajánlórendszereket akár a regressziós problémák közé is sorolhatnánk.

Azonban az ajánlórendszerek hátterében álló adatstruktúra, a „szokványos" regressziós eljárásokhoz képest, jelentősen különböző. További lényeges eltérés az, *ahogyan* az legsikeresebb ajánló algoritmusok a becsült értékeket kiszámolják. Az utóbbi években a témában született szinte hihetetlen mennyiségű tudományos cikk eredményeiből az rajzolódik ki, hogy az ajánlórendszerek hátterében álló adatokat érdemes egy ritka mátrixként elképezelni, lásd az 1.4 ábrát. Ritka mátrix alatt itt azt értjük, hogy a mátrix celláinak nagy része kitöltetlen. Az ajánló algoritmusok többsége az ismert cellák alapján becsüli meg az ismeretlen cellák értékeit, általában olyan módon, hogy a mátrixot kettő vagy több kisebb mátrix szorzatára bontja [Takács, 2008, Koren, 2009]. Ezeket mátrixfaktorizációs eljárásoknak nevezzük.

Idősorok bányászata: Az adatbányászati alapfeladatok idősorokkal kapcsolatos változatai – mint például idősorok osztályozása, idősorok klaszterezése, gyakori minták (motívumok) keresése, idősorok következő értékének előrejelzése – új kihívásokat rejt, melyekkel a 7. fejezetben foglalkozunk.

Attribútumok közötti kapcsolatok: Gyakran hasznos, ha a példányokra úgy tekintünk, mint az attribútumok megvalósulásaira és keressük az összefüggéseket az attribútumok között. Többféle összefüggés létezik.

Ilyenek az asszociációs- és korrelációs szabályok, a funkcionális függőségek és hasonlóságok. Az *osztályozás* is attribútumok közötti összefüggések felfedezésére szolgál. Az osztályozásnál egy kitüntetett attribútum értékét kell megjósolnunk a többi attribútum értéke alapján. Ezt egy modell felépítésével tesszük. Leggyakrabban a modell egy döntési fa, de lehet if-then szabályok sorozata, valamilyen matematikai formula, vagy akár egy neurális hálózat is.

Webes adatbányászat: Az Interneten óriási adattömeg található, így az interneten alapuló információ-kinyerő algoritmusok is az adatbányászat területéhez sorolhatóak. Szintén ide tartozónak tekinthetjük az oldalak rangsorolásának, illetve hasonló tartalmú oldalak megtalálásának feladatát, a kéretlen elektronikus levelek (spamek) felismerését vagy az interneten megjelenő tartalmakhoz kapcsolódó előrejelzési feladatokat (például: várhatóan hány felhasználó fog betölteni egy weblapot vagy hányan fognak megnézni egy youtube-ra feltöltött videót).

1.3. Sikeres alkalmazások

A következőkben az adatbányászat számos sikeres alkalmazása közül sorolunk fel néhányat a teljesség igénye nélkül:

- Osztályozó algoritmusok segítségével sikeresen oldották meg a nemkívánatos elektronikus levelek (spam-ek) felismerését[7] [Blanzieri és Bryl, 2008, Cormack, 2007].

- Az online kereskedelemben használt ajánlórendszerek a legelterjedtebb adatbányászati alkalmazások közé tartoznak, lásd például amazon.com, youtube vagy facebook ajánlórendszereit (az ajánlórendszerekről bővebben a 1.2. fejezetben írunk).

- Webes keresőrendszerek[8] esetében egy-egy szótöredéket beírva a rendszer lehetséges szavakat, szóösszetételeket kínál fel, megtippelvén, hogy mire keresünk. Ehhez hasonlóan, ha egy közösségi címkézőrendszer (social tagging system) egy felhasználója valamilyen címkével szándékozik ellátni egy képet, videót, hangfájlt, stb., a rendszer lehetséges címkéket javasol [Jäschke és tsa., 2008].

[7] http://en.wikipedia.org/wiki/Bayesian_spam_filtering
[8] Pl. Google: www.google.com

- Az ember (vagy más élőlény) genotípusának elemzéséhez a gének nagy száma miatt szintén adatbányászati algoritmusok szükségesek. Csak néhány példát említünk a sikeres alkalmazások közül: a cukorbetegség bizonyos változataiért felelős géncsoportok feltárását, valamint a transzkripciós faktor kapcsolódási helyek[9] (transcription factor binding site) azonosítását. Az utóbbihoz gyakori mintákat kereső algoritmusokat használtak. Az emberi genom feltárásával, a személyreszabott gyógyászat (personalized medicine) fejlődésével ez a terület várhatóan egyre fontosabb lesz [Roden és tsa., 2009].

- Osztályozó eljárásokat sikeresen használtak orvosi adatok elemzésére. [Reiz és Csató]

- A bankok gyakran alkalmaznak olyan automatikusan előállított döntési fákat, amelyek alapján egy program javaslatot tesz egy hitel megítéléséről. Ezt a kérelmezők személyes adatai valamint korábbi hitelfelvételi és törlesztési adatai alapján teszi [Thomas, 2000]. Igazolták, hogy a hitelbírálat minősége javult az USA-ban, amikor a bankok áttértek a kötelezően alkalmazott, írásban rögzített szabályok alkalmazására [Thomas, 2000]. Ezeket a szabályokat pedig az adatbányászat segítségével állították össze.

- A vásárlói szokások felderítése áruházakban hasznos lehet az áruház terméktérképének kialakításánál, akciók, eladáshelyi reklámok, leárazások szervezésénél [Liao és tsa., 2008].

- Adatbányászati eljárásokat sikeresen alkalmaztak csillagászati feladatokra [Way és tsa., 2012].

- Utazásszervezéssel kapcsolatos minták kinyerésével hatékonyabban (és ennek következtében nagyobb nyereséggel) megszervezhetők a nagy költségfaktorú tényezők, pl. szállodai szobák, repülőjegyek leárazása, vagy áremelése.

- Gyártási folyamatok során gyakran a beállítási paraméterek finomhangolására van szükség. A kőolaj és a földgáz szétválasztása az olajfinomítás egyik lépése, az elválasztási folyamat kontrollálása nem könnyű feladat. A British Petroleum olajvállalat a gépi tanulás technikáját használta a paraméter-beállítás szabályainak megalkotására. Az új eljárásnak köszönhetően tíz percre csökkentették a paraméter-beállításhoz szükséges

[9] A DNS kitüntetett részei az transzkripciós faktor kapcsolódási helyek (transcription factor binding site), melyek olyan szakszai az DNS-nek, ahová fehérjék kapcsolódhatnak, és segítségükkel a kapcsolódási helyet követő DNS szakasz átíródhat RNS-sé, hogy később az RNS-ről fehérjék szintetizálódhassanak.

időt, míg a feladat korábban a szakértők több, mint egy napi munkáját jelentette [Langley és Simon, 1995].

- A Westinghouse cég nukleáris tüzelőanyag-cellák gyártása során ütközött problémákba, és szintén a gépi tanulás segítségével hoztak létre folyamatkontrollálási szabályokat. Ezzel 10 millió dollárt sikerült megspórolniuk az 1984-es évben. A Tennessee állambeli R.R. Donelly nyomdaipari cég is adatbányászati technikákat alkalmazott a retogravúr nyomdagépek irányítására, így csökkentve a hibás paraméter-beállítások következtében keletkező selejtes nyomatok számát évi 500-ról 30-ra.

- A vírusölő programok az ismert vírusokat lenyomataik alapján detektálják, az ismeretleneket pedig többnyire heurisztikus módon szűrik. Adatbányászati algoritmusok felhasználásával az ismert vírusok tulajdonságai alapján olyan modellt állítottak fel, ami jól leírja a vírusok tulajdonságait [Schultz és tsa., 2001a, Schultz és tsa., 2001b]. A modellt sikeresen alkalmazták új vírusok kiszűrésére.

- Az új-zélandi tejgazdaságoknak minden évben kemény üzleti döntést kell meghozniuk: ki kell választani, hogy a szarvasmarha állomány mely egyedeit tartják meg, és melyeket értékesítik vágóhidaknak. Tipikusan minden gazdaság ötödik egyede kerül mészárszékre a fejési idény végén, ahogy az élelmezési tartalékok kiapadnak. A döntést az egyes példányok tenyészadatai és múltbéli tejtermelékenységi mutatója befolyásolja. További kritikus faktorok az egyed kora, kórtörténete, szülési komplikációk, agresszivitás, illetve az, hogy a következő szezonban vemhes-e. Több millió szarvasmarha egyedenként több mint 700 tulajdonságát rögzítették az évek során. A kutatók azt vizsgálják, hogyan használható fel a gépi tanulás annak megállapítására, hogy a sikeres farmerek mely faktorokat veszik számításba a szelektálásnál. Ezzel nem a döntési folyamat gépesítése a céljuk, hanem a sikerstratégia kitanulása, és annak közkinccsé tétele [Witten és tsa., 2011].

1.4. Az adatbányászat megközelítései

Az adatbányászatban a hipotézisek megtalálása áll a középpontban, míg a statisztika több hangsúlyt fektet hipotézisek vizsgálatára.

Az adatbányászat egy gyakorlatorientált terület, kevesebb súlyt kapnak az elméleti elemzések. Viszont központi kérdés egy algoritmus futási ideje és memóriaigénye. Ezért az adatbányászati algoritmusok bemutatása során ki fogunk térni az adatstruktúrákkal kapcsolatos és akár implementációs kérdésekre is.

A következőkben néhány példán keresztül szemléltjük, hogy milyen megközelítést követ az adatbányászat.

1. Tegyük fel, hogy egy adatbázisban sokmillió ember DNS-szekvenciáit és tulajdonságait tároljuk. Egy jellegzetes statisztikai kérdés lehet például az, hogy van-e szignifikáns összefüggés egy adott DNS-részszekvencia és egy adott betegség között. Ennek eldöntésére statisztikai próbát alkalmazhatunk. Egy adatbányász nem kérdezne rá egy konkrét részszekvencia és egy konkrét betegség közötti összefüggésre, hanem egy általánosabb kérdést tenne fel, például azt, hogy milyen összefüggés van a betegségek és a részszekvenciák között és, hogy mely részszekvenciák mely betegségek kialakulásának esélyét növelik?[10]

2. Egy statisztikai elemzés során megvizsgálhatjuk, hogy a nők illetve férfiak hány százaléka dohányzik, milyen szignifikáns eltérés van a két csoport között. Egy adatbányászati elemzés során most is általánosabb kérdést tennénk fel, például azt, hogy milyen jellegzetes csoportok vannak a dohányzásra nézve? A csoportokat tehát nem adjuk meg előre, nem helyezzük a nőket az egyik, a férfiakat pedig a másik csoportba. Az adatbányász feladata, hogy úgy csoportosítsa az embereket, hogy a hasonlók egy csoportba kerüljenek. Az adatbányászatban az ilyen feladatokat nem hosszas emberi munka és intuíció árán oldjuk meg, hanem törekszünk a minél nagyobb fokú automatizálásra. Eredményként könnyen lehet, hogy nem nemek szerinti csoportosítást kapunk, hanem olyat, melyben ugyanazon csoportokba férfiak és nők is kerültek, akik dohányzásra vonatkozó jellemző tulajdonságaik alapján hasonlóak.

3. Az előbbi példában más irányba is általánosíthatunk: lehet, hogy arra vagyunk kíváncsiak, hogy mi a különbség a férfiak és a nők között. Ismerjük tehát a két csoportot, de nem adjuk meg, hogy mely tulajdonságok jellemzőek egy-egy csoportra. Ekkor egy osztályozási feladattal állunk szemben, a csoportokat osztályoknak nevezzük. Amint látni fogjuk, az osztályozási feladatot megoldó algoritmusok egyik csoportja döntési fákat készít. Egy *döntési fa* látható az 1.5. ábrán. Az ábra felülről lefele

[10]Az ember DNS-e jelenlegi ismereteink szerint kb. 23000 gént tartalmaz, egy-egy gént százezres nagyságrendű bázispár ír le. Leegyszerűsítve úgy képzelhetjük el, hogy az egyes gének aktívak vagy inaktívak lehetnek: ha például egy személy valamely gén olyan mutációjával rendelkezik, amely hozzájárul egy betegség kialakulásához, abból nem következik, hogy a betegség ténylegesen is ki fog alakulni, mert szerencsés esetben az adott gén inaktív. A gének aktív vagy inaktív voltát környezeti tényezők, úgy mint táplálkozás, életmód, dohányzás, pszichológiai tényezők befolyásolják. Egy valós elemzés során tehát az ilyen, területspecifikus háttérismereteket is figyelembe kell vennünk.

olvasandó. A legfelső csomópontot nevezzük gyökérnek, azon csomópontokat pedig, amelyekből nem indulnak ki további élek, leveleknek hívjuk. A levelek az osztályoknak (nők illetve férfiak) felelnek meg. A gyökérben és a fa közbülső csomópontjaiban egy-egy attribútum (adattáblabeli oszlop) neve látható. A fa egy csomópontjából kiinduló ágak az adott csomóponthoz tartozó attribútum egy-egy lehetséges értékének felelnek meg. Egy döntési fa azt mutatja meg, hogy ha nem ismernénk, hogy egy ember melyik osztályba tartozik, akkor hogyan dönthetnénk ezt el. A legfelső csomópontból, a fa gyökeréből indulunk ki, és az attribútumok értékét követve a fa ágai mentén haladunk egészen addig, amíg egy levélhez nem érünk. Például a fogamzásgátlót szedő hallgatók nők. Az *idealbmirange* attribútum a válaszadó által saját maga számára ideálisnak tartott testsúly alapján számított testtömegindex.{ootnoteBMI = body mass index Látható, hogy a fogamzásgátlót nem szedő hallgatók közül az alacsonyabb testsúlyt ideálisnak tartók nők, míg a magasabb testsúlyt ideálisnak tartók férfiak.

1.5. Adatbányászati algoritmusokkal szembeni alapkövetelmények

Adatbányászati algoritmusokkal szemben két alapkövetelményt fogalmazhatunk meg: egyrészt a feltárt összefüggések, az elemzés eredményének gyakorlati hasznosíthatóságát, a másrészt pedig az, hogy az algoritmus skálázható legyen óriási méretű adatbázisokhoz: az adatbázis méretének növekedése mellett az algoritmus futásideje ne növekedjen elfogadhatatlanul nagyra. A következőkben ezt a két követelményt részletezzük.

Előfordulhat, hogy az adatbányászati rendszer, például asszociációs szabályok keresésekor, még megfelelően megválasztott paraméterek mellett is túl sok szabályt, összefüggést tár fel. Az egyik legnehezebb kérdés az, hogy ezek közül melyek az érdekesek. Érdekességi mutatókról általánosságban nem sok mondható el, mert a különböző felhasználási területeken más-más minta lehet hasznos. Megkülönböztetünk szubjektív és objektív érdekességi mutatókat. Egy minta mindenképpen érdekes, ha meglepő, azaz eddigi tudásunknak ellentmond, vagy újszerű, azaz tudásunkat kiegészíti. Ugyanakkor egy információ csak akkor érdekes, ha tudunk valamit kezdeni vele [Silberschatz és Tuzhilin, 1995]. Azt, hogy egy szabály mennyire meglepő – több-kevesebb sikerrel – tudjuk formalizálni. Az újszerűségről és a felhasználhatóságról azonban csak a terület szakértője tud nyilatkozni.

Annak ellenére, hogy az adatbányászat új terület, a fentiekből látható, hogy

régi, már ismert problémákat is magába foglal. Gondoljunk itt arra, hogy klaszterező algoritmusokat már a 60-as években is javasoltak, vagy arra, hogy az osztályozás irodalmával több könyvespolcot is meg lehetne tölteni. Tehát az adatbányászatban gyakran nem maga a probléma új, hanem az adatok mérete, továbbá az a követelmény, hogy az algoritmusok futási ideje kellően rövid legyen. Az alkalmazásokban nem ritkák a tera- sőt petabájt nagyságú adathalmazok. [Edelstein, 1999] cikkében például egy beszámolót olvashatunk egy bank adatbázisának elemzéséről adatbányászati eszközökkel, ahol az ügyfelek száma elérte a 190 milliót az adatok mérete pedig a 4 TB-ot. Ilyen méretek mellett már kvadratikus lépésigényű algoritmusokat sem engedhetünk meg. Látni fogjuk, hogy a legtöbb adatbányászati algoritmus a teljes adatbázist kevés alkalommal olvassa végig.

Skálázható (scalable) és hatékony (efficient) algoritmusokat keresünk, amelyek megbirkóznak nagy méretű adatbázisokkal. Elvárjuk, hogy az adatbázis fontosabb paramétereinek ismeretében az algoritmusok futási ideje megjósolható legyen. Az óriási memóriaméretek miatt a legtöbb elemzendő adatbázis – megfelelő átalakításokkal – valószínűleg elfér a memóriában, de mégis sokszor azt feltételezzük, hogy az adat a háttértáron található.

Az adatbázisok méretének növekedése miatt egyre fontosabbak a párhuzamosítható algoritmusok (lásd például partíciós algoritmusról szóló részt). Ezek az adatbázist részekre osztják, majd az egyes részeket külön memóriával és háttértárral rendelkező egységek dolgozzák fel, és végül egy kitüntetett egység egyesíti a részeredményeket. Szintén a méretnövekedés az oka azon algoritmusok népszerűségének, amelyek futási ideje nagy mértékben csökkenthető valamilyen előzetes információk (például korábbi futási eredmények) ismeretében (lásd asszociációs szabályok karbantartásával foglalkozó szakaszt).

1.6. Szabványok

Kezdetben sok adatbányászati projektre jellemző volt, hogy az adatbányászok megkapták az adatokat és némi információt az alkalmazási területről és cserébe várták tőlük a kinyer információkat. A szoros együttműködés hiánya azonban csak olyan információkhoz vezetett, amelyekkel az alkalmazási terület emberei nem sokat tudtak kezdeni. Az adatbányászat elterjedésével jött az igény, hogy legyen egy szabvány az adatbányászati projektek lebonyolításáról. Így született meg a CRISP-DM (CRoss Industry Standard Process for Data Mining) [Chapman és tsa., 1999], amely adatbányászati eszköztől és felhasználási területtől függetlenül leírja, hogy miként kellene kinéznie egy adatbányászati projektnek, illetve meghatározza a fontos lépéseket és a potenciális veszélyeket. A CRISP-DM szerint a tudáskinyerés az 1.6. ábra szerinti módon jön létre.

Az adatbányászati folyamat szabványosítása mellett egyre nagyobb az igény a folyamat egyes lépéseiben felmerülő megoldások, problémák és eszközök szabványosítására. Ezek közül a legismertebbek:

- az XML alapú PMML (Predictive Modeling Markup Language), amely az adatbányászati eredmények szabványos leírását szolgálja;
- a Microsoft analysis szerver adatbányászati funkciókkal kibővített szabványa (OLE DB for data mining);
- az ISO törekvései multimédia és alkalmazás specifikus SQL típusok és a hozzá tartozó eljárások definiálására (SQL/MM);
- Java adatbányászati API (JDMAPI).

1.7. Adatbányászati rendszer architektúrája

Egy adatbányászati rendszernek kapcsolatban kell lennie az adatbázissal, a felhasználóval és esetleg egy tudásalapú rendszerrel. Ezek alapján egy tipikus adatbányászati architektúra az 1.7. ábrán látható.

Adatbázis, adattárház vagy más adat raktár: Itt találhatók a tényleges adatok, ami lehet egy adatbázis, vagy adattárház, akár egy munkalap vagy bármilyen tárolt információ. Az adattisztítás és integráció közvetlenül az adatokon is elvégezhető.

Adatbázis vagy adattárház szerver: A szerver felelős a felhasználó által kért adat kézbesítéséért.

Tudásbázis: A területre jellemző, részlegesen vagy teljesen formalizálható tudás található itt. Fontos szerepe lehet ennek a keresési tér szűkítésénél, a kinyert minták érdekességének meghatározásánál, különböző paraméterek és küszöbszámok meghatározásánál.

Adatbányász motor: Az adatbányász motorban futnak a különböző adatbányászati algoritmusok.

Mintákat kiértékelő modul: Felelős a kinyert minta vagy összefüggések kiértékeléséért. Minél jobban egybe tudjuk építeni az adatbányászatot a minta kiértékelésével, annál hatékonyabb és gyorsabb a tudásfeltárás.

Grafikus felhasználói felület: Itt zajlik a kommunikáció a felhasználó és az adatbányászati rendszer között. A felhasználó itt adhatja meg, hogy melyik adatbázisban milyen jellegű összefüggéseket keres és ezen a rétegen keresztül láthatja a végeredményt. Az összefüggések átlátható és érthető tálalása rendkívül fontos, hiszen ennek hiánya elriasztja a felhasználót az adatbányászattól.

1.8. Az adatbányászat feltételei

Tagadhatatlan, hogy a sikertelen adatbányászati projektek száma nagy, és az adatbányászat sok esetben nem váltotta be a hozzá fűzött reményeket. Ennek oka egyrészről az adatbányászati szakemberhiány, másrészről az, hogy alapvető feltételek nem teljesültek a projektek kivitelezése során. A sikeres adatbányászati projekt egyik legfontosabb feltétele az adatbányász és a terület szakértőjének szoros együttműködése. A további feltételek az alábbiak:

Nagy mennyiségű adat, amely a kinyert szabályok statisztikai jelentőségét növeli. Minél nagyobb az adatmennyiség, annál biztosabban tudjuk kizárni bizonyos összefüggések esetiségét, azaz annál kisebb az esélye, hogy a talált összefüggés a véletlen eredménye. Sajnos sok adatot sokáig tart feldolgozni, sőt az algoritmusok egy jelentős része akkor működik csak igazán hatékonyan, ha az adatbázis elfér-e a memóriában (lásd például a tranzakciós adatbázist szófában tároló algoritmusokat a 5.2. fejezetben).

Sok attribútum: Ha az objektumokat leíró attribútumok száma kicsi, akkor hagyományos eszközökkel (grafikonok, egyszerű táblázatok, kis dimenziós, forgatható, színes ábrák, stb.) is fel tudjuk tárni a tudást. Kevés attribútum esetén a kinyerhető tudás sem lehet túl sokféle. Az adatbányászat ereje akkor mutatkozik meg, amikor az attribútumszám olyan nagy, hogy a hagyományos módszerekkel nem tudjuk feldolgozni az adatot.

Tiszta adat: Az adatok jó minősége az adatbányászat egyik alapfeltétele. A zajok, a hibás bejegyzések jó esetben csak nehezítik az adatbányászatot (például amikor ismerjük az adatokban található zaj, ill. bizonytalanság fokát), rosszabb esetben azonban hamis eredményekhez vezetnek. Az ilyen rossz minőségű adatokra remek példa hazánk orvosi adatbázisa (rengeteg hibás bejegyzés, kitöltetlen mező, eltérő mértékegység alapú bejegyzések, szöveges bejegyzések), pedig az ezekből kinyert információk értékesek lennének. A "szeméthalmazban" való kutakodást tréfásan

GIGO-nak (garbage in, garbage out[11]) nevezik.

Torzítatlan adat: Az adatbányászat sikeressége múlhat az adatok nem megfelelő kiválasztásán. Ide tartozó fogalom a BIBO (bias in, bias out[12]), mely arra figyelmeztet, hogy ha egy részsokaság alapján akarunk következtetni az alapsokaságra, akkor figyelembe kell vennünk a részsokaság kiválasztásának szempontjait, illetve az abból adódó (esetleges) torzításokat. Például, ha a lakosságot az anyagi helyzet szerint akarjuk csoportokba sorolni, de csak nyugat-magyarországi adatok állnak rendelkezésünkre, akkor tudnunk kell, hogy a kapott eredmény (a csoportok leírása) torz lesz, hiszen a részsokaság átlag életszínvonala jobb az alapsokaságénál.

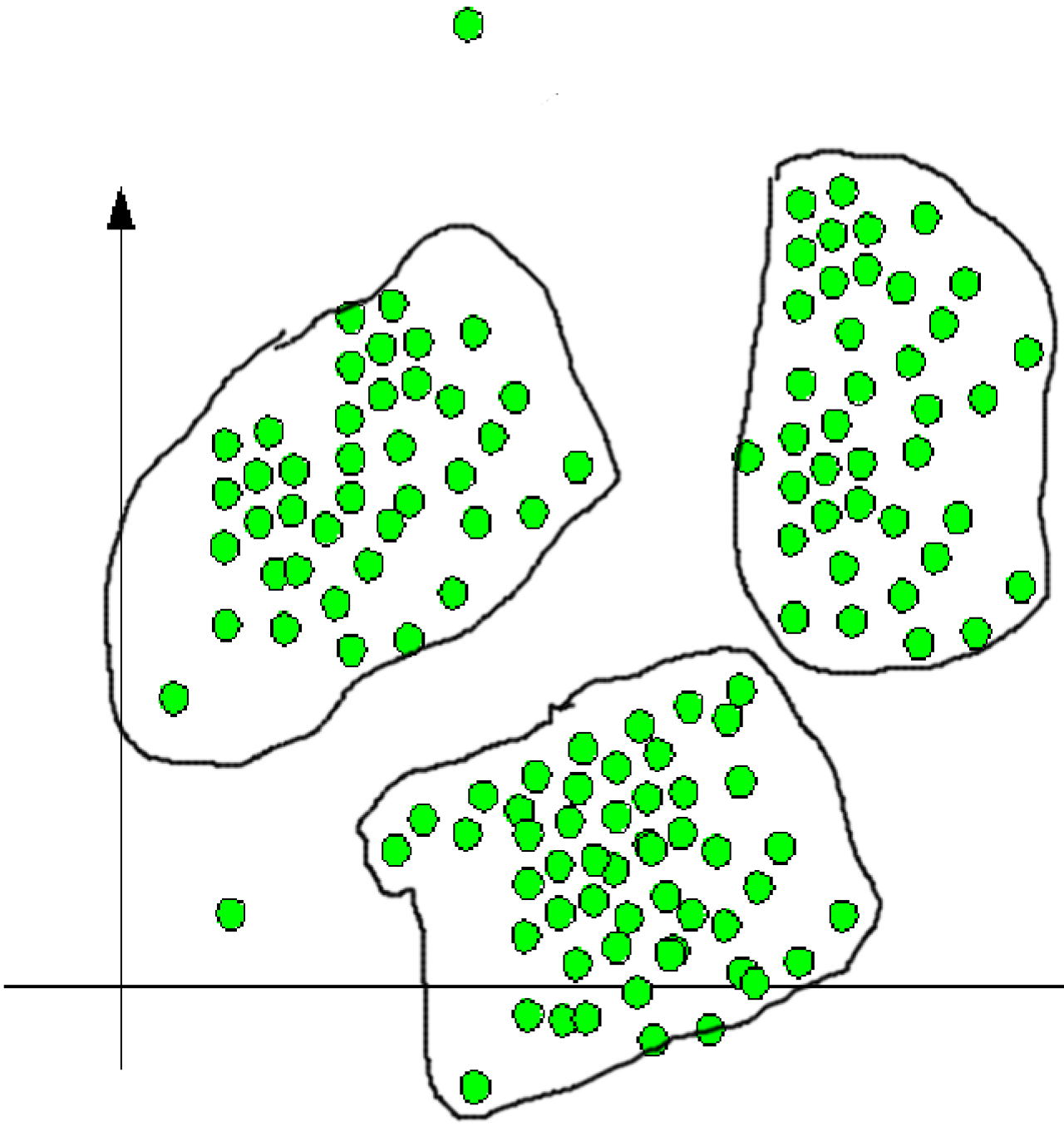
Alkalmazási terület akcióképessége: Gyakran előfordul, hogy a tudást kinyerik ugyan, de a felhasználása elmarad. Gyakran a felhasználási területek túl merevek, vagy a változtatás túlságosan magas költségekkel járna. A legtöbb adatbányászati esettanulmányban a tudás kinyerésének módjáról esik szó, a tudás felhasználásáról pedig ritkán hallunk.

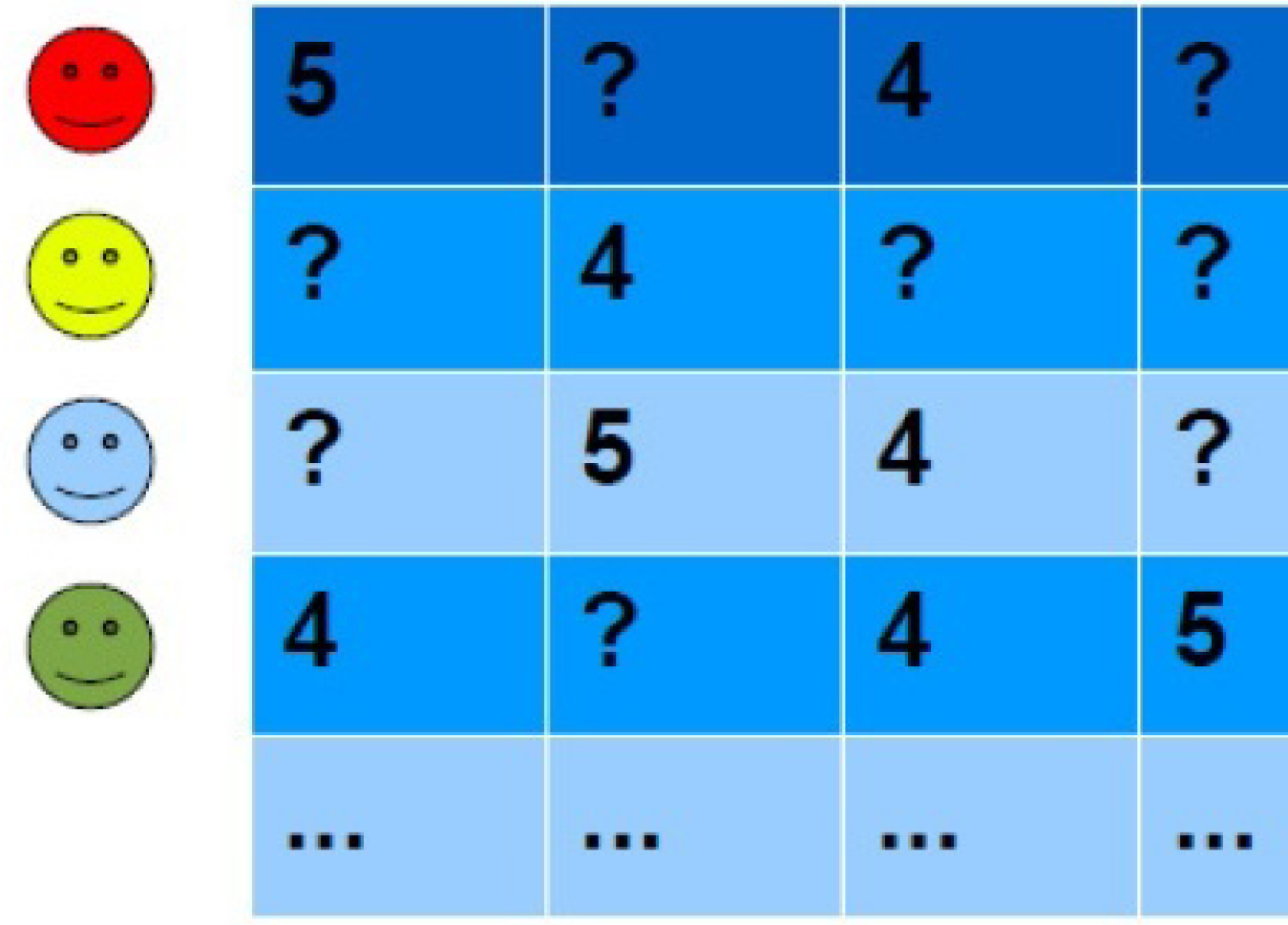
A befektetés megtérülésének (Return On Investment) mérhetősége: Egy adatbányászati projektről akkor állíthatjuk biztosan, hogy sikeres, ha a befektetés hatását mérni, vagy viszonylag pontosan becsülni tudjuk.

[11] szemét be, szemét ki
[12] torzítás be, torzítás ki

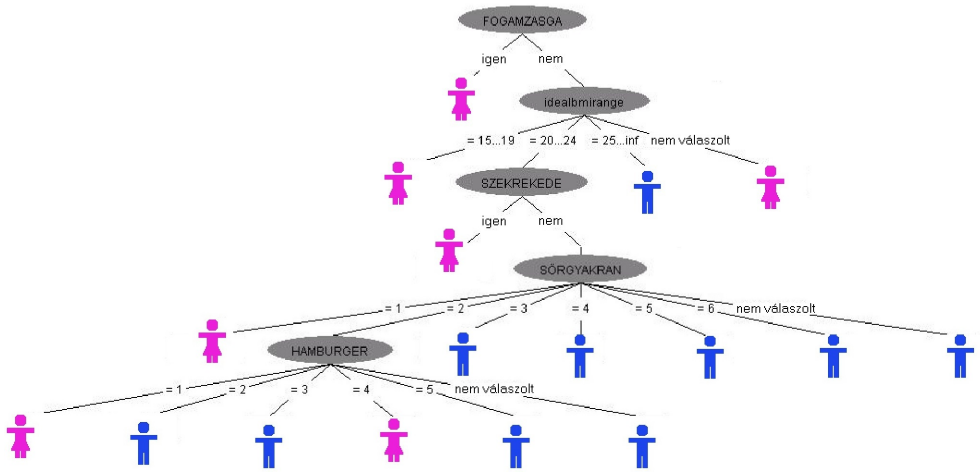
Sorsz.	Életkor	Testsúly
1	fiatal	alacsony
2	idős	közepes
3	középkorú	magas
4	idős	közepes
5	fiatal	magas
6	középkorú	alacsony
7	idős	alacsony
8	fiatal	közepes
9	középkorú	magas
10	idős	közepes

1.2. ábra. Példa: Egy adattábla

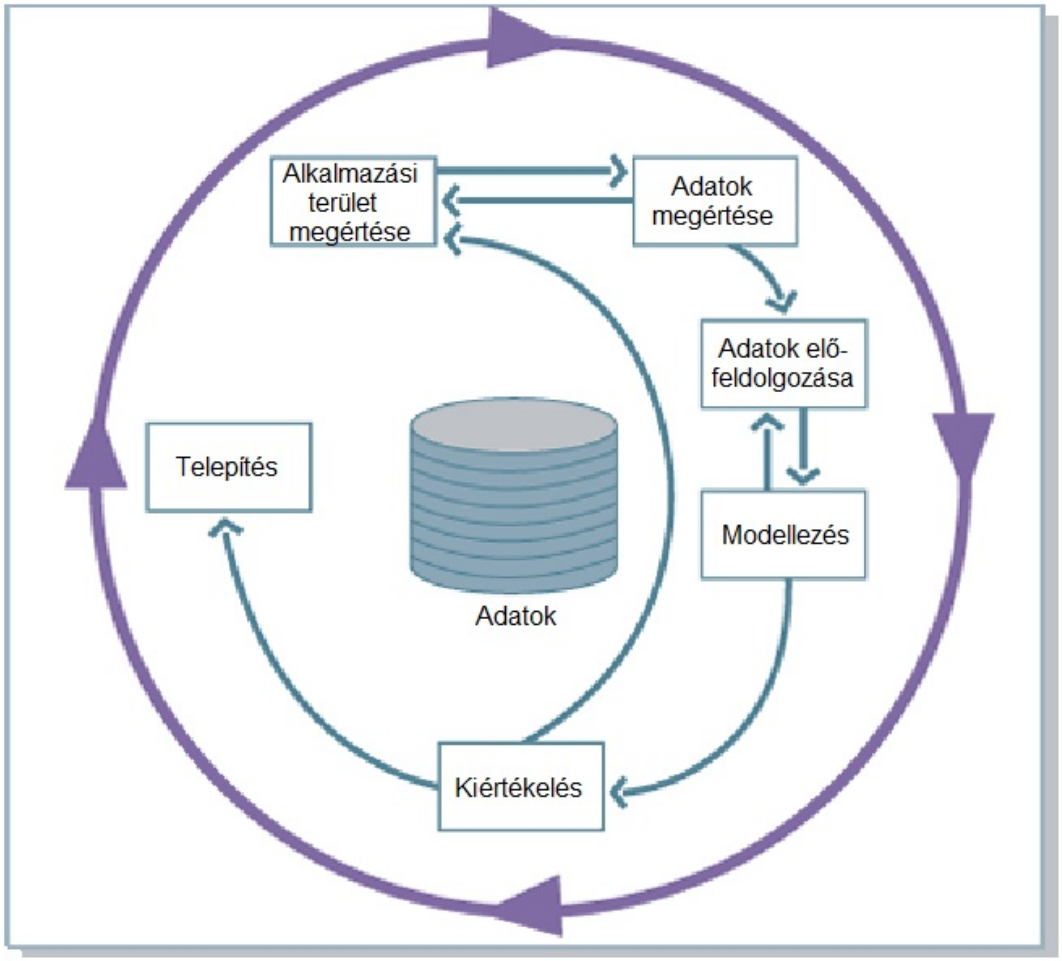




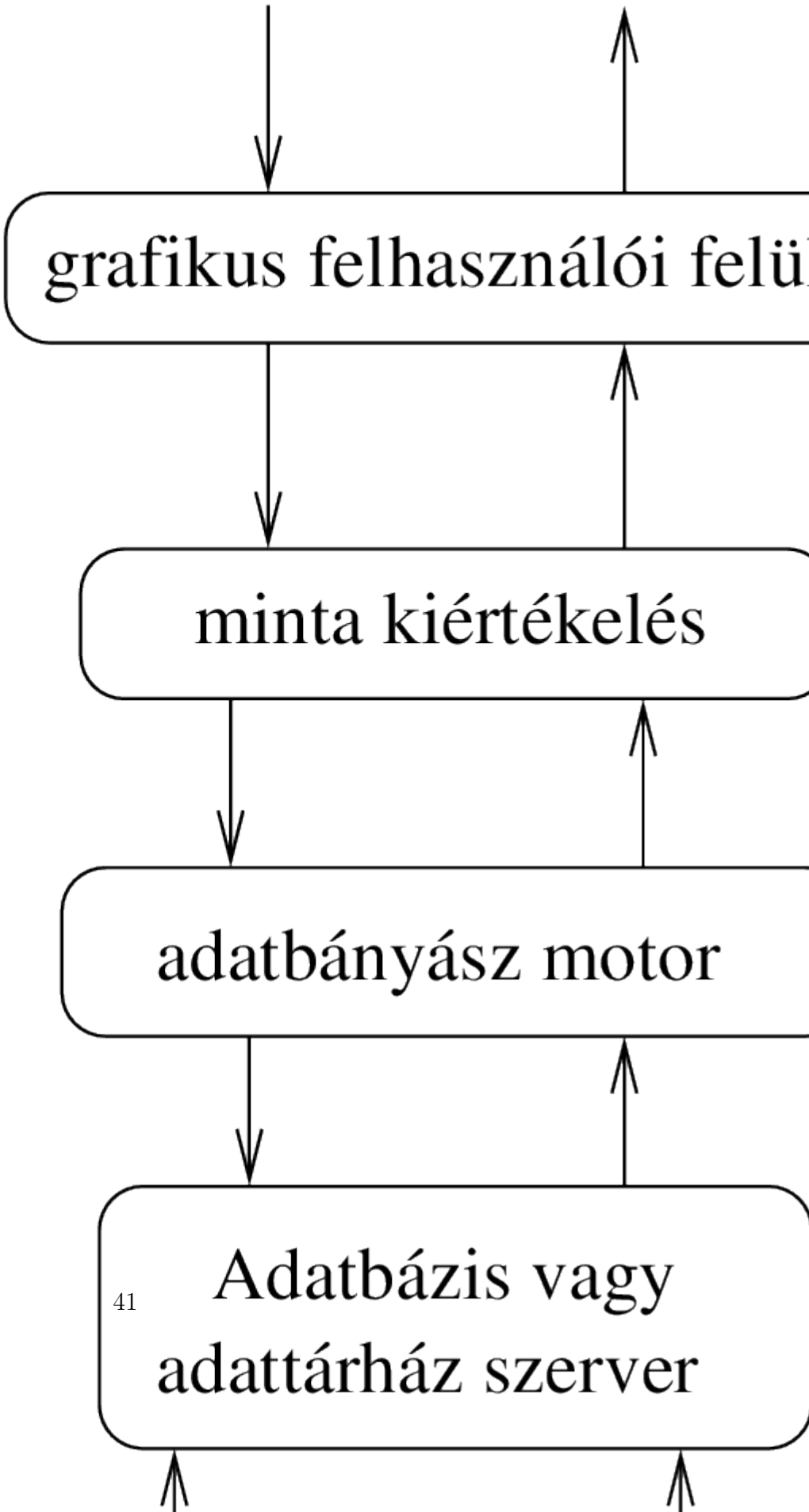
1.4. ábra. Az ajánlórendszerek hátterében álló adatokat általában egy ritka mátrix elemeinek szokták tekinteni. A mátrix sorai a felhasználóknak felelnek meg, oszlopai az egyes termékeknek, a példában ezek a termékek filmeknek. Feltehetjük, hogy néhány terméket a felhasználók 1-től 5-ig terjedő skálán értékeltek. A termékek nagyrészéről azonban nem tudjuk, hogy egy-egy felhasználónak tetszenek-e vagy sem, ezeket az esetekel jelöltük kérdőjelekkel. A feladat az, hogy eldőntsük mely termékeket érdemes az egyes felhasználók számára reklámozni, azaz: becsüljük meg, hogy mely termékek fognak várhatóan



1.5. ábra. Döntési fa: nők és férfiak közötti különbségek a Semmelweis Egyetem hallgatóinak körében végzett felmérés alapján.



1.6. ábra. A tudásfeltárás folyamata a CRISP-DM szabvány szerint [Chapman és tsa., 1999]


grafikus felhasználói felü
minta kiértékelés
adatbányász motor
Adatbázis vagy
adattárház szerver

2. fejezet

Alapfogalmak, jelölések

Ebben a részben definiáljuk a jegyzetben használt fogalmakat és összefoglaljuk a jegyzet megértéséhez szükséges legfontosabb háttérismereteket. Ez az áttekintés szükség szerűen rendkívül tömör. További részletek tekintetében ajánljuk Fazekas, Feller és Rényi tankönyveit [Fazekas, 2000, Feller, 1978, Rényi, 1968], valamint Fleiner Tamás jegyzeteit [Fleiner, 2011, Fleiner, 2012]. Célszerű akkor átnéznünk e fejezet egyes részeit, amikor az olvasás során olyan részbe ütközünk, ami nem teljesen világos.

A bemutatásra kerülő halmazelméleti alapfogalmakat (2.1. fejezet) az egyes adatbányászati feladatok formális megfogalmazásához, valamint a gyakori elemhalmazokat kereső algoritmusok leírásához használjuk. A valószínűségi változók következőkben tárgyalt eloszlásai (2.2. fejezet) valamint az ezekkel kapcsolatos összefüggések a mintavételezés és a gyakori minták illetve asszociációs szabályok értékelésére használt statisztikai próbák (2.3. fejezet) alapjait képezik. Az entrópia fogalmát (2.2.4. fejezet) osztályozó algoritmusban (például döntési fák), mintavételezésnél és klaszterező eljárások kiértékelésénél használhatjuk fel. A modern processzorok sajátosságait az algoritmusok hatékony implementációja során vesszük figyelembe. Szintén a hatékony implementációt segítik elő a bemutatott adatstruktúrák.

2.1. Halmazok, relációk, függvények, sorozatok

A *halmaz* különböző objektumok együttese, amelyeket a halmaz *elemeinek* hívunk. Ha x eleme a H halmaznak, akkor azt így jelöljük: $x \in H$, a halmaz elemeinek számát (rövidebben *elemszámát*) pedig $|H|$-val. A jegyzetben a természetes számok halmazát ($\{0,1,\dots\}$) $\mathbb{N}$-el jelöljük, a valós számok halmazát $\mathbb{R}$-el, az egész számok halmazát $\mathbb{Z}$-vel, az üres halmazt (egyetlen elemet sem tartalmazó halmaz) $\emptyset$-val. Két halmaz akkor egyezik meg, ha ugyanazok

az elemeik. X részhalmaza Y-nak ($X \subseteq Y$), ha X minden eleme Y-nak is eleme. Ha $X \subseteq Y$, de $X \neq Y$, akkor X *valódi részhalmaza* Y-nak. A valódi jelzőt gyakran fogjuk használni, és a valódi részhalmaz analógiájára azt értjük rajta, hogy az egyenlőséget kizárjuk. Sajnos a superset angol szónak nincsen általánosan elfogadott fordítása, pedig sokszor szeretnénk használni. Azt fogjuk mondani, hogy Y *bővebb* X-nél, ha ($X \subseteq Y$). A halmazműveletek jelölése és pontos jelentésük: metszet: $X \cap Y = \{z : z \in X \text{ és } z \in Y\}$, unió: $X \cup Y = \{z : z \in X \text{ vagy } z \in Y\}$, különbség: $X \setminus Y = \{z : z \in X \text{ és } z \notin Y\}$.

Két halmaz (X, Y) *Descartes-szorzata* ($X \times Y$) az összes olyan rendezett párból álló halmaz, amelynek az első komponense (tagja) X-ben, a második Y-ban van. Az X, Y halmazokon értelmezett *bináris reláció* az $X \times Y$ részhalmaza. Ha (x, y) eleme a ϕ relációnak, akkor azt így is jelölhetjük: $x\phi y$. A $\preceq$ reláció *részben rendezés* (vagy parciális rendezés), ha *reflexív* ($x \preceq x$), *antiszimmetrikus* ($x \preceq y$ és $y \preceq x$ feltételekből következik, hogy $x = y$), tranzitív ($x \preceq y$ és $y \preceq z$ feltételekből következik, hogy $x \preceq z$). Ha az előző 3 feltételben az antiszimmetrikus helyett szimmetrikusat ($x \preceq y$-ből következik, hogy $y \preceq x$) mondunk, akkor *ekvivalencia-relációról* beszélünk. A továbbiakban, tetszőleges $\preceq$ rendezés esetén, ha $x \neq y$ és $x \preceq y$, akkor azt így jelöljük $x \prec y$. Legyen X részhalmaza X'. A X' halmaznak $y \in X$ egy *alsó korlátja*, ha $y \preceq x$ minden $x \in X'$-re. Az y *legnagyobb alsó korlát*, ha minden y' alsó korlátra $y' \preceq y$. Az y *maximális alsó korlátja* X'-nak, ha nem létezik olyan y-tól különböző y' alsó korlát, amire $y \preceq y'$. Hasonlóan értelmezhető a felső, legkisebb felső, minimális felső korlát fogalmak is. A $\prec$ rendezés *teljes rendezés*, ha minden $x \neq y$ elemre $x \prec y$, $y \prec x$ közül az egyik fennáll. Az $(X, \preceq)$ párost *hálónak* nevezzük, ha $\preceq$ az X-en értelmezett parciális rendezés, és tetszőleges $x, y \in X$ elemeknek létezik legnagyobb alsó (jelölésben: $x \wedge y$) és legkisebb felső korlátjuk ($x \vee y$).

Központi fogalom a *lexikografikus rendezés*. Nézzük először ennek a matematikai definícióját. Legyen X és Y két halmaz, amelyeken értelmezve van egy-egy parciális rendezés ($\prec_X, \prec_Y$). Azt mondjuk, hogy a $(x_1, y_1) \in X \times Y$ lexikografikusan megelőzi $(x_2, y_2) \in X \times Y$ párt, ha $x_1 \prec_X x_2$, vagy $x_1 = x_2$ és $y_1 \prec_Y y_2$. A lexikografikus rendezést tetszőleges számú halmaz Descartes-szorzatára is kiterjeszthetjük rekurzív módon az alábbiak alapján: $X \times Y \times Z = X \times (Y \times Z)$. Látható, hogy a lexikografikus rendezést Descartes szorzatokon értelmezzük, vagy más szóval olyan összetett struktúrákon, amelyeknek ugyanannyi tagjuk van (n-eseknek is hívják ezeket). Mi ezt szeretnénk általánosítani, hiszen például szavak sorba rendezésénél is előfordulnak eltérő hosszúságú szavak. Ha a rövidebb szó megegyezik a hosszabb szó első felével (például komp és kompenzál szavak), akkor megegyezés alapján a rövidebb szó előzi meg lexikografikusan a hosszabbikat. Ezek alapján definiálható a lexikografikus rendezés eltérő számú halmazok Descartes szorzatára. A legtöbb esetben a Descartes szorzat tagjainak halmaza és a rajtuk definiált rendezések megegyeznek (pl.:

$X = Y$ és $\prec_X = \prec_Y$). Ilyenre, adott rendezés szerinti lexikografikus rendezésként hivatkozunk.

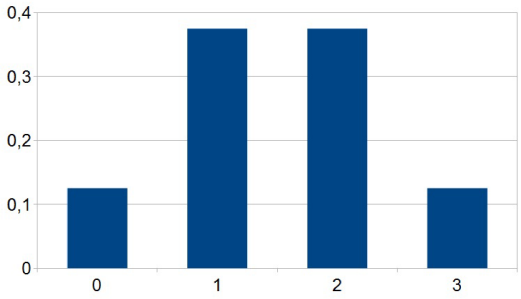
Az X, Y halmazokon értelmezett f bináris reláció *függvény*, ha bármely $x \in X$ esetén pontosan egy olyan $y \in Y$ létezik, hogy $(x, y) \in f$. Ez jelölésben $f : X \to Y$, és, ha $(x, y) \in f$, akkor $y = f(x)$. Az X halmazt a f *értelmezési tartományának* hívjuk (vagy máshogy: f az X-en értelmezett), Y-t az f képhalmazának, az $f(X)$ halmazt[1] pedig az f *értékkészletének*. Azt a függvényt, amelyet úgy kapunk, hogy először a f, majd az g függvényt alkalmazzuk $g \circ f$-el jelöljük. *Predikátum* egy függvény, ha az értékkészlete az $\{igaz, hamis\}$ halmaz. *Szürjektív* egy függvény, ha a képhalmaza megegyezik az értékkészletével, *injektív* (vagy más néven egy-egy értelmű leképzés), ha az értelmezési tartomány bármely két különböző eleméhez különböző értéket rendel és *bijektív* (másképpen a függvény egy *bijekció*), ha szürjektív és injektív is egyben.

Legyen H tetszőleges halmaz. Az $f : \overbrace{H \times \cdots \times H}^{n} \to H$ függvényt n változós *műveletnek* nevezzük. A H halmazon értelmezett kétváltozós $\star$ műveletet *asszociatívnak* nevezzük, ha tetszőleges $a, b, c \in H$ esetén $(a \star b) \star c = a \star (b \star c)$. A $(H, \star)$ párt *félcsoportnak* nevezzük, ha $\star$ a H-n értelmezett asszociatív művelet. A $(H, \star)$ félcsoport elemein a H elemeit értjük. Ha a $(H, \star)$ félcsoport elemei között létezik olyan e elem, amelyre $e \star a = a \star e = a$ minden $a \in H$ elemre, akkor e-t *egységelemnek* hívjuk és egységelemes félcsoportól beszélünk. Ha egy egységelemes félcsoportban minden elemnek létezik inverze, akkor *csoportról* beszélünk. Az a inverze (a^{-1}) olyan elem, amelyre teljesül, hogy $a \star a^{-1} = a^{-1} \star a = e$. A csoport *Ábel-csoport*, ha a $\star$ művelet *kommutatív* ($a \star b = b \star a$) is. A $(H, \star, +)$ hármas egy *gyűrű*, amennyiben $(H, \star)$ Ábel csoport, $(H, +)$ félcsoport és a $\star, +$ műveletek *disztributívek* egymásra nézve, azaz $(a + b) \star c = a \star c + b \star c$. A $\star$ és a $+$ műveletek egységelemeit az 1 és a 0 szimbólumok jelölik. *Testnek* hívjuk az olyan kommutatív gyűrűt, ahol az $1 \neq 0$ és a 0-án kívül a H minden elemének van inverze.

A H halmaz felett értelmezett *multihalmaznak* vagy *zsáknak* nevezzük azt a halmazt, amelynek elemei olyan párok, amelyek első tagja H egy eleme, második tagja pedig egy pozitív egész szám. Egy multihalmazt szokás úgy ábrázolni mintha olyan halmaz lenne, amely egy elemet többször is tartalmazhat. Ilyenkor a pár első tagját annyiszor írjuk le, amennyi a pár második tagja. Például a $\{(A, 1), (C, 3)\}$-at $\{A, C, C, C\}$-vel ábrázoljuk. A multihalmaz *méretén* a párok második tagjainak összegét, elemszámán pedig a párok számát értjük.

Sokat fogjuk használni a *sorozat* fogalmát. Legyen S egy halmaz. Az $f : \mathbb{N} \to S$ függvényt az S felett értelmezett sorozatnak hívjuk. Leírására az $f(0), f(1), \ldots$ helyett a $\langle s_0, s_1, \ldots \rangle$ jelölést fogjuk használni. *Véges so-*

[1] $X = \{x_1, x_2, ..., x_m\}$ esetén $f(X) = \{f(x_1), f(x_2), ..., f(x_m)\}$.



2.1. ábra. Jelölje az X valószínűségi változó a lánygyermekek számát három gyermekes családokban. A hisztogram X lehetséges értékeinek valószínűségeit mutatja.

rozatok esetében az f értelmezési tartománya általában az $\{1,2,\ldots,n\}$ véges halmaz. Véges sorozat *hossza* az értelmezési tartományának elemszáma. Az $S = \langle s_1, s_2, \ldots s_n \rangle$ és $S' = \langle s'_1, s'_2, \ldots s'_{n'} \rangle$ sorozatok konkatenációján a két sorozat összefűzéseként keletkező $\langle s_1, s_2, \ldots s_n, s'_1, s'_2, \ldots s'_{n'} \rangle$ sorozatot értjük, melyet $\langle S, S' \rangle$-el jelöljük.

2.2. Valószínűségszámítás

A *valószínűségi változó*, *eloszlásfüggvény* és *sűrűségfüggvény* fogalmakat egy példán keresztül szemléltetjük.

Tekintsünk három gyermekes családokat! Jelölje X a lánygyermekek számát. X lehetséges értékei tehát $0, 1, 2$ és 3. Kiszámolhatjuk, hogy mekkora a valószínűsége X egyes értékeinek. Például annak a valószínűsége, hogy $X = 3$, azaz mind a három gyermek lány:

$$\mathbb{P}(X = 3) = 0.5 \times 0.5 \times 0.5 = 0.125.$$

X lehetséges értékeinek valószínűségeit mutatja a 2.1. ábrán látható hisztogram. X egy diszkrét értékű valószínűségi változó. Ha azt a kérdést tesszük fel, hogy mi a valószínűsége annak, hogy X értéke 1 és 3 közötti, beleértve a 1-t és a 3-at, nincs más dolgunk, mint a hisztogramon látható oszlopok közül kiválasztani azokat, amelyek a kérdéses tarománynak felelnek meg, és magasságaikat összeadni:

$$\mathbb{P}(1 \leq X \leq 3) = 0.375 + 0.375 + 0.125 = 0.875.$$

Ebben a példában X egy valószínűségi változó. A X valamely lehetséges

értékét x-szel jelöljük, azaz például az előbbi összeget így is írhatjuk:

$$\mathbb{P}(1 \leq X \leq 3) = \sum_{x \in \{1,2,3\}} \mathbb{P}(X = x).$$

Tekintsük most azt az esetet, hogy X egy véletlenszerűen választott ember testmagasságát jelöli. Feltételezzük, hogy a testmagasságot tetszőleges pontossággal tudjuk mérni, így X egy folytonos valószínűségi változó, elméletileg végtelenül sokféle különböző értéket vehet fel. Ezért az előbbivel ellentétben most nem egy oszlopdiagramot rajzolunk, hanem egy folytonos görbét, lásd a 2.2. ábra bal oldalát. A diszkrét valószínűségi változó esetéhez hasonlóan, most is feltehetjük például azt a kérdést, hogy mi a valószínűsége annak, hogy X értéke 170 és 175 centiméter közötti. Az oszlopok nagyságának összeadása helyett, most a 2.2. ábra bal oldalán látható görbe alatti területet kell kiszámolnunk 170 és 175 között. Ehhez jelöljük $f(x)$-szel a 2.2. ábra bal oldalán látható függvényt:

$$\mathbb{P}(170 \leq X < 175) = \int_{x=170}^{x=175} f(x)dx$$

A 2.2. ábra bal oldalán látható görbét nevezzük valószínűségi sűrűségfüggvénynek. Legfontosabb tulajdonsága, hogy két x_0 és x_1 érték közti határozott integrálja annak a valószínűségét adja, hogy a valószínűségi változó értéke x_0 és x_1 közötti. A folytonos valószínűségi változó eloszlásfüggvényét, melyet $F(x)$-szel jelölünk, lásd a 2.2. ábra jobb oldalát, a valószínűségi sűrűségfüggvényből így kapjuk:

$$F(x) = \mathbb{P}(X < x) = \int_{x_0=-\infty}^{x_0=x} f(x_0)dx_0$$

Jelöljük az X valószínűségi változó értékkészletét $\mathbb{X}$-szel. A diszkrét X valószínűségi változó *várható értéke*:

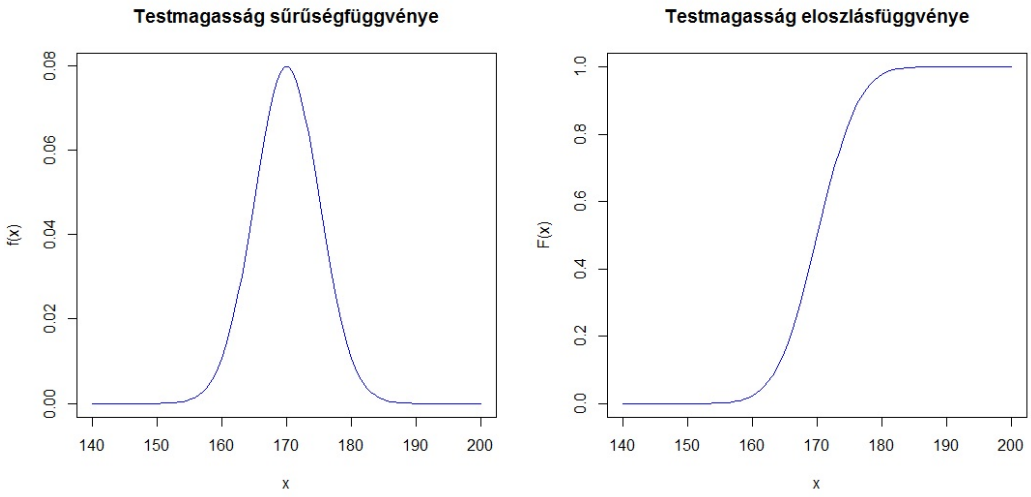
$$\mathbb{E}[X] = \mu = \sum_{x \in \mathbb{X}} x \cdot \mathbb{P}(X = x).$$

Folytonos esetben, az előbbi analógiájára, integrálással definiálhatjuk a várható értéket:

$$\mathbb{E}[X] = \mu = \int_{x=-\infty}^{x=\infty} x \cdot f(x)dx.$$

A várható érték legfontosabb tulajdonságai:

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y],$$



2.2. ábra. Példa: folytonos valószínűségi változó sűrűségfüggvénye (bal oldalon) és eloszlásfüggvénye (jobb oldalon).

ahol X és Y két valószínűségi változó. Továbbá:

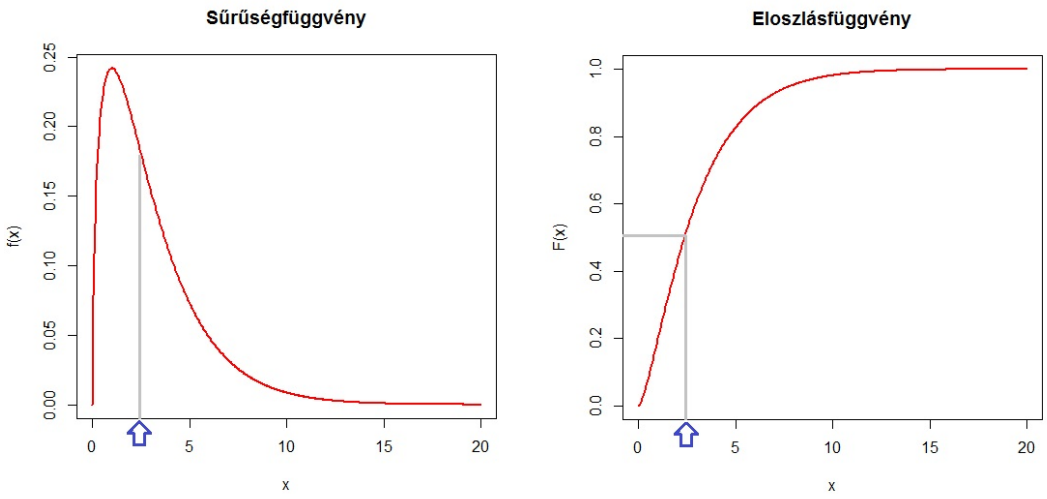
$$\mathbb{E}[X] = \mathbb{E}[\mathbb{E}[X|Y]],$$

ahol az $\mathbb{E}[\mathbb{E}[X|Y]]$ jelölés a következőképpen értendő. Ha egyszerre két valószínűségi változóval, X-szel és Y-nal van dolgunk, feltehetjük a kérdést, hogy mi X eloszlása, azaz milyen értéket mekkora valószínűséggel vesz fel X, ha *tudjuk*, hogy a Y valószínűségi változó egy adott y értéket vett fel. A kérdésre adott válasz az X valószínűségi változó adott $Y = y$ melletti feltételes eloszlása. $\mathbb{P}(X = x|Y = y)$-nal annak az eseménynek a valószínűségét jelöljük, hogy az X valószínűségi változó a x értéket veszi fel Y valószínűségi változó adott y értéke esetén. $\mathbb{P}(X = x|Y = y)$-t az $X = x$ esemény $Y = y$ melletti *feltételes valószínűségének* nevezzük.

Ha Y semmilyen hatással nincs X-re, akkor X feltételes eloszlása tetszőleges $Y = y$ mellett ugyanaz, mint X eloszlása Y ismeretlen értéke mellett. Adott $Y = y$ mellett kiszámolhatjuk X feltételes várható értékét, $\mathbb{E}[X|Y = y]$-t. Az $\mathbb{E}[\mathbb{E}[X|Y]]$ jelölés úgy értendő, hogy Y minden lehetséges y értéke mellett kiszámoljuk $\mathbb{E}[X|Y = y]$-t, majd ezen $\mathbb{E}[X|Y = y]$ értékek $\mathbb{P}(Y = y)$-oknal súlyozott összegét vesszük a várható érték definíciója szerint.

Az X valószínűségi változó *varianciája* vagy más szóval *szórásának négyzete*:

$$D^2[X] = \sigma^2[X] = \mathbb{E}[(X - \mu)^2]$$



2.3. ábra. Egy három szabadságfokú χ^2 eloszlású valószínűségi változó sűrűségfüggvénye (bal oldalon) és eloszlásfüggvénye (jobb oldalon) valamint mediánja.

és *n-edik centrális momentuma*:

$$D^n[X] = \mathbb{E}[(X - \mu)^n].$$

Két valószínűségi változó közti kovariancia, $Cov(X, Y)$, és korreláció, $Corr(X, Y)$, azt méri, hogy van-e összefüggés a két valószínűségi változó között:

$$Cov(X, Y) = \mathbb{E}[(X - \mu)(Y - \nu)]$$

illetve

$$Corr(X, Y) = \frac{Cov(X, Y)}{\sigma_X \sigma_Y}.$$

A korreláció $+1$ és -1 közti értékeket vehet fel. Ha a korreláció pozitív, a két változó együtt mozog, azaz amikor az egyik – saját értékkészletéhez viszonyítva – nagy értéket vesz fel, akkor tipikusan a másik értéke is – saját értékkészletéhez viszonyítva – nagy lesz. Negatív korreláció esetén a két változó egymással ellentétesen mozog. Ha a korreláció nulla, akkor nincs összefüggés a két változó között.

A sűrűségfüggvény (vagy diszkrét eloszlás) maximumhelyét az eloszlás móduszának hívjuk. Az F eloszlásfüggvény *p-kvartilisét* az a K szám adja, amelyre $F(K) < p$ és $F(K + 0) \geq p$. Az $1/2$-hez tartozó kvartilist *mediánnak* nevezzük (lásd 2.3. ábra).

2.2.1. Valószínűségi változók feltételes függetlensége

Az X és Y valószínűségi változó független, ha bármely lehetséges $x_i \in \mathbb{X}$ illetve $y_j \in \mathbb{Y}$ értékre igaz, hogy

$$\mathbb{P}(X = x_i | Y = y_j) = \mathbb{P}(X = x_i).$$

A Bayes-osztályozók megértéséhez szükségünk lesz a feltételes függetlenség fogalmára:

2.2.1. Definíció *Legyen X, Y és Z három valószínűségi változó. Az X feltételesen független Y-tól adott Z esetén, ha*

$$\mathbb{P}(X = x_i | Y = y_j, Z = z_k) = \mathbb{P}(X = x_i | Z = z_k)$$

minden lehetséges $x_i \in \mathbb{X}, y_j \in \mathbb{Y}, z_k \in \mathbb{Z}$ hármasra.

A feltételes függetlenség fogalmát két példán keresztül szemléltetjük:

1. Egy gyermek olvasási képessége adott életkor esetén feltételesen független a testmagasságától: ha például a 8 éves tanulókat tekintjük, az, hogy egy adott diák mennyire jól olvas, nem függ attól, hogy milyen magas. Ugyanakkor az olvasási képesség nyilván összefügg a testmagassággal, hiszen az idősebb gyerekek magasabbak és jobban is olvasnak.

2. Ha például az `eső`, `vihar`, `villámlás` diszkrét valószínűségi változókat tekintjük, akkor a `vihar` feltételesen független az `esőtől`, ha a `villámlást` ismerjük. A `villámlás` ugyanis `vihart` okoz (a `villámlás` hiánya pedig azt jelenti, hogy nincs `vihar`), ezért az `eső` ténye semmilyen további információval nem szolgál a `viharra` vonatkozóan. Természetesen van összefüggés a `vihar` és az `eső` között, de nincs köztük feltételes összefüggés, ha a `villámlás` értékét ismerjük.

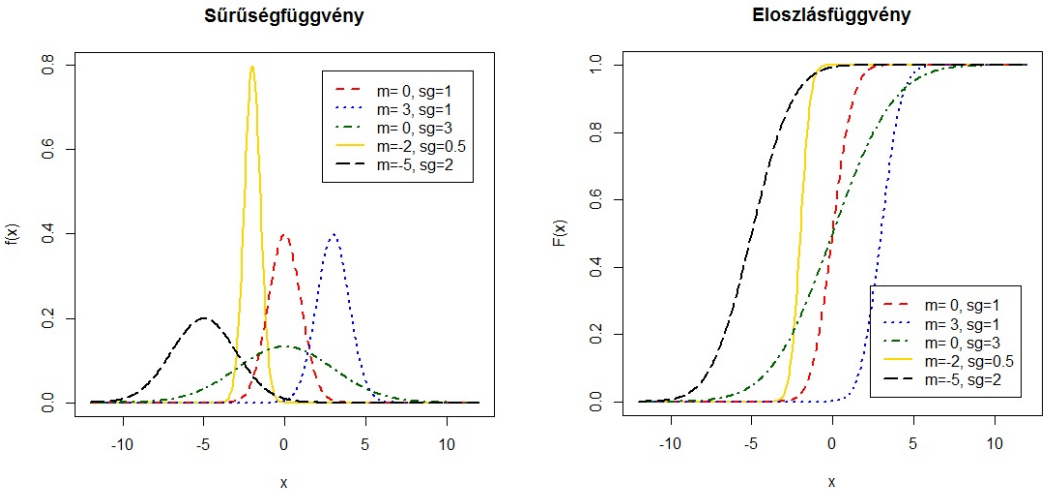
2.2.2. Nevezetes eloszlások

A következő nevezetes eloszlásokkal fogunk találkozni tanulmányaink során.

Normális eloszlás

A μ középértékű, σ szórású *normális eloszlást*, más néven *Gauss-eloszlást* egy harang alakú valószínűségi sűrűségfüggvény jellemzi, melynek képlete:

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$



2.4. ábra. Normális eloszlású valószínűségi változó sűrűségfüggvénye (bal oldalon) és eloszlásfüggvénye (jobb oldalon) μ (m) és σ (sg) különböző értékei mellett.

A középérték a haranggörbe csúcsának helye, a szórás pedig a haranggörbe szélességét jellemzi, lásd a 2.4. ábrát. A normális eloszlás eloszlásfüggvénye $\Phi(x)$, amelyet így definiálhatunk:

$$\Phi(x;\mu,\sigma) = \mathbb{P}(X < x;\mu,\sigma) = \int_{x_0=-\infty}^{x_0=x} f(x_0;\mu,\sigma)dx_0.$$

Standard normális eloszlásról beszélünk $\mu = 0$, $\sigma^2 = 1$ esetben.

Binomiális és Poisson eloszlás

Jelölje $p = P(A) > 0$ az A esemény bekövetkezésének valószínűségét. Hajtsunk végre n-szeres független kísérletsorozatot és legyen X valószínűségi változó értéke annyi, ahányszor A bekövetkezett a kísérletsorozatban. X-et ekkor n, p paraméterű binomiális eloszlású valószínűségi változónak nevezzük, jele $X \in B(n,p)$. X eloszlása:

$$\mathbb{P}(X = k) = \binom{n}{k} p^k (1-p)^{n-k},$$

várható értéke: $\mathbb{E}[X] = np$, szórása: $\sigma^2[X] = np(1-p)$.

A Poisson-eloszlás a binomiális eloszlás határesete:

$$\lim_{n\to\infty, p\to 0, np=\lambda} \binom{n}{k} p^k q^{n-k} = \frac{\lambda^k}{k!} e^{-\lambda}$$

A Moivre-Laplace tétel[2] szerint, az n-ed rendű p paraméterű binomiális eloszlás standardizáltja n minden határon túl való növelése esetén normális eloszlású:

$$\forall x \in \mathbb{R} : \lim_{n\to\infty} \sum_{\frac{k-np}{\sqrt{npq}}<x} \binom{n}{k} p^k q^{n-k} = \Phi(x),$$

ahol $\Phi(x)$ a standard normál eloszlás eloszlásfüggvénye (lásd 2.2.2. fejezetet).

Hipergeometrikus eloszlás

Tegyük fel, hogy van N különböző elemünk, amelyből R darab rossz. A hipergeometrikus eloszlás adja meg annak az esélyét, hogy r darab rossz elem lesz, ha az N elemből n darabot kiveszünk véletlenszerűen. Elemi kombinatorikus úton a valószínűség kiszámítható ($0 \le r \le \min\{n, R\}$):

$$\mathbb{P}(r, N, R, n) = \frac{\binom{R}{r}\binom{N-R}{n-r}}{\binom{N}{n}}$$

A fenti sűrűségfüggvénnyel rendelkező diszkrét valószínűségi eloszlást hívjuk *hipergeometrikus eloszlásnak*. Amennyiben $n \ll N$, akkor a hipergeometrikus eloszlást közelíthetjük az $n, R/N$ paraméterű binomiális eloszlással.

χ^2 eloszlás

Legyenek $\xi_1, \xi_2, \ldots, \xi_k$ egymástól független, standard normális eloszlású valószínűségi változók. Ekkor az $\sum_{i=1}^{k} \xi_i^2$ valószínűségi változó eloszlását k-ad rendű χ^2 *eloszlásnak* nevezzük, melyet χ_k^2-vel jelölünk. A k paramétert gyakran az eloszlás szabadsági fokának is nevezik. A χ_k^2 eloszlás sűrűségfüggvényét és eloszlásfüggvényét mutatja a 2.5. ábra a k paraméter különböző értékei mellett.

2.2.3. Egyenlőtlenségek

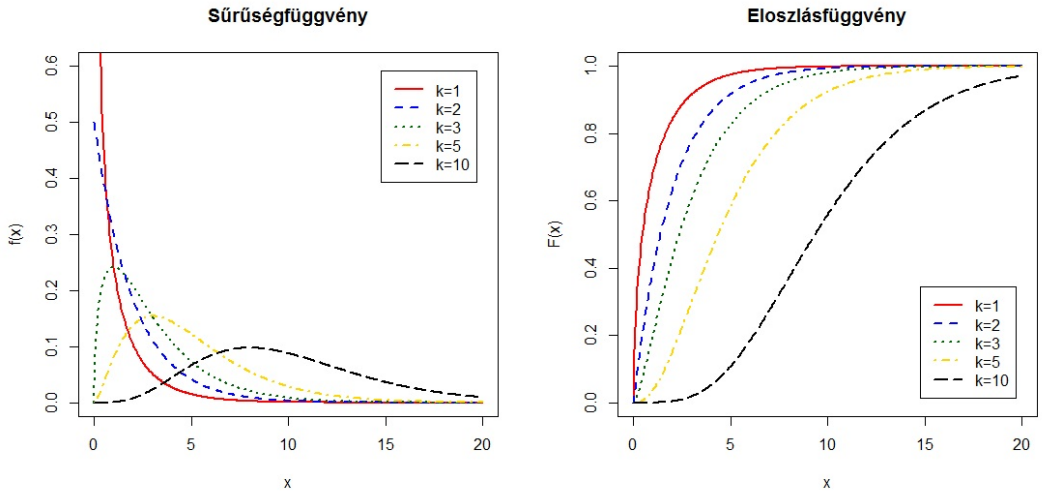
Legyen X egy $\mathbb{E}[X]$ várható értékű valószínűségi változó. A Markov egyenlőtlenség szerint $\mathbb{P}(|X| \ge a) \le \frac{\mathbb{E}[|X|]}{a}$, ahol $a > 0$.

A Hoeffding-korlát[3] a mintavételzéssel kapcsolatos állítások alapja:

2.2.2. Lemma *Legyen X_i, $1 \le i \le n$ μ várható értékű, független, azonos eloszlású valószínűségi változók és $a \le X_i \le b$ minden i-re. Ekkor tetszőleges $\lambda > 0$-ra fennáll a következő egyenlőtlenség:*

$$\mathbb{P}\Big[\Big|\frac{1}{n}\sum_{i=1} X_i - \mu\Big| \ge \lambda\ \Big] \le 2e^{-2\lambda^2 n/(b-a)^2}.$$

[2]http://mathworld.wolfram.com/deMoivre-LaplaceTheorem.html
[3]http://www.stat.cmu.edu/~larry/=stat705/Lecture2.pdf



2.5. ábra. χ^2_k eloszlás sűrűségfüggvénye (bal oldalon) és eloszlásfüggvénye (jobb oldalon) a k paraméter (szabadsági fok) különböző értékei mellett.

2.2.4. Entrópia

Legyen X egy diszkrét valószínűségi változó, amely értékeit egy $\mathbb{X}$ halmazból veheti fel. X entrópiáját – $H(X)$-et – az alábbi módon definiáljuk:

$$H(X) = -\sum_{x\in\mathbb{X}} \mathbb{P}(X = x) \log_2 \mathbb{P}(X = x).$$

Az entrópia valamiképpen a változó *bizonytalanság*át fejezi ki. Ha $\mathbb{X}$ elemszáma rögzített és az X változó csak egy értéket vehet fel (mert az egyik érték valószínűsége 1), akkor $H(X)$ értéke 0 (nincs bizonytalanság), ha pedig X eloszlása egyenletes eloszlást követ, akkor az entrópia a maximumát veszi fel, a $\log_2(|\mathbb{X}|)$ értéket.

Legyen X és Y két diszkrét értékű valószínűségi változó. Az X-nek az Y feltétellel vett feltételes entrópiája:

$$H(X|Y) = -\sum_{y\in\mathbb{Y}} \sum_{x\in\mathbb{X}} \mathbb{P}(X = x, Y = y) \log_2 \mathbb{P}(X = x|Y = y),$$

vagy egy kicsit átalakítva kapjuk, hogy

$$H(X|Y) = -\sum_{y\in\mathbb{Y}} \mathbb{P}(Y = y) \sum_{x\in\mathbb{X}} \mathbb{P}(X = x|Y = y) \log_2 \mathbb{P}(X = x|Y = y).$$

Be lehet bizonyítani, hogy $H(X|Y) = H(XY) - H(Y)$, ami informálisan úgy lehet megfogalmazni, hogy a feltételes entrópia megadja, hogy mennyi bizonytalanság marad X-ben, ha elvesszük az Y bizonytalanságát.

A feltételes entrópia számos tulajdonsága közül mi csak az alábbit fogjuk felhasználni:

$$0 \leq H(X|Y) \leq H(X).$$

2.3. Statisztika

A statisztikában általában $X_1, X_2, \ldots, X_n$ független, azonos eloszlású valószínűségi változók, *minták* vannak megadva. Az eloszlást nem ismerjük pontosan, de rendelkezésünkre állnak a megfigyelések.

A $\bar{X} = \frac{X_1+X_2+\cdots+X_n}{n}$ valószínűségi változót *empirikus középnek*, vagy *mintaátlagnak*, a $s_n^{*2} = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \bar{X})^2$ valószínűségi változót pedig *korrigált empirikus szórásnégyzetnek* nevezzük.

A χ^2 eloszlás definíciójából következik, hogy az $\frac{(n-1)s^{*2}}{\sigma^2}$ valószínűségi változó eloszlása χ_n^2, amennyiben a s^{*2} σ szórású, normális eloszlású valószínűségi változók korrigált empirikus szórásnégyzetét jelöli.

2.3.1. Hipotézisvizsgálat

A hipotézisvizsgálat feladata egy állítás helyességének vizsgálata. Ezt az állítást *nullhipotézis*nek nevezzük, jele H_0. A nullhipotézis általában egy valószínűségi változó valamely paraméterére vagy a változó viselkedésére vonatkozó állítás. Az állítás igazolásához vagy elvetéséhez minták állnak rendelkezésünkre. Összefoglalva tehát, adott egy állítás, egy paraméter (α) és minták sorozata. Feladatunk, hogy a minták alapján cáfoljuk vagy igazoljuk az állítást úgy, hogy bizonyíthatóan α-nál kisebb legyen annak valószínűsége, hogy az állítás igaz, holott mi cáfoljuk. A hipotézisvizsgálatnál a minták eredményeit felhasználva kiszámítunk egy *próbastatisztika* értéket. Ha a hipotézis fennáll, akkor a próbastatisztika értéke egy adott eloszlást követ: hogy konkrétan melyik eloszlást, az attól függ, hogy milyen próbát alkalmazunk (néhány próbát alább részletesebben is leírunk). Az alapötlet azonban minden esetben ugyanaz: a próbastatisztika értékét összevetjük az adott eloszlással, ennek függvényében döntjük el, hogy elfogadjuk-e a hipotézist vagy sem, aszerint, hogy az adott eloszlás mellett mennyire valószínű vagy valószínűtlen a próbastatisztika kapott értéke.

Ha a minták alapján a nullhipotézist elvetjük, holott az igaz, akkor *elsőfajú hibát* követünk el. Ellenkező esetben – amikor a nullhipotézis hamis, de mi elfogadjuk – *másodfajú hibáról* beszélünk.

Pusztán minták segítségével nem tudunk teljesen biztos választ adni arra, hogy a hipotézis fennáll-e. A gyakorlatban ezért egy paraméterrel (α) rögzítik az elsőfajú hiba elkövetésének megengedett valószínűségét. Az $1-\alpha$ értéket a *próba szignifikanciaszintjének* hívjuk. Az α értékét kicsinek, legtöbbször 0.05-nek, 0.01-nek vagy 0.001-nek szokás megválasztani.[4]

2.3.2. Az F-próba

2.3.1. Definíció *Legyenek X_0 és Y_0 két olyan valószínűségi változó, amelyek eloszlása rendre χ^2_n és χ^2_m. Ekkor a $Z = \frac{X_0/n}{Y_0/m}$ valószínűségi változó eloszlását $F_{n,m}$ eloszlásnak hívjuk.*

Az F-próba arra szolgál, hogy két független, normális eloszlású valószínűségi változó (X, Y) szórásának egyenlőségét eldöntsük.

$$H_0 : \sigma_X = \sigma_Y.$$

Tudjuk, hogy $\frac{(n_X-1)s^{*2}_X}{\sigma^2_X}$ és $\frac{(n_Y-1)s^{*2}_Y}{\sigma^2_Y}$ χ^2 eloszlásúak (n_X-1) illetve (n_Y-1) paraméterrel. Ha a nullhipotézis fennáll, azaz a két szórás egyenlő, akkor az

$$F = \frac{s^{*2}_X}{s^{*2}_Y}$$

próbastatisztika F-eloszlású (n_X-1, n_Y-1) paraméterrel. Azonban $\frac{1}{F}$ szintén F-eloszlású (n_Y-1, n_X-1) paraméterrel, ezért a gyakorlatban $F^* = \max\{F, 1/F\} \geq 1$ statisztikát szokás használni. Ha ennek értéke nagyobb az α-nál, azaz a próba a szignifikaciaszintjétől függő küszöbszámnál, akkor elvetjük azon nullhipotézisünket, hogy a két változó szórása egyenlő.

2.3.3. A χ^2-próba

A χ^2-próbák az alábbi tételt használják fel.

2.3.2. Tétel *Legyen $A_1, A_2, \ldots, A_r$ egy teljes eseményrendszer ($r \geq 3$), legyen $p_i = \mathbb{P}(A_i) > 0, i = 1, \ldots, r$. Ismételjük a kísérletet n-szer egymástól függetlenül. Jelölje X_i az A_i esemény bekövetkezésének számát. Belátható, hogy ekkor a*

$$\sum_{j=1}^{r} \frac{(X_j - np_j)^2}{np_j}$$

valószínűségi változó eloszlása $n \to \infty$ esetén χ^2_{r-1} eloszláshoz konvergál.

[4] Gondolkozzunk el azon, hogy mi történne, ha α-nak nagyon kis értéket választanánk!

A χ^2 eloszlás kvantiliseit függvény-táblázatokban megtalálhatjuk.

A χ^2-próba legfontosabb alkalmazási területei az (1.) illeszkedés-, (2.) függetlenség- és (3.) homogenitásvizsgálat. A jegyzetben a függetlenség-vizsgálatot asszociációs szabályok kiértékelésénél fogjuk használni, így a továbbiakban ezt részletezzük. A χ^2-próba iránt érdeklődőknek a [Fazekas, 2000] magyar nyelvű irodalmat ajánljuk.

2.3.4. Függetlenségvizsgálat

Legyen $A_1, A_2, \ldots, A_r$ és $B_1, B_2, \ldots, B_s$ két teljes eseményrendszer. Végezzünk n kísérletet. Nullhipotézisünk az, hogy az eseményrendszerek függetlenek.

$$H_0 : \mathbb{P}(A_i, B_j) = \mathbb{P}(A_i)\mathbb{P}(B_j), \qquad i = 1, \ldots, r \quad j = 1, \ldots, s,$$

ahol $\mathbb{P}(A_i, B_j)$ annak a valószínűségét jelöli, hogy az A_i és B_j események egyaránt bekövetkeznek. Ha az események valószínűségei, $p_1 = \mathbb{P}(A_1), \ldots, p_r = \mathbb{P}(A_r)$ és $q_1 = \mathbb{P}(B_1), \ldots, q_s = \mathbb{P}(B_s)$ adottak, akkor tiszta illeszkedésvizsgálati feladatról beszélünk. Jelölje k_{ij} azt, ahányszor A_i és B_j események együttes bekövetkezéseit figyeltük meg, n pedig az összes megfigyelés számát. Ekkor ki kell számítanunk a

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{s} \frac{(k_{ij} - np_iq_j)^2}{np_iq_j}$$

próbastatisztika értéket. Jobban megvizsgálva χ^2-et láthatjuk, hogy az egy

$$\sum \frac{(\text{megfigyelt érték - várt érték})^2}{\text{várt érték}}$$

jellegű kifejezés. Amennyiben χ^2 kicsi, akkor a megfigyelt értékek közel vannak azokhoz, amit H_0 fennállása esetén vártunk, tehát a nullhipotézist elfogadjuk.

Hogy pontosan mit jelent az, hogy „kicsi", azt a 2.3.2.-as tétel alapján χ^2_{rs-1} és az α paraméter határozza meg. A 2.1 táblázat mutatja a küszöbértéket néhány esetben: ha például $\alpha = 0.05$ és $rs - 1 = 3$, a küszöbérték 7.82. Amennyiben a küszöbérték nagyobb a fent kiszámított χ^2 értéknél, akkor a nullhipotézist elfogadjuk, ellenkező esetben elvetjük.

A gyakorlatban sokkal többször fordul elő az az eset, amikor az események valószínűségeit nem ismerjük. Ekkor a valószínűségeket az események relatív gyakoriságával becsüljük meg. Jelöljük az A_i esemény gyakoriságát $k_{i.}$-vel, tehát $k_{i.} = \sum_{j=1}^{s} k_{ij}$ és hasonlóan B_j esemény gyakoriságát $k_{.j}$-vel. χ^2-próbák során az adatok szemléltetésének gyakran használt eszköze az ún. kontingenciatáblázat. Ez egy többdimenziós táblázat, amely celláiban a megfelelő esemény bekövetkezésének száma található. Egy ilyen 2-dimenziós kontingenciatáblázatot láthatunk a következő ábrán.

k	$\alpha = 0.05$	$\alpha = 0.01$	$\alpha = 0.001$
1	3.84	6.64	10.83
2	5.99	9.21	13.82
3	7.82	11.34	16.27
4	9.49	13.28	18.47
5	11.07	15.09	20.52
6	12.59	16.81	22.46
7	14.07	18.48	24.32
8	15.51	20.09	26.12
9	16.92	21.67	27.88
10	18.31	23.21	29.59

2.1. táblázat. A χ^2 póbastatisztika küszöbértékei néhány esetben

	B_1	B_2	$\dots$	B_s	$\sum$
A_1	k_{11}	k_{12}	$\dots$	k_{1s}	$k_{1.}$
A_2	k_{21}	k_{22}	$\dots$	k_{2s}	$k_{2.}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
A_r	k_{r1}	k_{r2}	$\dots$	k_{rs}	$k_{r.}$
$\sum$	$k_{.1}$	$k_{.2}$	$\dots$	$k_{.s}$	n

Az A_i és B_j együttes bekövetkezéseinek megfigyelt darabszáma k_{ij}, míg együttes bekövetkezésük várt darabszáma H_0 esetén $n \cdot \frac{k_{i.}}{n} \cdot \frac{k_{.j}}{n}$. Ezek alapján χ^2 értéke:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{s} \frac{(k_{ij} - \frac{k_{i.}k_{.j}}{n})^2}{\frac{k_{i.}k_{.j}}{n}}$$

Mivel a függetlenség fennállása esetén $r-1$ darab p_i-t és $s-1$ darab q_j valószínűséget kell megbecsülni, így a fenti H_0 fennállása esetén $\chi^2_{rs-1-(r+s-2)} = \chi^2_{(r-1)(s-1)}$ eloszlású.

A χ^2 eloszlás közelítése csak abban az esetben pontos, ha a k_{ij} értékek nagyok. Persze nincs pontos szabály arra nézve, hogy mennyire kell nagynak lennie. Azt szokták mondani, hogy a kontingencia táblázat elemeinek 90%-a nagyobb legyen ötnél.

Példa

A χ^2 próbával történő függetlenségvizsgálatot egy példán keresztül szemléltetjük. Tekintsünk egy biztosítótársaságot, amely kötelező gépjármű felelősség-

	Okozott-e balesetet?		Összesen
	igen	nem	
nő	12	273	285
férfi	25	352	377
Összesen	37	625	662

2.2. táblázat. Példa: egy biztosítótársaság nemenként és okozott balesetenként csoportosítja ügyfeleit. Azt szeretnénk eldönteni, vajon van-e összefüggés a gépjárművezető neme és aközött, hogy okoz-e balesetet.

biztosítással foglalkozik. Ügyfeleiket kétféle szempont szerint csoportosítják: nemük szerint és aszerint, hogy okoztak-e balesetet az elmúlt évben. Az egyes csoportokba tartozó ügyfelek számát mutatja a 2.2. táblázat. A χ^2 próbastatisztika értéke:

$$\chi^2 = \frac{\left(12 - \frac{37\times285}{662}\right)^2}{\frac{37\times285}{662}} + \frac{\left(273 - \frac{625\times285}{662}\right)^2}{\frac{625\times285}{662}} + \frac{\left(25 - \frac{37\times377}{662}\right)^2}{\frac{37\times377}{662}} + \frac{\left(352 - \frac{625\times377}{662}\right)^2}{\frac{625\times377}{662}}$$

$$\chi^2 \approx 1.8$$

Mivel a 2.2. táblázatnak két sora és két oszlopa van, ezért $r = s = 2$, tehát: $(r-1)(s-1) = 1$. Így a 2.1 táblázat $k = 1$ sorához tartozó küszöbértékeket tekintjük. A táblázatban szereplő mindhárom küszöbszintnél (3.84, 6.64 és 10.83) kisebb az általunk számított érték, ezért α mindhárom értéke mellett arra a következtetésre jutunk, hogy nincs összefüggés az autóvezetők neme és aközött, hogy okoztak-e balesetet.

2.4. Gráfelmélet

Irányított gráf egy $G = (V, E)$ pár, ahol V *csúcsok* (vagy *pontok*) véges halmaza, E pedig egy bináris reláció V-n. E elemeit *éleknek* nevezzük. Ha $(u, v) \in E$, akkor az u, v csúcsok egymás *szomszédai. Irányítatlan gráfról* beszélünk, ha az E reláció szimmetrikus. A *címkézett* (vagy *súlyozott*) gráfnál a csúcsokhoz, *címkézett élű* (vagy *élsúlyozott*) gráfnál pedig az élekhez rendelünk címkéket. A címkézett (élű) gráfot *súlyozott* gráfnak hívjuk, ha a címkék számokkal kifejezhető súlyokat jelentenek. A gráf méretén ($|G|$) a csúcsok számát értjük. Egy csúcs *fokán* a csúcsot tartalmazó éleket értjük. Irányított gráfoknál megkülönböztetünk *kifokot* és *befokot.* A G irányítatlan gráf k*-reguláris*, ha minden csúcs foka pontosan k.

A $G' = (V', E')$ gráf a $G = (V, E)$ *részgráfja*, ha $V' \subseteq V$ és $E' \subseteq E$. A $G = (V, E)$ gráf $V' \subseteq V$ *által feszített részgráfja* (induced subgraph) az a $G' = (V', E')$ gráf, ahol $E' = \{(u, v) \in E : u, v \in V'\}$. A $G_1 = (V_1, E_1)$ *izomorf* a $G_2 = (V_2, E_2)$ gráffal, jelölésben $G_1 \cong G_2$, ha létezik $\phi : V_1 \to V_2$ bijekció, amelyre $(u, v) \in E_1$ esetén $(\phi(u), \phi(v)) \in E_2$ is fennáll. Címkézett gráfoknál emellett megköveteljük, hogy az u csúcs címkéje megegyezzék a $\phi(u)$ címkéjével minden $u \in V_1$-re, címkézett élű gráfnál pedig az (u, v) címkéje egyezzen meg a $(\phi(u), \phi(v))$ él címkéjével. A G gráfot önmagába leképező izomprfizmus esetén *automorfizmusról* beszélünk.

A gráfok ábrázolásának elterjedt módja a *szomszédossági mátrix* (adjacency matrix) és a *szomszédosság lista*. Az $|G| \times |G|$ méretű A szomszédossági mátrix a_{ij} eleme 1 (élcímkézett esetben az él címkéje), ha a G gráf i-edik csúcsából indul él a j-edik csúcsba, különben 0. Természetesen a szomszédossági mátrixot a gráfon kívül az határozza meg, hogy melyik csúcsot hívjuk az elsőnek, másodiknak, ... A szomszédossági mátrixot tehát a gráf és az $f : V \to \{1, \ldots, |V|\}$ bijekció adja meg. Hurokél nélküli[5], címkézett gráfban a szomszédossági mátrix a_{ii} eleme az i csúcs címkéjét tárolja. A szomszédossági lista $|G|$ darab lista, ahol az i-edik lista tárolja az i-edik csúcs szomszédait.

Az u csúcsot az u' csúccsal összekötő k-hosszú *út*on csúcsoknak egy olyan (véges) $\langle v_0, v_1, \ldots, v_k \rangle$ sorozatát értjük, amelyre $u = v_0$, $u' = v_k$, és $(v_{i-1}, v_i) \in E$ $(i = 1, 2, \ldots, k)$. Egy út *egyszerű*, ha a benne szereplő csúcsok páronként különbözők. A $\langle v_0, v_1, \ldots, v_k \rangle$ út *kör*, ha $v_0 = v_k$, és az út legalább egy élt tartalmaz. Egy gráfot *összefüggőnek* hívunk, ha bármely két csúcsa összeköthető úttal. A körmenetes, irányítás nélküli gráfot *erdőnek* hívjuk. Ha az erdő összefüggő, akkor pedig *fának*. Az olyan fát, amely tartalmazza egy G gráf minden csúcsát, a G *feszítőfájának* hívjuk.

A *gyökeres fában* az egyik csúcsnak kitüntetett szerepe van. Ezt a csúcsot *gyökérnek* nevezzük. A gyökérből egy tetszőleges x csúcsba vezető (egyértelműen meghatározott) út által tartalmazott bármely y csúcsot az x *ősének* nevezünk. Azt is mondjuk ekkor, hogy x az y *leszármazottja*. Ha $x \neq y$, akkor *valódi ősről* és *valódi leszármazottról* beszélünk. Ha az úton x egy élen keresztül érhető el y-ból, akkor x az y *gyereke* és y az x *szülője*. Ha két csúcsnak ugyanaz a szülője, akkor *testvéreknek* mondjuk őket.

A $G = (V, E)$ gráf $S, V \backslash S$ *vágásán* a V halmaz kétrészes partícióját értjük. Az $(u, v) \in E$ él *keresztezi* az $S, V \backslash S$ vágást, ha annak egyik végpontja S-ben a másik $V \backslash S$-ben van. Egy vágás *súlya* – súlyozott gráfok esetében – megegyezik a vágást keresztező élek összsúlyával.

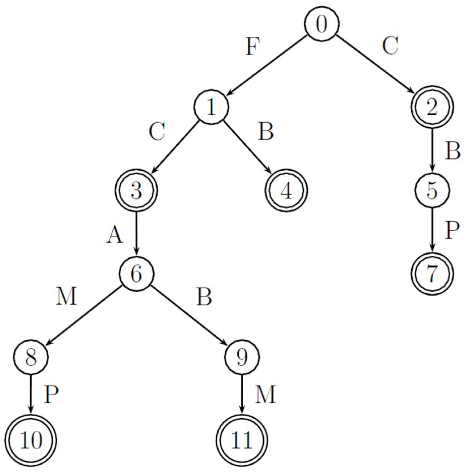
[5]Hurokélnek nevezzük egy olyan e élet, amelynek mindkét végpontja egyazon v csúcs.

2.5. Adatstruktúrák

Az adatbányászatban leginkább kedvelt adatstruktúrák, a lista (vektor) és tömb mellett, a *szófa* (trie), vagy más néven prefix-fa (prefix-tree), valamint a piros-fekete fa, illetve a hash-tábla.

2.5.1. Szófák

A szófát eredetileg szótár szavainak tárolásánál alkalmazták, annak érdekében, hogy gyorsan el lehessen dönteni, hogy egy adott szó szerepel-e a szótárban [Briandais, 1959], [Fredkin, 1960]. A szavak egy alfabeta (ábécé) felett értelmezett sorozatok, így általánosan azt mondhatjuk, hogy egy szófa egy adott véges elemhalmaz feletti sorozatok tárolására és gyors visszakeresésére alkalmas adatstruktúra. A szófa angol neve (trie, amit úgy ejtünk, mint a try szót) a visszakeresés angol fordításából származik (re`trie`val). A továbbiakban az alaphalmazt $\mathbb{I}$-vel jelöljük, az alaphalmaz felett értelmezett, adott sorozatok halmazát szótárnak hívjuk. A 2.6 ábrán egy szófát láthatunk, mely az C, FC, FB, CBP, $FCAMP$, $FCABM$ sorozatokat tárolja.



2.6. ábra. Példa szófára

A szófa egy (lefelé) irányított gyökeres címkézett fa. Egy d-edik szintű pontból csak $d+1$-edik szintű pontba mutathat él. Néha a hatékonyság kedvéért minden pontból a pont szülőjére is mutat él. A gyökeret 0. szintűnek tekintjük. A címkék az $\mathbb{I}$-nek egy-egy elemei. Minden pont egy elemsorozatot reprezentál, amely a gyökérből ebbe a pontba vezető éleken található elemekből áll. Akkor tartalmazza a szófa az S sorozatot, ha van olyan pont, amely az S-t reprezentálja.

Ha egy sorozatot tartalmaz egy szófa, akkor annak tetszőleges prefixét is tartalmazza. A prefix azonban nem biztos, hogy eleme a szótárnak. Ezt a problémát kétféleképpen lehet kiküszöbölni. Egyrészről megkülönböztetünk *elfogadó* és *nem elfogadó* pontokat. Egy sorozatot akkor tartalmazza a szófa, ha van olyan elfogadó állapot, amely a sorozatot reprezentálja. Másrészről bevezethetünk egy speciális elemet, amit minden sorozat végére illesztünk, továbbá sorozatot csak levél reprezentálhat.

A szófának két implementációját különböztetjük meg attól függően, hogy milyen technikát alkalmazunk az élek tárolására. Az ún. *táblázatos implementációban* (tabular implementation) [Fredkin, 1960] minden ponthoz egy $|\mathbb{I}|$ hosszúságú, mutatókat tartalmazó vektort veszünk fel. Az i-edik mutató mutat a szótár i-edik eleméhez tartozó él végpontjára. Ha a pontnak nincs ilyen címkéjű éle, akkor a mutató értéke NULL. A vektor hossza az $\mathbb{I}$ elemszámával egyezik meg.

A *láncolt listás implementációban* [Briandais, 1959] az éleket egy láncolt listában tároljuk. A lista elemei (élcímke, gyermekmutató) párok. A láncolt lista következő elemére mutató mutatókat megspórolhatjuk, ha egy vektort alkalmazunk, aminek hossza megegyezik a pont éleinek számával, és elemei szintén (élcímke, gyerekmutató) párok. Ez azért is jó megoldás, mert egy lépéssel tudunk tetszőleges indexű elemre lépni (a címke, mutató pár memóriaszükségletének ismeretében), és nem kell a mutatókon keresztül egyesével lépegetnünk.

Szófák esetében a legfontosabb elemi művelet annak eldöntése, hogy egy adott pontnak van-e adott címkéjű éle, és ha van, akkor ez hova mutat. Táblázatos implementációnál ezt a feladatot egy lépésben megoldhatjuk a megfelelő indexű elem megvizsgálásával. Láncolt listás, illetve változó hosszúságú vektor esetén a megoldás lassabb művelet. A vektor minden párját ellenőriznünk kell, hogy a pár címkéje megegyezik-e az adott címkével. A hatékonyságot növelhetjük, ha a párokat címkék szerint rendezve tároljuk, és bináris keresést végzünk.

Érdemes összehasonlítanunk a két vektoros implementációban a pontok memóriaigényét. Amennyiben a mutatók, és a címkék is 4 bájtot foglalnak, akkor a táblázatos implementációban egy pont memóriaigénye (a vektor fejléc memóriaigényétől eltekintve) $|\mathbb{I}| \cdot 4$ bájt, a listás implementációé $n \cdot 2 \cdot 4$ bájt, ahol n az adott pontból induló élek száma, amire igaz, hogy $0 \leq n \leq |\mathbb{I}|$. Ha a szófa pontjai olyanok, hogy kevés élük van, akkor a listás implementációnak lesz kevesebb memóriaigénye, ha azonban egy-egy pont sok éllel rendelkezik, a táblázatos implementáció a jobb megoldás. A két technikát ötvözhetjük akár egy adott szófán belül is [Severance, 1974], [Yao, 1975]: ha a pont éleinek száma meghalad egy korlátot (általában $\mathbb{I}/2$-t), akkor táblázatos implementációt használunk, ellenkező esetben maradunk a listás megoldásnál.

Megemlítünk két szófa leszármazottat. Ezek a *nyesett szófák* (pruned trie)

és a *Patrícia fák*. Mindkét fa abban különbözik az eredeti szófától, hogy kiiktatják az olyan utakat a fából, amelyekben nincsen elágazás. A nyesett fánál ezt kizárólag levélhez vezető utakkal teszik, Patrícia fáknál ez a korlátozás nem áll fenn.

Patrícia-fák és nyesett szófák

Egy irányított utat láncnak hívunk, ha minden pontjának csak egy gyereke van. A Patrícia-fa a szófából származtatható úgy, hogy a szófa nem bővíthető láncait egy-egy éllé vonjuk össze. Az új él a lánc utolsó pontjába mutat, címkéje a lánc éleinek címkéiből álló sorozat. Ha a láncösszevonást csak a levélben végződő láncokra hajtjuk végre, akkor nyesett szófát kapunk, amelyet Patrícia* fának is neveznek.

Ha a szófa sok láncot tartalmaz, akkor a Patrícia-fa sokkal hatékonyabb, mint az eredeti szófa. Ellenkező esetben viszont több memóriát használ, mivel a címkéket vektorokban tároljuk, ami egyetlen elem tárolása esetén nem célravezető a nagy többletköltség miatt.

2.5.2. Piros-fekete fák

A piros-fekete (RB-tree vagy symmetric binary B-trees) fák a kiegyensúlyozott bináris fák (balanced binary tree) egy típusa. Minden csúcsnak színe van, hagyományosan piros vagy fekete. Speciális forgatásokat használó beszúrás művelet biztosítja, hogy bármely a gyökérből levélbe vezető út hossza ne legyen nagyobb, mint a legrövidebb ilyen út hosszának kétszerese. Egy piros-fekete fa a következő tulajdonságokkal rendelkezik:

1. Minden csúcsnak a színe piros vagy fekete.
2. Minden levél színe fekete.
3. Minden piros csúcsnak mindkét fia fekete.
4. Bármely két, azonos csúcsból induló, levélig vezető úton ugyanannyi fekete csúcs van.

A fentiekből következik, hogy bármely n belső csúccsal rendelkező piros-fekete fa magassága legfeljebb $2\lg(n+1)$. A bizonyítás és a fa építésének menete megtalálható az irodalomban [Rónyai és tsa.,1998].

2.5.3. Hash-tábla

A hash-tábla (ritkán használt elnevezése: hasító tábla) elemek gyors elhelyezésére és visszakeresésére használt adatstruktúra. A táblázatnak cellái vannak, amibe elemeket helyezhetünk. Minden cellának van egy címe (vagy indexe). A hash-táblás tárolásban központi szerepet tölt be az elemeken értelmezett ún. *hash-függvény*, ami megadja az elem *hash-értékét*. Egy elemet arra a címre helyezünk be a hash-táblában, amelyet a hash-értéke megad. Előfordulhat, hogy különböző elemekhez a hash-függvény ugyanazokat a hash-értéket rendeli. Ezt *ütközésnek* hívjuk.

2.6. Számítógép-architektúrák

Sok kutató alkalmazza a külső táras modellt algoritmusok hatékonyságának vizsgálatakor. Mára az óriási memóriaméreteknek köszönhetően a legtöbb adatbázis elfér a memóriában, valamilyen szűrt formában. Ilyen esetekben az elemzéshez használt modell leegyszerűsödik a közvetlen hozzáférésű modellre (RAM-modellre), amely Neumann-modell néven is ismert, mivel a magyar születésű Neumann János javasolta először [Neumann, 1945].

De a modern processzorok, amelyeken a programokat futtatják, sokkal kifinomultabbak a RAM-modellnél [Hennessy és Patterson, 2011]. A modell túlzott egyszerűsítése ahhoz vezet, hogy az elemzések nem elég pontosak: a modell alapján várt eredmények nem mindig esnek egybe a valósággal. A modern processzorok miatt ezért egy új modellt használhatunk az elemzéshez, amely új elvárásokat támaszt az algoritmusokkal szemben. Ezekről egy kiváló áttekintés olvasható a [Meyer és tsa., 2003] tanulmányban. A modern processzorok legfontosabb sajátossága a többszintű memória és a csővezetékes (pipeline-) feldolgozás.

2.6.1. Többszintű memória, adatlokalitás

A memória nem egyelten nagy blokk, sokkal inkább különböző méretű, késleltetésű memóriákból álló hierarchikus rendszer. Minél nagyobb a memória mérete, annál több idő kell a hozzáféréshez. A hierarchia elemei méret szerint növekvő sorrendben a következők: regiszterek, pár kilobájtos elsőszintű gyorsítótár, pár megabájtos másodszintű gyorsítótár, esetleges harmadszintű gyorsítótár, rendszermemória és merevlemez. Az adatot a rendszermemóriából a másodszintű gyorsítótárba, a másodszintűből az elsőszintű gyorsítótárba blokkonként másolhatjuk. A blokkméret egy Pentium 4-es processzor esetén 128 bájt.

A blokkonkénti feldolgozás más megvilágításba helyezi az algoritmusok vizsgálatát: egyetlen bit eléréséhez és beolvasásához egy lassú memóriából ugyanannyi idő kell, mint a bitet tartalmazó teljes blokk eléréséhez és beolvasásához. Másik adat elérése ugyanebben a blokkban viszont nem igényli már a hozzáférést a lassú memóriához. Így rendkívül fontos követelménnyé válik az adatlokalitás, azaz hogy az adatok, amelyeket időben egymáshoz közel dolgozunk fel, a memóriában is közel legyenek egymáshoz.

Az adatot feldolgozásakor be kell hozni a regiszterekbe. Előfordulhat, hogy már eleve ott van, mert az előző műveletekhez szükség volt rá. A korlátozott számú regiszterek miatt azonban sokkal valószínűbb, hogy az egyik gyorsítótárban vagy a rendszermemóriában helyezkedik el. Sőt, az is lehet, hogy a merevlemezen található, ha az algoritmus memóriaigénye annyira nagy, hogy lapozásra van szükség. Ha a másodszintű gyorsítótárban vagy a rendszermemóriában helyezkedik el a kívánt adat, akkor az adathozzáférés ún. cache miss-t okoz. Amíg ez az adat bekerül a regiszterekbe, a processzor végrehajthat más műveleteket (ezer alapművelet, például összeadás elvégzésére képes ez idő alatt), ennek ellenére a teljesítménye messze elmaradhat ilyenkor a maximálistól. Tehát az adatstruktúra és algoritmus tervezésekor törekednünk kell a minél jobb adatlokalitásra.

2.6.2. Csővezetékes feldolgozás, elágazás-előrejelzés

A programozók által használt műveleteket a fordító mikroutasítások sorozatára bontja. A műveleteket nem külön-külön, egymás után hajtjuk végre, az n-dik művelet végrehajtása nem akkor kezdődik, amikor az $(n-1)$-dik művelet végeredménye már ki van számolva. Még mielőtt az $(n-1)$-dik művelet végrehajtása befejeződne, már megkezdődik a következő néhány művelet végrehatása. Úgy képzelhetjük el, mintha a műveletek végrehajtásuk során egy csővezetéken haladnának szorosan egymást követve. Sajnos azonban az adatfüggőség és a feltételes ugrások sokat rontanak a feldolgozás hatékonyságán. Adatfüggőségről akkor beszélünk, ha egy utasítás egy előző utasítás eredményétől függ. Ilyen lehet például egy *if*-szerű elágazás: a feltétel igaz vagy hamis voltától függően vagy az egyik vagy a másik ágon kell folytatódnia a végrehajtásnak. Azt, hogy a feltétel igaz-e vagy hamis, viszont csak a feltétel kiértékelése *után* tudjuk meg. A processzor azonban még a feltétel kiértékelése előtt megpróbálja megjósolni a feltétel kimenetét (elágazás-előrejelzés), és még a feltétel kiértékelése előtt belekezd a jóslat szerinti ágon lévő utasítások végrehajtásába. Ha a jóslás hamisnak bizonyul, akkor a csővezetéket ki kell üríteni, és be kell tölteni a helyes utasításokat. Ezt a problémát gyakran kiküszöbölhetjük különböző technikák alkalmazásával, mint például kódátrendezéssel, amelyet automatikusan elvégez a fordító. Számításigényes algoritmus tervezésekor azonban nekünk

kell ügyelnünk az adatfüggetlenségre és az elágazás-előrejelzésre.

A csővezetékes feldolgozás lehetővé teszi, hogy egy órajel alatt több műveletet is elvégezzünk. A fent említett problémák miatt azonban a processzor átlagos teljesítménye messze nem éri el az optimumot. A felesleges feltételek ronthatják a hatékonyságot. Az elágazás-előrejelzés intelligens olyan szempontból, hogy ha egy feltétel kimenete sohasem változik, akkor a processzor ezt figyelembe veszi és a későbbiekben ennek megfelelően jósol. Így a mindig igaz (vagy hamis) feltételek nem befolyásolják a hatékonyságot.

3. fejezet

Előfeldolgozás, távolságfüggvények

Különböző adatbányászati témák — úgy mint: osztályozás, klaszterezés, anomáliakeresés — egyik kulcsfogalma az objektumok közti hasonlóság, amelyet legtöbbször távolsági függvények segítségével fogunk jellemezni. Ahhoz hogy objektumok közti távolságot vagy hasonlóságokat definiálhassunk, először az attribútumok típusait kell alaposabban szemügyre vennünk.

A fejezetben foglalkozunk továbbá a legfontosabb előfeldolgozási műveletekkel. A távolsági függvényekhez hasonlóan az előfeldolgozási műveletek is univerzálisak abban az értelemben, hogy különböző adatbányászati feladatok megoldásához lesz rájuk szükségünk, nem kötődnek szorosan egyik vagy másik adatbányászati területhez.

3.1. Attribútum típusok

Adatainkat a legegyszerűbb esetben egy nagy táblázatként képzelhetjük el, lásd az 1.2. ábrát. A táblázat egy-egy sora felel meg egy-egy ügyfélnek, páciensnek, terméknek, stb. A táblázat sorait objektumoknak, példányoknak vagy rekordoknak nevezzük. A táblázat oszlopai az objektumok egyes tulajdonságainak felelnek meg, ezért az oszlopokat attribútumoknak nevezzük.

Hacsak ennek ellenkezőjét külön nem jelezzük, általában adattábla típusú adatokat tételezünk fel. Nem jelezzük külön, ha az adattábla típusú adatok tanulmányozása során kapott megállapításaink triviálisan általánosíthatók más esetekre: az attribútumtípusok például ugyanúgy értelmezhetőek akkor is, ha a különböző objektumok különböző attribútumokkal rendelkeznek, és ezért az adataink nem illeszkednek egy nagy táblázatba.

Jelöljük az A attribútum két értékét a-val és a'-vel.

1. A *kategória típusú* (*nominal*) attribútumnál az értékek között csak azonosságot tudunk vizsgálni. Tehát csak azt tudjuk, hogy $a = a'$ vagy

$a \neq a'$. A kategória típusú attribútum egy speciális esete a *bináris attribútum*, ahol az attribútum csak két értéket vehet fel. A kategória típusú attribútumokat az irodalom néha *felsorolás* (enumerated) vagy *diszkrét* típusnak is hívja. Másodlagos jelentésük miatt ebben a tanulmányban ezeket az elnevezéseket kerüljük.[1]

2. A *sorrend típusú* (*ordinal*) attribútumoknál az értékeket sorba tudjuk rendezni, azaz az attribútum értéken teljes rendezést tudunk megadni. Ha tehát $a \neq a'$, akkor még azt is tudjuk, hogy $a > a'$ és $a < a'$ közül melyik igaz.

3. Ha az eddigiek mellett értelmezett az attribútumértékek összeadása, azaz meg tudunk adni egy + függvényt, amivel az attribútumértékek, mint elemek csoportot alkotnak, akkor *intervallum típusú* (*interval scale*) attribútumról beszélünk.

4. Ha egy intervallum típusú attribútumnál meg lehet adni zérus értéket és értelmezett két attribútumérték hányadosa, vagy pontosabban: az attribútumértékek, mint elemek gyűrűt alkotnak, akkor az attribútum *arány skálájú* (*ratio scale*). Az arány skálájú attribútumot gyakran fogjuk *valós* attribútumnak hívni, hiszen a gyakorlati esetek többségében az arány skálájú attribútumok megadásához valós számokat használunk. Azonban ne felejtsük el az arány skálájú attribútum eredeti definícióját, illetve azt, hogy az arány skálájú attribútumok nem feltétlenül valós számokat tartalmaznak.

Az intervallum típusú és arány skálájú attribútumokat együttesen szokás *numerikus* típusú attribútumoknak is nevezni.

Például egy ügyfeleket leíró adatbázisban vannak bináris (pl.: büntetett előéletű-e), kategorikus (pl.: vallás, családi állapot) és intervallum (pl.: dátum) típusú attribútumok is.

Fontos, hogy nem mindig triviális az, hogy egy attribútum milyen típusú. Például az időjárás jellemzésére használt `napsütéses`, `borús`, `esős` értékekre mondhatjuk, hogy ez kategória típusú. Ugyanakkor érezzük, hogy a `borús` valahol a `napsütéses` és az `esős` között helyezkedik el, így inkább sorrend típusú az attribútum.

Az intervallum típusú attribútumok megadására is számokat használunk, amelyeknél értelme van a különbség számításának, de a hányados képzésnek nincs. Tulajdonképpen azt, hogy mikor beszélünk intervallum és mikor arány

[1] Például a felsorolás típus említésénél a legtöbb informatikusnak a C++, java, C#-beli felsorolás típusú változó jut eszébe, amelyek mindig egyértelmű megfeleltetésben állnak egy egész számmal.

skálájú típusról az dönti el, hogy egyértelmű-e a zérus pont definiálása. Gondoljuk meg, hogy például az évszámoknál hány fajta nullát ismerünk. Hasonló a helyzet a hőmérséklet esetében (Fahrenheit kontra Celsius).

Szinte minden adatbányász/statisztikai program megadja minden intervallum típusú attribútumnak a legfontosabb statisztikáit. Ezek a

- középértékre vonatkozó adatok: mintaátlag, medián, módusz,
- szóródásra vonatkozó adatok: empirikus szórásnégyzet, minimum, maximum, terjedelem (max és min érték közötti különbség)
- eloszlásra vonatkozó adatok: empirikus kvantilisek, ferdeség, lapultság.

A *ferdeség* egy eloszlás szimmetriáját számszerűsíti. Ha a ferdeség nulla, akkor az eloszlás szimmetrikus (például normális eloszlásoknál), ellenkező esetben a várható értéktől balra (negatív ferdeség esetében) vagy jobbra „nyúlik el". A ferdeségnek több mutatóját definiálták; ezek közül a legelterjedtebb az eloszlás harmadik standardizált momentuma:

$$\gamma_1 = \frac{D^3[X]}{(D^2[X])^{3/2}} = \frac{\mathbb{E}\Big[(X-\mu)^3\Big]}{\sigma^3},$$

ahol $D[X]$ az X attribútum szórását jelöli. Gyakran használják a $\beta_1 = \sqrt{\gamma_1}$-et is az eloszlás ferdeségének mérésére.

A *lapultság* egy eloszlás csúcsosságát adja meg. A lapultságnak is több elfogadott definíciója létezik. Legelterjedtebbek a

$$\beta_2 = \frac{D^4[X]}{(D^2[X])^2},$$

és a

$$\gamma_2 = \beta_2 - 3,$$

értékek. Az előbbit kurtosis proper-nek, az utóbbit kurtosis excess-nek nevezik. A normális eloszlás β_2 lapultsági értéke három, a normálisnál laposabbaké háromnál kisebb. A ferdeséget és a lapultságot annak eldöntésénél szokták használni, hogy egy adott minta származhat-e normális eloszlásból. Mint látni fogjuk, a ferdeség fogalmának kulcsszerepe lesz a csomósodás (presence of hubs) jelenségének vizsgálatakor is.

3.2. Távolsági függvények

Az adatbányászatban gyakran szükségünk van arra, hogy attribútumokkal leírt objektumok között hasonlóságot definiáljunk. Természetesen elvárjuk, hogy ha

minél több azonos érték szerepel az attribútumaik között, illetve minél kisebb az eltérés a (numerikus) attribútumaik között, annál hasonlóbbak legyenek az objektumok. A gyakorlatban hasonlósági mérték helyett gyakran *távolságmértékekkel* vagy más néven *különbözőségi mértékkel, távolsági függvénnyel* dolgozunk, amely a hasonlóság inverze: minél hasonlóbb két objektum, annál kevésbé különbözők. Elvárjuk, hogy az adatbázisbeli bármely két objektum, x és y, távolságát (különbözőségét), $d(x, y)$-t, ki lehessen fejezni egy nemnegatív valós számmal, melyet metrikának nevezünk és az alábbi tulajdonságokkal rendelkezik:

1. egy objektum önmagától ne különbözzön: $d(x, x) = 0$ tetszőleges x-re,
2. a távolság szimmetrikus legyen: $d(x, y) = d(y, x)$ bármely x-re és y-ra, és
3. teljesüljön a háromszög egyenlőtlenség: $d(x, y) \leq d(x, z)+d(y, z)$ bármely x-re, y-ra és z-re.

Ha a 3. tulajdonság nem teljesül, akkor szemi-metrikáról beszélünk, ha az erősebb $d(x, y) \leq \max\{d(x, z), d(y, z)\}$ tulajdonság áll fenn, akkor pedig ultrametrikáról (más néven nem-archimédeszi távolságról).

Mivel a távolság és hasonlóság rokon fogalmak, a távolsági függvények gyakran hasonlósági függvénnyé alakíthatók (és fordítva). A hasonlósági függvény általában 0 és 1 közötti értékeket vesznek fel, 1-t, ha a két objektum azonos, 1-hez közi értéket, ha a két objektum nagyon hasonló, 0-hoz közeli értéket, ha a két objektum nagyon különböző. Ugyanazon adatbányászati algoritmus nem ritkán – kis módosítással – távolsági függvény helyett hasonlósági függvénnyel is működhet (és fordítva), ezért az angol irodalomban gyakran előfordul, hogy a távolság és hasonlóság fogalmára együtt hivatkoznak *proximity* néven.

A következőkben sorra vesszük, hogyan definiáljuk a távolságot különböző típusú attribútumok esetében, és azt, hogy miként lehet egyes attribútumok fontosságát (súlyát) figyelembe venni.

3.2.1. Bináris attribútum

Egy bináris attribútum olyan kategória típusú attribútum, amely két értéket vehet fel (pl.: 0 és 1). Hogyan határozzuk meg x és y objektumok (példányok) hasonlóságát, ha azok m darab bináris attribútummal vannak leírva? Készítsük el a következő összefoglaló táblázatot.

		y		
		1	0	$\sum$
x	1	q	r	q+r
	0	s	t	s+t
	$\sum$	q+s	r+t	m

Például az 1-es sor 0-ás oszlopához tartozó érték azt jelenti, hogy r darab olyan attribútum van, amelyek az x objektumnál 1-et, y-nál 0-át vesznek fel.

Ez alapján definiálhatjuk az ún. invariáns és variáns távolságot. Az invariáns távolságot olyan eseményeknél használjuk, amikor a bináris attribútum két értéke ugyanolyan fontos (szimmetrikus attribútum), tehát mindegy, hogy melyiket kódoljuk 0-val, illetve 1-essel. Ilyen attribútum például egy ember neme. Azért kapta ez a távolság az invariáns jelzőt, mert nem változik az értéke, ha valaki máshogy kódolja az attribútumokat (tehát kódolásinvariáns). A legegyszerűbb invariáns távolság az eltérő attribútumok relatív száma:

$$d(x, y) = \frac{r + s}{m}.$$

Aszimmetrikus attribútum esetében a két lehetséges érték nem egyenrangú. Ilyen attribútum lehet például egy orvosi vizsgálat eredménye: nagyobb súlya van annak, hogy valaki fertőzött, mint annak, hogy nem az. A konvencióknak megfelelően 1-essel kódoljuk a lényeges (általában ritka) kimenetet. A legegyszerűbb variáns távolsági mérték a Jaccard-koefficiens [Levandowsky és Winter, 1970] komplementere:

$$d(x, y) = 1 - \frac{q}{m - t} = \frac{r + s}{m - t},$$

ahol nem tulajdonítunk jelentőséget a nem jelentős kimenetek egyezésének.

Amennyiben szimmetrikus és aszimmetrikus értékek is szerepelnek a bináris attribútumok között, akkor azokat vegyes attribútumként kell kezelni (lásd a 3.2.5-os részt).

3.2.2. Kategória típusú attribútum

Általános esetben a kategória típusú attribútum nem csak kettő, hanem véges sok különböző értéket vehet fel. Ilyen attribútum például az ember szeme színe, családi állapota, vallása stb. A legegyszerűbb távolság a nemegyezések relatív száma:

$$d(x, y) = \frac{u}{m},$$

ahol m a kategória típusú attribútumok száma, u pedig azt adja meg, hogy x és y objektumokat tekintve ezek közül mennyi nem egyezett. Természetesen a kategória típusú attribútumok sem feltétlenül szimmetrikusak, mert lehet, hogy az alapértelmezett értékek egyezése nem igazán fontos. A Jaccard-koefficiens komplementerét kategória típusú attribútumokra is felírhatjuk. Tekintsük példaként azt, hogy ügyfelek hasonlóságát szeretnénk számszerűsíteni egy kérdőív kérdéseire adott válaszaik alapján. Véletlenszeűnek tételezzük fel azt, hogy valaki egy kérdésre nem válaszol, a hiányzó válaszok tehát nem képezik alapját annak, hogy hasonlónak tekintsünk két ügyfelet. Ha adottak

x_1 =(életkor = 20–25, autó = Opel, végzettség = egyetem, nem = férfi, családi állapot = nőtlen)

és

x_2 =(életkor = 20–25, végzettség = egyetem, nem = nő, családi állapot = házas, hobbi = könyvek olvasása, vallás = buddhista)

ügyfelek, a Jaccard-koefficiens komplementere szerinti távolságuk számításához a megegyező attribútumaik számát és az összes megadott attribútum számát használjuk:

$$d(x_1, x_2) = 1 - \frac{2}{7} = 0.714.$$

3.2.3. Sorrend típusú attribútum

Sorrend típusú attribútum például az iskolai végzettség: 8 általános, befejezett középiskola, érettségi, főiskolai diploma, egyetemi diploma, doktori cím. Vannak arány skálájú attribútumok, amelyeket inkább sorrend típusú attribútumnak kezelünk. Például a Forma 1-es versenyeken sem az egyes körök futási ideje számít, hanem az, hogy ki lett az első, második, harmadik, stb.

A sorrend típusú attribútumokat általában egész számokkal helyettesítik – tipikusan 1 és M közötti egész számokkal. Ha több sorrend típusú attribútumunk van, amelyek a fontos állapotok számában eltérnek, akkor célszerű mindegyiket a [0,1] intervallumba képezni az $\frac{x-1}{M-1}$ művelettel. Így mindegyik egyenlő súllyal szerepel majd a végső távolsági mértékben. Ezután alkalmazhatjuk a következő szakaszban bemutatásra kerülő intervallum típusú távolságok valamelyikét.

3.2.4. Intervallum típusú attribútum

Az intervallum típusú attribútumokat általában valós számok írják le. Ilyen attribútumra példa egy ember súlya, magassága, vagy egy ország éves átlaghőmérséklete. Tekinthetünk úgy egy elemre, mint egy pontra az m-dimenziós vektortérben. Az elemek közötti különbözőséget a vektoraik különbségének normájával (hosszával) definiáljuk ($d(\vec{x}, \vec{y}) = ||\vec{x} - \vec{y}||$). Legtermészetesebb talán az Euklideszi-norma, de alkalmazhatjuk a Manhattan-normát is. Mindkét mérték a Minkowski-norma speciális esete.

Legyen $\vec{z} = \vec{x} - \vec{y}$ és jelöljük $\vec{z}$ koordinátáit rendre $z_1, z_2, ..., z_m$-mel. Ekkor így definiálhatjuk az Euklideszi-, Manhattan-, és Minkowski-normákat:

Euklideszi-norma: $L_2(\vec{z}) = \sqrt{|z_1|^2 + |z_2|^2 + \cdots + |z_m|^2}$

Manhattan-norma: $L_1(\vec{z}) = |z_1| + |z_2| + \cdots + |z_m|$

Minkowski-norma: $L_p(\vec{z}) = (|z_1|^p + |z_2|^p + \cdots + |z_m|^p)^{1/p}$

A $p = \infty$ esetén két vektor távolsága megegyezik a koordinátáinak a legnagyobb eltérésével: $L_\infty(\vec{z}) = \max_i\{|z_i|\}$.

Példa: Számítsuk ki a következő két objektum L_1 (Manhattan-norma), L_2 (Euklideszi-norma), L_4 (Minkowski-norma $p = 4$ mellett), és L_∞ távolságát: $\vec{x} = (5, 8, 3, 5)$ és $\vec{y} = (6, 13, 4, 2)$

$\vec{z} = \vec{x} - \vec{y} = (-1, -5, -1, 3)$.
$L_1(\vec{x} - \vec{y}) = L_1(\vec{z}) = |-1| + |-5| + |-1| + |3| = 10$.
$L_2(\vec{x} - \vec{y}) = L_2(\vec{z}) = \sqrt{(-1)^2 + (-5)^2 + (-1)^2 + (3)^2} = 6$.
$L_4(\vec{x} - \vec{y}) = L_4(\vec{z}) = (|-1|^4 + |-5|^4 + |-1|^4 + |3|^4)^{1/4} = \sqrt[4]{708}$.
$L_\infty(\vec{x} - \vec{y}) = L_\infty(\vec{z}) = \max\{|-1|, |-5|, |-1|, |3|\} = 5$.

„*Az ideális korkülönbség férj és feleség között hat év. Egy svéd kutatás szerint ilyen esetben van maximális lehetőség az utódok születésére.*"
Forrás: http://hvg.hu/egeszseg/20070913_idealis_korkulonbseg.aspx

Habár az elemek leírásában már csak számok szerepelnek, a háttérben megbújó mértékegységeknek nagy szerepük van. Gondoljuk meg, ha méter helyett milliméterben számolunk, akkor sokkal nagyobb értékek fognak szerepelni az elemek leírásában, és így a különbségek is megnőnek. A nagy értékekkel rendelkező attribútumoknak nagyobb hatásuk van a hasonlóság értékére, mint a kis értékkel rendelkezőknek. Ha az attribútumok értékeinek nagyságrendje jelentősen különbözik, fontos lehet az egyes attribútumok normalizálása, azaz transzformálhatjuk az összes attribútumot például a [0,1] intervallumba, majd ezen transzformált attribútumok alapján számíthatjuk a távolságokat (lásd 3.3.5. fejezetet).

Gyakran előfordul, hogy a különbözőség megállapításánál bizonyos attribútumokra nagyobb súlyt szeretnénk helyezni. Például két ember összehasonlításánál a hajszínnek nagyobb szerepe van, mint annak, hogy melyik lábujja a legnagyobb. Ha figyelembe vesszük az attribútumok súlyait, akkor például az Euklideszi-távolság így módosul:

$$d(x, y) = \sqrt{w_1|x_1 - y_1|^2 + w_2|x_2 - y_2|^2 + \cdots + w_m|x_m - y_m|^2},$$

ahol w_i-vel jelöltük i-edik attribútum súlyát és a súlyokat úgy választjuk meg, hogy $\sum_{i=1}^{m} w_i = 1$.

Előfordulhat, hogy olyan attribútummal van dolgunk, amely értékeit nemlineáris léptékben ábrázoljuk, ezeket nemlineáris növekedésű attribútumnak szokás hívni. Például a baktérium populációk növekedését vagy algoritmusok futási idejét exponenciális skálán érdemes ábrázolni. Az ilyen attribútumoknál nem célszerű az attribútum eredeti értékén közvetlenül számolni a távolságot, mert ez óriási különbözőségeket eredményez azokon a helyeken, ahol kis különbözőséget várunk. Legyen például az A algoritmus futásideje (egy adott számítógépen) 10 másodperc, a B algoritmusé 20 másodperc, a C és D algoritmusoké rendre 1 óra és 2 óra. Az általánosan alkalmazott megközelítés szerint, a futásidők közti másodpercben, percben, órában kifejezett abszolút különbség helyett elsődlegesen arra irányul figyelmünk, hogy a B és D algoritmusok 2-szer több ideig futnak, mint az A és a C algoritmusok. Ilyen megközelítésben az A és B algoritmusok távolsága (futásidejük alapján) megegyezik a C és D algoritmusok távolságával, hiszen az egyik futásideje – mindkét esetben – kétszerese a másikénak.

A nemlineáris növekedésű attribútumok esetén két megközelítés között szokás választani. Egyrészt használhatjuk az intervallum alapú hasonlóságot, de nem az attribútum eredeti értékén, hanem annak logaritmusán. Másrészt vehetjük csak az értékek közti sorrendet a hasonlóság alapjául.

3.2.5. Vegyes attribútumok

Az előző részekben azt tekintettük át, hogyan definiáljuk a távolságot két objektum között adott típusú attribútumok esetén. Mit tegyünk akkor, ha egy objektum leírásánál vegyesen adottak a különböző típusú – intervallum, bináris, kategória – attribútumok? Csoportosítsuk az egyes attribútumokat típusuk szerint, és határozzuk meg a két objektum távolságát minden csoportra nézve. A kapott távolságokat képezzük a [0,1] intervallumba. Minden csoportnak feleltessünk meg egy-egy dimenziót a térben, így két objektum távolságához hozzárendelhetünk egy vektort a vektortérben. A távolság értékét feleltessük meg a vektor hosszának.

Ennek a megközelítésnek a hátránya, hogy ha például egyetlen kategória típusú attribútum van, akkor az ugyanolyan súllyal fog szerepelni, mint akár tíz bináris attribútum összesen. Célszerű ezért az egyes csoportok (attribútumtípusok) által szolgáltatott értékeket súlyozni a hozzájuk tartozó attribútumok számával.

3.2.6. Speciális esetek

Egyre több olyan alkalmazás kerül elő, ahol a fent definiált általános távolsági függvények nem ragadják meg jól két objektum különbözőségét. A teljesség igé-

nye nélkül bemutatunk két olyan esetet, amikor speciális távolsági függvényre van szükség.

Elemsorozatok távolsága

Elemsorozaton egy véges halmazból vett elemek sorozatát értjük. Például a magyar nyelven értelmezett szavak elemsorozatok. Nézzük az $S = \langle abcde \rangle$ sorozatot. Legtöbben azt mondanánk, hogy a $\langle bcdxye \rangle$ sorozat jobban hasonlít S-re, mint az $\langle xxxddd \rangle$ sorozat. Nem ezt kapnánk, ha a pozíciókban megegyező elemek relatív számával definiálnánk a távolságot.

Egy elterjedt mérték az elemsorozatok távolságára az ún. *szerkesztési távolság* [Peltola és tsa., 1984, Nerbonne és tsa., 1999]. Két sorozatnak kicsi a szerkesztési távolsága, ha az egyik sorozatból kevés elem törlésével ill. beszúrásával megkaphatjuk a másikat. Pontosabban: két sorozat szerkesztési távolsága azt adja meg, hogy legkevesebb hány beszúrás és törlés művelettel kaphatjuk meg az egyik sorozatból a másikat. A szerkesztési távolság alapján csoportosíthatunk dokumentumokat, weboldalakat, DNS sorozatokat, vagy kereshetünk illegális másolatokat. Szerkesztési távolságon alapuló távolsági mértékek nagyon népszerűek az idősor típusú adatok esetében (lásd 7. fejezetet).

Bezárt szögön alapuló távolság

Vannak alkalmazások, ahol nem a vektorok különbségének a hossza a lényeges, hanem a vektorok által bezárt szög. Például dokumentumok hasonlóságával kapcsolatban számos okfejtést olvashatunk, hogy miért jobb szögekkel dolgozni, mint a vektorok hosszával. Egy dokumentumot többféleképpen is ábrázolhatunk vektorként. A legegyszerűbb esetben a vektor egyes koordinátái szavaknak felelnek meg. A koordináták értéke attól függ, hogy hányszor (akár 0-szor) fordult elő egy koordinátának megfelelő szó a dokumentumban.

Ilyen esetben gyakran használják a koszinusz-távolságot:

$$d(x, y) = \arccos \frac{\vec{x}^T \vec{y}}{||\vec{x}|| \cdot ||\vec{y}||}.$$

Az előbbi képlet két objektumot (dokumentumot) reprezentáló vektorok által bezárt szög nagyságát adja. Sokszor azonban a bezárt szög helyett egyszerűen annak koszinuszával dolgozunk (koszinusz-hasonlóság):

$$s(x, y) = \frac{\vec{x}^T \vec{y}}{||\vec{x}|| \cdot ||\vec{y}||}.$$

Vegyük észre, hogy a bezárt szög egy távolsági mérték, annak koszinusza azonban hasonlósági mérték: azonos dokumentumok esetén a vektorok bezárt szöge

nulla, melynek koszinusza 1, különböző dokumentumok esetén a bezárt szög nullánál nagyobb lehet, melynek koszinusza kisebb 1-nél.

3.3. Előfeldolgozás

A következőkben a legfontosabb előfeldolgozási lépéseket tekintjük át. Külön fejezetben (9. fejezet) írunk arról, hogyan tudjuk ezeket az előfeldolgozási lépéseket Weka-ban végrehajtani.

3.3.1. Hiányzó értékek kezelése

Míg néhány adatbányászati algoritmus (pl. Naive Bayes, lásd 4.7.1. fejezetben) különösebb gond nélkül elboldogul olyan adatbázisokon, amelyekben az objektumok néhány attribútumának értéke hiányzik, számos algoritmusok csak olyan objektumokat tud kezelni, amelyeknek minden attribútuma adott. Valós adatbázisoknál ez nem mindig áll fenn, könnyen lehet, hogy bizonyos celláknak nincs értékük. Például az orvos bizonyos teszteken nem végzett el a pácienssen, mert nem találta szükségesnek, vagy az adott attribútum egyes esetekben nem értelmezhető, ilyen lehet az dohányzásról való leszokás óta eltelt idő olyan embereknél, akik korábban nem dohányoztak.

Legegyszerűbb megoldáként törölhetjük azokat az objektumokat, amelyek tartalmaznak hiányzó attribútumokat, de lehet, hogy ekkor annyira lecsökken az adatbázis mérete, hogy az alkalmatlan lesz az elemzésre, vagy legalábbis adott konfidencia mellett keveset tudunk mondani az összefüggésekről. A hiányzó értékeket tartalmazó objektumok hasznos információt tartalmazhatnak, ne dobjuk el őket, ha nem muszáj.

A hiányzó cellákat fel kell töltenünk valamilyen értékkel, vagy a hiányzást mint külön attribútumértéket kell kezelnünk. Ez utóbbit teszi például a C4.5 nevű döntési fákat előállító módszer [Quinlan, 1993] is.

Sokféleképpen helyettesíthetjük a hiányzó értékeket. Ha a hiányzó attribútum kategória típusú, akkor vehetünk egy alapértelmezett értéket, vagy az attribútum leggyakrabban előforduló értékét (módusz). Létrehozhatunk egy objektum helyett sok új teljes objektumot úgy, hogy a hiányzó attribútum helyére az összes lehetséges értéket beírjuk. Az újonnan létrejött objektumokat súlyozhatjuk aszerint, hogy melyik helyettesítés mennyire valószínű. Intervallum attribútumok esetén szokás a hiányzó értéket az átlaggal vagy a mediánnal helyettesíteni.

Ha osztályozási feladattal van dolgunk, akkor a fenti értékek számításánál szorítkozhatunk csak az adott osztályba tartozó objektumokra. Sőt ezt a gondolatot vihetjük tovább és az értékek számításánál tekinthetjük csak azokat az

objektumokat (ha vannak ilyenek), amelyek attribútumainak értékei megegyeznek a hiányzó értéket tartalmazó objektum ismert attribútumainak értékeivel. Itt érdemes gondosan eljárni és csak a fontos attribútumokat vizsgálni (gondoljuk meg, ha például az azonosító attribútumot nem zárjuk ki, akkor egyetlen elemet sem fogunk figyelembe venni).

A hiányzó értékek ismert értékek alapján történő becslésére használhatunk osztályozó és regressziós algoritmusokat [Farhangfar és tsa., 2008] vagy egyszerűen keresünk egy hasonló objektumot, amelynek az adott attribútumának értéke ismert. Tegyük fel, hogy egy x objektum A attribútumának értéke hiányzik, de találunk egy x-hez hasonló y objektumot, amelynek ugyanezen attribútumának értéke ismert. Ekkor az x objektum A attribútumának értéket egyszerűen ugyanarra az értékre állíthatjuk, mint ami az y objektum A attribútumának éréke. Ilyenkor azt mondjuk, hogy az y objektum *donor* szerepet játszik x számára. Felmerül, hogy korlátozzuk, hogy egy-egy objektum legfeljebb hányszor lehet donor: ha például az előbbi y objektum túl gyakran szerepel donorként az A attribútum hiánya miatt, az esetlegesen torzíthatja az A attribútum értékeinek eloszlását. A donorszerep korlátozásának hatását vizsgálja Joenssen és Bankhofer cikke [Joenssen és Bankhofer, 2012].

3.3.2. Attribútumtranszformációk

A következőkben attribútumok létrehozásáról és törléséről lesz szó, külön fejezetekben foglalkozunk az attribútumok diszkretizálásával (3.3.4. fejezet) és normalizálásával (3.3.5. fejezet).

Új attribútumok létrehozása

Előfordulhat, hogy egy attribútumérték előrejelzésénél vagy becslésénél a többi attribútum külön-külön kevésbé bizonyulnak használhatónak, mint azok valamilyen kombinációja. Például rendelkezésünkre állhat az emberek magassága és a testtömege, egy betegséggel szembeni rizikófaktor becslésekor azonban a testtömeg index (body mass index, BMI) jobban használható lehet, mint a magasság és testtömeg külön-külön. A testtömeg index a magasságból és testtömegből számolható, ezért akár el is várhatnánk, hogy az osztályozó algoritmus automatikusan felismerje, hogy a két attribútum milyen függvénye lényeges az adott felismerési feladat szempontjából, hiszen, mint azt látni fogjuk, az osztályozás maga is egy függvény approximáció. A gyakorlatban azonban az előzetes ismeretek, apriori tudás bevitelével szinte mindig javul az osztályozás minősége. Ne várjunk csodát az osztályozótól, amikor tudunk, segítsünk neki.

Attribútumok törlése

Az adatbányász algoritmustól elvárjuk, hogy a lényegtelen attribútumokat ne vegye figyelembe. Szokták mondani, hogy a döntési fák nagy előnye, hogy a döntését csak a lényeges attribútumok alapján hozzák meg. Ez azt sugallja, hogy nyugodtan összekapcsolhatjuk az adattáblákat és létrehozhatunk egy sok attribútumot tartalmazó táblát, a csodamódszerek majd figyelmen kívül hagyják a lényegtelen attribútumokat. Sajnos ez nem mindig van így, a felesleges attribútumok által okozott zaj ugyanis általában rontja a módszerek teljesítményét. Erre a problémára a döntési fáknál visszatérünk.

Ha tehetjük, segítsünk az adatbányász módszereken és töröljük azokat az attribútumokat (például egyedi azonosító), amelyekről tudjuk, hogy nem fontosak az elemzés céljából.

Lényegtelen attriútumok felismerése

Külön adatbányászati téma a lényegtelen attribútumok automatikus (gépi) felismerése és kiszűrése. Az ilyen célból kidolgozott eljárások két csoportba sorolhatók: a filter módszerek valamilyen "külső" kritérium (pl. osztálycímkével vagy más attribútumokkal való korreláció, feltételes entrópia, stb.) alapján értékelik az egyes attribútumokat, és csak a releváns(nak látszó) attribútumokat tartják meg. A wrapper módszerek az értékelésbe bevonják az osztályozó modellt is: kipróbálják az osztályozó algoritmust különböző attribútumokhalmazok esetén, megviszgálják, hogy mikor adja a legjobb eredményt. (Az osztályozók kiértékeléséről a későbbiekben lesz szó.)

3.3.3. Adatok torzítása

Miért akarnánk torzítani, rontani az adathalmazt? Több okunk is lehet rá. Például vizsgálni szeretnénk, hogy egy adott módszer mennyire érzékeny a zajra. Az is lehet, hogy egy cég publikussá teszi bizonyos adatait, de először azt kicsit átalakítja/lerontja úgy, hogy az adatelemzés technikailag kivitelezhető legyen, de a konkurrencia ne tudjon hasznos információhoz jutni. A torzítás oka lehet továbbá a magánszféra, a személyes adatok védelme.

Sok esetben egyáltalán nem nyilvánvaló, hogy az adatok torzítása szükséges a személyes információk védelmében. Korolova és társai (2009) beszámolnak arról[2], hogy 2006-ban az AOL "anonimizált" módon nyilvánosságra hozta, hogy milyen kifejezésekre keresnek a keresőrendszer felhasználói. Az anonimizáció abban állt, hogy a felhasználóneveket és IP-címeket véletlenszerű azonosítókra

[2] A konferencián elhangzott előadás: http://videolectures.net/www09_korolova_rsqcp/ , a cikk: http://www2009.org/proceedings/pdf/p171.pdf

cserélték. Elsőre azt gondolnánk, hogy ez jóval több, mint elégséges, az ilyen módon anonimizált adatból érzékeny, személyes információk nem nyerhetők ki, csak általános trendek. Hamar kiderült azonban, hogy a valóság épp ennek ellenkezője. Az alapvető probléma abból adódik, hogy amikor a felhasználók egy keresőrendszerrel kommunikálnak, implicite feltételezik, hogy mindaz, amit a keresőrendszertől kérdeznek, kettőjük közt, a felhasználó és a keresőrendszer közt fog maradni, harmadik személy nem látja azt. Ezért a felhasználók érzékeny, személyes informćiókra is keresnek. A felhasználók jelentős része időnként rákeres saját nevére, TAJ-számára (social security number), bankkártyaszámára, vélt vagy valós betegségeire, tüneteire, stb. A látszólag anoním módon publikált adatokban egy újságíró sikeresen beazonosított egy 60 év körül hölgyet, aki sok személyes információt adott ki magáról, például azt, hogy társat keres. Mindez óriási botrányt kavart, két alkalmazottat kirúgtak, jogi eljárás indult, és mondanunk sem kell, hogy az eset sokat ártott az AOL jóhírének.

A kérdés tehát az, hogyan lehet adatokat a személyes információk védelme mellett publikálni? A válasz, természetesen, az adatok torzítása. Amint az előbbi példában is láttuk, az ad hoc ötletek szerint történő torzítások nemkívánt eredményre vezethetnek. Ezért egy rendkívül izgalmas, új kutatási terület a *bizonyítható biztonság* témaköréhez kapcsolódik: hogyan publikáljunk adatokat oly módon, hogy bizonyítható legyen, hogy azokból nem következtethetünk érzékeny információkra? Korolova és szerzőtársai egy olyan módszert javasoltak, mely segítségével keresőrendszereknek feltett keresőkérdéseket lehet aggregált és torzított formában publikálni olyan módon, hogy abból bizonyíthatóan nem lehet visszakövetkeztetni érzékeny személyes adatokra. Bemutatták azt is, hogy a torzítások és aggregáció ellenére a publikált adatok számos alkalmazásban jól használhatóak, majdnem ugyanolyan jól, mint az eredeti adatok [Korolova és tsa., 2009].

3.3.4. Diszkretizálás

A diszkretizálás/kvantálás során szám típusú attribútumot kategória típusúvá alakítjuk. Az attribútum értékkészletét intervallumokra/csoportokra osztjuk és minden intervallumhoz egy kategóriát rendelünk. A diszkretizálás során nyilván információt veszítünk viszont segíthetünk az adatbányász algoritmuson. Számos módszer létezik diszkretizációra.

Kialakíthatunk egyenő szélességű vagy egyenő gyakoriságú intervallumokat. Az egyenlő gyakoriságú intervallumoknál minden intervallumba ugyanannyi adatpont esik.

PKI (Proportional k-Interval Discretization) diszkretizációs módszerként hivatkoznak arra az esetre, amikor egyenlő gyakoriságú intervallumokat alakítunk ki és az intervallumok száma az adatpontok négyzetgyökével egyezik meg

[Yang, 2001].

1R módszer

Az 1R tulajdonképpen egy egyszerű osztályozó módszer, amely tartalmaz egy diszkretizációs eljárást. Egy példán keresztül szemléltetjük az algoritmust. A diszkretizálandó attribútum a hőmérsékletet adja meg Fahrenheitban mérve. A tanítómintában az egyes hőmérsékletekhez a következő osztályértékek tartoznak (az attribútumértékeket nagyság szerint növekvően sorba kell rendezni):

64	65	68	69	70	71	72	72	75	75	80	81	83	85
1	0	1	1	1	0	0	1	1	1	0	1	1	0

Egy lehetséges csoportosítás szerint induljuk el a legkisebb értékektől és akkor zárjuk le az aktuális intervallumot, ha változik az osztály. A példában nyolc csoportot hoznánk létre:

64	65	68	69	70	71	72	72	75	75	80	81	83	85
1	0	1	1	1	0	0	1	1	1	0	1	1	0
1	0		1			0			1		0	1	0

A határokat a felezőpontokban megválasztva a következő határokat hoznánk létre: 64.5, 66.5, 70.5, 72, 77.5, 80.5, 84. A felosztás persze nem egyértelmű, hiszen ugyanahhoz a ponthoz tartozhatnak különböző osztályok is. Erre példa a 72. Ha van egy osztály, amely a leggyakrabban fordul elő a kérdéses tanítópontok között, akkor azt az osztályt rendeljük a ponthoz. Ellenkező esetben a leggyakoribb osztályok közül azt, amelyik a legkevesebb csoportot/felosztást adja.

A túl sok kicsi intervallum létrehozásának elkerülése végett célszerű megadni egy minimális elemszám küszöböt, legalább ennyi elemet kell tartalmaznia minden csoportnak, kivéve az utolsót. Ha ez a minimum érték három, akkor a következő csoportokat hozzuk létre.

64	65	68	69	70	71	72	72	75	75	80	81	83	85
1	0	1	1	1	0	0	1	1	1	0	1	1	0
		1					1				0 v 1		

Amikor a szomszédos csoportokban megegyezik a legtöbbször előforduló osztályérték, akkor a két csoport közötti határt eltörölhetjük. Ez alapján csak két intervallumot fogunk előállítani, a határvonalat a 77.5 adja. Az utolsó csoporthoz önkényesen rendeltük a 0-ás osztályértéket. Ha nem így teszünk, akkor egyáltalán nem jelölünk ki határt és minden pont egy intervallumba tartozik.

Vegyük észre, hogy különböző felosztás kaphatunk, attól függően, hogy a sor melyik végétől kezdjük a módszert.

További diszkretizáló eljárások közül megemlítjük az entrópia alapú diszkretizálást, melynek során olyan úgy keressük meg az intervallumok határait, hogy az intervallumokba tartozó objektumok osztálycímkéinek entrópiáját minimalizáljuk [Tan és tsa., 2005].

3.3.5. Normalizálás

Normalizáláson azt értjük, hogy az attribútum elemeit egy másik intervallum elemeivel helyettesítjük úgy, hogy a helyettesített értékek eloszlása megegyezzen az eredeti értékek eloszlásával. Tegyük fel, hogy az A attribútum eredetileg az $a_1, a_2, \ldots, a_l$ értékeket veszi fel. Az a_j, $j = 1, \ldots, l$ érték normáltját a'_j-vel jelöljük. Normalizálásra két módszer terjedt el.

Min-max normalizálás: egy lineáris transzformáció:

$$a'_j = \frac{a_j - min_A}{max_A - min_A},$$

ahol min_A (max_A) jelöli az A attribútum eredeti értékei közül a legkisebbet (legnagyobbat). Ezen transzformáció után minden elem a [0,1] intervallumba fog esni.

Standard normalizálás (z-score normalization):

$$a'_j = \frac{a_j - \bar{A}}{\sigma_A},$$

ahol $\bar{A}$ az A attribútum átlaga, σ_A pedig a szórása. A hagyományos szórás

$$\sigma_A = \sqrt{\frac{\sum_{i=1}^{l}(a_i - \bar{A})^2}{l}}$$

helyett az abszolút szórást

$$\sigma'_A = \frac{\sum_{i=1}^{l}|a_i - \bar{A}|}{l}$$

is használni szokták. Ennek előnye, hogy csökkenti az átlagtól távol eső pontok (különcök, outlier-ek) hatását.

3.3.6. Mintavételezés

Az adatbányászati algoritmusok általában erőforrás-igényesek. Ha a bemeneti adathalmaznak csak egy kis szeletét dolgozzuk fel, akkor hamarabb kapunk eredményt. A mintavételezés következménye, hogy az így kapott eredmény nem biztos, hogy eléggé pontos. Vannak esetek, amikor a pontos eredménynél fontosabb a gyors adatfeldolgozás. Ilyen esetekben nagyon hasznos egy olyan mintaméret meghatározása, aminél az algoritmus gyors, és a hibázás valószínűsége kicsi.

Az adatbányászat és a statisztika által követett megközelítések közti különbséget a mintavételezés során tetten érhetjük [Tan és tsa., 2005]. A statisztikában jellemzően azért mintavételeznek, mert a teljes populáció *megfigyelése* valamilyen értelemben túl drága vagy más okból nem kivitelezhető, ezért csak egy (remélhetőleg reprezentatív) mintát figyelnek meg (pl. néhány ezer ember megkérdezéseként végzett közvéleménykutatásból próbálnak következtetni a teljes lakosság véleményére). Ezzel szemben egy adatbányászati elemzés során rendelkezésünkre állnak a teljes populációt leíró adatok, de az *adatbázis óriási mérete* miatt kényszerülünk arra, hogy az adatok egy részével dolgozzunk csak, mert a tervezett (mélyreható) elemzés elvégzése a teljes adatbázison túlságosan költséges lenne, sok időt venne igénybe.

Az alábbi példában azt látjuk, hogy egy gyógyszer hatékonyságát egy tízezer fős mintán igazolták:

„*Az Elevit hatékonyságát igazoló klinikai vizsgálatokat közel tízezer magyar kismama bevonásával végezték. A vizsgálatok során az Elevit szedésével kilencvenkét százalékkal csökkent az idegrendszeri fejlődési rendellenességek előfordulása.*" Forrás: Baba Patika X. évfolyam 10. szám, 44. old., 2007. okt.

A mintaméret becslése Csernov-korláttal

A hiba mértékéről csak abban az esetben tudunk bővebben nyilatkozni, ha tudjuk, milyen jellegű összefüggéseket nyerünk ki. Most azt a speciális esetet nézzük meg, amikor elemek előfordulásának valószínűségét akarjuk közelíteni a relatív gyakoriságukkal. Gyakori minták és asszociációs szabályok bányászatánál, χ^2 alapú függetlenségvizsgálatnál ez az eset áll fenn.

Tegyük fel, hogy elemek halmazából egy tetszőleges x elem előfordulásának valószínűsége p és m megfigyelés/minta áll rendelkezésünkre. A mintavételezés hibázik, amennyiben x relatív gyakorisága eltér p-től, pontosabban a mintavételezés hibája:

$$\text{hiba}(m) = \mathbb{P}\Big(\big|\text{rel. gyakoriság}(x) - p\big| \geq \epsilon\Big).$$

Jelölje X_i azt a valószínűségi változót, amely 1, ha x-et választottuk egy i-edik húzásnál, különben 0, és legyen $Y = \sum_{i=1}^{m} X_i$. Mivel a húzások egymástól függetlenek, az Y eloszlása m, p paraméterű binomiális eloszlást követ. Ezt felhasználva:

$$\begin{aligned}\text{hiba}(m) &= \mathbb{P}\Big(\Big|\frac{Y}{m} - p\Big| \geq \epsilon\Big) = \mathbb{P}\Big(\Big|Y - m \cdot p\Big| \geq m \cdot \epsilon\Big) \\ &= \mathbb{P}\Big(\Big|Y - \mathbb{E}\big[Y\big]\Big| \geq m \cdot \epsilon\Big) \\ &= \mathbb{P}\Big(Y \geq m \cdot (\mathbb{E}[X] + \epsilon)\Big) + \mathbb{P}\Big(Y \leq m \cdot (\mathbb{E}[X] - \epsilon)\Big)\end{aligned}$$

A második egyenlőségnél kihasználtuk, hogy a binomiális eloszlás várható értéke $m \cdot p$. Tetszőleges eloszlás esetén a várható értékétől való eltérés valószínűségére több ismert korlát is létezik [Alon and Spencer, 2000]. A Csernov-korlát (amely a Hoeffding korlát egy speciális esete) a következőket adja:

$$\mathbb{P}\Big(Y \geq m \cdot (\mathbb{E}[X] + \epsilon)\Big) \leq e^{-2\epsilon^2 m}$$

és

$$\mathbb{P}\Big(Y \leq m \cdot (\mathbb{E}[X] - \epsilon)\Big) \leq e^{-2\epsilon^2 m}$$

amiből megkapjuk, hogy:

$$\text{hiba}(m) \leq 2 \cdot e^{-2\epsilon^2 m}.$$

Amennyiben a hibakorlátot δ-val jelöljük, akkor igaznak kell lennie, hogy

$$m \geq \frac{1}{2\epsilon^2} \ln \frac{2}{\delta}.$$

Csak a véletlen műve, ha egy elem megfigyelt relatív gyakorisága *pontosan* egybeesik az adott elem előfordulásának valószínűségével. Ha azonban a minta elég nagy, akkor nagy valószínűséggel *kicsi* lesz az eltérés a megfigyelt relatív gyakoriság és az adott elem valószínűsége között. Ha például azt szeretnénk, hogy az eltérés az elem megfigyelt relatív gyakorisága és valószínűsége között legfeljebb 0.01 legyen, és azt várjuk el, hogy 1 %-nál kisebb legyen annak a valószínűsége, hogy az eltérés mégis nagyobb 0.01-nél, akkor a minta mérete legalább 27000 kell legyen. A 3.1 táblázatban adott eltérés- és valószínűség-korlátokhoz tartozó minimális mintaméret található.

További eljárások a mintaméret becslésére

Gyanús, hogy az előző szakasz végén kapott képletben nem szerepel az x elem előfordulásának valószínűsége, p. Ez nem Csernov hibája, hanem a abból adódik, hogy túl gyenge korlátot használtunk, olyat, amelyik nem vette figyelembe

ϵ	δ	m
0.05	0.01	1060
0.01	0.01	27000
0.01	0.001	38000
0.01	0.0001	50000
0.001	0.01	2700000
0.001	0.001	3800000
0.001	0.0001	5000000

3.1. táblázat. A minimális minta mérete rögzített ϵ, δ mellett

az X eloszlását, ezért előbbi becslésünk túlságosan pesszimista: igaz ugyan, hogy 27000 méretű minta mellett legfeljebb 1 % lesz a valószínűsége annak, hogy a megfigyelt relatív gyakoriság és a valószínűség közti eltérés nagyobb 0.01-nél, de valójában ennél kisebb minta is elég lenne ugyanekkor pontossághoz. A következőkben az előző szakaszban adott becslésnél pontosabb becslést keresünk a mintaméretre.

A Csernov-Hoeffding korlát feltételezi, hogy X binomiális eloszlású es 0,1 értékeket vehet fel:

$$\text{hiba}(m) \leq e^{-D(p+\epsilon||p)m} + e^{-D(p-\epsilon||p)m},$$

ahol D a Kullback-Leibler divergenciafüggvényt jelöli:

$$D(a||b) = a \log \frac{a}{b} + (1-a) \log \frac{1-a}{1-b}.$$

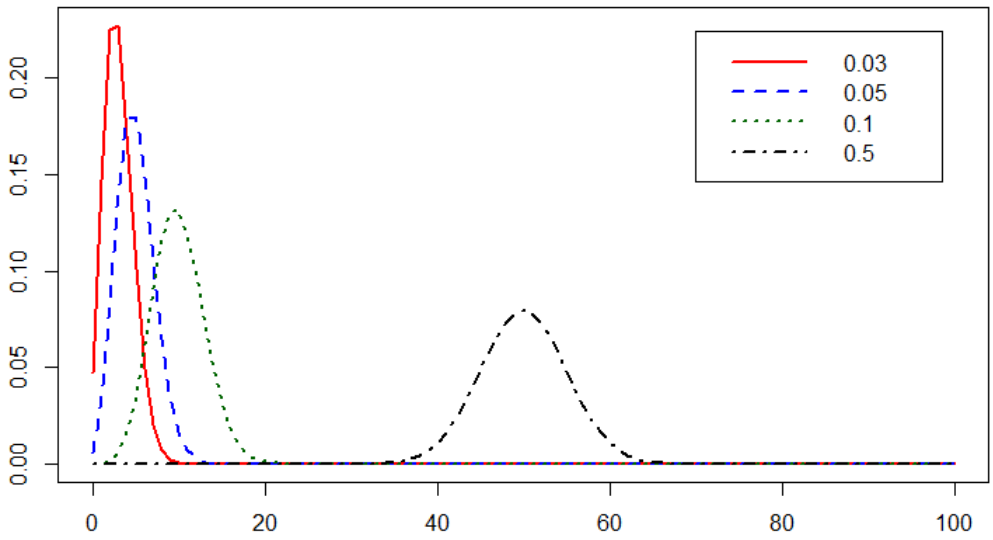
A Csernov korlátot megkapjuk, ha észrevesszük, hogy $D(p+\epsilon||p) \geq 2\epsilon^2$.

Mivel ismerjük Y sűrűségfüggvényét, így tetszőleges intervallumra meg tudjuk mondani az előfordulás valószínűségét. Megkísérelhetjük, hogy ez alapján adjunk becslést a minta méretére:

$$\mathbb{P}\Big(\Big|Y - m \cdot p\Big| \geq m \cdot \epsilon\Big) = 1 - \sum_{i=\max\{\lceil mp-m\epsilon\rceil,0\}}^{\min\{\lfloor mp+m\epsilon\rfloor,m\}} \binom{m}{i} p^i (1-p)^{m-i}.$$

Az (m,p) paraméterű binomiális eloszlás eloszlásfüggvényét $F(x,m,p)$-vel jelölölve:

$$\begin{aligned}\mathbb{P}\Big(\Big|Y - m \cdot p\Big| \geq m \cdot \epsilon\Big) = 1 &+ F(\max\{\lfloor mp - m\epsilon\rfloor, 0\}, m, p) \\ &- F(\min\{\lceil mp + m\epsilon\rceil, m\} - 1, m, p).\end{aligned}$$



3.1. ábra. Különböző p paraméterű binomiális eloszlások

Sajnos a fentiek alapján nem tudunk zárt képletet adni a minta méretének alsó korlátja és az ϵ, δ páros közötti kapcsolatra.

Azt gondolhatjuk, hogy minél kisebb a p, annál nagyobb mintát kell venni a pontos becsléshez. Mint látni fogjuk, ez nincs így. Mivel a binomiális eloszlás szimmetrikus, ezért a $p \leq 0.5$ esetekre szorítkozunk.

Amennyiben $p \leq \epsilon$, akkor a $mp - m\epsilon \leq 0$ és így a hiba $1 - F(\lfloor mp + m\epsilon \rfloor, m, p)$-re egyszerüsödik. Ez viszont nullához tart, amennyiben $p \to 0$, hiszen

$$1 - F(\lfloor mp + m\epsilon \rfloor, m, p) \leq 1 - F(\lfloor m\epsilon \rfloor, m, p) = \mathbb{P}(Y \geq \lfloor m\epsilon \rfloor) \leq \frac{mp}{\lfloor m\epsilon \rfloor}.$$

Az utolsó egyenlőtlenségnél a Markov egyenlőtlenséget használtuk fel. A 0 határértéket megkaphattuk volna úgy is, ha a Hoeffding-korlát határértékét számítjuk ki $p \to 0$ esetén. Az eredmény ellentmond elvárásainknak, hiszen eszerint kis valószínűségeket kisebb mintával tudunk jól közelíteni.

A következőkben megvizsgáljuk $p \geq \epsilon$ esetét. Továbbra is igaz, hogy a p növelésével növekszik a hiba? A válasz igenlő. Ezt az állítást csak szemléltetni fogjuk. Vessünk egy pillantást a 3.1 ábrára, amelyen két, különböző p paraméterű binomiális eloszlást láthatunk.

Két dolgot vehetünk észre. A kisebb p-hez tartozó maximális valószínűség nagyobb. A nagy valószínűségek a várható érték kisebb környezetében találhatók. Az észrevételeink általánosan is igazak. A második észrevétel például

a szórással van kapcsolatban. A kisebb p paraméterű eloszlás szórása kisebb. Legyen a két paraméter p és q és legyen $p < q < 0.5$. Ekkor:

$$\begin{aligned} mp(1-p) = \sigma_p^2 &< \sigma_q^2 = mq(1-q) \\ p - p^2 &< q - q^2 \\ 0 &< (q-p)(1-p-q). \end{aligned}$$

A kisebb valószínűségeknél a várható érték szűkebb környezetében vannak a nagy valószínűségek, ezért a várható érték $\pm\epsilon m$ környezetén kívüli pontok valószínűséginek összege kisebb, azaz a hiba kisebb!

A következő ábrákon az érvelést támasztjuk alá. A 3.2 ábrán a hibát ábrázoljuk a minta mérete és a valószínűség függvényében rögzített ϵ mellett. Látjuk, hogy ha növekszik p (vagy csökken m), akkor csökken a hiba valószínűsége.

A 3.3 ábrán megint a mintavételezés hibáját ábrázoltuk, de most az ϵ (0.035) mellett a minta mérete (200) is rögzítve van. Itt még jobban látszik, hogy ahogy csökken p úgy csökken a hiba is.

A 3.2 táblázatban a binomiális eloszlásból számolt hibát és a Hoeffding-korlátot láthatjuk néhány p valószínűségre. Nyilvánvaló, hogy a Hoeffding-korlát használhatóbb, mint a Csernov-korlát és jól mutatja, hogy a p csökkenésével a hiba is csökken, ugyanakkor a tényleges valószínűségek elég távol vannak a felső korláttól.

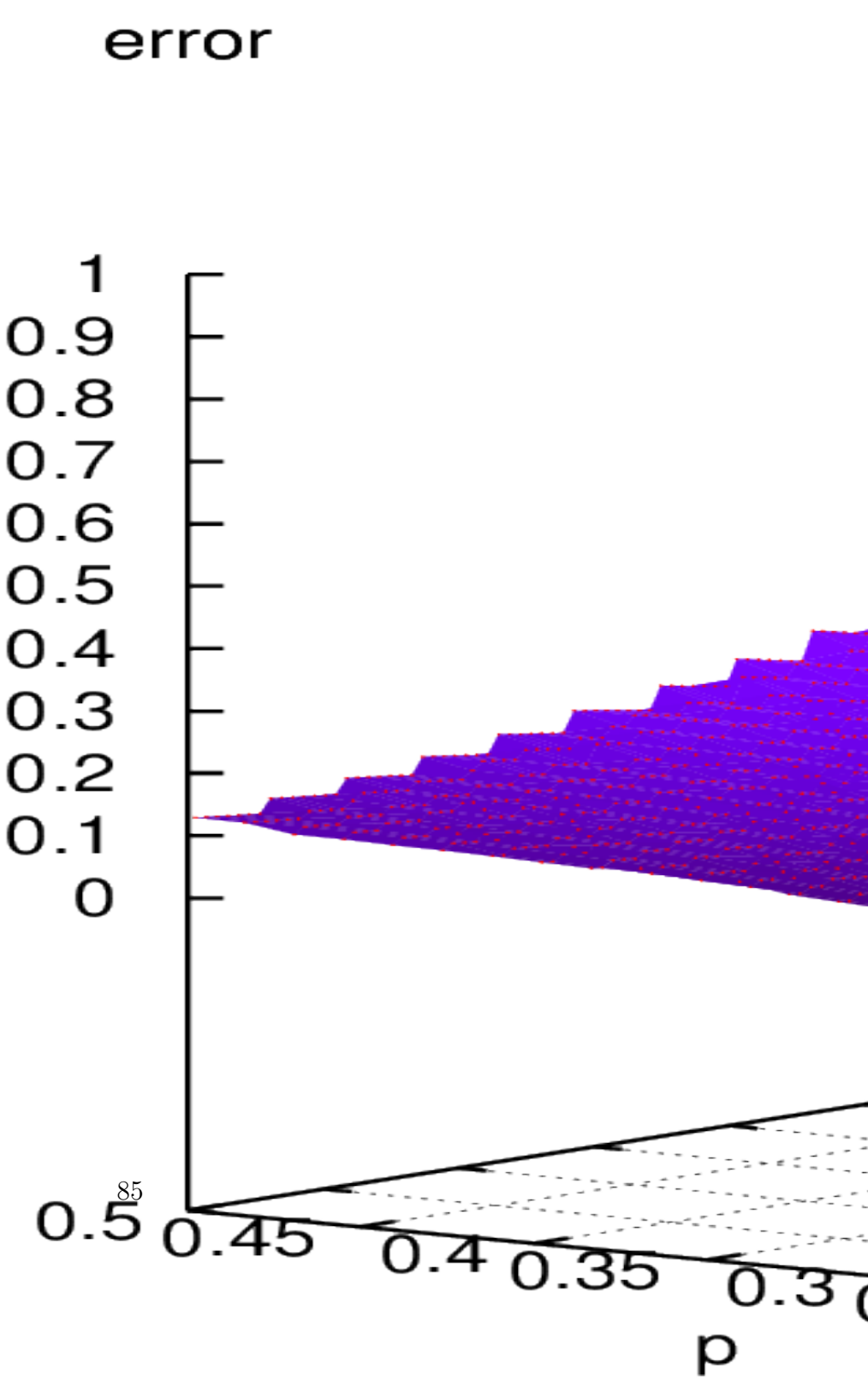
p	$\mathbb{P}\Big(\big\lvert Y - m \cdot p\big\rvert \geq m \cdot \epsilon\Big)$	Hoeffding
0.02	0.00078	0.01420
0.04	0.00728	0.08386
0.06	0.02431	0.21903
0.1	0.07547	0.50479
0.2	0.18433	0.92763
0.4	0.27896	1.19989

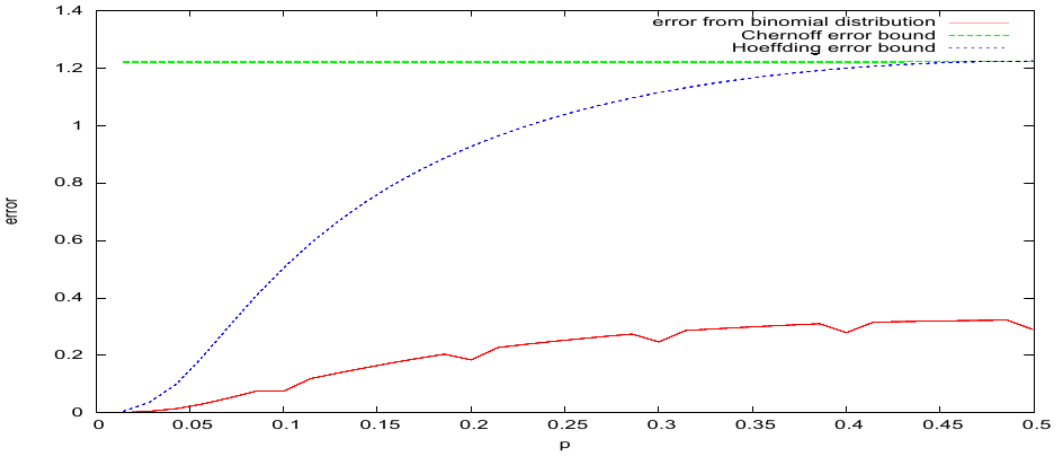
3.2. táblázat. A mintavételezés hibája és a hibára adott Hoeffding korlát néhány előfordulás valószínűségre $m = 200$ és $\epsilon = 0.035$ esetén

Ha ezeknél a paramétereknél a Csernov-korlátot alkalmazzuk, akkor azt kapjuk, hogy a hiba kisebb 1.2-nél. Mivel a hibát egy valószínűséggel definiáltuk ez elég semmitmondó korlát.

Az elemzés során az intuíciónkkal ellentétes eredményre jutottunk. Ennek okát keresve, idézzük fel a hiba definícióját:

$$\text{hiba}(m) = \mathbb{P}\Big(\big\lvert\text{rel. gyakoriság}(x) - p\big\rvert \geq \epsilon\Big),$$


error
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.5
0.45
0.4
0.35
0.3
p



3.3. ábra. A mintavételezés hibája és a hibára adott felső korlátok az előfordulás valószínűségének függvényében ($m = 200$, $\epsilon = 0.035$)

azaz hibát követünk el, ha a relatív gyakoriság és a tényleges valószínűség közötti különbség nagyobb egy adott konstansnál, amelyet ϵ-nal jelöltünk. A relatív gyakoriságnak a valószínűség egy rögzített szélességű környezetében kell lennie.

Szerencsés az, hogy a hibát a relatív gyakoriság és a valószínűség különbségével mérjük? Ez alapján például ugyanakkora hibát követünk el, ha $p = 0.8$ esetén a relatív gyakoriság 0.81 és ha $p = 0.01$ esetén a relatív gyakoriság nulla, azaz az esemény nem következett be egyszer sem. Az embernek az az érzése van, hogy az első esetben kisebbet hibáztunk.

A fenti érvelés alapján célszerűbb a hibát a valószínűség és a relatív gyakoriság hányadosával mérni. Jobban érdekel minket az, hogy hány százalékkal nagyobb vagy kisebb a relatív gyakoriság a valószínűségnél, mint az abszolút különbség. Ha elfogadjuk ezt az érvelést, akkor a hibát a következőképpen definiáljuk:

$$\begin{aligned}\text{hiba}(m) &= \mathbb{P}\Big(\text{rel. gyakoriság}(x)/p \geq 1+\epsilon\Big) + \mathbb{P}\Big(\text{rel. gyakoriság}(x)/p \leq \frac{1}{1+\epsilon}\Big) \\ &= 1 - \mathbb{P}\Big(\frac{1}{1+\epsilon} < \text{rel. gyakoriság}(x)/p < 1+\epsilon\Big) \\ &= 1 + F(\lfloor mp/(1+\epsilon)\rfloor, m, p) - F(\min\{\lceil mp(1+\epsilon)\rceil, m\} - 1, m, p),\end{aligned}$$

ahol $\epsilon > 0$ valós szám.

Felső korlát ismét létezik [Hagerup és Rüb, 1990].

$$\mathbb{P}\Big(Y/mp \geq 1+\epsilon\Big) \leq \left(\frac{e^{\epsilon}}{(1+\epsilon)^{(1+\epsilon)}}\right)^{mp},$$

továbbá

$$\mathbb{P}\Big(Y/mp \leq 1-\epsilon'\Big) \leq e^{-mp\epsilon'^2/2},$$

amelyből $\epsilon' = \epsilon/(1+\epsilon)$ helyettesítéssel kapjuk, hogy

$$\mathbb{P}\Big(Y/mp \leq 1-\frac{\epsilon}{1+\epsilon} = \frac{1}{1+\epsilon}\Big) \leq e^{-mp\epsilon^2/(2(1+\epsilon)^2)},$$

amiből kapjuk, hogy

$$\text{hiba}(m) \leq \left(\frac{e^{\epsilon}}{(1+\epsilon)^{(1+\epsilon)}}\right)^{mp} + e^{-mp\epsilon^2/(2(1+\epsilon)^2)}.$$

A relatív hibamérés esetén már igaz, hogy minél kisebb az előfordulás valószínűsége, annál nagyobb lesz a hiba, tehát annál nagyobb mintát kell vennünk.

Vegyük észre, hogy csak nagyvonalakban igaz, hogy kisebb p esetén nagyobb a hiba. Ennek oka, hogy a binomiális eloszlás diszkrét eloszlás és ezért ahogy csökkentjük a p-t és úgy tolódik a nem hibát jelentő intervallum a nulla pont felé és előfordulhat az, hogy egy újabb pont bekerül az intervallumba. Például $\epsilon = 0.035$ és $m = 1500$ esetében a $[pm/(1+\epsilon), pm(1+\epsilon)]$ intervallumba nem esik egész érték $p = 0.007$ esetében (hiszen a nem hibát jelentő intervallum [10.1,10.9]), míg $p = 0.006$ esetén igen (ekkor a vizsgált intervallum [8.7,9.3]).

Ha p tart nullához, akkor a hiba egyhez tart. Amennyiben a p kisebb $1/m(1+\epsilon)$, akkor a $\left(\frac{mp}{1+\epsilon}, mp(1+\epsilon)\right)$ intervallumba nem eshet egész érték, ezért az X előfordulásától függetlenül a hiba értéke egy lesz.

A Csernov-korlát alkalmazásánál jobb megoldás tehát a hibát a valószínűség és a relatív gyakoriság hányadosából származtatni és a binomiális eloszlást használni. Mivel a végeredmény nem egy zárt képlet lesz, ezért a hiba vagy a szükséges mintaméret kiszámítása bonyolultabb.

A binomiális eloszlás sem a legpontosabb eredményt adja. Az elemzés során ugyanis feltételeztük, hogy az esemény bekövetkezésének valószínűsége ismert. A valóságban a mintát egy nagy alaphalmazból vesszük. Például a népszavazást megelőző közvélemény-kutatásokban a mintát a felnőtt lakosságból vesszük, amely egy véges halmaz. Ha úgy tesszük fel a kérdést, hogy egy M alaphalmazból mekkora m mintát kell vennünk, hogy a mintában az x relatív gyakorisága kis mértékben térjen el az x M-beli relatív gyakoriságától, akkor a binomiális eloszlás helyett hipergeometrikus eloszlást kell használnunk.

Arányos mintavételezés

Az előző fejezetekben azt tételeztük fel, hogy a mintavételezés során véletlenszerűen választunk elemeket. A gyakorlatban nem kell feltétlenül teljesen véletlenszerűen választani az elemeket, fontosabb szempont, hogy a kapott minta *reprezentatív* legyen. Általánosan azt mondhatjuk, hogy egy minta akkor reprezentatív, ha a mintán végzett elemzés ugyanazt az eredményt adja, mintha a teljes adathalmazzal dolgoznánk. Látható, hogy a reprezentativitás, ezen általános meghatározás mellett, alkalmazásfüggő.

Amennyiben az adatbázisbeli objektumok (példányok) osztályokba, előre definiált csoportokba tartoznak, elvárhatjuk, hogy az egyes osztályok ugyanolyan arányban legyenek képviselve a mintában, mint az eredeti adatbázisban. Ilyen esetben beszélünk arányos mintavételezésről (stratified sampling).

3.3.7. Sokdimenziós adatok, dimenziócsökkentés

Amint már volt róla szó, az adatbázisbeli objektumokat attribútumokkal írjuk le. Amikor egy-egy objektumot nagyon sok attribútummal írunk le, sokdimenziós adatokról beszélünk. Az elnevezés onnan ered, hogy az adatbázisbeli objektumokat egy sokdimenziós vektortér pontjainak tekinthetjük, ha az objektumok numerikus attribútumokkal írhatók le.

Ha például egy objektum egy szövegnek felel meg, és minden egyes attribútum egy-egy szó adott szövegbeli előfordulásainak számát adja, több ezer attribútummal kell dolgoznunk, az interneten együtt előforduló szópárok keresésénél 10^9 körüli lehet a dimenziószám.

A dimenzióátok

Elsőre azt gondolnánk, hogy minél nagyobb a dimenzionalitás, minél többet attribútum adott, annál könnyebben dolgunk van egy adatbányászati feladat (osztályozás, klaszterezés, anomálikeresés) megoldásakor, hiszen annál több az információnk egy-egy objektumról. Nem biztos azonban, hogy a sokadik attribútum valóban lényeges többletinformációt hordoz a korábbiakhoz képest, az egyes attribútumok egymással erősen korrelálhatnak. Sőt, az attribútumok egy része teljesen irreleváns lehet a konkrét feladat szempontjából, ezek csak zajt jelentenek.

Az irodalomban curse of dimensionality [Bishop, 2006, Tan és tsa., 2005] – magyarul: dimenzióátok – néven szokták összefoglalni a sokdimenziós adatok bányászata során felmerülő problémákat. Ezek leginkább abból adódnak, hogy a dimeziószám növekedésével az adatok sűrűsége óhatatlanul csökken. Ennek illusztrálásaként képzeljük el, hogy van egy 1000 objektumot tartalmazó adatbá-

zisunk. Ha egy kétdimenziós adatbázisról van szó, és minden dimenziótengely 0 és 10 közötti értékeket vehet fel, egy kétdimenziós egységkockába (egységnyi oldalú négyzetbe) átlagosan $1000/(10 \times 10) = 10$ objektum esik. Ha egy százdimenziós adatbázisról van szó, egy egységkockába már csak $1000/10^{100} = 10^{-97}$ objektum esik átlagosan. A sűrűség ilyen drasztikus csökkenése megzavarhatja a klaszterező algoritmusokat, amelyek tulajdonképpen egy-egy sűrűbb régiót keresnek, mint klasztert. A ritka adatok miatt az osztályozó algoritmusok is alulteljesíthetnek.

A dimenzionalitás átkaként szokták számon tartani a távolságok koncentrációjának jelenségét is. Ennek megértéséhez végezzük el az alábbi kísérletet.

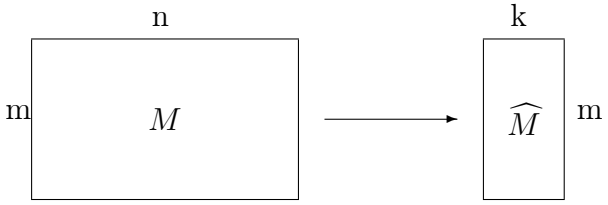
Feladat – Generáljuk véletlenszerűen pontokat egy d dimenziós térben, tekintsük a legközelebbi és legtávolabbi pontok távolságának a különbségét, jelöljük ezt l_d-vel. Ismételjük a kísérletet d növelése mellett, számoljuk ki l_d értékét különböző d-kre. Mivel a dimenziószám növekedése mellett a pontok távolsága természetes módon növekszik – gondoljunk bele, hogy egy háromdimenziós egységkocka és egy százdimenziós egységkocka (leghosszabb) átlója milyen hosszú –, annak érdekében, hogy az előbb számított l_d értékek összemérhetőek legyenek, a kapott értékeket osszuk el az adott dimenziószám melletti legközelebbi két pont távolságával, a kapott értékeket jelöljük l'_d-vel, majd ábrázoljuk diagramon az l'_d értékeket a d dimenziószám függvényében.

Következtetés – Azt tapasztaljuk, hogy a d dimenziószám növekedésével l'_d tart a nullához [Tan és tsa., 2005]. Ez alapján arra következtetünk, hogy a távolságfogalom egyre kevésbé lesz használható nagydimenziós terek esetén.

Dimenziócsökkentő eljárások haszna

A fejezet következő szakaszaiban dimenzió-csökkentésről lesz szó, mely részben megoldást jelent a dimenzionalitás átkaként leírt problémákra. A dimenziócsökkentés során az objektumok sok attribútummal való leírását szeretnénk helyettesíteni kevesebb attribútumot használó leírással. *Hasonlóságtartó* dimenzió-csökkentésről fogunk beszélni, ami azt jelenti, hogy tudunk adni egy olyan hasonlósági definíciót az új leírásban, ami jó becslése az eredeti hasonlóságnak.

Az eredeti adathalmazt az $m \times n$-es M mátrixnak tekintjük, az új leírást pedig az $m \times k$-s $\widehat{M}$ mátrixnak. Ahogy már írtuk, az n nagyon nagy lehet, ami azt jelenti, hogy az adatbázis nem biztos, hogy elfér a memóriában. Ezt a problémát szeretnénk megkerülni azzal, hogy az M-et az $\widehat{M}$ mátrixszal helyettesítjük úgy, hogy $k \ll n$ annyira, hogy $\widehat{M}$ elférjen a memóriában. Ezáltal lehetővé válik olyan algoritmusok futtatása, amelyek feltételezik, hogy az adatokat leíró



3.4. ábra. Dimenziócsökkentés sémája

mátrix a gyors elérésű memóriában található.

A dimenziócsökkentés hasznos lehet akkor is, ha az adatainkat vizualizálni szeretnénk: egy sokdimenziós adatbázist az ábrázoláshoz akár kettő vagy három dimenziósra csökkenthetünk. Még ha nem is élünk a dimenziószám ilyen szélsőséges csökkentésével, ábrázolhatjuk egy viszonylag kis dimenziószámúra csökkentett adatbázis két- vagy háromdimenziós vetületeit.

A következőkben két speciális feladatot tárgyalunk részletesen. Az elsőben az attribútumok valós számok és két objektum különbözőségén az Euklideszi távolságukat értjük. A második esetben az attribútumok csak binárisak lehetnek, és két objektum hasonlóságát a Jaccard-koefficiens (lásd 3.2.1 rész) adja meg.

A dimenziócsökkentés során csak a legfontosabb dimenziókat tartjuk meg, azokat, amelyekről úgy gondoljuk, hogy a legnagyobb szerepet játszanak két objektum hasonlóságának megállapításánál. A többi attribútumot elhagyjuk, ezért a dimenziócsökkentés zajszűrésnek is tekinthető.

Szinguláris felbontás

A szinguláris felbontás[3] az elméleti szempontból egyik legtöbbet vizsgált, klasszikus lineáris algebrai eszközöket használó dimenzió-csökkentési eljárás[4]. Ennek alkalmazása után nyert $\widehat{M}$ mátrix soraiból jól közelíthető az euklideszi távolság, illetve az attribútumok vektoraiból számított skaláris szorzattal mért hasonlóság. Utóbbi megegyezik a koszinusz mértékkel, ha a mátrix sorai normáltak. Ebben a szakaszban néhány jelölés és alapvető fogalom után definiáljuk a szinguláris felbontást, igazoljuk a felbontás létezését, majd megmutatjuk, hogy miként használható a felbontás dimenzió-csökkentésre. Megjegyezzük, hogy a szakasz nem mutat a gyakorlatban numerikus szempontból jól alkalmazható módszert a felbontás kiszámítására. Kisebb adathalmaz esetén általános lineá-

[3] A szinguláris felbontásról szóló rész Fogaras Dániel munkája.

[4] A szinguláris felbontáshoz nagyon hasonló eljárás a főkomponens analizis (angolul: principal component analysis).

ris algebrai programcsomag (Matlab, Octave, Maple) használata javasolt, míg nagyobb adatbázisoknál az adatok sajátosságát kihasználó szinguláris felbontó program (SVDPack[5]) használata ajánlott.

Egy $U \in \mathbb{R}^{n\times n}$ mátrixot *ortogonálisnak* nevezünk, ha oszlopai ortogonális rendszert alkotnak, azaz $U^TU = I_n$, ahol I_n az $n \times n$ méretű egységmátrixot, és U^T az U transzponáltját jelöli. Másképpen mondva U invertálható és U^{-1}-gyel jelölt inverzére teljesül, hogy $U^{-1} = U^T$. Mátrix ortogonalitásának szemléletes tárgyalásához szükségünk lesz a vektorok hosszának általánosítására, a norma fogalmára. A 2-norma általánosítása az $M \in \mathbb{R}^{m\times n}$ mátrixra értelmezett $\|M\|_F$ *Frobenius-norma*, amelynek definíciója $\|M\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} M_{i,j}^2}$.

Egy ortogonális mátrix által reprezentált lineáris transzformáció egy forgatás, mely a vektorok hosszát nem változtatja. Ezen szemlélet alapja, hogy tetszőleges $U \in \mathbb{R}^{n\times n}$ ortogonális mátrix és $x \in \mathbb{R}^n$ vektor esetén

$$\|Ux\|_2 = \|x\|_2$$

teljesül. Az azonosság az alábbi elemi lépésekből következik:

$$\|Ux\|_2^2 = (Ux)^T(Ux) = x^T(U^TU)x = x^Tx = \|x\|_2^2 .$$

Hasonlóan belátható, hogy tetszőleges $X \in \mathbb{R}^{m\times n}$ mátrix esetén és $U \in \mathbb{R}^{m\times m}$ illetve $V \in \mathbb{R}^{n\times n}$ ortogonális mátrixok esetén igaz, hogy

$$\left\|UXV^T\right\|_F = \|X\|_F .$$

A rövid bevezető után rátérünk a szinguláris felbontás definíciójára. Egy nem szükségszerűen négyzetes $M \in \mathbb{R}^{m\times n}$ mátrix *szinguláris érték felbontásán* (singular value decomposition, SVD) az olyan

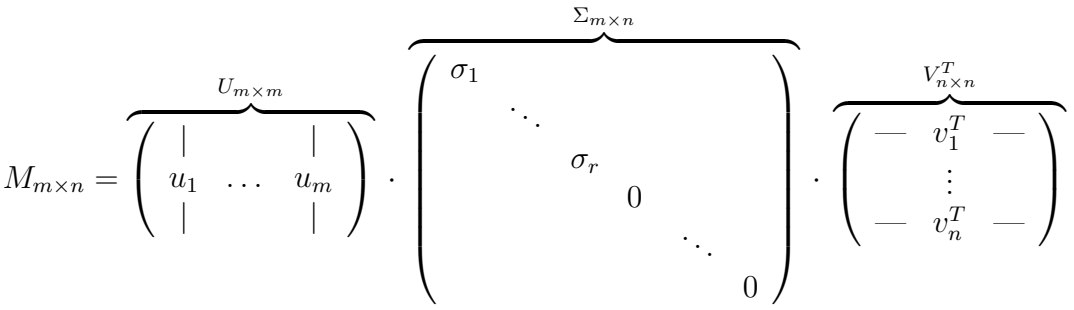
$$M = U\Sigma V^T$$

szorzattá bontást értjük, ahol $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ ortogonális mátrixok, továbbá a Σ mátrix M-mel megegyező méretű és a főátlóban elhelyezkedő $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ pozitív számokat csupa 0 követi és a többi elem szintén 0. A σ_i számokat *szinguláris értékeknek* nevezzük, és a $\sigma_i = 0$ választással terjesztjük ki az $i > r$ esetre. A felbontásból látható, hogy $\operatorname{rang}(M) = \operatorname{rang}(\Sigma) = r$. Az U és a V oszlopait *bal-, illetve jobboldali szinguláris vektoroknak* mondjuk. A jelölések áttekintése a 3.5. ábrán látható.

3.3.1. Tétel *Tetszőleges $M \in \mathbb{R}^{m\times n}$ mátrixnak létezik szinguláris érték felbontása, azaz léteznek $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$ ortogonális mátrixok, melyekkel*

$$M = U\Sigma V^T,$$

[5]http://www.netlib.org/svdpack/



3.5. ábra. A szinguláris felbontás sematikus vázlata.

ahol

$$\Sigma \in \mathbb{R}^{m\times n}, \qquad\qquad \Sigma = \begin{pmatrix} \Sigma_+ & 0 \\ 0 & 0 \end{pmatrix},$$

továbbá Σ_+ *egy* $r \times r$ *méretű diagonális mátrix, amelynek főátlójában a* $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$ *számok helyezkednek el sorrendben.*

Bizonyítás. Az M^TM mátrix szimmetrikus, ezért ortogonális transzformációval diagonalizálható és sajátértékei valósak. Továbbá pozitív szemidefinit, mert tetszőleges $x \in \mathbb{R}^{n\times n}$ vektor esetén $x^TM^TMx = (Mx)^T(Mx) = \|Mx\|_2^2 \geq 0$, ezért a sajátértékek nem negatívak. A sajátértékek legyenek $\sigma_1^2 \geq \sigma_2^2 \geq \cdots \geq \sigma_r^2 > 0$. Az ezekhez tartozó sajátvektorokból alkotott ortogonális mátrixot jelölje V, ekkor

$$V^TM^TMV = \begin{pmatrix} \Sigma_+^2 & 0 \\ 0 & 0 \end{pmatrix}.$$

A mátrixot két részre osztva $V = (V_r\, V_2)$, ahol $V_r \in \mathbb{R}^{n\times r}$ a pozitív sajátértékhez tartozó sajátvektorokat tartalmazza. Vagyis

$$V_r^TM^TMV_r = \Sigma_+^2.$$

Vezessük be az

$$U_r = MV_r\Sigma_+^{-1}$$

jelölést, ekkor

$$M = U_r\Sigma_+V_r^T.$$

Az U_r vektorai ortogonális vektorrendszert alkotnak, ezt tetszőlegesen kiegészítve $U = (U_r\, U_2)$ ortogonális mátrixszá

$$M = U\begin{pmatrix} \Sigma_+ & 0 \\ 0 & 0 \end{pmatrix}V^T.$$

Most megmutatjuk, hogy szinguláris felbontás segítségével hogyan lehet dimenzió-csökkentést végrehajtani. Emlékeztetünk rá, hogy az M mátrix n-dimenziós sorvektorai objektumokat jellemeznek. Dimenzió-csökkentéskor az n attribútumot szeretnénk $k < n$ dimenziójú vektorokkal jellemezni úgy, hogy közben az objektumok euklideszi távolsága vagy skaláris szorzattal mért hasonlósága csak kis mértékben változzon. A mátrixszorzás elemi tulajdonsága, hogy a szinguláris felbontás az alábbi formában is írható.

$$M = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T,$$

ahol $u_i v_i^T$ a bal- illetve a jobboldali szinguláris vektorokból képzett diádszorzat, azaz egy oszlop- és egy sorvektor szorzataként felírt $m \times n$ méretű 1-rangú mátrix. Látható, hogy az $u_i v_i^T$ diádok monoton csökkenő σ_i súllyal szerepelnek az összegben. Innen adódik az ötlet, hogy $k < r$ esetén csak az első k legnagyobb súlyú diád összegével közelítsük az M mátrixot. Azaz

$$M_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T = U_k \Sigma_k V_k^T,$$

ahol $U_k = (u_1\, u_2\, \ldots\, u_k)$ és $V_k = (v_1\, v_2\, \ldots\, v_k)$, valamit Σ_k egy $k \times k$ méretű diagonális mátrix, melynek főátlójában a $\sigma_1, \sigma_2, \ldots, \sigma_k$ értékek vannak. Könnyen látható, hogy M_k sorai egy k-dimenziós altérben helyezkednek el, hiszen $\operatorname{rang}(M_k) = \operatorname{rang}(\Sigma_k) = k$. Sokkal mélyebb eredmény a következő, M_k hibájára vontakozó tétel, melynek bizonyítását mellőzzük.

3.3.2. Tétel *Legyen M egy legalább k rangú mátrix és legyen M_k a fenti módon számított közelítése. Ha a közelítés hibáját Frobenius-normával mérjük, akkor a k-rangú mátrixok közül az M_k mátrix a lehető legjobban közelíti M-et, azaz*

$$\|M - M_k\|_F = \min_{N:\, \operatorname{rang}(N)=k} \|M - N\|_F .$$

Továbbá a közelítés hibája a σ_i szinguláris értékekkel kifejezhető:

$$\|M - M_k\|_F = \sqrt{\sum_{i=k+1}^{r} \sigma_i^2}.$$

A közelítés relatív pontosságán a hibanégyzet egytől vett különbségét értjük, azaz

$$\frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2}. \tag{3.2}$$

Az M_k mátrix sorai az M-éhez hasonlóan n méretűek, de most már egy k-dimenziós altérnek az elemei. Ennek az altérnek egy bázisát alkotják a V_k^T sorai, és az

$$M' = U_k \Sigma_k$$

mátrix k-dimenziós sorvektorai e bázisban fejezik ki az M_k sorait. Tehát a dimenzió-csökkentés eredménye, hogy az M mátrix n-dimenziós sorait a vetítés után az M' mátrix k-dimenziós soraival közelítjük. A V_k^T sorainak ortogonalitásából könnyen belátható, hogy az M_k, illetve az M' soraiból számított euklideszi távolságok és skaláris szorzatok is megegyeznek. Tehát a közelítés alatt torzítás kizárólag az M-ből M_k-ba történő vetítés során történik, melynek mértéke a 3.3.2.. tétel alapján felülről becsülhető.

Multidimensional Scaling és ISOMAP

Egy további dimenziócsökkentő eljárás a multidimensional scaling (MDS) [Borg és Groenen, 2005]. Az MDS abból indul ki, hogy az objektumok közti távolságok egy távolságmátrix-szal adottak. A korábbiakhoz hasonlóan az a cél, hogy megtaláljuk az objektumok egy olyan, kisebb dimenziós reprezentációját, amelynél a páronkénti távolságok minél jobban közelítik az eredeti páronkénti távolságokat. Ennek érdekében az MDS egy célfüggvényt definiál, melyet optimalizál. Ebből adódik az MDS egyik legnagyobb előnye: nem csak olyan esetben használható, amikor az eredeti adat egy sokdimenziós térben adott, hanem bármilyen olyan esetben, amikor távolságot tudunk definiálni az eredeti adatbázis objektumai között. Ilyen lehet például, ha az objektumaink különböző hosszúságú idősorok vagy karakterláncok (sztring-ek).

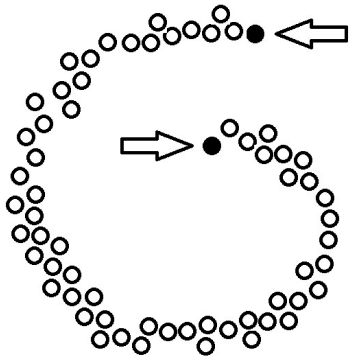
Jelöljük $d_{i,j}$-vel az i-dik és j-dik objektumok eredeti távolságát, és $d'_{i,j}$-vel az i-dik és j-dik objektum leképezés utáni távolságát. Az MDS ekkor az alábbi módon definiált *stressz*t, mint célfüggvényt minimalizálja:

$$\text{stressz} = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{n}(d'_{i,j} - d_{i,j})^2}{\sum_{i=1}^{n}\sum_{j=1}^{n} d_{i,j}^2}},$$

ahol n az adatbázisbeli objektumok száma.

Az MDS algoritmust nem tárgyaljuk részletesen, szemléltetésként csak annyit mondunk, hogy az algoritmus kezdetben valahogyan elhelyezi az objektumoknak megfelelő pontokat a kis dimenziószámú térben, és ezeket a pontokat mozgatja úgy, hogy közben a fenti stressz értéke csökkenjen.

Az ISOMAP algoritmus abban különbözik az MDS-től, hogy mit tekint az objektumok (pontok) $d_{i,j}$ távolságának a stressz számításakor. Adott az objektumok valamely távolságfüggvény szerinti $d^0_{i,j}$ távolsága, például Euklideszi



3.6. ábra. Az ISOMAP figyelembe veszik az adatok strukturáját: a példában a távolságokat a csigavonal mentén számítja, a két jelölt pontot tekinti legtávolabbinak, holott az Euklideszi távolsága más pontpároknak nagyobb.

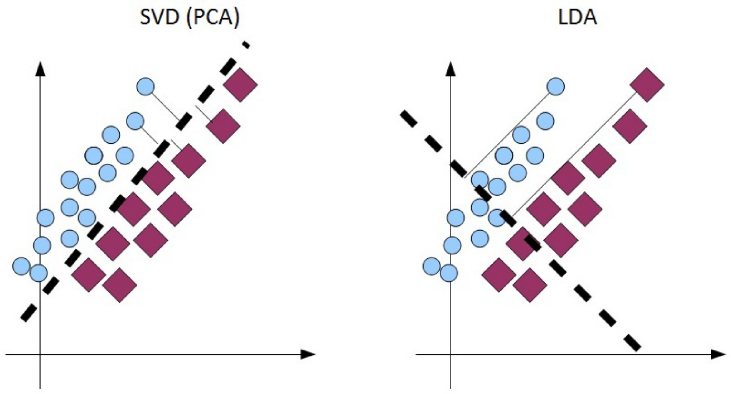
távolságuk. Ezen távolságok alapján az ISOMAP algortimus felépít egy szomszédossági gráfot: minden objektumot összeköt a k darab legközelebbi szomszédjával. Ezt követően kiszámolja az objektumok közti legközelebbi szomszéd gráfbeli legrövidebb utak hosszát: a $d_{i,j}$ távolság tehát az i és j objektumok közti legközelebbi szomszéd gráfbeli legrövidebb út hossza lesz.

Vegyük észre, hogy i és j pontok (objektumok) közti legközelebbi szomszéd gráfbeli legrövidebb út hossza nagyban különbözhet az i és j pontok Euklideszi távolságától: ha például pontjaink egy csigavonal (spirál) mentén helyezkednek el, a legközelebbi szomszéd gráfbeli legrövidebb út hossza, nagyjából, a csigavonal mentén történő távolságot fogja jelenteni. Ilyen értelemben az ISOMAP figyelembe veszi az adatok strukturáját a (kisebb dimenziószámú) térbe történő leképezés során (3.6. ábra).

Felügyelt dimenziószámcsökkentés, LDA

Ha címkézett adatokkal dolgozunk, azaz az adatbázisbeli objektumok különböző osztályokba sorolhatóak, és legalább az objektumok egy részéről tudjuk, hogy azok mely osztályba tartoznak, a korábbiakban bemutatottak helyett választhatunk olyan dimenziócsökkentő eljárást is, amely kitüntetett figyelmet szentel az objektumok osztályattribútumának, az osztálycímkének. Ilyen eljárások egyike az LDA (Linear Discriminant Analysis).

Az SVD (PCA) és LDA közti különbséget a 3.7. ábrán szemléltetjük. A példában egy kétdimenziós adatot csökkentünk egydimenziósra. A bal oldali ábrán az SVD-vel azon irányt találjuk meg, amely mentén legnagyobb az objektumok szórása. Ezt szaggatott vonal jelöli. Az SVD-t úgy képzelhetjük



3.7. ábra. Az SVD (PCA) és LDA dimenziócsökkentő eljárások.

el, hogy erre a vonalra vetíti az adatokat. Az LDA ezzel szemben figyelembe veszi az osztálycímkéket és olyan irányt keres, amelyre vetítve az osztályok minél jobban elkülönülnek. Az LDA-val talált irányt az ábra jobboldali részén látható szaggatott vonal mutatja. Az LDA-t úgy képzelhetjük el, hogy erre a vonalra vetíti az objektumokat.

Látható, hogy ha az LDA-val egyetlen dimenziósra csökkentünk egy adatbázist, és meghatározunk egy küszöbszámot, az LDA-t osztályozási feladatok megoldásához használhatjuk.

Minhash alapú lenyomat

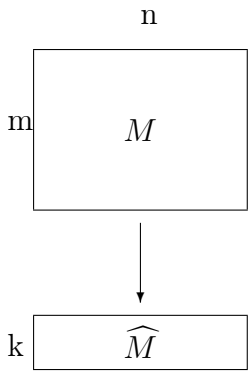
Eddig azt feltételeztük, hogy az adattábla egyes sorai felenek meg az adatbázisbeli objektumoknak és a táblázat oszlopai az egyes attribútumoknak. A sorokat és oszlopokat nyilván felcserélhetjük. Ezzel fogunk élünk a Minhash [Datar és tsa., 2004] eljárás bemutatása során: a Minhash konvencióinak megfelelően most azt tételezzük fel, hogy az sorok felelnek meg az attribútumoknak, az oszlopok pedig az egyes példányoknak.

A következőkben tehát az adathalmaz sok objektumot és még több attribútumot tartalmaz. Célunk az attribútumok számának csökkentése. A feladatot a következő ábra szemlélteti.

Az M mátrix bináris és két oszlop (vektor) hasonlóságát a Jaccard-koefficiens adja meg:

$$d_{i,j} = \frac{||m^i \cap m^j||}{||m^i \cup m^j||} = \frac{(m^i)^T m^j}{||m^i||^2 + ||m^j||^2 - (m^i)^T m^j},$$

hiszen az $m^i(m^j)^T$ bináris vektorok esetében az azonos pozíciókban lévő 1-esek számát adja meg, $||m^i||^2$ pedig a vektor egyeseinek számát. Feltételezzük,



3.8. ábra. A Mishash szemléltetése

hogy a bináris vektorok ritkák azaz, ha r-el jelöljük a sorokban az 1-esek átlagos számát, akkor $r \ll n$.

Az $\widehat{M}$ mátrixot az M lenyomatmátrixának fogjuk hívni. A lenyomatmátrixnak nem kell binárisnak lennie, de azt természetesen most is elvárjuk, hogy a memóriaigénye jóval kevesebb legyen, mint az M memóriaigénye. További kikötés, hogy az adatok sorfolytonosan vannak tárolva, azaz először kiolvashatjuk az első sort, majd a másodikat, és így tovább.

Ez a helyzet áll fel hasonló weboldalak kiszűrésénél, koppintások, kalózmásolatok felderítésénél, hasonló tulajdonságú felhasználók keresésénél stb. Továbbá ezt a módszert alkalmazhatjuk, amikor hasonló eladású termékpárokat keresünk. Amennyiben a termékeket kis tételben értékesítik, akkor az asszociációs szabályokat kinyerő technikák (lásd 5.2. fejezet) nem alkalmazhatóak.

Gondolkozzunk el azon, hogy működik-e az alábbi algoritmus. Válasszunk ki néhány sort véletlenszerűen és tekintsük ezeket lenyomatoknak. Két lenyomat hasonlóságának várható értéke meg fog egyezni az oszlopaik hasonlóságával. Ez alapján azt mondhatnánk, hogy a sorok egy véletlenszerűen választott halmaza jó lenyomat.

A fentiek ellenére ez az egyszerű módszer nagyon rossz eredményt adna. Ennek oka az, hogy a mátrixunk nagyon ritka ($r \ll n$), tehát egy oszlopban a legtöbb elem 0, így nagy valószínűséggel a legtöbb lenyomat is csupa 0 elemből állna.

A minhash alapú lenyomat egy elemét a következőképpen állítjuk elő. Véletlenszerűen permutáljuk meg a sorokat, majd válasszuk az j-edik oszlopok hash értékének (h) azt a legkisebb sorindexet, ahol 1-es szerepel a j-edik oszlopban. A véletlen permutáció természetesen csak elméleti megközelítés, diszken található nagy adatbázis esetén túl lassú művelet. Ehelyett sorsoljunk ki minden

sorhoz egy véletlen hash értéket. Amennyiben feltehetjük, hogy a mátrix sorainak száma 2^{16}-nál kisebb, akkor a születésnapi paradoxon[6] alapján válasszunk 32 bit szélességű egyenletes eloszlású véletlen számot. Az algoritmus tényleges implementálása során tehát egyesével olvassuk a sorokat, véletlen számot generálunk, és minden oszlopnak folyamatosan frissítjük azt a változóját, ami megadja a legkisebb, 1-est tartalmazó sorindexet.

Mivel egy lenyomatnak k darab eleme van, ezért minden oszlophoz k darab véletlen számot állítunk elő, és k darab hash értéket tároló változót tartunk karban. Vegyük észre, hogy a lenyomat előállításhoz egyszer megyünk végig a mátrixon.

Két lenyomat hasonlóságát a páronként egyező lenyomatok számának k-hoz vett aránya adja meg, azaz

$$\widehat{d}_{ij} = \frac{|\{\ell : \widehat{M}_{i,\ell} = \widehat{M}_{j,\ell}\}|}{k},$$

ahol $\widehat{M}_{i,\ell}$ az $\widehat{M}$ mátrix i-edik oszlopának ℓ-edik elemét jelöli.

Be fogjuk bizonyítani, hogy $\widehat{d}_{ij}$ jó becslése d_{ij}-nek abban az értelemben, hogy ha i és j oszlopok nagyon hasonlók, akkor azok lenyomatai is nagy valószínűséggel hasonlók. Ehhez a következő észrevételt használjuk fel.

Észrevétel. Tetszőleges (i, j) oszloppárra igaz, hogy

$$\mathbb{P}[\widehat{M}_{i,\ell} = \widehat{M}_{j,\ell}] = d_{ij}.$$

Bizonyítás. Csak akkor lehet a két lenyomat azonos, ha a legalább az egyik oszlopban az 1-est tartalmazó indexek közül olyan sor kapta a legkisebb véletlen számot, amelynél mindkét oszlopban 1-es szerepel. Ennek valószínűsége éppen d_{ij}, amennyiben a permutáció egyenletesen szórja szét az egyeseket.

És most a hasonlóság megőrzésével kapcsolatos állítás:

3.3.3. Tétel *Legyenek $0 < \delta < 1$, és $\epsilon > 0$ valós számok. Amennyiben $k > -\frac{\ln \delta/2}{2\epsilon^2}$, akkor δ-nál kisebb a valószínűsége annak, hogy a lenyomat és az eredeti hasonlóság különbsége ϵ-nál nagyobb.*

[6] A születésnap paradoxonnal kapcsolatos kérdés a következő: „Mekkora a valószínűsége annak az eseménynek, hogy emberek egy véletlenszerűen választott r fős csoportjában van legalább két személy, akik egy napon ünneplik a születésnapjukat?”. Elemi kombinatorikus úton a válasz meghatározható: $p_r = 1 - \frac{\binom{365}{r} \cdot r!}{365^r} \approx 1 - \exp \frac{-r^2}{3 \cdot 365}$. A feladat következménye az az állítás, miszerint 2^n elemnek 2^{2n} elemű halmazból kell egyenletes eloszlás szerint véletlenszerűen egyesével kulcsot sorsolni, hogy kicsi ($\exp(-3) < 0.05$) legyen annak valószínűsége, hogy két elem ugyanazt a kulcsot kapja.

Bizonyítás. Tekintsük az i, j oszlopokat. Definiáljuk X_l valószínűségi változót, ami 1 $\widehat{M}_{i,\ell} = \widehat{M}_{j,\ell}$ esetén, különben 0. Legyen $Y = X_1 + \ldots + X_k$.

X_l binomiális eloszlású és az előzőekben kimondott észrevétel miatt $E[X_l] = p = \mathbb{P}(\widehat{M}_{i,\ell} = \widehat{M}_{j,\ell}) = d_{ij}$. A lenyomatok hasonlóságának definíciójából adódik, hogy $\widehat{d}_{ij} = \frac{Y}{k}$. Írjuk fel Y-re 2.2.3 -es tételét:

$$\mathbb{P}\big(|Y - E[Y]| > k\epsilon\big) \leq 2e^{-2\epsilon^2 k},$$

amiből adódik, hogy

$$\mathbb{P}\big(|\widehat{d}_{ij} - d_{ij}| > \epsilon\big) \leq 2e^{-2\epsilon^2 k}.$$

További dimenziócsökkentő eljárások

Számos további dimenziószámcsökkentő eljárás létezik. Használhatunk például neurális hálókat[7] is dimenziószámcsökkentésre: egy feed-forward típusú, egy rejtett réteggel rendelkező neurális háló rejtett rétegében lévő neuronjainak aktiválását tekinthetjük a háló bemenetén aktuálisan lévő objektum reprezentációjának egy másik (adott esetben kisebb dimenziószámú) térben.

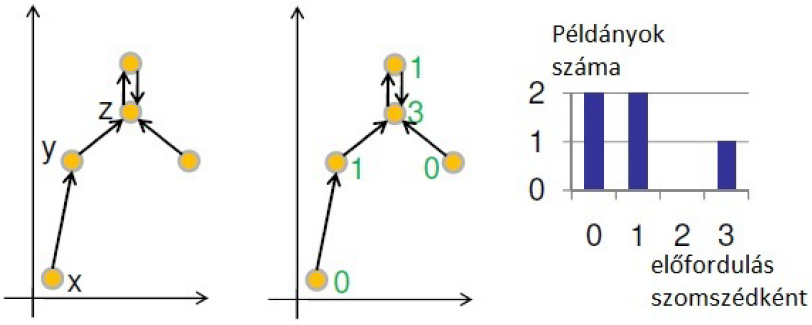
Egy szupport vektor gép által talált elválasztó hipersíkra vetíteve az adatbázisbeli objektumokat, egyel csökkenthetjük azok dimenzionalitását. Ezt iteratíve alkalmazva dimenziócsökkentő eljárást kapunk [Pitelis és Tefas, 2012].

A csomósodás jelensége

Sokdimenziós térbeli adatokkal kapcsolatos egyik újabb eredmény a csomósodás (presence of hubs) jelenségének megfigyelése [Symeonidis és tsa., 2010, Radovanović, 2011]. A következőkben ezt részletezzük.

Vegyük észre, hogy a legközelebbi szonszéd reláció nem feltétlenül szimmetrikus: ha egy x objektum legközelebbi szomszédja y, abból nem következik, hogy y legközelebbi szomszédja x (lásd 3.9. ábra bal oldalán). Ezek szerint megszámolhatjuk, hogy egy-egy objektum hányszor fordul elő más objektumok legközelebbi szomszédjaként (lásd 3.9. ábra középső részét). Egy x objektum előfordulási számát, azt, hogy hányszor fordul elő más objektumok legközelebbi szomszédjaként, $N(x)$-szel jelöljük. Következő lépésként felrajzolhatjuk $N(x)$ eloszlását, azt, hogy hány olyan pont van, amely 0, 1, 2... további pont legközelebbi szomszédjaként fordul elő: a 3.9. ábra jobb oldalán lévő hisztogram utolsó oszlopa például azt mutatja, hogy ebben a példában egy olyan pont van, amelyik három másiknak a legközelebbi szomszédja.

[7] A neurális hálókat és szupport vektor gépeket az osztályozó algoritmusok közt tárgyaljuk.



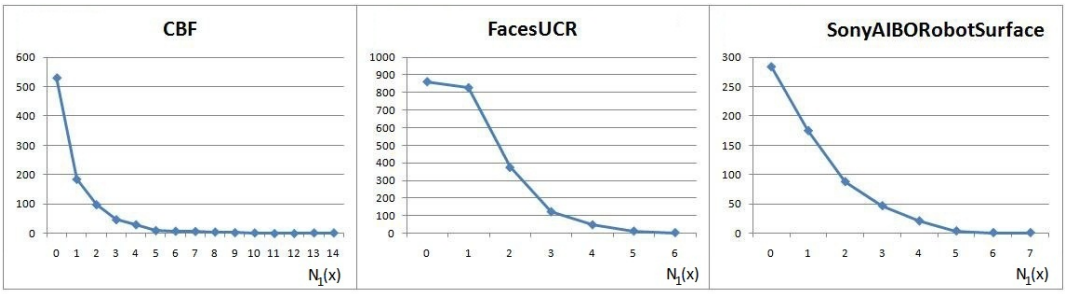
3.9. ábra. A legközelebbi szomszéd reláció asszimetrikus (bal oldalt), megvizsgálhatjuk, hogy egy-egy objektum hányszor fordul elő más objektumok legközelebbi szomszédjaként (középen), a legközelebbi szomszédkénti előfordulások számának eloszlása (jobb oldalt).

Az egyszerűség kedvéért a 3.9. ábrán látható példában csak egyetlen legközelebbi szomszédot vettünk figyelembe. Világos, hogy tekinthetjük minden objektum k darab legközelebbi szomszédját, az objektumhoz leginkább hasonló k darab további objektumot, és az előbbi eljárást követve definiálhatjuk $N_k(x)$-t, azt, hogy egy x objektum hány további objektum k legközelebbi szomszédja közt fordul elő, majd ábrázolhatjuk $N_k(x)$, azaz a k-legközelebbi szomszédkénti előfordulás eloszlását.

Ha az osztálycímkék is adottak, beszélhetünk jó és rossz szomszédokról: x objektum *jó* (rossz) k-legközelebbi szomszédainak egyike az y objektum, amennyiben: (i) az y objektum x k legközelebbi szomszédainak egyike, valamint (ii) x és y osztálycímkéi *megegyeznek* (különbözőek). $GN_k(x)$-szel illetve $BN_k(x)$-szel jelöljük, hogy egy x objektum hány további objektum jó illetve rossz k legközelebbi szomszédjaként fordul elő. Nyilván: $N_k(x) = GN_k(x) + BN_k(x)$.

A kérdés az, hogy valós adatok mellett hogy néz ki $N_k(x)$, $GN_k(x)$ és $BN_k(x)$ eloszlása? $N_1(x)$ eloszlását mutatja a 3.10 ábra néhány, valós adatot tartalmazó adatbázis esetén [Buza, 2011a]. $GN_k(x)$ és $BN_k(x)$ eloszlásai is hasonlóak. Látható, hogy az eloszlás meglehetősen asszimetrikus, más szóval ferde: sok olyan objektum akad, amely 0 vagy 1 további objektum legközelebbi szomszédjaként fordul elő, míg akad néhány objektum, amely feltűnően sok további objektum legközelebbi szomszédja. Ez utóbbi objektumokat nevezzük hub-oknak vagy csomópontoknak.

A csomósodás jelenségét a 3.1. fejezetben bemutatott ferdeség segítségével szokták számszerűsíteni. Radovanovic, Nanopoulos és Ivanovic kutatásai azt mutatják, hogy a legtöbb esetben a dimenziószám növekedésével együtt $N_k(x)$,



3.10. ábra. $N_1(x)$ eloszlása néhány valós adatokat tartalmazó adatbázis esetén. A vízszintes tengely $N_1(x)$ értékeit mutatja, a függőleges tengely pedig az objektumok számát.

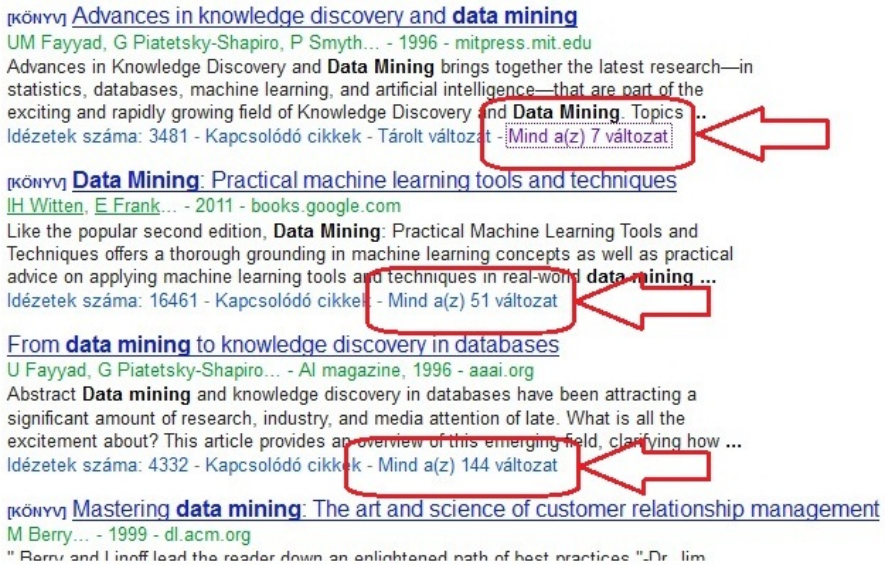
$GN_k(x)$ és $BN_k(x)$ ferdesége is növekszik, ezért a csomósodás jelenségét figyelembe vevő adatbányászati eljárásokat dolgoztak ki sokdimenziós adatok kezelésére, melyek számos esetben felülmúlják a korábbi algoritmusokat. Ezekre az eljárásokra a megfelelő adatbányászati feladatoknál hivatkozunk.

3.3.8. Duplikátumok kiszűrése

Az egyik leggyakoribb előfeldolgozási, adattisztítási lépés a duplikátumok kiszűrése. Duplikátumok több okból keletkezhetnek: tévedésből ugyanazon ügyfelet többször visszük fel az adatbázisba, ugyanazon objektum több adatforrásban is jelen van, és ezért az adatok integrációja során többszörösen is bekerül az integrált adatbázisba. Duplikátumok kiszűrése során az a célunk, hogy a valós világ egy objektuma csak egyszer szerepeljen az adatbázisban.

A legegyszerűbb esetben pontosan egyező duplikátumokat keresünk, azaz olyan objektumokat, amelyek minden egyes attribútuma pontosan megegyezik. Ekkor egyszerű annak eldöntése, hogy két objektum duplikátum-e vagy sem, ugyanakkor egy naív implementáció, amely során minden egyes objektumpárt vizsgálunk, túlságosan sok ideig tarthat nagy adatbázisok esetén, hiszen az objektumpárok száma négyzetes az objektumok számában. Ezért a naív implementáció helyett szokás az objektumok sorbarendezése. Az objektumok darabszámát n-nel jelölve, a sorbarendezés számítási költsége $\mathcal{O}(n \log n)$, amely jóval kisebb $\mathcal{O}(n^2)$-nél. A sorbarendezés után a duplikátumok egymás mellé kerülnek, elég egyszer végigolvasni az adatbázist a sorrendezésnek megfelelően a duplikátumok megtalálásához.

A pontosan egyező duplikátumok keresésénél sokkal nagyobb kihívást jelent a közelítőleg egyező duplikátumok keresése: olyan *hasonló* bejegyzések azonosí-



3.11. ábra. A Google Scholar a duplikátumok figyelembe vételével listázza a találatokat.

tása, melyek egyazon valós objektumnak felelnek meg. Ilyen eset például akkor fordul elő, amikor egy ügyfelet többször vittek fel a rendszerbe, de nevét és címét eltérő írásmóddal rögzítették. Hasonló kihívással állunk szemben több, különböző helyről, például különböző webes áruházból származó termékleírások integrációja során is. Keresőrendszerek szintén használnak duplikátumkeresést (3.11. ábra). A következőkben a közelítőleg egyező duplikátumok keresésével foglalkozunk.

Sorted neighborhood technika

Feltételezzük, hogy létezik egy szabályrendszer, amely alapján el tudjuk dönteni két adatbázisbeli objektumról, hogy azok duplikátumok-e vagy sem. Ez természetesen alkalmazásfüggő, de szakértők segítségével számos alkalmazásban lehetséges elég jó szabályokat definiálni.

A szabályrendszert minden egyes párra alkalmazni túlságosan számításigényes, hiszen a párok száma négyzetes az adatbázisbeli objektumok számában. Ezen probléma kikerülésére használhatjuk az ún. *sorted neighborhood* technikát. Ennek során minden egyes adatbázisbeli objektumhoz egy-egy kulcsértéket rendelünk. A kulcsérték például egy néhány karakter hosszú karakterlánc lehet. Egy objektumhoz rendelt kulcsértéket az attribútumok alapján határozzák meg. A kulcsérték képzésének szabályait úgy választjuk meg, hogy

a közel azonos objektumok – amelyek potenciálisan duplikátumok – hasonló kulcsértékeket kapjanak. Ezt követően rendezzük az adatbázist a kulcsérték szerint. Ha jól választottuk meg a kulcsképzés szabályait, a duplikátumok a rendezés hatására egymás közelébe kerültek. Ezt követően a rendezés szerinti sorrendben végigolvassuk az adatbázist, és az egymástól legfeljebb w pozíciónyi távolságban lévő objektumokra alkalmazzuk a korábban definiált szabályokat és eldöntjük róluk, hogy duplikátumok-e, így jóval kevesebb objektum-párt vizsgálunk, mintha minden párt vizsgálnánk.

Az eljárás sikere nagyban múlik azon, hogyan választottuk meg a kulcsképzési eljárást. Lehet, hogy a duplikátumok egy részének megtalálásához egy adott kulcsképzési eljárás ideális, míg más típusú duplikátumokat más módon képzett kulccsal vagyunk képesek megtalálni. Ezért érdemes többféle kulcsképzési ejárással végrehajtani a sorted neighborhood technika szerinti duplikátumkeresést, és a kapott eredményeket egyesíteni.

Regressziós és osztályozó modellek a duplikátumok szűrésére

Eddig azt feltételeztük, hogy létezik egy szabályrendszer, amely alapján el tudjuk dönteni két adatbázisbeli objektumról, hogy azok duplikátumok-e vagy sem. Ilyen szabályok azonban nem feltétlenül adottak, és nem mindig triviális, hogyan definiáljunk ilyen szabályokat.

Ha a felhasználó néhány száz objektumpárról megadja, hogy azok duplikátumok-e vagy sem, akkor a használhatjuk a következő fejezetben bemutatandó oszályozó és regressziós algoritmusokat annak eldöntésére, hogy a két objektum duplikátum-e vagy sem. Az osztályozó vagy regressziós algoritmus számára egy objektum ilyenkor két eredeti objektumból képzett *párnak* felel meg. Két osztály van: egyik osztály annak felel meg, hogy a párt alkotó eredeti objektumok duplikátumok, a másik pedig annak, hogy a párt alkotó eredeti objektumok nem duplikátumok [Christen, 2008].

Ezt az eljárást kombinálhatjuk a sorted neighborhood technikával. Érdekességként jegyezzük meg, hogy az osztályozó és regressziós modellek segítségével végzett duplikátumkereséshez nagyon hasonló technikákat használhatunk weblapok automatikus csoportosítására [Romano és tsa., 2009] és fényképek események szerinti automatikus csoportosítására [Reuter és tsa., 2011].

3.3.9. Aggregáció

Aggregációra egyrészt szükség lehet az adatok méretének csökkentésére, hogy az adathalmaz beférjen a memóriába, másrészt pedig azért, mert a mintázatok nem minden szinten ismerhetők fel: lehet, hogy elemi szinten nem áll rendelkezésre elég információ vagy túl nagy az elemi szintű mérések szórása. Ha

például egy bolthálózat eladásait kívánjuk elemezni, előfeldolgozási lépésként célszerű lehet az eladásokat ügyfelenként összesíteni, vagy – a konkrét elemzési feladattól függően – akár magasabb szinten is összesíthetjük az eladásokat, például körzetenként, napok vagy hetek szintjén, stb. Hasonlóképpen, ha a csapadékmennyiséget egy négyzetkilométeres területeken mérjük, ez túl részletes lehet, ezért összevonhatjuk a szomszédos cellákat és nagyobb egységeket alakíthatunk ki.

Az aggregációval vigyáznunk kell: ha rosszul választjuk meg, hogy milyen szinten aggregálunk vagy túl sok elemi megfigyelést vonunk össze, lehet, hogy nem leszünk képesek felismerni az adatbázisbeli mintázatokat.

3.3.10. Monotonizáció

Osztályozási feladatok esetében az osztályattribútum (más néven osztálycímke) gyakran ordinális. Ha például egy banki vagy biztosítási környezetben osztályozunk ügyfeleket aszerint, hogy mennyire kockázatosak, és tíz osztályt hozunk létre, az első osztály jelölheti a legkevésbé kockázatos ügyfelek csoportját, a tizes pedig a legkockázatosabb ügyfeleket. Termékek ajánlásakor az ötös osztály felelhet meg azon termékeknek, amelyeket egy adott felhasználó legnagyobb eséllyel vásárol meg, a négyes osztály termékeit ugyanezen felhasználó viszonylag jó esélyel vásárolja meg, de kisebb eséllyel, mint az ötös osztály termékeit, az 1-es osztályba pedig azokat a termékeket sorolhatjuk, amelyeket az adott felhasználó szinte biztosan nem vásárol meg. Ilyen esetekben, az általános célú osztályozási algoritmusok mellett, speciális, úgynevezett *monoton klasszifikációs* algoritmusokat is használhatunk [Horváth, 2006]. A monoton klasszifikációs algoritmusok sikeres alkalmazásához szükséges, hogy az adatbázisbeli objektumokra (példányokra) teljesüljön az alábbi monotonitás: tegyük fel, hogy az objektumainkat az A_1, A_2, ..., A_k attribútumokkal írjuk le, jelöljük a_1^x-szel, a_2^x-szel, ... a_k^x-szel illetve a_1^y-nal, a_2^y-nal, ... a_k^y-nal az attribútumok értékeit adott x és y objektumok esetében, az x és y objektumok ordinális osztályát c_x-szel és c_y-nal jelöljük, ekkor:

$$a_x^1 \leq a_y^1 \wedge a_x^2 \leq a_y^2 \wedge ... \Rightarrow c_x \leq c_y.$$

Azt tételezzük tehát fel, hogy mind az osztályok, mind az attribútum-értékek között van valamilyen preferencia: ha egy y objektum egyetlen attribútumértéke sem kevésbé kedvező, mint az x-é, akkor y osztálya sem lehet kevésbé kedvező, mint x-é.

A következőkben a monotonitást illetve egy az adatbázis monotonná alakítását szemléltetjük egy példán keresztül. Tegyük fel, hogy egy egészségügyi alkalmazásban tároljuk az emberek testsúlyát, testmagasságát, azt, hogy saját

bevallásuk szerint mennyi cigarettát szívnak és mennyi alkoholt fogyasztanak. Az embereket kockázati osztályokba soroljuk aszerint, hogy életmódjuk alapján mennyire várható, hogy egészségügyi problémákkal néznek majd szembe. Minél többet dohányzik valaki, annál nagyobb az egészségügyi problémák esélye, ezért tehát a dohányzást leíró attribútumra teljesül a monotonitás. Feltehetjük azt is, hogy az alkoholfogyasztás is monoton. Ugyanez azonban nem igaz a testsúlyra: túl kicsi és túl nagy testsúly egyaránt kockázatot jelent. A testsúlyból és a testmagasságból kiszámolhatjuk a 3.3.2. fejezetben már említett testtömeg indexet, és megnézhetjük az ideális testtömegindextől való abszolút eltérést. Az ideális testtömegindextől való abszolút eltérés már monoton lesz: minél nagyobb az eltérés, annál kockázatosabb az egyén. Ha tehát a testsúly és testmagasság attribútumokat lecseréljük a testtömegindex ideálistól való eltérésével, akkor monotonizáltuk az adatbázist.

Amint a példa mutatja, egyes esetekben monotonizálhatjuk adatainkat szakterületi háttértudás alapján. Ennek hiányában azonban szükség van automatikus monotonizációs eljárásokra, melyekkel ebben a jegyzetben nem foglalkozunk részletesebben. Az érdeklődőknek ajánljuk Horváth és társai (2011) cikkét, amelyben egy automatikus monotonizációs eljárásra láthatunk példát.

4. fejezet

Osztályozás és regresszió

.

Ismeretlen, attribútumok értékének előrejelzése más megfigyelhető attribútumok ismeretében régóta aktív kutatás tárgya. A kérdés gyakorlati jelentőségét nehéz lenne túlértékelni. Ebben a fejezetben vázlatosan ismertetjük, hogy miként alkalmazhatók a statisztika és a gépi tanulás területén kifejlesztett módszerek az adatbányászatban.

Azt a változót, amelynek értéket előrejelzeni vagy becsülni szeretnénk, *magyarázott (vagy magyarázandó) változónak*, más néven osztályattribútumnak (class attribute, target) hívjuk. A magyarázott változótól (osztályattribútumtól) különböző összes többi változót *magyarázó változónak* () nevezzük. A jegyzetben akkor beszélünk regresszióról, ha a magyarázott változót intervallum skálán mérjük. Amennyiben a magyarázott változó diszkrét értékkészletű, nominális vagy ordinális skálán mért, akkor osztályozásról vagy klasszifikációról (csoportba sorolásról) beszélünk. A klasszifikációt a statisztikai irodalom gyakran diszkriminancia elemzés néven illeti. A gépi tanulás területén az eljárásokat összefoglalóan felügyelt tanulásnak (supervised learning) nevezik.

Az adatbányászatban leggyakrabban alkalmazott osztályozó és regressziós eljárások: (i) legközelebbi szomszéd módszerek, (ii) lineáris és logisztikus regresszió, (iii) mesterséges neurális hálózatok, (iv) döntési szabályok, sorozatok és fák, (v) naiv Bayes módszer és a Bayes-hálózatok, (vi) szupport vektor gépek (support vector machines, SVM-ek), (vii) metaalgoritmusok (boosting, bagging, stacking). Ebben a fejezetben részletesen foglalkozunk ezekkel.

Mindegyik eljárásról elmondható, hogy (legalább) két lépcsőben működik.

1. Először a tanító adatbázison felépítjük a modellt. A tanító adatbázison a magyarázandó változó értékei ismertek, például azért, mert a tanító adatbázis múltbeli megfigyelésekre vonatkozik. A modell építése lényegében annak a feltárása, hogy a magyarázandó változó hogyan függ a többi

változótól. A modellépítés során a regressziós vagy osztályozó eljárás valamilyen szabályokat keres, amelyek alapján a magyarázandó változó értékét a többi változó alapján meg lehet határozni.

2. Később alkalmazzuk a modellt új adatokra, amelyeken a magyarázott változó értéke nem ismert, de ismerni szeretnénk.

Amikor regressziós vagy klasszifikáló modellt választunk, a következő tulajdonságait célszerű figyelembe venni:

- **Előrejelzés teljesítménye:** Milyen jól becsüli meg a modell a nem megfigyelhető magyarázott változót? Milyen gyakran téved (osztályozásnál), illetve mennyit téved átlagosan (regressziónál)? Lásd a 4.1. szakaszt.
- **Gyorsaság:** A modell előállításának és használatának időigénye.
- **Robusztusság:** Érzékeny-e a modell hiányzó attribútum-értékekre, vagy a sokaságtól nagyban eltérő objektumokra?
- **Skálázhatóság:** Használható-e a modell nagyon nagy adathalmazokra is?
- **Értelmezhetőség:** Kinyerhetünk-e az emberek számára értelmezhető tudást a modell belső szerkezetéből?
- **Skála-invariancia:** A klaszterezés lehetetlenség-elméletét (6.1.1 rész) adaptálva skála-invariánsnak hívunk egy osztályozó eljárást, ha a módszer kimenete nem változik abban az esetben, ha bármelyik intervallum típusú magyarázó változó értékei helyett az értékek $\alpha > 0$-szorosát vesszük. Általában érdemes olyan modellt választanunk, amelyre teljesül ez a tulajdonság.

Az adatbányász közösség leginkább a korábban is ismert regressziós és klasszifikáló eljárások skálázhatóságának továbbfejlesztésében ért el eredményeket. Különösen a döntési fák területén fejlesztettek ki olyan algoritmusokat, amelyek akár milliós esetszámú tanuló adatbázis esetén is alkalmazhatók.

A fejezet hátralévő részében először a klasszifikálók és regressziós modellek elméletének legfontosabb elemeivel foglalkozunk, majd az eljárásokat ismertetjük, ezt követően az osztályozók kiértékléséről írunk. A fejezetet az osztályozók gyakorlati alkalazásával kapcsolatos megjegyzésekkel zárjuk. A hagyományos statisztikai módszerek (diszkriminancia analízis, lásd. például [Fazekas, 2000]) ismertetésétől eltekintünk, helyettük inkább az „egzotikusabbakra" koncentrálunk: a dőntési fák, a mesterséges neuronhálózatok, a Bayes-hálózatok, mátrix

faktorizációs algoritmusok és néhány további eljárás főbb jellemzőit mutatjuk be [Johnson és Wichern, 2002], [Han és Kamber, 2006], [Futó, 1999] valamint [Mena, 2000] írások alapján.

4.1. Az osztályozás és a regresszió feladata, jelölések

Az osztályozás és regresszió során n-esekkel (angolul *tuple*) fogunk dolgozni, amelyeket objektumoknak fogunk hívni. Adott lesz objektumok sorozata, amelyet tanító mintáknak, tanító pontoknak, tanító halmaznak (habár a halmaz szó használata itt helytelen, hiszen ugyanaz az objektum többször is előfordulhat) nevezünk. A tanítópontok halmazát $\mathcal{T}$-vel, a tanítópontok számát m-mel vagy $|\mathcal{T}|$-val fogjuk jelölni.

Ahogy említettük, a tanítópontok esetében ismert a magyarázott változó értéke. Valójában tanításra a címkézett adatoknak[1] csak egy részét fogjuk használni. Tehát a $\mathcal{T}$ tanítóhalmaz a címkézett adatoknak egy részhalmaza lesz. A többi címkézett adatot az eljárás teszteléséhez fogjuk használni. A modell tesztelése során ugyanis azt *szimuláljuk*, hogy nem ismerjük a magyarázandó változó értékét: az osztályozó vagy regressziós eljárás számára megmutatjuk ugyan a teszadatokat, de a magyarázandó változó (osztályattribútum) értéke *nélkül*. Ahhoz, hogy mérni tudjuk, mennyire jól teljesít az eljárás, az eljárás által becsült (előrejelzett) osztálycímkéket hasonlítjuk össze a magyarázandó változó általunk ismert értékével, lásd bővebben a 4.10. szakaszt.

A fent említett n-es (tuple) j-edik elemét j-edik *attribútumnak* hívjuk. Egy attribútumra névvel is hivatkozhatunk (pl. kor, magasság, szélesség attribútumok), nem csak sorszámmal. Minden attribútumnak saját értékkészlete van. Az A attribútumváltozón olyan változót értünk, amely az A értékkészletéből vehet fel értékeket.

Általános módon egy klasszifikáló vagy regressziós módszer teljesítményét várható hasznosságával mérhetjük. Legyen a magyarázandó attribútumváltozó (osztályattribútum) Y, a magyarázó attribútumváltozó(k) pedig $\vec{X}$. (Ezen jelölés mellett tehát $\vec{X}$ nem egyetlen attribútumot jelöl, hanem az összes magyarázó attribútumot.) Az $\vec{X}$ komponenseit, az egyes attribútumokat $X_1, \ldots X_k$-val jelöljük. Az egyes attribútumok egy adott objektum esetén felvett konkrét értékeit $x_1, \ldots, x_k$-val jelöljük, $\vec{x} = (x_1, ..., x_k)$. Ha azt is megadjuk, hogy az i-dik objektum attribútumainak értékeiről van szó, akkor ezt felső index-szel jelöljük: $\vec{x}^i = (x_1^i, ..., x_k^i)$.

[1] Az olyan adatokat, amelyek esetében ismert a magyarázott változó (osztályattribútum) értéke, *címkézett adatoknak* nevezzük.

Az eljárásunkat úgy tekinthetjük, hogy egy olyan f függvényt keresünk, amely adott $\vec{x}$-hez a hozzátartozó y értéket rendeli. Az f tehát az $\vec{X}$ értékkészletéről az Y értékkészletére képez. Ekkor célunk $\mathbb{E}_{(\vec{x},y)\in\mathbb{X}\times\mathbb{Y}}\left[U\left(y, f(\vec{x})\right)\right]$ maximalizálása, ahol $\mathbb{E}$ a várható értéket jelöli; $\mathbb{X}$ és $\mathbb{Y}$ az $\vec{X}$ és Y értelmezési tartományát; $U\left(y, \widehat{y}\right)$ pedig az előrejelzett $\widehat{y}$ hasznosságát (utility function), ha tudjuk, hogy a valódi érték y. A gyakorlatban nem ismerjük $\vec{X}$ és Y valódi együttes eloszlását, ezért legtöbbször azzal a feltételezéssel élünk, hogy $\mathcal{T}$ reprezentatív az $\vec{X}$ és Y teljes értelmezési tartományára nézve, és az adott a $\mathcal{T}$ tanítóhalmaz elemei felett számolt hasznosság várható értékét maximalizáljuk. Bináris Y esetén *bináris osztályozásról* beszélünk.

A feladatot $\mathbb{E}_{(\vec{x},y)\in\mathbb{X}\times\mathbb{Y}}\left[L\left(y, f(\vec{x})\right)\right]$ minimalizálásaként is megfogalmazhatjuk, ahol L az U inverze, egy veszteséget mérő függvény (loss function). Az $\mathbb{E}_{(\vec{x},y)\in\mathbb{X}\times\mathbb{Y}}\left[L\left(y, f(\vec{x})\right)\right]$ értéket *várható előrejelzési hibának (expected prediction error)* nevezzük és VEH-val jelöljük. Mivel a várható érték változóiban additív és a konstanssal való eltolás nem változtat az optimalizáláson, ezért $L(y, y) = 0$ feltehető. A hibát a gyakorlatban egy távolságfüggvénnyel (lásd 3.2 rész) definiálják.

Regresszió esetén a két legelterjedtebb megoldás a hiba mérésére a négyzetes hiba $L\left(y, \widehat{y}\right) = (y - \widehat{y})^2$ és az abszolút hiba $L\left(y, \widehat{y}\right) = |y - \widehat{y}|$ alkalmazása.

4.1.1. Definíció *A regressziós eljárás feladata, négyzetes hiba esetén, egy olyan f függvényt találni, amelyre az alábbi $VEH(f)$ minimális; f-et* regressziós modellnek *nevezzük.*

$$\begin{aligned} VEH(f) =& \mathbb{E}_{(\vec{x},y)\in\mathbb{X}\times\mathbb{Y}}\left[\left(y - f(\vec{x})\right)^2\right] \\ =& \int\limits_{(\vec{x},y)\in\mathbb{X}\times\mathbb{Y}} (y - f(\vec{x}))^2 f_{\vec{X},Y}(\vec{x}, y) d\vec{x}, dy, \end{aligned}$$

ahol $f_{\vec{X},Y}(\vec{x}, y)$ a $\vec{X}$ és y együttes valószínűségi sűrűségfüggvényét jelöli.

Osztályozás esetén négyzetes hibáról nincs értelme beszélnünk. Hibafüggvény helyett, k osztály esetén, egy $c \times c$ méretű hibamátrixot (L) adhatunk meg, amely i-edik sorának j-edik eleme ($L[i, j]$) megadja a hiba mértékét, ha i-edik osztály helyett a j-edik osztályt jelezzük előre. A mátrix főátlóján nulla értékek szerepelnek. A várható osztályozási hiba

$$VOH(f) = \mathbb{E}_{(\vec{x},y)\in\mathbb{X}\times\mathbb{Y}}\left[L[y, f(\vec{x})]\right].$$

4.1.2. Definíció *Az osztályozó eljárások feladata egy olyan f függvényt találni, amelyre a fenti $VOH(f)$ minimális. Az f-et* osztályozó modellnek *nevezzük.*

4.1.1. Az elméleti regressziós görbe

A regresszió feladatának előző fejezetbeli definíciója szerinti legkisebb hiba [Bishop, 2006] akkor adódik, ha

$$f(\vec{x}) = \mathbb{E}[Y|\vec{X} = \vec{x}], \tag{4.1}$$

ugyanis

$$\begin{aligned}\mathbb{E}\left[\left(Y - f(\vec{X})\right)^2\right] &= \mathbb{E}\left[\left(Y - \mathbb{E}[Y|\vec{X}] + \mathbb{E}[Y|\vec{X}] - f(\vec{X})\right)^2\right] \\ &= \mathbb{E}\left[\left(Y - \mathbb{E}[Y|\vec{X}]\right)^2\right] + \mathbb{E}\left[\left(\mathbb{E}[Y|\vec{X}] - f(\vec{X})\right)^2\right] \geq \mathbb{E}\left[\left(Y - \mathbb{E}[Y|\vec{X}]\right)^2\right],\end{aligned}$$

mert a kereszttag

$$\begin{aligned}\mathbb{E}\left[\left(Y - \mathbb{E}[Y|\vec{X}]\right)\left(\mathbb{E}[Y|\vec{X}] - f(\vec{X})\right)\right] &= \mathbb{E}_X\mathbb{E}_Y\left[\left(Y - \mathbb{E}[Y|\vec{X}]\right)\left(\mathbb{E}[Y|\vec{X}] - f(\vec{X})\right)|\vec{X}\right] \\ &= \mathbb{E}\left[\left(\mathbb{E}[Y|\vec{X}] - f(\vec{X})\right)\mathbb{E}[Y - \mathbb{E}[Y|\vec{X}]|\vec{X}]\right] \\ &= \mathbb{E}\left[\left(\mathbb{E}[Y|\vec{X}] - f(\vec{X})\right)\left(\mathbb{E}[Y|\vec{X}] - \mathbb{E}[Y|\vec{X}]\right)\right] = 0\end{aligned}$$

A második egyenlőségnél felhasználtuk, hogy $\mathbb{E}(V) = \mathbb{E}_W\mathbb{E}_V(V|W)$, a harmadik egyenlőségnél felcseréltük a szorzat két tagját és felhasználtuk, hogy a $\mathbb{E}[Y|\vec{X}] - f(\vec{X})$ független Y-tól, ezért a várható érték elé mozgatható. Végezetül ismét a $\mathbb{E}(V) = \mathbb{E}_W\mathbb{E}_V(V|W)$ trükköt használtuk, $V = \mathbb{E}[Y|\vec{X}]$ és $W = \vec{X}$ helyettesítéssel.

Az $f(\vec{x}) = \mathbb{E}[Y|\vec{X} = \vec{x}]$ függvényt *elméleti regressziós görbének* nevezik.

Ha a hiba mérésénél a négyzetösszeg helyett (L_2 norma) a különbségösszeget használjuk (L_1 norma), akkor az elméleti regressziós görbe:

$$f(\vec{x}) = median(Y|\vec{X} = \vec{x}). \tag{4.2}$$

4.1.2. Maximum likelihood osztályozás

A . fejezetben definiált várható osztályozási hibát minimalizáló függvény

$$f(\vec{x}) = \text{argmin}_{y_\ell \in Y} \sum_{i=1}^{c} L(y_i, y_\ell)\mathbb{P}(y_i|\vec{X} = \vec{x})$$

A legismertebb veszteség mátrix a nulla-egy mátrix, amelyben a fődiagonálison kívül minden elem egy. Emellett a fenti kifejezés a következőre egyszerűsödik:

$$f(\vec{x}) = \text{argmin}_{y_l \in Y}[1 - \mathbb{P}(y_l|\vec{X} = \vec{x})],$$

vagy egyszerűen:

$$f(\vec{x}) = y_k, \text{ amennyiben } \mathbb{P}(y_k|\vec{X} = \vec{x}) = \max_{y_l \in Y} \mathbb{P}(y_l|\vec{X} = \vec{x}).$$

A fenti osztályozó a Bayes vagy maximum likelihood osztályozó, amely azt állítja, hogy az adott $\vec{x}$ megfigyelés esetén legvalószínűbb osztály lesz az osztályozó kimenete.

Ha a várható értéket meghatározó valódi eloszlásokat ismernénk, akkor megtalálható a legjobb osztályozó (klasszifikáló). Például (azonos kovarianciájú) többdimenziós normális eloszlásokat feltételezve kvadratikus (lineáris) döntési szabályokat kapunk [Thomas, 2000], [Fazekas, 2000]. A gyakorlatban azonban az eloszlás paramétereit általában még akkor is becsülnünk kell, ha feltételezünk egy adott típusú eloszlást.

Adatbányászati alkalmazásokban a normális eloszlás feltételezése gyakran egyáltalán nem reális (gondoljunk a sok nominális változóra), sőt, legtöbbször nincs elegendő háttérismeretünk ahhoz, hogy bármilyen eloszlást is feltételezhessünk. Ezért az adatbányászati módszerek nem élnek feltevésekkel az eloszlással kapcsolatban.

Mivel a valódi eloszlásokat nem ismerjük, a most látott maximum likelihood osztályozás nem a gyakorlatban használható osztályozó algoritmusok egyike, hanem egy elméleti lehetőség, amelyet az elképzelhető legjobb osztályozó modellnek tekintünk. A maximum likelihood osztályozó leginkább azért érdekes, mert néhány gyakorlatban is létező modellről bizonyítható, hogy megfelelő feltételek mellett nem nyújt sokkal rosszabb teljesítményt, mint a maximum likelihood osztályozás (lásd például a legközelebbi szomszéd módszert).

4.2. k-legközelebbi szomszéd módszer

A k-legközelebbi szomszéd módszere osztályozásra és regresszióra egyaránt használható. A k-legközelebbi szomszéd módszere egy „lusta" eljárás, amely nem épít modellt. Alapelgondolása, hogy a hasonló objektumok hasonló tulajdonságokkal bírnak. A hasonlóságot a 3.2. részben bemutatott távolságfüggvények valamelyikével mérjük. A tanuló adatbázist teljes egészében tároljuk, és amikor egy ismeretlen osztályú x' objektumot kell klasszifikálnunk, akkor megkeressük az x'-höz a távolságfüggvény szerint legközelebbi k darab objektumot, ezeket nevezzük az x' szomszédainak, és az x' objektumot abba a kategóriába soroljuk, amely a leggyakoribb a szomszédok között, többségi szavazást végzünk. Ha például $k = 5$ és egy x'-höz talált öt legközelebbi objektum közül egy tartozik az A osztályba, három a B-be, és egy a C-be, akkor x'-t a B osztályba sorolja az eljárás. Regresszió esetén a szomszédok osztályértékeinek átlaga lesz a kimenet.

A k-legközelebbi szomszéd egy módosítását *súlyozott legközelebbi szomszéd módszernek* hívják. Ekkor az osztályozási feladatoknál a többségi szavazás során a k szomszéd minden tagjának súlya az osztályozandó ponttól vett távolságától függ: a távolságnak inverze vagy valamilyen csökkenő függvénye. Az osztályozandó ponthoz közel fekvő tanítópontoknak tehát nagyobb szavuk van a végső osztály meghatározásában, mint a távolabb eső pontoknak. Ehhez hasonlóan egy regressziós feladat esetén az egyszerű átlag helyett számolhatunk súlyozott átlagot is: minél kisebb egy adott szomszéd távolsága, annál nagyobb súllyal szerepel a súlyozott átlagban.

A módszer egyfajta lokális sűrűségfüggvény becslő eljárásnak is tekinthető. A k-legközelebbi szomszéd a következő regressziós függvényt adja tetszőleges x pontra:

$$\widehat{f}(\vec{x}) = \frac{1}{k} \sum_{y_i \in N_k^{(y)}(\vec{x})} y_i,$$

ahol $N_k^{(y)}(\vec{x})$ az $\vec{x}$ pont tanítóhalmazbeli k-legközelebbi szomszédjainak címkéiből álló halmazt jelöl. Osztályozás esetén pedig:

$$\widehat{f}(\vec{x}) = y_k, \text{ amennyiben } freq(y_k|N_k^{(y)}(\vec{x})) = \max_{y_\ell \in Y} freq(y_\ell|N_k^{(y)}(\vec{x})),$$

ahol $freq(y|N_k^{(y)})$ az y osztálycímke gyakoriságát jelöli az $N_k^{(y)}$-ban.

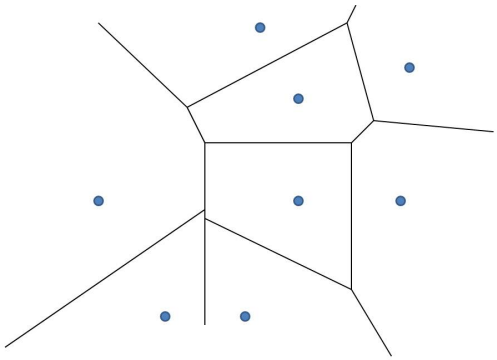
Az $\widehat{f}(\vec{x})$ tulajdonképpen az $f(\vec{x})$ közelítése. A közelítés két okból következik:

1. regresszió esetén a várható érték helyett a mintaátlagot használtuk, osztályozás esetén pedig a valószínűség helyett a relatív gyakoriságot,

2. az $\vec{x}$ pontban vett feltétel helyett az $\vec{x}$ környezetét vettük.

Sok tanítópont esetében tetszőleges ponthoz közel lesznek a szomszédai, továbbá az átlag egyre stabilabb lesz, amennyiben k egyre nagyobb. Be lehet látni, hogy $P(\vec{X}, Y)$-ra tett enyhe feltételek mellett $\widehat{f}(\vec{x}) \to \mathbb{E}[Y|\vec{X} = \vec{x}]$, amennyiben $m, k \to \infty$ és $m/k \to 0$. Ezek szerint a k-legközelebbi szomszéd módszere egy univerzális approximátor. Amint láttuk, kellőképpen nagy tanítóadatbázis esetén a legközelebbi szomszéd módszer regressziós változatának kimenete megközelítőleg az elméleti regressziós görbe. Az is belátható, hogy a tanítóhalmaz méretével a végtelenhez tartva, a legközelebbi szomszéd módszere osztályozás során legfeljebb kétszer annyi hibát követ el, mint a maximum likelihood osztályozó, amelyet az elméletileg lehetséges legjobb osztályozónak tekintettünk [Devroye és tsa., 1996]. Azt gondolhatnánk tehát, hogy nem is érdemes további osztályzókkal foglalkoznunk.

A gyakorlatban azonban nem áll rendelkezésünkre elegendően sok tanítópont, magas dimenziószám mellett a konvergencia lassú. A módszer további



4.1. ábra. Tanítópontok a síkon és a Voronoi tartományok.

problémáival a 4.2.2. fejezetben foglalkozunk. Ha fel tudunk tenni az osztályozásra valamilyen struktúrális feltételt, például azt, hogy az osztályokat egy hipersík választja el (lásd 4.3. és 4.8. fejezeteket), akkor ezt kihasználva a gyakorlatban pontosabb modellt építhetünk, mint a k-legközelebbi szomszéd módszere.

A legközelebbi szomszéd módszer ábrázolásánál $k = 1$ esetén kedvelt eszköz a Voronoi diagramm. A felületet felosztjuk tartományokra úgy, hogy minden tartományba egy tanító pont essen és igaz legyen, hogy a tartományon belüli bármely pont a tanítópontok közül a tartomány tanítópontjához van a legközelebb. Egy ilyen felosztást láthatunk a 4.1 ábrán.

Az osztályozáshoz természetesen nem kell meghatározni a tartományokat és megnézni, hogy az osztályozandó pont melyik tartományba tartozik. Egyszerűen nézzük végig a tanítópontokat és válasszuk ki a k darab leginkább hasonlót.

4.2.1. Dimenzióátok és a legközelebbi szomszéd módszere

A legközelebbi szomszéd módszer egy univerzális approximátor, tetszőleges osztályozó függvényt képes reprodukálni, csak elég tanítópont kell hozzá. A módszert lokális approximátornak is szokás hívni, mert tetszőleges pont osztályértékét a (lokális) kőrnyezetének tanítóértékeinek átlagával helyettesíti. A módszer jól működik alacsony dimenzióknál, de magas dimenzióknál könnyen csődőt mondhat. Ez szorosan összefügg a 3.3.7. fejezetben tárgyalt dimenzióátokkal, a sűrűség csökkenésével.

Ahhoz, hogy a k legközelebbi szomszéd jól működjön, tetszőleges pont környezetében elegendő tanítópontnak kell lennie. Tetszőleges $\vec{x}$ pont környezetén

az $\vec{x}$-től legfeljebb ϵ távolságra lévő pontokat értjük. Ez egydimenziós esetben egy 2ϵ hosszú szakaszt, kétdimenziós esetben ϵ sugarú kört, háromdimenziós esetben ϵ sugarú gömböt jelent. Ha azt szeretnénk, hogy a keresési térben a tanítópontok sűrűsége rögzített legyen, akkor a tanítópontok számának exponenciálisan kell nőnie a dimenzió növelésével. A gyakorlatban a tanítópontok adottak, ami általában behatárolja a dimenziók és így a figyelembe vehető attribútumok számát.

Újabb kutatások azt mutatják, hogy a k-legközelebbi szomszéd módszer sokdimenziós adatokra történő alkalmazásakor érdemes figyelembe venni a csomósodás jelenségét [Symeonidis és tsa., 2010, Radovanović, 2011], amelyet a 3.3.7. fezetben tárgyaltunk. A kutatás eredményei azt mutatják, hogy javít az osztályozás pontosságán, ha a tanítóhalmazbeli objektumokat súlyozzuk aszerint, hogy hányszor fordulnak elő más tanítóhalmazbeli objektumok jó illetve rossz szomszédjaiként, egy-egy objektum súlyának meghatározására ezt a súlyfüggvényt javasolják:

$$w_{k'}(\vec{x}) = e^{-h_B(\vec{x},k')},$$

ahol $\vec{x} \in \mathcal{T}$ ($\vec{x}$ egy tanítópont), k' a 3.3.7. fejezetben leírt $BN_{k'}$ számításakor figyelembe vett legközelebbi szomszédok száma, $h_B(\vec{x}, k')$ pedig

$$h_B(\vec{x}, k') = \frac{BN_{k'}(\vec{x}) - \mu_{BN_{k'}}}{\sigma_{BN_{k'}}},$$

ahol μ és σ a $BN_{k'}$ átlagát és szórását jelöli.

A fenti súlyozás szignifikánsan javít ugyan az osztályozás pontosságán, de nem oldja meg azt az alapvető problémát, amely a sokdimenziós adatok óhatatlanul alacsony sűrűségéből adódik. Az alacsony sűrűség nem jelenti azt, hogy magas dimenziókban nem lehet jó osztályozó függvényt találni, csak megkötést kell tennünk az osztályozó függvény típusára vonatkozóan. Például, ha azt feltételezzük, hogy az osztályozó egy hipersíkkal leírható, akkor a dimenziók számának növelésével csak lineárisan növekszik a szükséges tanítópontok száma, hiszen kétdimenziós esetben két pont határoz meg egy egyenest, három dimenziónál három pont határoz meg egy síkot, stb.

4.2.2. A legközelebbi szomszéd érzékenysége

A legközelebbi szomszéd módszer hátránya, hogy érzékeny a független attribútumokra. Ezt egy példán keresztül szemléltetjük. Feladatunk, hogy egy olyan modellt találjunk, amely képes eldönteni egy-egy diákról, hogy szorgalmas-e. Az egyik attribútum a görgetett tanulmányi átlag a másik a hajhossz. A 4.2 ábra mutatja a tanító pontokat, cél a csillaggal jelölt tanuló osztályozása. Ha

csak a jegyátlagot tekintjük, akkor a szorgalmasak közé soroljuk. Ha a távolság megállapításánál a hajhosszt is figyelembe vesszük, akkor egy olyan hallgató lesz hozzá a legközelebb, akiről tudjuk, hogy szorgalmatlan. Sőt, ha euklideszi távolságot használunk és a független attribútum értékei jóval nagyobbak a függő attribútum értékeinél, akkor a független attribútum „elnyomja" a függő attribútumot.

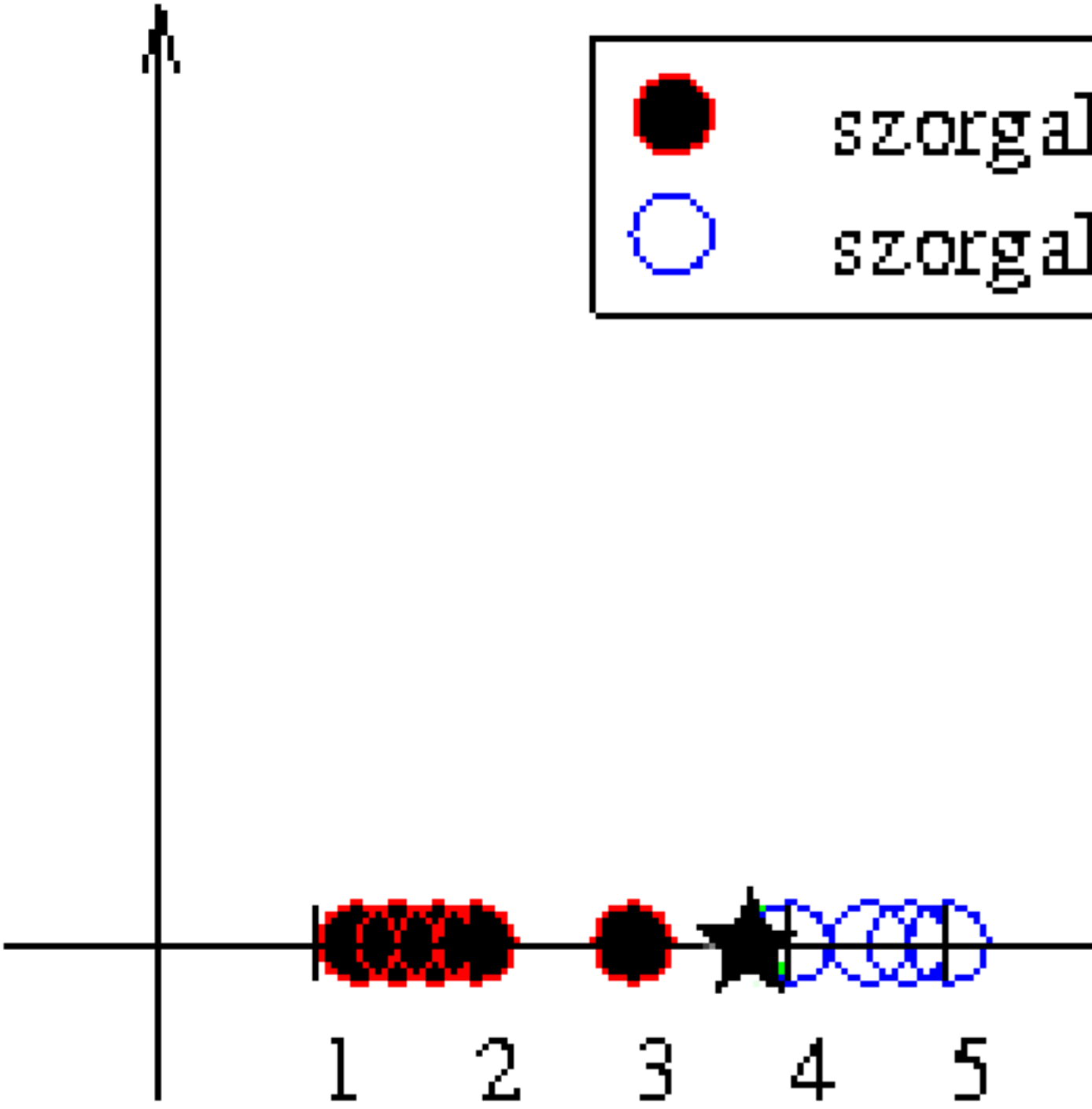
Számos megoldást javasolnak a független attribútum által okozott hiba kiküszöbölésére: (i) ha tehetjük használjunk *több tanító pontot*; (ii) kérdezzük meg az *alkalmazási terület szakértőjét*, hogy a távolság meghatározásánál mely attribútumokat vegyük számításba; vagy (iii) alkalmazzunk *statisztikai tesztet* a függetlenség megállapítására; (iv) amennyiben nincs sok attribútumunk, akkor meghatározhatjuk az osztályozás pontosságát az *összes attribútum részhalmaz* esetén majd kiválaszthatjuk a legjobbat.

Sok attribútum esetén az összes attribútumhalmaz kipróbálása túl sok időt és erőforrást kíván. Egy (v) *mohó bővítő eljárás* (*forwald selection*) egyesével bővítené a tesztelendő attribútumhalmazt úgy, hogy az a legjobb osztályozást adja. Ha az osztályozás minősége nem javul, akkor befejezzük a bővítést. Ez a módszer kiselejtezné az X_1 és X_2 bináris attribútumokat annál az osztályozásnál, amelyben a magyarázandó attribútum értéke X_1 és X_2 moduló kettővel vett összege és X_1, X_2 egymástól (és a magyarázandó attribútumtól is) teljesen függetlenek. Egy (vi) *csökkentő módszer* (*backward selection*) a teljes attribútumhalmazból indulna ki és minden lépésben egy attribútumot dobna ki.

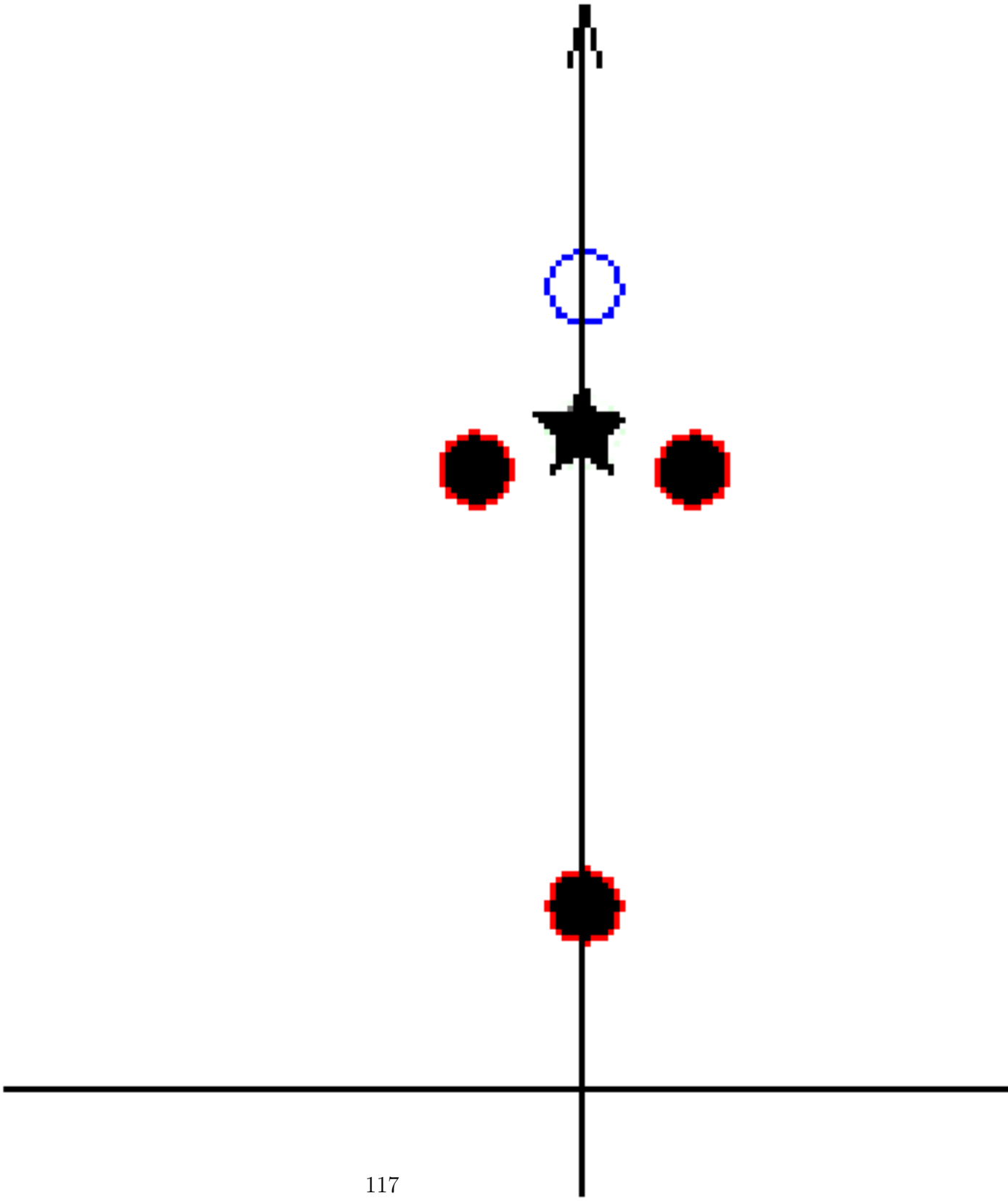
A legközelebbi szomszéd módszer érzékeny a mértékegységre is. Ez logikus, hiszen a legközelebbi szomszéd módszer érzékeny a távolság definíciójára, az pedig nagyban függ az egyes attribútumok mértékegységétől. A problémát a 4.3 ábra szemlélteti.

Az egyik attribútum jelöli egy ember hajhosszának eltérését az átlagos hajhossztól, a másik attribútum az adott ember jövedelmét jelöli dollárban. Az első ábrán a hosszt méterben mérjük a másodikban pedig lábban. Az osztályozandó (csillaggal jelölt) ponthoz egy teli van a legközelebb az első esetben, míg a második esetben üres pont a legközelebbi. A példa mutatja, hogy a legközelebbi szomszéd módszer nem skálainvariáns, azaz egy attribútum értékének konstanssal történő szorzása változtathat az osztályozás eredményén.

Az említett problémák nem feltétlenül az osztályozó hibái. A legközelebbi szomszéd módszerben a távolságfüggvény játszik központi szerepet. A helyes távolságfüggvény meghatározásához válasszuk ki a fontos attribútumokat, normalizáljuk, ha szükséges, illetve fontosságuk alapján súlyozzuk őket. Korábban volt szó az objektumok súlyozásáról távolságuk szerint, illetve aszerint, hogy hányszor fordulnak elő rossz szomszédként. Ne keverjük ezt az attribútumok súlyozásával: az attribútumokat a távolságfüggvény számításához súlyozhatjuk. Ha szükséges, a kétféle súlyozást egyszerre is alkalmazhatjuk.



4.2. ábra. Független attribútumok hatása a legközelebbi szomszéd osztályozásra



4.3. ábra. Mértékegység hatása a legközelebbi szomszéd osztályozóra

4.2.3. Az osztályozás felgyorsítása

Egy új elem osztályozásánál meg kell határoznunk a k-legközelebbi szomszédot. Ez a teljes adatbázis egyszeri, lineáris végigolvasását jelenti. A mai számítógépes architektúráknak kedvez ez a feladat. A tanítópontok gyakran elférnek a memóriában és a modern processzorokban alkalmazott csővezetékes feldolgozás (lásd 2.6. fejezet) nagyban gyorsítja a keresést. Talán ez az oka, hogy teszteredmények szerint a futásidő tekintetében a kifinomultabb, bonyolultabb módszerek is legfeljebb egy nagyságrendet vernek a lineáris módszerre.

A 70-es évektől egyre több írás született, amelynek témája a lineáris módszernél gyorsabb algoritmusok kidolgozása. Az új módszerek általában az ún. *branch-and-bound* technikát alkalmazzák, melynek során a tanító pontok terét felosztják és csak bizonyos részeket vizsgálnak a keresés során. Az előfeldolgozási lépésben transzformálják az adatbázist, melynek eredményeként általában egy speciális adatstruktúrában tárolják a tanítópontokat. Amennyiben a tanítás során egyetlen numerikus attribútumot veszünk figyelembe és előfeldolgozási lépésként sorba rendezzük az adatokat ($O(m \log m)$ idő alatt), akkor a legközelebbi szomszédok meghatározásához $O(\log m)$ lépés elég. Futási idő szempontjából azonos asszimptotikával rendelkező algoritmust adtak két magyarázó változó esetében is [Aurenhammer, 1991].

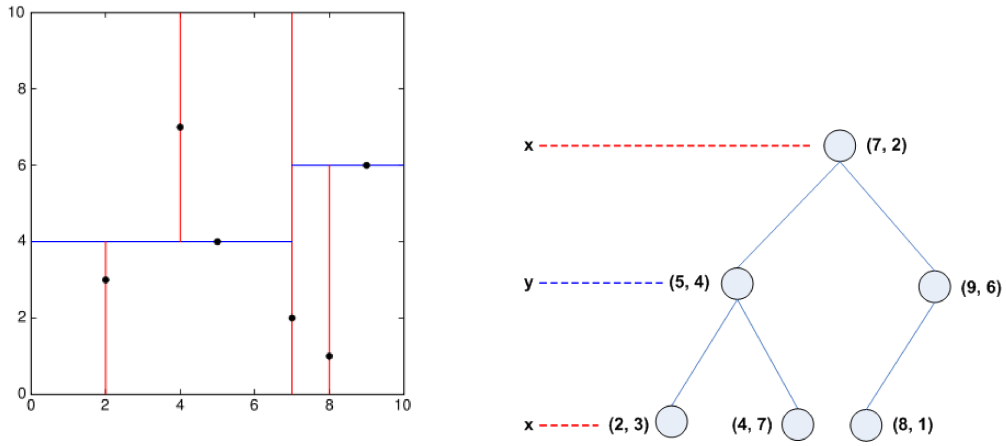
Több magyarázó attribútum esetén (nagyobb dimenzióknál) a legismertebb módszer KD-fákat használ [Bentley, 1975]. Az algoritmus az előfeldolgozás során a teret hipertéglatestekre osztja, egyes téglatesteket pedig további téglatestekre. A hipertéglatestek oldalai párhuzamosak egymással és a tengelyekkel és egy téglatest kettéosztása mindig egy az oldalfallal párhuzamos sík mentén való kettéosztást jelent. Egy téglatestet nem oszt tovább, ha a téglatestben található pontok száma adott korlát alatt van.

A KD-fa bináris és minden csomópontjának megfelel egy hipertéglatest. A levelekhez hozzá vannak rendelve azok a tanítópontok, amelyek a levél által meghatározott téglatesbe esnek. Tetszőleges csomópont gyermekeihez tartozó hipertéglatest a csomóponthoz tartozó hipertéglatest kettéosztásából jött létre. A 4.4 ábrán néhány tanítópont felosztása látható és a felosztáshoz tartozó KD-fa. Felhívjuk a figyelmet, hogy a téglatestek által kijelölt terek nem osztályozási tartományoknak felelnek meg.

Osztályozásnál nem csak azokat a tanítópontokat veszi figyelembe, amelyek abban a téglatestben vannak, amelyet az osztályozandó pont kijelöl. Az osztályozás menete a következő:

1. A fa csúcsából kiindulva jussunk el addig a levélig, amely téglatestje tartalmazza az osztályozandó pontot.

2. Határozzuk meg a legközelebbi pontot (az előző lépésben talált téglatest



4.4. ábra. Tartományok meghatározása KD-fa építésénél (bal oldali ábra) és a KD-fa (jobb oldali ábra). Először a vízszintes tengelyt vágtuk ketté a (7,2) koordinátájú ponton áthaladó fallal, majd a függőleges (Y) tengelyt vágtuk az (5,4) és (9,6) koordinátájú pontokon keresztül haladó falakkal, stb. Forrás: wikipedia.org

pontjai közül).

3. Amennyiben a legközelebbi pont közelebb van, mint bármelyik oldalfal – másképp fogalmazva, az osztályozandó pontból a legközelebbi ponttól vett távolsággal rajzolt hipergömb nem metsz oldalfalat –, akkor leállunk. Ellenkező esetben meg kell vizsgálni azt a hipertéglatestet (illetőleg azon hipertéglatesteket), amely(ek) fala közelebb van, mint a 2. lépésben talált legközelebbi pont. Ez a téglatest (ezen téglatestek) ugyanis tartalmazhat(nak) olyan pontot, amely közelebb van, mint az eddig talált legközelebbi pont.

Az utolsó lépésben vizsgálandó téglatestek nem biztos, hogy az osztályozandó pont által kijelölt téglatest szomszédai a KD-fában. Vegyük észre, hogy az egyszerű konstrukció következtében (értsd: oldalfalak párhuzamosak a tengelyekkel) nagyon gyorsan el tudjuk dönteni, hogy egy adott téglatestnek lehet-e olyan pontja, amely egy adott ponttol, adott távolságnál közelebb van.

A KD-fa építésénél két célt tartunk szem előtt:

1. A fa kiegyensúlyozott legyen, abban a tekintetben, hogy minden téglatest nagyjából ugyanannyi tanítópontot tartalmazzon. Ha ki tudunk zárni egy téglatestet a vizsgálatból, akkor ezzel sok pontot szeretnénk kizárni.

2. A hipertéglalapok legyenek kockák. Ekkor ugyanis nem fordulhat elő, hogy az osztályozandó pont által kijelölt téglatesttel nem érintkező téglatest tartalmazza a legközelebbi szomszédot. Az elnyújtott téglatestek nem kedveznek az algoritmusnak.

Habár a második elvárásnak nem biztos, hogy eleget tesz az alábbi módszer, mégis a gyakorlatban jó eredményt ad. Ki kell jelölni az új fal tengelyét, majd meg kell határozni a helyét. A tengely kijelöléséhez nézzük meg mekkora a szórás az egyes tengelyekre nézve. Legyen a fal a legnagyobb szórást eredményező tengelyre merőleges. A fal helyét pedig a medián határozza meg, így a pontok egyik fele az egyik téglatestbe a másik fele a másik téglatestbe fog kerülni.

A KD-fánál vannak újabb adatstruktúrák, ezek közül a legismertebbek a Metric tree (Ball tree) [Omohundro, 1989, Uhlmann, 1991] és a Cover Tree [Beygelzimer és tsa., 2006]. Ugyanakkor Kibriya és Frank (2007) valódi és generált adatbázisokon végzett kísérletek eredményei alapján azt állítják, hogy ezek az új módszerek nem mutatnak számottevő javulást a KD-fához képest.

4.3. Lineárisan szeparálható osztályok

Két osztály lineárisan szeparálható, ha egy hipersík segítségével el tudjuk különíteni a két osztály pontjait. Amennyiben a pontok n dimenziósak, akkor $n-1$ dimenziós hipersíkot kell meghatároznunk. Egy $\vec{x}$ pont osztályozásához az

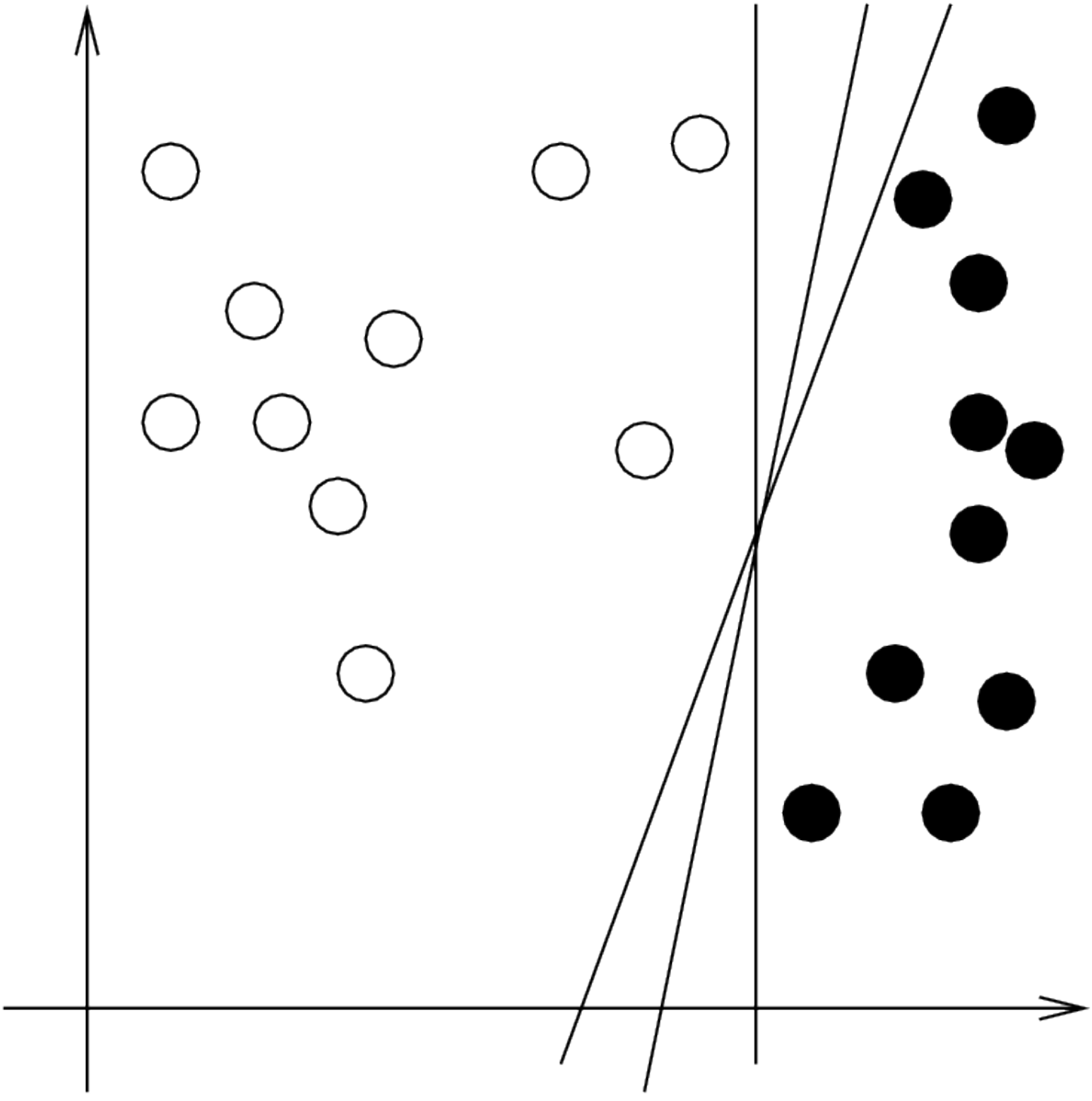
$$f_0(\vec{x}) = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b, \quad \vec{x} = (x_1, ..., x_n). \tag{4.3}$$

függvényt hívjuk segítségül. Ha $f_0(\vec{x}) > 0$, akkor az $\vec{x}$-t az első osztályba soroljuk, egyébként a másodikba.

Az osztályozó algoritmus tanulási fázisa a w súlyok meghatározásából áll. Új elemek osztályozásához meghatározzuk az új elem attribútumainak w értékekkel történő súlyozott összegét. Ha az összeg nagyobb nulla, akkor az első osztályba tartozik, ellenkező esetben a másodikba.[2] Lineárisan szeparálható osztályokra láthatunk példát a 4.5 ábrán. Látható, hogy adott tanítóhalmazhoz több szeparáló hipersík is létezik.

Mennyire erős az a megkötés, hogy az osztályok lineárisan szeparálhatók legyenek? Új attribútumok bevezetésével, amelyek az eredeti attribútumok nemlineáris transzformáltjai olyan térbe kerülhetünk, amelyben már lehet lineáris

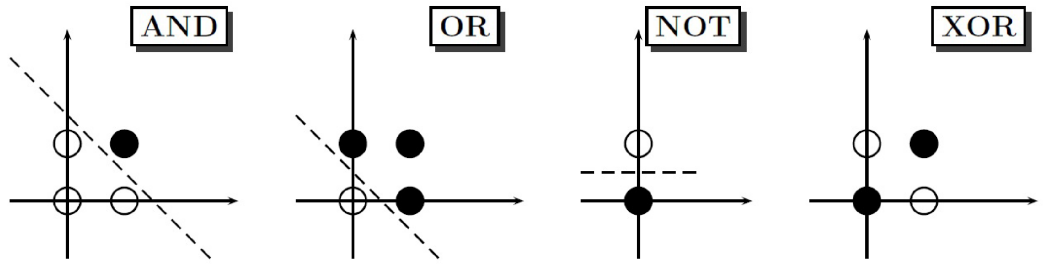
[2]Látható, hogy ez az eljárás ebben a formában két osztályos (bináris) osztályozási feladatokra alkalmazható, a későbbiekben (4.13.1. fejezet) foglalkozunk azzal, mit tehetünk, ha ezt a modellt szeretnénk többosztályos osztályozási feladatokra alkalmazni.



4.5. ábra. Példa lineárisan szeparálható osztályokra

szeparálást végezni. Kérdés azonban, hogy ehhez az eredeti attribútumainkat konkrétan hogyan transzformáljuk egy adott feladat estetében?

Tekintsük azt az esetet, hogy minden attribútum bináris, azaz 0 és 1 értékeket vehet fel. Miként a 4.6 ábra mutatja, az AND, OR, NOT függvények lineárisan szeparálható osztályokat hoznak létre.



4.6. ábra. AND, OR, NOT logikai függvények tanulása, XOR függvény

Sajnos ugyanez nem mondható el az XOR függvényre. Tehát már egy ilyen egyszerű logikai függvényt, mint az XOR, sem tud megtanulni egy lineáris osztályozó. A neurális hálózatoknál vissza fogunk térni az XOR kérdéséhez. Látni fogjuk, hogy a neurális hálózatok már tetszőleges logikai függvényt képesek megtanulni.

A perceptron és a Winnow módszereket fogjuk először szemügyre venni. Ezek kezdetben konstans[3] értékeket tartalmazó súlyvektorból kiindulnak ki, és a tanítópontok hatására a súlyvektort addig módosítják, amíg minden pontot jól szeparál a súlyvektor. A módszerek előnye, hogy jól használható online környezetben is, ahol néha új tanítópont érkezik, amely hatására módosítanunk kell a súlyvektort.

Ismertetjük még a Rocchio-eljárást, amely szintén lineáris szeparálást hajt végre annak ellenére, hogy nem állít elő szeparáló hipersíkot. Végül elmélyedünk a logisztikus regreszió rejtelmeiben. A logisztikus regressziónál és az SVM osztályozónál fog felmerülni az a kérdés, hogy a több lehetséges szeparáló hipersík közül melyik választja el a legjobban a két osztályt, melyik az a hipersík, amelytől legtávolabb vannak a pontok.

[3]perceptronnál nulla, Winnownál egy

4.3.1. Perceptron tanulási szabály

A következőkben feltételezzük, hogy az összes attribútum valós.[4] Az attribútumok számát az eddigiekhez hasonlóan n-nel jelölve, a hipersík dimenziója n lesz, ugyanis felveszünk egy extra attribútumot, melynek értéke minden pontnál konstans 1 lesz. Ezt az extra attribútumot az angol irodalomban bias-nak hívnak. Így az osztályozáshoz használt (4.3) egyenletet

$$f_0(\vec{x}) = w_0x_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n = \vec{x}^T\vec{w} \tag{4.4}$$

alakban írhatjuk, ahol $x_0 = 1$, így w_0 a korábbi formalizmusbeli b-t helyettesíti, $\vec{w} = (w_0, w_1, ..., w_n)$, a vektorokat egyetlen oszlopból álló mátrixnak tekintjük, a T felsőindex pedig a mátrix transzponálát jelöli.

A perceptron algoritmus pszeudokódja alább olvasható, az áttekinthetőbb leírás érdekében használjuk a $\vec{w} = (w_0, w_1, ..., w_n)$ jelölést.

Algoritmus Perceptron tanulási szabály

Require: $\mathcal{T}$: tanítópontok halmaza

```
w = (0,0,...,0)
while van rosszul osztályozott t ∈ T do
  for all minden t ∈ T do
    if t rosszul van osztályozva then
      if t az első osztályba tartozik then
        w = w + t
      else
        w = w − t
      end if
    end if
  end for
end while
```

Kezdetben az összes $w_0, w_1, ..., w_n$ súly értéke 0. Az algoritmus egyenként végighalad a tanítópontokon, akár többször is, ameddig van rosszul osztályozott tanítópont. Amennyiben eközben rosszul osztályozott ponttal találkozunk, úgy módosítjuk a hipersíkot, hogy a rosszul osztályozott tanító pont közelebb kerül hozzá, sőt akár át is kerülhet a sík másik oldalára. Ha egy $\vec{t}$ tanítópont a valóságban az első osztályba tartozik, de az aktuális súlyok alapján az algoritmus rosszul osztályozza, azaz $f_0(\vec{t}) < 0$, akkor a $\vec{w}$ súlyvektort $\vec{w} + \vec{t}$-re változtatjuk. Ennek hatására $f_0(\vec{t})$, azaz a $\vec{t}$ az attribútumértékeinek (4.4) szerinti súlyozott összege, $\sum w_it_i$-ről $\sum(w_i + t_i)t_i$-re változik. Az $f_0(\vec{t})$ új értéke

[4] A Perceptron algoritmus eredetileg bináris, 0 illetve 1 értékű attribútumokra vezették be.

biztosan nem kisebb, mint a régi, hiszen $f_0^{(\text{új})}(\vec{t}) = \sum(w_i+t_i)t_i = \sum(w_i t_i+t_i^2) = f_0^{(\text{régi})}(\vec{t}) + \sum t_i^2$. Hasonlóképpen, ha egy rosszul osztályozott tanítópont a *második* osztályba tartozik, szintén úgy módosítjuk a súlyokat, hogy a helytelenül osztályozott pont vagy átkerüljön a hipersík túloldalára vagy legalább közelebb kerüljön a hipersíkhoz.

A hipersík módosításai egymásnak ellentétesek lehetnek (olyan, mintha a tanítópontoktól jobbról és balról kapná a pofonokat), de szerencsére biztosak lehetünk benne, hogy a sok módosításnak előbb-utóbb vége lesz:

4.3.1. Lemma *Perceptron tanulási algoritmus véges lépesen belül leáll, ha az osztályok lineárisan szeparálhatók.*

Hátrány, hogy ha a tanító pontok nem szeparálhatóak lineárisan, akkor az algoritmus nem áll le. A gyakorlatban ezért egy maximális iterációs számot adnak meg, amelynek elérésekor sikertelen üzenettel leáll az algoritmus.

Gyakran az eddig ismertetett Percepton-algoritmus egy módosított változatát használják: $\vec{w} = \vec{w}+\vec{t}$ illetve $\vec{w} = \vec{w}-\vec{t}$ helyett $\vec{w} = \vec{w}+\epsilon\vec{t}$ illetve $\vec{w} = \vec{w}-\epsilon\vec{t}$ lépésekben módosítják a $\vec{w} = (w_0, w_1, ..., w_n)$ súlyvektort [Tan és tsa., 2005], ahol ϵ az adatbányász felhasználó által választott, általában kicsi szám (pl. $\epsilon = 0.01$). Az ϵ-t *tanulási rátának* hívják. Megjegyezzük továbbá, hogy a Perceptron algoritmus módosított változata használható a lineáris regresszióhoz, lásd 4.3.4. fejezetet.

4.3.2. Winnow módszer

Winnow módszerét akkor alkalmazhatjuk, ha az objektumok minden attribútuma bináris. Az eltérés a perceptron tanulástól annyi csak, hogy (i) kezdetben a $w_1, ..., w_n$ súlyok értéke nem 0, hanem 1 és (ii) a rossz osztályozás esetén a súlyvektorhoz nem hozzáadjuk a tanítópont vektorát, hanem a súlyvektor bizonyos elemeit megszorozzuk vagy eloszjuk $\alpha > 1$ konstanssal, attól függően, hogy a tanítópont melyik osztályba tartozik. Az osztályozó akkor sorol egy $\vec{x}$ pontot az első osztályba, ha

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n > \Theta,$$

ahol Θ előre megadott konstans. A szorzást vagy osztást azon elemekre végezzük, amelyre a tanítópont vektora egyest tartalmaz. Ha egy második osztályba tartozó $\vec{x} = (x_1, ..., x_n)$ tanítópontot tévesen az első osztályba soroltunk, a fenti képlet túl nagy eredményt adott, tehát ekkor α-val osztjuk a megfelelő súlyokat. Ellenkező hiba esetén értelemszerűen α-val szorozzuk a megfelelő súlyokat.

Bevezetve az X_0 bias változót, amelynek értéke minden objektumra konstans $x_0 = 1$, most is formalizálhatjuk $\Theta = -w_0$ mellett az osztályozási függvényt úgy, hogy akkor sorolunk egy $\vec{x} = (x_1, ..., x_n)$ objektumot az első osztályba, ha

$$w_0 x_0 + w_1 x_1 + w_2 x_2 + \ldots + w_n x_n > 0.$$

Ilyen formalizmus mellett könnyen látható, hogy nem feltétlenül szükséges a $\Theta = -w_0$ értékét előre megadnunk, hanem a w_0 súlyt is tanulhatjuk a többi $w_1, ..., w_n$ súllyal együtt.

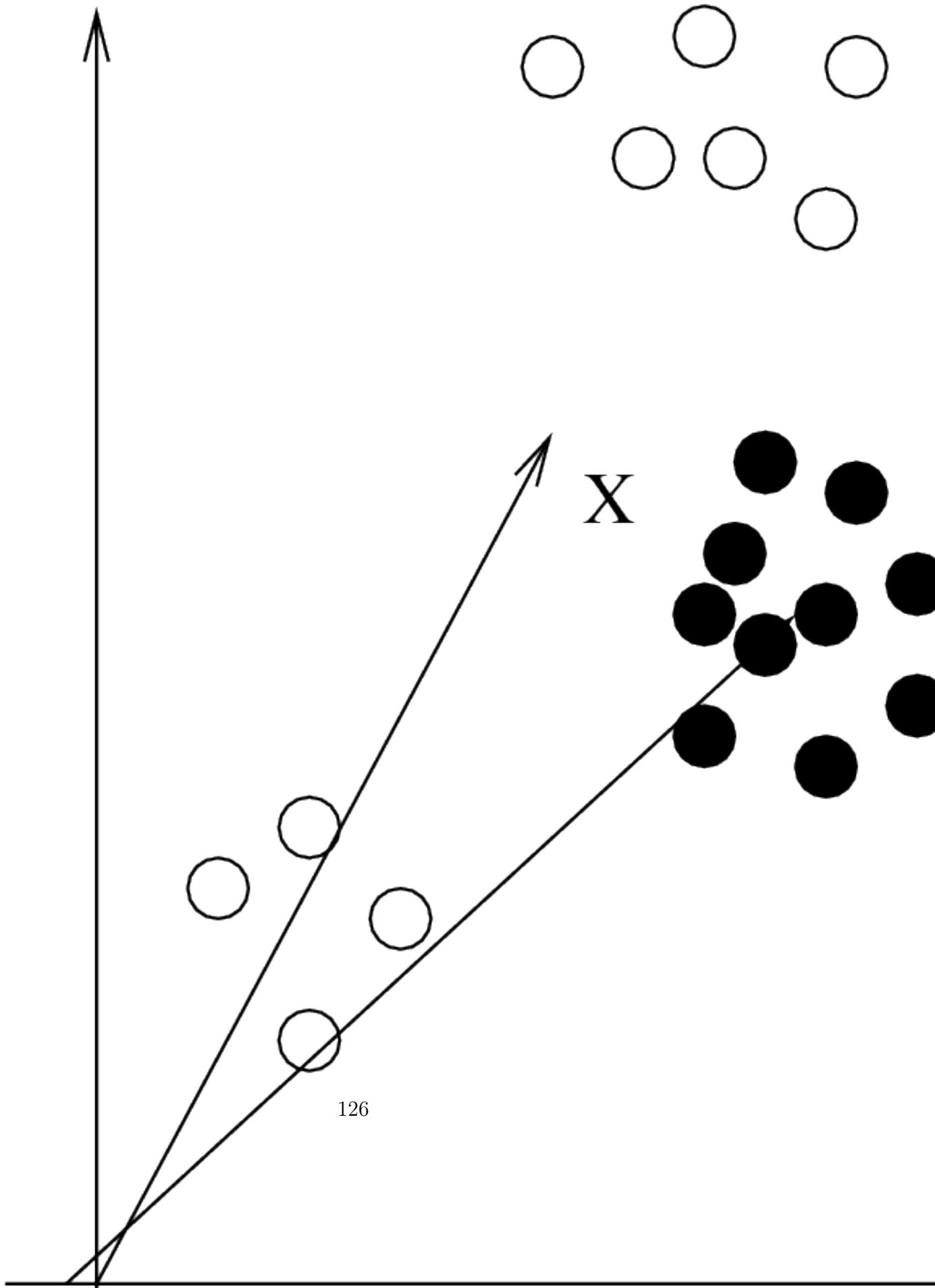
Mivel α pozitív és a kezdeti súlyvektor minden eleme egy, ezért a súlyvektor minden eleme mindig pozitív marad. Vannak azonban alkalmazások, ahol negatív súlyokat is meg kell engedni. Ekkor a *kiegyensúlyozott Winnow* (balanced Winnow) módszert alkalmazhatjuk, ahol két súlyvektort módosítunk ($\vec{w}^+$, $\vec{w}^-$). Az osztályozáshoz a $\vec{w}^+ - \vec{w}^-$ vektort használjuk. A rossz osztályozás esetén a $\vec{w}^+$-t ugyanúgy módosítjuk, mint a Winnow alapverziójánál, a $\vec{w}^-$-t ellenkezőképpen: amikor w_i^+ egy elemét szorozzuk α-val, akkor a w_i^- megfelelő elemét osztjuk vele.

4.3.3. Rocchio-eljárás

A *Rocchio-eljárás* klasszikus módszernek számít az információ-visszakeresés területén. Osztályozási feladatra először Hull adaptálta (1994), és azóta is sok kutatás foglalkozott vele, lásd pl. [Sebastiani, 2002]. Az eljárás feltételezi, hogy minden attribútum valós típusú. Minden c osztályhoz megalkotunk egy *prototípusvektor*t, amit a c osztályba tartozó tanítópontok átlagaként számítunk ki (centroid), és ehhez hasonlítjuk az ismeretlen osztályba tartozó (vagy tesztelésre használt) objektum vektorát. Az osztályozandó objektum és egy kategória prototípusvektorának távolságát koszinusz- vagy más távolságmértékkel számolhatjuk.

A módszernek kicsiny a számításigénye, ezért a tanulás nagyon gyors. Hátránya viszont, hogy rossz eredményt ad, ha az egy osztályba tartozó pontok nem jellemezhetők egy vektorral (pl. amőba alapú osztályok, vagy az objektumok két, egymástól jól elkülönülő csoportja tartozik egyazon osztályhoz). Ezt szemlélteti a 4.7 ábra. Az üres körök az első, a feketével töltött körök a második osztályba tartoznak. Az üres körök osztályának prototípusvektora távol esik a tanítópontok közt ténylegesen szereplő üres köröktől. Ezért az $\vec{x}$-szel jelölt osztályozandó pontot a Rocchio az üres körök osztályba sorolná a teli körök osztálya helyett.

A módszer hatékonysága lényegesen javítható, ha a prototípusvektorok


X

megalkotásánál a negatív tanulóadatokat[5] is figyelembe vesszük. Ekkor a

$$\vec{c} = w \cdot \frac{1}{|\mathcal{T}_c|} \sum_{t_j \in \mathcal{T}_c} \vec{t_j} - \gamma \cdot \frac{1}{|\mathcal{T} \setminus \mathcal{T}_c|} \sum_{t_j \in \mathcal{T} \setminus \mathcal{T}_c} \vec{t_j} \tag{4.5}$$

képlettel számítható a c osztály prototípusvektora. A képletben $\mathcal{T}_c$ a c osztályba tartozó tanítópéldákat jelöli, w és γ pedig az adatbányász felhasználó által választott paraméterek.[6] Ha a második tagban nem az összes negatív tanulópontok, hanem csak a majdnem pozitív tanulópontok átlagát vesszük — ezek ugyanis azok, amelyektől a legnehezebb megkülönböztetni a pozitív tanulóadatokat, akkor további lényeges hatékonysági javulás érhető el [Schapire és tsa., 1998, Wiener és tsa., 1995].

4.3.4. Lineáris regresszió

A lineáris regresszió számos adatbányászati módszer alapja. A lineáris regresszió abban az esetben használható, ha minden attribútum valós típusú, beleértve a magyarázandó attribútumot (osztályváltozót) is.

Feltételezzük, hogy az $X_1, ..., X_n$ magyarázó változók és a magyarázandó Y változó között lineáris kapcsolat áll fenn. Úgy tekintjük tehát, hogy egy $\vec{x} = (x_1, ..., x_n)$ objektumhoz tartozó magyarázott változó y-nal jelölt értéke a

$$\widehat{y} = w_0 + \sum_{j=1}^{n} x_j w_j$$

összefüggéssel becsülhető, ahol $\widehat{y}$ az y becslését jelöli. Ha a korábbiakhoz hasonlóan felveszünk egy extra változót, X_0-t, melynek értéke minden pontra $x_0 = 1$, akkor a vektoros felírást használva tovább egyszerűsíthetjük a képletet:

$$\widehat{y} = \vec{x}^T \vec{w},$$

ahol a T felsőindex a mátrix transzponálását jelöli. A $\vec{w}$ oszlopvektort kell meghatároznunk úgy, hogy adott tanítópontok mellett négyzetes hibaösszeg minimális legyen. Ahogy korábban említettük, az adatbázis i-dik tanítópontját (pontosabban annak magyarázó attribútumainak értékeiből álló vektort) $\vec{x}^i$-vel, az i-dik tanítóponthoz tartozó magyarázandó váltotó értékét y^i-vel jelöljük. A minimalizálandó négyzetes hibaösszeget tehát így írhatjuk fel:

$$E(\vec{w}|\mathcal{T}) = \sum_{i=1}^{|\mathcal{T}|} (y^i - (\vec{x}^i)^T \vec{w})^2.$$

[5] Ebben a kontextusban negatív tanulópontoknak a tanítóhalmaz c-től különböző osztályba tartozó objektumait tekintjük, pozitívnak pedig a c osztályba tartozó tanítópontokat.

[6] A pontok centroidjaként számolt prototípusvektort a $w = 1$, $\gamma = 0$ paraméterek mellett kapjuk meg.

Adott tanítóhalmaz esetén E a $\vec{w}$ függvénye. Az $E(\vec{w}|\mathcal{T})$ függvény a w-ban négyzetes, így minimuma mindig létezik és egyértelmű. Amennyiben a tanítópontokat egy $|\mathcal{T}| \times n$-es $\mathbf{X}$ mátrixszal ábrázoljuk (egy tanítópontnak a mátrix egy sora felel meg), a magyarázandó változó tanítópontokhoz tartozó értékeit pedig az $\vec{y}$ oszlopvektorral jelöljük, akkor a fenti függvényt átírhatjuk más formába:

$$E(\vec{w}|\mathcal{T}) = (\vec{y} - \mathbf{X}\vec{w})^T(\vec{y} - \mathbf{X}\vec{w})$$

Ennek $\vec{w}$ szerinti deriváltja:

$$-2\mathbf{X}^T\vec{y} + 2\mathbf{X}^T\mathbf{X}\vec{w}$$

Ha a deriváltat egyenlővé tesszük nullával, akkor egyszerűsítés után a következőhöz jutunk:

$$\mathbf{X}^T(\vec{y} - \mathbf{X}\vec{w}) = 0$$

amelyből nemszingularitást feltételezve kapjuk, hogy

$$\widehat{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\vec{y},$$

azaz $\widehat{w} = \mathbf{X}^+\vec{y}$, ahol a $\mathbf{X}^+$ az $\mathbf{X}$) mátrix Moore-Penrose-féle pszeudoinverzét jelöli.

A gyakorlatban a mátix invertálása nagy mátrixok esetén nem mindig célszerű, mert túl sok ideig tarthat, ezért a w súlyok meghatározásához a Perceptron algoritmus egy változatát szokták használni. Ekkor az iteratív algoritmusban a w-t az nem eredeti $\vec{w} = \vec{w} + \vec{t}$ illetve $\vec{w} = \vec{w} - \vec{t}$ lépésekben módosítják, hanem a

$$\vec{w} = \vec{w} + \epsilon(y^t - \vec{t}^T\vec{w})\vec{w}$$

lépésben, ahol t egy tanítópont, azaz a $\mathcal{T}$ tanítóadatbázis egy objektuma, y^t a (numerikus) magyarázandó változó értéke a t objektum esetén, $\vec{t}^T\vec{w}$ a magyarázó változó aktuális $\vec{w} = (w_0, w_1, ..., w_n)$ súlyok által becsült értéke, ϵ a tanulási együttható. A regresszió esetében a magyarázott változó folytonos, így az algoritmusban nem ágazunk el aszerint, hogy első osztálybeli pontot soroltunk-e a második osztályba, vagy fordítva, hanem egyszerűen a fenti képletet használjuk, amikor a magyarázott változó az aktuálisan becsült értéke különbözik annak valós értékétől, y^t-től. Az $\vec{w}$-vel való szorzásnak köszönhetően, mivel $\vec{w}$ komponensei pozitívak és negatívak is lehetnek, épp a jó irányba mozdítjuk el a hipersíkot.

Vegyük észre, hogy az előbbi algoritmus valójában az $E(\vec{w}|\mathcal{T})$ egy lokális optimumát keresi adott $\mathcal{T}$ tanítóadatbázis mellett. Mivel az $E(\vec{w}|\mathcal{T})$ hibafüggvény konvex és optimuma egyértelmű, azaz egyetlen lokális optimummal

rendelkezik, amely egyben a globális optimum is, ezért az algoritmus megfelelően megválasztott iterációszám és ϵ mellett a hibafüggvény optimumához tartozó $w_0, w_1, ..., w_n$ súlyokat találja meg jó közelítéssel.

Lineáris regresszióra visszavezethető számos nemlineáris kapcsolat is. Például az $y = ax_1^b$ alakú összefüggés esetén y lineárisan függ x_1^b-től. Előfeldolgozási lépésként tehát x_1-t a b-dik hatványra emeljük, azaz az adatbázis összes objektuma esetében lecseréljük x_1 értékét x_1^b-re és ezt követően használhatjuk a lineáris regessziót.

4.3.5. Logisztikus regresszió

Ha a lineáris regressziót osztályozásra akarjuk használni (de a magyarázó változók továbbra is valós számok), akkor az egyes osztályoknak egy-egy valós számot kell megfeleltetnünk. Bináris osztályozásnál a 0-t és az 1-t szokás használni. Ezzel azonban nem oldottuk meg a problémát. A lineáris regresszió egy tanítópont osztályozásnál egy számot fog előállítani és a hibát a tanítópont ettől a számtól vett különbségével definiálja. Tehát egyes típusú tanítópont esetén ugyanakkora lesz a hiba 0 és 2 kimenetek esetén, ami nem túl jó.

Egy $\vec{x}$ oszlopvektorral leírt pont osztályának felismerésénél meg kell határoznunk az $\vec{x}^T w$ értéket. Amennyiben ez nagyobb, mint 0.5, akkor az 1-eshez tartozó osztály a felismerés eredménye, ellenkező esetben pedig a 0-hoz tartozó osztály. Az egyszerűség kedvéért jelöljük az $\vec{x}^T\vec{w} - 0.5$ értéket $\widehat{y}$-nal. A jelölés további egyszerűsítése érdekében, a korábbiakhoz hasonlóan most is vezessünk be egy X_0 extra attribútumot, amelynek értéke az összes adatbázisbeli objektumra $x_0 = 1$, így $w_0 = -0.5$ mellett az $\vec{x}^T\vec{w}$ szorzatot jelöljük $\widehat{y}$-nal.

Az a függvény, amely nullánál kisebb értékekre 0-át ad, nagyobbakra pedig 1-et, eléggé hasonlít az előjel (szignum) függvényre. Ha megengedjük, hogy értelmetlen eredményt kapjunk $\widehat{y} = 0$ esetében – amelyet értelmezhetünk úgy, hogy az osztályozó nem képes dőnteni –, akkor az osztályozó által előrejelzett osztályt megkaphatjuk az

$$\frac{1 + \mathrm{sgn}(\widehat{y})}{2} \tag{4.6}$$

kiszámításával.

Ha így definiáljuk a kimenetet, akkor a hiba definíciója is megváltozott és az előző fejezetben látott lineáris regresszió nem használható a w vektor meghatározásához. Kérdés, hogy tudunk-e árnyaltabb kimenetet adni, mint pusztán egy osztályindikátor (nulla vagy egy)?

Minél közelebb vagyunk az osztályok határához, annál bizonytalanabbak vagyunk a döntést illetően. Tehát természetes, hogy az osztályok előrejelzése

helyett az osztály előfordulásának esélyét[7] becsüljük. Ehhez csak annyit kell tennünk, hogy a 4.6 függvényt „lesimítjuk", azaz egy olyan $f(\widehat{y})$ függvénnyel helyettesítjük, amely

1. értéke 1-hez közelít, ha $\widehat{y}$ tart végtelenhez,
2. értéke 0-hoz közelít, ha $\widehat{y}$ tart mínusz végtelenhez,
3. $f(0) = 0.5$,
4. szimmetrikus nullára nézve, tehát $f(\widehat{y}) + f(-\widehat{y}) = 1 = 2f(0)$,
5. differenciálható minden pontban és
6. monoton növekvő.

Az ilyen függvényeket nevezzük *szigmoid* függvényeknek.

Sok függvény teljesíti a fenti elvárásokat. Könnyű belátni, hogy az

$$f_a(\widehat{y}) = 1/(1 + a^{-\widehat{y}})$$

függvények minden $a > 1$ esetében megfelelnek az elvárásoknak. Amennyiben $a = e$, akkor az ún. *logisztikus függvényt* kapjuk

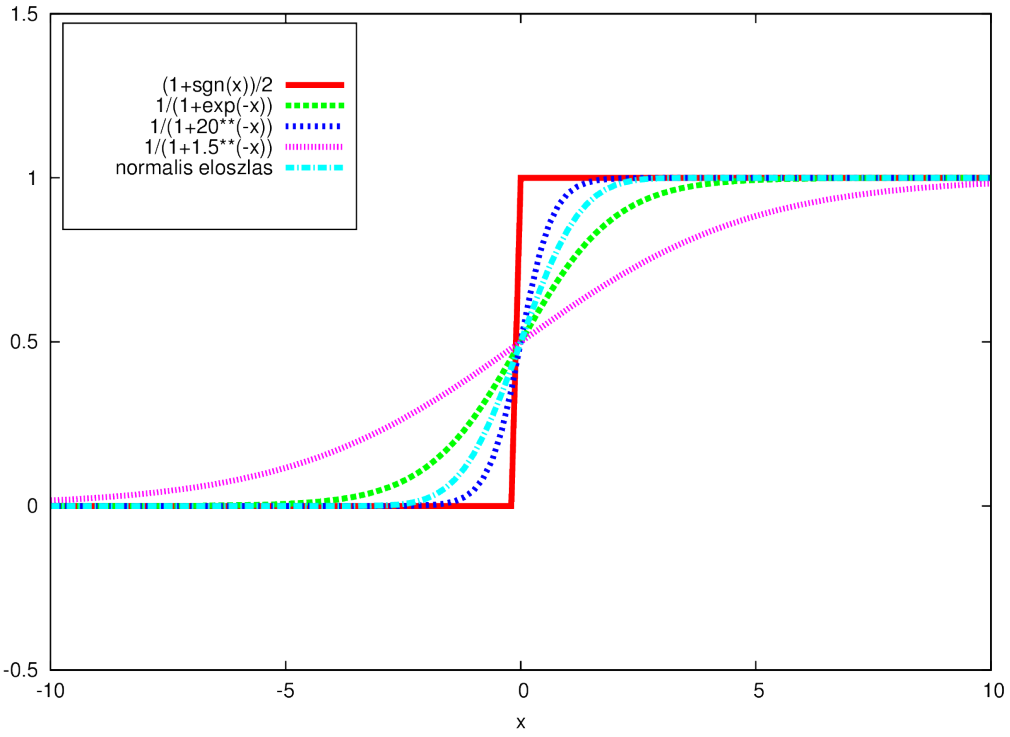
$$f_e(\widehat{y}) = \widehat{\mathbb{P}}(Y = 1|\vec{X}) = \frac{1}{1 + e^{-\widehat{y}}}, \tag{4.7}$$

A logisztikus függvény inverzét, $ln(\frac{x}{1-x})$, *logit függvénynek* hívjuk. A logisztikus függvény szépsége, hogy a deriváltja $f(\widehat{y})(1-f(\widehat{y}))$, amely a mi esetünkben $\mathbb{P}(Y = 1|\vec{X})\mathbb{P}(Y = 0|\vec{X})$-el egyezik meg.

A fenti feltételeket további függvények is teljesítik. Valószínűségi változók eloszlásfüggvénye is nullából indul mínusz végtelenben és egyhez tart a végtelenben. A harmadik és negyedik feltétel ($f(0) = 0.5$, $f(\widehat{y}) + f(-\widehat{y})$) megkívánja, hogy a sűrűségfüggvény szimmetrikus legyen, azaz az $f'(x) = f'(-x)$ teljesüljön minden x valós számra. A nulla várható értékű normális eloszlás eloszlásfüggvénye megfelel a feltételeknek.

A (4.6) szerinti "szögletes" előjelfüggvény "simított" változatát, a $1/(1 + a^{-\widehat{y}})$ típusú függvényeket különböző a-kra és a normális eloszlás eloszlásfüggvényét a 4.8 ábra mutatja.

[7]Szándékosan nem használjuk a valószínűség szót, mert nem matematikai értelemben vett valószínűségről van szó.



4.8. ábra. Az eltolt előjelfüggvény és néhány „simítása”

Ezzel el is jutottunk a logisztikus regresszió feladatához. Szemben a lineáris regresszióval, lineáris kapcsolat nem az $\vec{X}$ és a magyarázadó Y változók között van, hanem $\ln(\frac{\mathbb{P}(Y=1|\vec{X})}{1-\mathbb{P}(Y=1|\vec{X})})$ és $\vec{x}^T\vec{w}$ között, tehát

$$\mathbb{P}(Y=1|\vec{X}=\vec{x}) = \frac{1}{1+e^{-\vec{x}^T\vec{w}}}, \tag{4.8}$$

$$\mathbb{P}(Y=0|\vec{X}=\vec{x}) = 1-\mathbb{P}(Y=1|\vec{X}=\vec{x}) = \frac{e^{-\vec{x}^T\vec{w}}}{1+e^{-\vec{x}^T\vec{w}}}. \tag{4.9}$$

A következőkben azzal foglalkozunk, hogyan határozzuk meg a $\vec{w}$ azon értéket, amelyik a legkisebb hibát adja, $\vec{w}$ ilyen értékét jelöljük $\vec{w}^*$-gal.

Sajnos a $\vec{w}^*$ meghatározására nincs olyan szép zárt képlet, mint ahogy a lineáris regresszió esetében volt. Iteratív, közelítő módszert használhatunk, amely gradiensképzésen alapul. A hiba minimalizálása helyett a feltételes valószínűségeket maximalizáljuk:

$$\vec{w}^* \leftarrow argmax_{\vec{w}} \sum_{i=1}^{|\mathcal{T}|} \ln \mathbb{P}(Y=y^i|\vec{X}=\vec{x}^i,\vec{w}).$$

A fenti képletben a regressziós függvény a szokásos $\mathbb{P}(Y = y^i|\vec{X} = \vec{x}^i)$ helyett $\mathbb{P}(Y = y^i|\vec{X} = \vec{x}^i, \vec{w})$, hiszen $\vec{w}$ most nem mint konstans szerepel, hanem mint változó: azon $\vec{w}$-t keressük, amelyre $\sum_{i=1}^{|\mathcal{T}|} \ln \mathbb{P}(Y = y^i|\vec{X} = \vec{x}^i)$ értéke maximális (lásd még a 4.8. és 4.9. egyenlőségeket).

Felhasználva, hogy az y csak nulla vagy egy értéket vehet fel, a maximálandó függvényt átírhatjuk:

$$l(\vec{w}) = \sum_{i=1}^{|\mathcal{T}|} \big(y^i \ln \mathbb{P}(Y = 1|\vec{X} = \vec{x}^i, \vec{w}) + (1 - y^i) \ln \mathbb{P}(Y = 0|\vec{X} = \vec{x}^{\,i}, \vec{w})\big).$$

Az iteratív algoritmus során kiindulunk valamilyen szabadon megválasztott $\vec{w}^{(0)}$ vektorból, majd a k-adik lépésben a $\vec{w}^{(k-1)}$ vektorhoz hozzáadjuk a $\frac{\delta l(\vec{w})}{\delta \vec{w}}$ vektor λ-szorosát, így megkapjuk a $\vec{w}^{(k)}$ vektort. A λ egy előre megadott konstans, amelynek értékét tipikusan kicsire, például 0.01-re, szokták állítani. A $\frac{\delta l(\vec{w})}{\delta \vec{w}}$ vektor a $\frac{\delta l(\vec{w})}{\delta w_j}$ parciális deriváltakból áll:

$$\frac{\delta l(\vec{w})}{\delta w_j} = \sum_{i=1}^{|\mathcal{T}|} x_j^i \big(y^i - \mathbb{P}(Y = 1|\vec{X} = \vec{x}^i, \vec{w})\big),$$

ahol $\mathbb{P}(Y = 1|\vec{X} = \vec{x}^i, \vec{w})$ a logisztikus regresszió által adott becslés (lásd 4.8. egyenlőséget). Az $y^i - \mathbb{P}(Y = 1|\vec{X} = \vec{x}^i, \vec{w})$ tag a hibát jelenti, amely meg van szorozva egy nagyság jellegű tényezővel: az x_j^i érték adja meg a becslésben a w_j szerepének nagyságát.

Az $l(w)$ konkáv ezért a gradiens módszer a globális maximumhoz fog konvergálni. A gyakorlat azt mutatja, hogy a konvergencia igen gyors, a w_j értékek néhány iteráció után már alig-alig változnak.

A lineáris regresszióról szóló korábbi fejezetben említettük, hogy a lineáris regresszióhoz tartozó $\vec{w}$ súlyvektort a Perceptron algoritmus módosított változatával is kiszámolhatjuk. Vegyük észre, hogy ha a most emített gradiens módszert alkalmazzuk az optimális $\vec{w}$ súlyvektor megtalálására, minimális különbségtől eltekintve ugyanarra az algoritmusra jutunk, mint lineáris regresszió esetén. Ez a minimális különbség abban áll, hogy míg a Perceptron algoritmus módosított változata egyesével tekinti a tanítóadatbázis pontjait (illetve a magyarázott változó aktuális $\vec{w}$ vektor melletti becslésekor elkövetett hibát), addig a fenti eljárásban a gradiens számításakor az összes tanítópontot tekintjük (összesített hibával dolgozunk).

Logisztikus regresszió általános osztályozásnál, one-versus-all technika

Az eddigiekben bináris osztályozással foglalkoztunk. Mi a teendő, ha a magyarázandó attribútum diszkrét és $k > 2$ féle különböző értéket vehet fel?

A többválaszú logisztikus regresszió (multiresponse/multinomial logistic regression) k darab osztály esetén k-szor alkalmazza a logisztikus regressziót. Veszi az első osztályt és a többit egy kalap alá vonva végrehajt egy logisztikus regressziót, mely ad egy valószínűséget.[8] Ezután a második osztályt emeli ki és az összes többi osztályt vonja egy kalap alá. Így az összes osztáyhoz meg tud határozni egy valószínűséget (bizonyosságot), mintha csak egy tagsági függvényt próbálna meghatározni.

Új objektum osztályozásánál az osztályzó arra az osztályra teszi le a voksát, amelyik a legnagyobb valószínűséget kapta. Ha csak a felismert (előrejelzett) osztály érdekel minket és a kapott valószínűségre nem vagyunk kíváncsiak, akkor az új objektum osztályozása során nem szükséges a logisztikus függvény alkalmazása, mivel a logisztikus függvény monoton növekvő. Ezt az eljárást *one-versus-all* technikának hívják az angol irodalomban [Rifkin és Klautau, 2004]. A one-versus-all eljárás nem csak logisztikus regresszió esetén alkalmazható: segítségével többosztályos osztályozási feladatokat oldhatunk meg bármely olyan bináris osztályozó algoritmussal, amely képes arra, hogy az egyik illetve másik osztályba tartozás bizonyosságát jellemző folytonos kimenetet adjon. A 4.13.1. fejezetben további lehetőségeket is fogunk látni arra, hogyan vezethetünk vissza a többosztályos osztályozási feladatokat kétosztályos (bináris) osztályozási feladatokra.

A one-versus-all technika logisztikus regresszióra történő alkalmazásakor a kapott "valószínűségek" összege nem feltétlenül egy, mivel az osztályozók függetlenek. Ezen a következőképpen segíthetünk: a fenti módszer helyett csak $k-1$ darab $\vec{w}^{\ell}$ vektort állítsunk elő úgy, hogy minden $\ell = 1, \ldots, k-1$-re

$$\widehat{\mathbb{P}}(Y = \ell | \vec{X} = \vec{x}) = \frac{e^{-\vec{x}^T \vec{w}^{\ell}}}{1 + \sum_{\ell'=1}^{k-1} e^{-\vec{x}^T \vec{w}^{\ell'}}},$$

valamint

$$\widehat{\mathbb{P}}(Y = k | \vec{X} = \vec{x}) = \frac{1}{1 + \sum_{\ell'=1}^{k-1} e^{-\vec{x}^T \vec{w}^{\ell'}}}.$$

A gradiens módszernél alkalmazott vektor – amely λ-szorosát hozzá kell adni az aktuális $\vec{w}^{(\ell)}, 1 \leq \ell \leq k-1$ vektorhoz – a következő komponensekből áll:

$$\frac{\delta l(\vec{w}^1, \vec{w}^2, ..., \vec{w}^{k-1})}{\delta w_j^{\ell}} = \sum_{i=1}^{|\mathcal{T}|} x_j^i \big(\delta(y^i = \ell) - \widehat{\mathbb{P}}(Y = y_{\ell} | \vec{X} = \vec{x}^i, \vec{w}^{\ell})\big), \qquad (4.10)$$

ahol $\delta(y^i = \ell) = 1$, ha az i-edik tanítópont osztálya ℓ, különben 0.

[8] Egy adott osztály valószínűsége alatt ebben a fejezetben egy folytonos számot értünk, amely annál nagyobb, minél nagyobb a bizonyosságunk afelől, hogy egy adott objektum az adott osztályba tartozik.

A logisztikus regresszió és a Bayes osztályozó kapcsolatáról a 4.7 részben szólunk.

4.4. Mesterséges neurális hálózatok

A logisztikus regresszió 4.9. ábrán látható modelljét egyrétegű mesterséges neurális hálózatnak is nevezik. Ez az alapja a „komolyabb" mesterséges neurális hálózatóknak, melyek – némileg az agyműködést utánzó biológiai analógiára is támaszkodva – logisztikus regressziók kapcsolatai.

A neurális hálók kifejezőereje nagyobb a lineáris modelleknél, melyet az alábbi példán szemléltetünk. Amint láttuk, a lineáis szeparációt végző modellek de nem képesek megtanulni az *xor* függvényt. Három logisztikus regresszió felhasználásával azonban az *xor*-t is ki tudjuk fejezni. Idézzük fel, hogy az *és* függvényt (jele:$\wedge$) az $x_1 + x_2 - 1.5 = 0$, a *vagy* függvényt (jele: $\vee$) az $x_1 + x_2 - 0.5 = 0$ egyenes, a *nem*-et (jele: felülvonás) pedig az $x_2 - 0.5 = 0$ egyenesek szeparálják (lásd a 4.6 ábra). Az *xor* függvény pedig felírható, mint $(x_1 \vee x_2) \wedge \overline{(x_1 \wedge x_2)}$. A 4.10 ábra ezt a konstrukciót mutatja. A felső szignum függvényhez tartozó logisztikus regresszió az *és*-t, a bal alsó a *vagy*-ot a jobb alsó pedig a *nem és*-t adja vissza.

Az építőelemeket ismerve tetszőleges logikai formulát kifejezhetünk logisztikus regressziók összekapcsolásával, ezért a logisztikus regressziók kapcsolata univerzális függvényapproximátor.

A legnépszerűbb modell a többrétegű előrecsatolt neuronhálózat (lásd 4.11. ábra). Az első réteg csomópontjai az bemeneti (input) neuronok, melyek a magyarázó változóknak felelnek meg (1, 2, 3-mal címkézett. neuronok a 4.11. ábrán). Az outputot (magyarázott változókat) a legutolsó réteg kimenete (6. neuron) adja. A közbenső rétegeket rejtett (hidden) rétegeknek (4-5. neuronok) nevezzük. Minden réteg minden neuronjának kimenete a következő réteg összes neuronjának bemenetével kapcsolatban áll. A kapcsolat szorosságát w_{ij} súlyok jellemzik.

Az bemeneti réteg neuronjai (1-3. neuronok a 4.11. ábrán) nem végeznek műveleteket, csupán a magyarázó változókat reprezentálják. A 4.11. ábrán látható 4-6. neuronok egyenként a 4.9. ábrán szereplő logisztikus regressziót végzik.

Mind a logisztikus regresszió, mind a neurális hálózatok nemlineáris függvényapproximátornak tekinthetők. A tapasztalatok és az elméleti eredmények (lásd [Futó, 1999]) szerint is ugyanannyi paramétert (súlyt) használva nemlineárisan paraméterezett függvényekkel gyakran jobb közelítést érhetünk el, mint lineárisan paraméterezett társaikkal.

Az alkalmas súlyokat nemlineáris optimalizációs technikával, gradiens mód-

szerrel kereshetjük meg gyakorlatilag ugyanúgy, mint a logisztikus regressziónál tettük.

A többrétegű előrecsatolt neuronhálózatok esetében az előrecsatolt topológiának köszönhetően az *egész* neuronháló hibafüggvényének w súlyok szerinti gradiensét könnyen kiszámíthatjuk. A súlyok megtalálása a tanító példák alapján az ún. backpropagation (hiba visszaterjesztés) eljárás szerint zajlik [Tan és tsa., 2005]. A Perceptron algoritmushoz hasonlóan egyesével végigmegyünk (akár többször is) a $\mathcal{T}$ tanítóadatbázis t objektumain. Ennek során:

1. Az inputokból előrehaladva kiszámítjuk az egyes neuronok outputjait.

2. Az utolsó output rétegből kiindulva, rétegről rétegre visszafelé haladva a megfelelő backpropagation szabály szerint módosítjuk $w_{i,j}$ értékeket, úgy, hogy a módosítás hatására a t-re vonatkozó hiba csökkenjen.

Vegyük észre, hogy a Percepton algoritmus a backpropagation algoritmus azon egyszerű esete, amikor a neurális háló egyetlen rejtett réteggel sem rendelkezik.

Mivel a neuronháló által reprezentált függvénynek lehetnek lokális maximumai ezért a módszer nem biztos, hogy a globális optimumot adja. A backpropagation eljárást ezért többször szokás futtatni különböző kezdeti paraméterekkel.

A neuronhálók hátránya, hogy a súlyok rendszere közvetlenül nem értelmezhető emberek számára: legtöbbször nem tudjuk szemléletesen indokolni, hogy mi alapján hozta meg a neuronháló a döntést. A neuronháló tehát egy fekete doboz a felhasználó szemszögéből. A magyarázó értelmezés hiánya sok területen elfogadható (gondoljunk példaként arra, amikor azt szeretnénk meghatározni, egy milyen termékeket reklámozzunk egy felhasználónak, amikor következő alkalommal belép egy webes áruházba). Más esetekben azonban a magyarázat hiánya korlátozza a neuronhálók alkalmazását. Léteznek ugyanakkor olyan eljárások, amelyek a neuronhálók súlyaiból emberek számára érthető, a döntéseket indokló szabályokat nyernek ki [Han és Kamber, 2006].

Egy városi legenda szerint a 80-as években az amerikai hadsereg szolgálatba akarta állítani a mesterséges intelligenciát. Céljuk volt minden tankra egy kamerát tenni, a kamera képét egy számítógépnek továbbítani, amely automatikusan felismeri az ellenséges tankot. A kutatók neurális hálózat alapú megközelítés mellett döntöttek. A tanításhoz előállítottak 100 darab olyan képet amelyen a fák mögött tank bújt meg és 100 olyat, amelyen tank nem volt látható. Néhány iteráció után a hálózat tökéletesen osztályozta a képeket. A kutatók nagyon meg voltak elégedve, ugyanakkor még maguk sem voltak biztosak abban, hogy a neurális hálózat valóban a tank fogalmát tanulta-e meg. Független szakértőktől kért verifikáció során azonban a háló rosszul szerepelt:

nem osztályozott pontosabban, mint egy teljesen véletlenszerűen tippelő osztályozó. Későbbi vizsgálatok során kiderült, hogy a tanításhoz használt összes tankos képen borult volt az idő, a tank nélküli képeken pedig sütött a nap.

Nem tudni, hogy a történet mennyiben igaz, az azonban tény, hogy a neurális háló nem ad magyarázatot az osztályozás okára. Ez komoly hátrány például a pénzügyi vagy orvosi világban.

4.5. Döntési szabályok

A `HŐMÉRSÉKLET = magas AND SZÉL = nincs` $\rightarrow$ `IDŐ_JÁTÉKRA alkalmas` egy döntési szabály, amely azt fejezi ki, hogy ha magas a hőmérséklet és nincs szél, akkor az idő alkalmas kültéri játékra.

A *valószínűségi döntési szabályokban* a következményrészben az osztályattribútumra vonatkozó valószínűségi eloszlás szerepel. Ilyen szabályra példa az autóbiztosítás területéről, hogy `nem = férfi AND gyerek száma = 0 AND autó teljesítmény > 150LE` $\rightarrow$ `kockázatos = (80%, 20%)`.

A feltételrészben elvileg az `és`, `vagy`, valamint a negáció tetszőleges kombinációját felhasználhatjuk. A gyakorlatban első sorban olyan szabályokkal foglalkoznak, amelyben alapfeltételek illetve azok negációjának `"és"` kapcsolatai szerepelnek. Ha az azonos következményrésszel rendelkező szabályokból egy szabályt készítünk úgy, hogy a feltételek `"vagy"` kapcsolatát képezzük, akkor elmondhatjuk, hogy a szabályok feltételrészében diszjunktív normál formulák állnak. Minden ítéletlogikában megadott formula átírható diszjunktív normál formulává a dupla negáció eliminálásával, a de Morgan és a disztributivitási szabály alkalmazásával.

4.5.1. Definíció *Az $\mathcal{A}$ attribútumhalmaz felett értelmezett döntési szabály alatt egy $R : \phi(\mathcal{A}) \rightarrow Y = y$ logikai implikációt értünk, amelyek feltételrészében szereplő ϕ egy logikai formula, amely az $\mathcal{A}$-beli attribútumokra vonatkozó feltételek logikai kapcsolataiból áll. A szabály következményrészében az osztályattribútumra (magyarázott változóra) vonatkozó ítélet szerepel.*

4.5.2. Definíció *Az $R : \phi(\mathcal{A}) \rightarrow Y = y$ szabályra* illeszkedik *a t objektum, ha a feltételrész attribútumváltozóiba a t megfelelő értékeit helyettesítve igaz értéket kapunk.*

Amennyiben a szabály következményrésze is igazra értékelődik az objektumon, akkor a szabály *fennáll* vagy *igaz* az objektumon. Például az alábbi objektum

HŐMÉRSÉKLET	PÁRATARTALOM	SZÉL	ESŐ	IDŐ_JÁTÉKRA
magas	alacsony	nincs	van	alkalmatlan

illeszkedik a

`HŐMÉRSÉKLET = magas AND SZÉL = nincs` $\rightarrow$ `IDŐ_JÁTÉKRA alkalmas`

szabályra.

4.5.3. Definíció *Az $R : \phi(\mathcal{A}) \rightarrow Y = y$ szabály* lefedi *az T objektumhalmazt, ha minden objektum illeszkedik a szabályra. Adott $\mathcal{T}$ tanító halmaz esetén az R által fedett tanítópontok halmazát $cover_{\mathcal{T}}(R)$-rel jelöljük.*

Helyesen fedi a T halmazt az $R : \phi(\mathcal{A}) \rightarrow Y = y$ szabály, ha R fedi T-t és a halmaz összes objektuma az y osztályba tartozik. Ha R fedi ugyan T-t, de a halmaz nem minden objektuma tartozik az y osztályba, *helytelen fedésről* vagy egyszerűbben rossz osztályozásról beszélünk. A fenti példában helytelen fedéssel van dolgunk, hiszen az objektum nem a következményrész szerinti osztályba tartozik. A $cover_{\mathcal{T}}$-ben az R által helyesen fedett pontok halmazát $cover^{+}_{\mathcal{T}}(R)$-rel jelöljük, a helytelenül fedettekét pedig $cover^{-}_{\mathcal{T}}(R)$-rel.

4.5.4. Definíció *Az R szabály* relatív fedési hibája *megegyezik a rosszul osztályozott objektumok számának a fedett tanítóobjektumokhoz vett arányával, tehát*

$$Er_{\mathcal{T}}(R) = \frac{cover^{-}_{\mathcal{T}}(R)}{cover_{\mathcal{T}}(R)}.$$

Döntési szabályok kifejezőereje

Kifejezőerejük szempontjából a döntési szabályok következő típusairól beszélünk:

Ítételkalkulus-alapú döntési szabály – A feltételrészben predikátumok logikai kapcsolata áll (ítéletkalkulus egy formulája, amelyben nem szerepelnek a $\rightarrow$ és $\longleftrightarrow$ műveleti jelek). Minden predikátum egy attribútumra vonatkozik. Amennyiben az attribútum kategória típusú, akkor $A = a$ vagy $A \in \mathcal{A}$ alakú a feltétel, ahol a egy konstans, $\mathcal{A}$ pedig az A értékkészletének egy részhalmaza. Sorrend vagy intervallum típusú attribútum esetében emellett $A \leq a$ és $a' \leq A \leq a''$ szabályokat is megengedünk.

Számos algoritmus létezik döntési szabályok kinyerésére. A legismertebbek a tanítóhalmaz mintáinak iteratív fedésén alapulnak: az algoritmus először egy nagy fedésű szabályt keres, majd az ezt követő iterációkban olyan szabályokat, amelyek a még lefedetlen vagy az eddigi szabályokkal rosszul osztályozott objektumok minél nagyobb részét fedik. [Tan és tsa., 2005]. Az algoritmusok többsége csak olyan egyszerű formulákat tud előállítani, amelyekben a predikátumok és kapcsolataik

állnak, például `MAGASSÁG` ≤ 170 `AND HAJSZÍN = barna AND SZEMSZÍN` $\in$ `{kék, zöld}`.

A csak ítéletkalkulus-alapú szabályokat tartalmazó döntési szabályokat *univariate* (egyváltozós) döntési szabályoknak hívjuk.

Reláció-alapú döntési szabály – Halmazelméleti szempontból az attribútumokra vonatkozó predikátumok bináris relációk, melyek egyik tagja egy változó, másik tagja egy konstans. A reláció-alapú döntési szabályokban a második tag attribútumváltozó is lehet. Itt például a `hajszín = szemszín` vagy a `szélesség < magasság` megengedett feltételek. A reláció-alapú szabályokat tartalmazó döntési szabályokat/fákat multivariate (tőbbváltozós) döntési szabályoknak/fáknak hívjuk. A reláció alapú döntési szabályoknak nem nagyobb a kifejező erejük, amennyiben az attribútumok értékészlete véges. Ekkor ugyanis egy relációs szabály helyettesíthető sok egyváltozós szabálypárral. A fenti példa megfelelője a `hajszín = barna AND szemszín = barna`, `hajszín = kék AND szemszín = kék`, `hajszín = mályva AND szemszín = mályva` szabályokkal.

Induktív logikai programozás Példaként tegyük fel, hogy egy építőelemet állónak nevezünk, ha a szélessége kisebb, mint a magassága. Tegyük fel továbbá, hogy építőelemek egy listáját toronynak hívjuk, amelynek legfelső elemére a `csúcs`, a maradék elemekre pedig a `maradék` attribútummal hivatkozunk. A `szélesség < magasság` $\rightarrow$ `ALAK = álló` szabályt úgy is írhatjuk, hogy `szélesség(építőelem) < magasság(építőelem)` $\rightarrow$ `álló(építőelem)`. Azt mondjuk, hogy egy torony álló, ha a torony minden építőelem álló:

```
szélesség(torony.csúcs) < magasság(torony.csúcs) AND
        álló(torony.maradék) → álló(torony).
```

Ez egy rekurzív kifejezés, amely szerint egy torony akkor álló, ha a legfelső elem magassága nagyobb a szélességénél és a maradék elem álló. A rekurziót le kell zárni: `torony = üres` $\rightarrow$ `álló(torony)`. A rekurzív szabályoknak nagyobb a kifejezőerejük, mint a reláció-alapú döntési szabályhalmaznak, hiszen kifejtve tetszőleges számú predikátumot tartalmazhatnak. A rekurzív szabályokat is tartalmazó szabályhalmazt *logikai programnak* nevezzük, ezekkel továbbiakban nem foglalkozunk.

4.5.1. Szabályhalmazok és szabálysorozatok

Szabályokon alapuló osztályozás esetén megkülönböztetjük a *szabályhalmazokat* és *szabályok sorozatát*. Halmazok esetén a szabályok függetlenek egymástól. A szabályhalmaz *egyértelmű*, ha tetszőleges objektum csak egy szabályra illeszkedik.

Sorozat esetében egy új objektum osztályozásakor egyesével sorra vesszük a szabályokat egészen addig, amíg olyat találunk, amelyre illeszkedik az objektum. Ennek a szabálynak a következményrésze adja meg az osztályattribútum (modell által becsült illetve előrejelzett) értékét.

Egy szabályrendszer (sorozat vagy halmaz) *teljes*, ha tetszőleges objektum illeszthető egy szabályra, vagyis az osztályozó minden esetben (tetszőleges osztályozandó elemre) döntést hoz. Sorozatok esetében a teljességet általában az utolsó, ún. *alapértelmezett* szabály biztosítja, amelynek feltételrésze üres, tehát minden objektum illeszkedik rá.

Szabálysorozat esetében nem kell beszélnünk egyértelműségről, hiszen több szabályra való illeszkedés esetén mindig a legelső illeszkedő szabály alapján osztályozunk. A szabályok közötti sorrend (vagy másképp prioritás) biztosításával kerüljük el azt a problémát, hogy milyen döntést hozzunk, ha egy objektumra több, különböző következményrésszel rendelkező szabály illeszkedik.

A sorrend létezésének ára van. Szabályhalmaz esetén ugyanis minden szabály a tudásunk egy töredékét rögzíti. Sorozatok esetében azonban egy szabályt nem emelhetünk ki a környezetéből; egy R szabály csak akkor süthető el, ha az R-et megelőző szabályok feltételrészei nem teljesülnek.

A szabályok sorozata átírható szabályok halmazába úgy, hogy egyesével vesszük a szabályokat az elsőtől és a feltételrészhez hozzáfűzzük az előtte álló szabályok feltételrészeinek negáltjainak *és* kapcsolatát. Az így kapott szabályhalmaz azonban túl bonyolult lesz. Sorozattal esetleg az összefüggés egy tömörebb, könnyebben értelmezhető formáját kapjuk.

4.5.2. Döntési táblázatok

Egy döntési táblázat minden oszlopa egy-egy attribútumnak felel meg, az utolsó oszlop az osztályattribútumnak. Az A attribútumhoz tartozó oszlopban az A értékére vonatkozó feltétel szerepelhet, leggyakrabban $A = a$ alakban (ítételkalkulus-alapú döntési szabály). A táblázat egy sora egy döntési szabályt rögzít. Ha az attribútumok a sorban szereplő feltételeket kielégítik, akkor az osztályattribútum értéke megegyezik a sor utolsó elemének értékével. A 4.1. táblázat egy döntési táblázatra mutat példát.

Egy döntési táblázat tulajdonképpen egy speciális döntési szabályhalmaz, amelyre igaz, hogy a szabályhalmazhoz tartozó összes szabály feltételrészében

időjárás	hőmérséklet	páratartalom	szél	játékidő?
napos	meleg	magas	nincs	nem
napos	meleg	magas	van	nem
borús	meleg	magas	nincs	nem
esős	enyhe	magas	nincs	igen
esős	hideg	magas	nincs	igen
esős	hideg	magas	nincs	igen
esős	hideg	magas	nincs	igen

4.1. táblázat. Egy döntési táblázat

pontosan ugyanazok az attribútumok szerepelnek.

Döntési táblák előállításánál a következő kérdéseket kell tisztázni:

1. Az attribútumok melyik részhalmazát érdemes kiválasztani? Ideális az lenne, ha minden részhalmazt ki tudnánk értékelni és kiválasztani azt, amelyik a legkisebb hibát (rosszul osztályozott tanítópontok száma) adja. Az attribútumok száma gyakran nagy, ekkor az összes részhalmaz kipróbálása túlságosan sok időbe telik.

2. Hogyan kezeljük a folytonos attribútumokat? A fenti példában a hőmérsékletet diszkretizáltuk. Meleg az idő, ha 25 foknál több van, alatta enyhe 5 fokig. Ha a hőmérséklet 5 fok alá megy, akkor hideg van. Ideális az lenne, ha a folytonos attribútumokat az algoritmus automatikusan tudná diszkretizálni. Az attribútumok összes lehetséges részhalmazának kipróbálásához hasonlóan az összes lehetséges diszkretizálás vizsgálata is túlságosan sok időbe telhet, a kombinatorikus robbanás veszélyét hordozza (nem feltétlenül elegendő ugyanis az egyes attribútumok lehetséges diszkretizálásai egymástól függetlenül vizsgálni, az összes lehetséges kombináció száma pedig általában nagy).

4.5.3. Az 1R algoritmus

A legegyszerűbb osztályzó algoritmusok egyike az 1R. Az algoritmus kiválaszt egy attribútumot és az osztályozásban kizárólag ezt használja. Annyi szabályt állít elő, ahány értéket felvesz a kiválasztott attribútum a tanítóhalmazban. Az $A = a \to Y = c$ szabály következményrészében szereplő c osztály a legtöbbször előforduló osztály az A attribútumában a értéket felvevő tanítóminták közül.

Nyilvánvaló, hogy az 1R egyértelmű szabályhalmazt állít elő abban az értelemben, hogy egy példányra mindig csak egy szabály illeszkedik, nem kell tehát a szabályok prioritásval foglalkoznunk.

Minden attribútumértékhez meg tudjuk határozni a rosszul osztályozott tanítópontok számát. Ha összeadjuk az A attribútum értékeihez tartozó rosszul osztályozott tanítópontok számát, akkor megkapjuk, hogy mennyi tanítópontot osztályoznánk rosszul, ha az A attribútum lenne a kiválasztott. A legkevesebb rosszul osztályozott tanítópontot adó attribútumot választjuk osztályzó attribútumnak. Hiányzó attribútumértékeket úgy kezeljük, mintha az attribútumnak lenne egy különleges, a többitől eltérő értéke.

Sorrend és intervallum típusú attribútumnál $A \leq a \rightarrow y$, $a' \leq A < a'' \rightarrow y$ és $a''' \leq A \rightarrow y$ típusú szabályokat célszerű előállítani, ahol y az osztályattribútum egy értékét jelöli. Ehhez csoportosítsuk az egymást követő értékeket úgy, hogy a hozzájuk tartozó osztályérték szempontjából homogén csoportokat hozzanak létre. Erre diszkretizálásként is hivatkozunk és az 1R során használt módszert az Előfeldolgozás fejezetben ismertettük (lásd 3.3.4 rész).

Habár a sorrend és intervallum típusú attribútum csoportosításán sokat lehet elmélkedni az 1R módszer nem túl bonyolult. Egyszerűsége ellenére a gyakorlatban sok esetben viszonylag jól működik: egy meglepő cikkben a szerzők arról írtak, hogy az 1R sokkal jobb osztályzó algoritmus, mint azt hinnénk [Holte, 1993]. A szerzők azon a 16 adatbázison értékelték ki a különböző osztályozó módszereket – köztük az 1R-t –, amelyeket a kutatók gyakran használnak cikkeikben. Az 1R zavarba ejtően jó helyen végzett, a pontosság tekintetében alig maradt el az újabb és jóval bonyolultabb eljárásoktól.

Az 1R nevében szereplő szám az osztályozás során felhasznált attribútum számára utal. Létezik 0R osztályozó is, amely nem használ fel egyetlen attribútumot sem. Az osztályozó ekkor egy feltétel nélküli szabály, amely ítéletrészében a leggyakoribb osztály áll. A 0R-t *triviális osztályozónak* is nevezhetjük.

Az 1R osztályozóra példa az alábbi:

$$\begin{aligned} \text{hőmérséklet} &= \text{meleg} \rightarrow \text{játékidő} = \text{nem} \\ \text{hőmérséklet} &= \text{enyhe} \rightarrow \text{játékidő} = \text{igen} \\ \text{hőmérséklet} &= \text{hideg} \rightarrow \text{játékidő} = \text{igen} \end{aligned}$$

4.5.4. A Prism módszer

A Prism módszer [Cendrowska, 1987] feltételezi, hogy a tanító adatbázisban nincs két olyan elem, amelynek a fontos magyarázó attribútumai megegyeznek, de más osztályba tartoznak. Ha mégis akadnak ilyen objektumok, akkor csak egyet tartsunk meg méghozzá olyat, amelyik a leggyakrabban előforduló osztályba tartozik. A leggyakoribb osztályt az azonos attribútumértékkel rendelkező pontok körében kell nézni.

A Prism módszer a *fedő módszerek* közé tartozik. A fedő algoritmus egyesével veszi az osztályattribútum értékeit és megpróbál olyan szabályokat előállí-

tani, amelyek helyesen fedik azon tanítópontokat, amelyek a vizsgált osztályba tartoznak. A szabályok előállításánál a feltételrészhez adunk hozzá egy-egy újabb részfeltételt törekedve arra, hogy olyan részfeltételt vegyünk, amely legnagyobb mértékben növeli a pontosságot. A módszer hasonlít a döntési fák előállítására (lásd következő fejezet). Ugyanakkor a döntési szabályoknál más a cél; a tanítópontokra való illeszkedés növelése helyett az osztályok közötti szeparációt szeretnénk maximalizálni.

A Prism menete a következő. Egyesével sorra vesszük az osztályattribútum értékeit. Minden értéknél kiindulunk egy olyan döntési szabályból, amelynek feltételrésze üres, következményrészében pedig az aktuális osztályérték szerepel. Minden lehetséges (A, a) párra – ahol A egy attribútum, a pedig az A egy lehetséges értéke – kiszámítjuk, hogy mennyi lenne a helytelenül osztályozott tanítópontok száma, ha az $A = a$ részfeltételt adnánk a feltételrészhez. Azt a részfeltételt választjuk, amely a legkisebb relatív fedési hibát adó szabályt eredményezi. A részfeltételek hozzáadását addig folytatjuk, amíg olyan szabályt kapunk, amelynek nem nulla a fedése, de nulla a relatív fedési hibája.

Ezután töröljük a tanítópontok közül azokat, amelyeket az újonnan előállított szabály lefed. Ha nincs több olyan tanítópont, amelynek osztályattribútuma az aktuális osztályértéket veszi fel, akkor a következő attribútumértéket vesszük a következményrészbe. Az algoritmus pszeudokódja alább látható:

Algoritmus Prism

Require: $\mathcal{T}$: tanítópontok halmaza,
Y: osztályattribútum változó,

for all $y \in$ osztályattribútum értékre **do**
  $E \leftarrow$ az y osztályba tartozó tanítópontok
  $\phi \leftarrow \emptyset$
  while $E \neq \emptyset$ **do**
    $R \leftarrow (\phi \rightarrow Y = y)$
    while $Er_{\mathcal{T}}(R) \neq 0$ **do**
      hiba $\leftarrow 1$
      for all (A, a) attribútum-érték párra **do**
        if $Er(\phi \text{ AND } A = a \rightarrow Y = y) < hiba$ **then**
          hiba $\leftarrow Er(\phi \text{ AND } A = a \rightarrow Y = y)$
          $A* \leftarrow A$
          $a* \leftarrow a$
        end if
      end for
      $\phi \leftarrow (\phi \text{ AND } A* = a*)$
    end while
    $\mathcal{T} \leftarrow \mathcal{T} \setminus cover(R)$
  end while
end for

A Prism algoritmus alkotta szabályokat szabálysorozatként célszerű értelmezni. A módszer mindig olyan szabályt hoz létre, amely lefed néhány tanítópontot. A következő szabály a maradék tanítópontokra szól ezért új objektum osztályozásakor akkor süssük el, ha az előző szabályt nem tudtuk illeszteni. A Prism algoritmusra, mint *separate and conquer* (*leválaszt majd lefed*) módszerre szoktak hivatkozni. A Prism először leválasztja a tanítópontok egy csoportját, majd megpróbálja lefedni azokat szabályokkal.

A Prism csak 100%-os pontosságú szabályokat állít elő. Az ilyen egzakt szabályok mindig a túltanulás veszélyét hordozzák magukban, amellyel a későbbiekben részletesen foglalkozni fogunk. Az ilyen szabályok sok feltételt tartalmaznak és általában kevés tanítópontot fednek. Hasznosabb lenne kisebb pontosságú, de több pontot fedő szabályokat előállítani. A tökéletességre való törekvés a Prism egy vitathatatlan hibája. Ha például egy feltétel két meghosszabbítása olyan, hogy az első lefed 1000 pontot, de egyet negatívan, a másik pedig csak egy pontot fed le (nyilván helyesen), akkor a Prism a második meghosszabbítást fogja választani. A Prism egyik változata a relatív fedési

hiba helyett egy információnyereség jellegű értékkel számol a pszeudokódbeli ϕ bővítésénél. Tegyük fel, hogy azt kívánjuk eldönteni, hogy bővítsük-e a ϕ-t az AND $A = a$ jelölt taggal. Jelöljük a ϕ AND $A = a \to Y = y$ szabályt R-rel. Ekkor a potenciális bővítés hibáját az alábbiak szerint számoljuk:

$$\text{hiba}^* = cover^+(R) \cdot \big[\log(Er(R)) - \log(Er(\phi \to Y = y))\big].$$

Az eddigiekhez hasonlóan az információnyereség-alapú Prism is addig bővíti a feltételrészt, amíg nem sikerül 0 hibájú (100%-os pontosságú) szabályt előállítani.

Összehasonlítva az információnyereség és a relatív fedési hiba alapján előállított szabályokat a következőket mondhatjuk: a relatív fedési hiba esetén eleinte kis fedésű szabályokat nyes le, hogy a kivételeket jelentő tanító pontokat lefedje. A komoly szabályokat a futás végére hagyja. Az információnyereség-alapú módszer fordítva működik, a speciális eseteket a végére hagyja.

4.6. Döntési fák

A döntési fák bonyolult összefüggéseket egyszerű döntések sorozatára vezetnek vissza. Egy ismeretlen osztályba tartozó objektum klasszifikálásakor a fa gyökeréből kiindulva a csomópontokban feltett kérdésekre adott válaszoknak megfelelően addig lépkedünk lefelé a fában, amíg egy levélbe nem érünk. A döntést a levél címkéje határozza meg. Egy hipotetikus, leegyszerűsített, hitelbírálatra alkalmazható döntési fát mutat be a 4.12. ábra.[9]

A döntési fák nagy előnye, hogy automatikusan felismerik a lényegtelen változókat. Ha egy változóból nem nyerhető információ a magyarázott változóról, akkor azt nem is tesztelik. Ez a tulajdonság azért előnyös, mert így a fák teljesítménye zaj jelenlétében sem romlik sokat, valamint a problémamegértésünket is nagyban segíti, ha megtudjuk, hogy mely változók fontosak, és melyek nem. Általában elmondható, hogy a legfontosabb változókat a fa a gyökér közelében teszteli. További előny, hogy a döntési fák nagyméretű adathalmazokra is hatékonyan felépíthetők. Egy olyan fa, amely pontjainak kettőnél több gyermeke van, mindig átrajzolható bináris fává. Számos algoritmus ezért csak bináris fát tud előállítani.

4.6.1. Döntési fák és döntési szabályok

A döntési fák előnyös tulajdonsága, hogy a gyökérből egy levélbe vezető út mentén a feltételeket összeolvasva könnyen értelmezhető szabályokat kapunk a

[9] Az ábrázolt döntési fa sem értékítéletet, sem valós hitelbírálati szabályokat nem tükröz, pusztán illusztráció.

döntés meghozatalára, illetve hasonlóan, egy laikus számára is érthető módon azt is meg tudjuk magyarázni, hogy a fa miért pont az adott döntést hozta.

4.6.1 Észrevétel *A döntési fákból nyert döntési szabályhalmazok egyértelműek.*

Ez nyilvánvaló, hiszen tetszőleges objektumot a fa egyértelműen besorol valamelyik levelébe. E levélhez tartozó szabályra az objektum illeszkedik, a többire nem.

Vannak olyan döntési feladatok, amikor a döntési fák túl bonyolult szabályokat állítanak elő. Ezt egy példával illusztráljuk.

Jelöljük a négy bináris magyarázó attribútumot A, B, C, D-vel. Legyen az osztályattribútum is bináris és jelöljük Y-nal. Álljon a döntési szabálysorozat három szabályból:

1. `A=1 AND B=1` $\rightarrow$ `Y=1`
2. `C=1 AND D=1` $\rightarrow$ `Y=1`
3. $\rightarrow$ `Y=0`

A szabálysorozat teljes, hiszen az utolsó, feltétel nélküli szabályra minden objektum illeszkedik. A fenti osztályozást a 4.13 ábrán látható döntési fa adja.

A fenti példában a döntési fa az osztályozás bonyolultabb leírását adja, mint a szabálysorozat. A sárga és kék részfák izomorfak. A részfa által adott osztályozást egyszerűen tudjuk kezelni a döntési szabálysorozatokkal. A döntési fa estében ugyanakkor kétszer is megjelent ugyanazon részfa. Ezt a problémát az irodalom *ismétlődő részfa problémaként (replicated subtree problem)* emlegeti és a döntési fák egy alapproblémájának tekinti. A döntési fák a megoldást nagymértékben elbonyolíthatják. Az előző példában, ha a magyarázó attribútumok nem binárisak, hanem három értéket vehetnek fel, akkor a megadott döntési sorozattal ekvivalens döntési fa a 4.14 ábrán látható. A szürkével jelölt részfa további háromszor megismétlődik. Az ismétlődő részfát egy háromszöggel helyettesítettük az áttekinthetőség érdekében. Természetesen a fa jóval egyszerűbb lenne, ha az attribútumot nem csak egy értékkel hasonlíthatnánk össze, hanem olyan tesztet is készíthetnénk, hogy az adott attribútum benne van-e egy adott értékhalmazban. Például a gyökérben csak kétfelé célszerű ágazni, attól függően, hogy $A = 1$ vagy $A \neq 1$ (másképp $A \in \{2, 3\}$). Ha ilyen feltételeket megengednénk, akkor – a címkéktől eltekintve – a 4.13 ábrán látható fával izomorf fát kapnánk.

4.6.2. A döntési fa előállítása

A fát a tanító adatbázisból *rekurzívan* állítjuk elő. Kiindulunk a teljes tanító adatbázisból és egy olyan kérdést keresünk, aminek segítségével a teljes tanulóhalmaz jól szétvágható kettő vagy több részre. Egy szétvágást akkor tekintünk jónak, ha a magyarázandó változó eloszlása a keletkezett részekben kevésbé szórt, kevésbé bizonytalan, mint a szétvágás előtt. Egyes algoritmusok arra is törekednek, hogy a keletkező részek kb. egyforma nagyok legyenek. A részekre rekurzívan alkalmazzuk a fenti eljárást. Ezt szemlélteti a 4.15. ábra:
a) A tanítóadatbázis, amely alapján a döntési fát elő kívánjuk állítani: arra vagyunk kíváncsiak, hogy az ügyfelek közül kiket érdekel egy akció, tehát az utolsó oszlop az osztályattribútum.
b) A teljes adatbázist tekintve, az életkor alapján lehet leginkább szétválasztani az ügyfeleket aszerint, hogy érdekli-e őket az akció. Ezért a döntési fa gyökerében az életkorra vonatkozó kérdést tesszük fel. A három ágon a tanítóadatok különböző részhalmazai láthatóak az életkor különböző értékeinek megfelelően.
c) Mindhárom ágon rekurzívan alkalmazzuk az eljárást az adott ághoz tartozó tanítóadatbázisra. A két szélső ágon minden példány egyazon osztályba tartozik, ezért egy-egy levél csomópontot hozunk létre. A középső ágon a testsúly szerint vágunk.
d) A kapott döntési fa.
A példa illusztratív: a valóságban gyakran előírjuk, hogy olyan leveleket hozzunk létre, hogy minden levélhez legalább n_{levl} darab adatbázisbeli objektum (példány) tartozzon, és előfordulhat, hogy akkor is létrehozunk egy levelet, ha az adott ághoz tartozó nem minden objektum tartozik ugyanazon osztályba, csak a nagytöbbségük. A következő fejezetekben részletezzük, hogy miként lehet kiszámolni azt, hogy melyik kérdést tegye fel a fa a különböző csomópontokban.

Néhány döntési fát előállító algoritmus egy csomópont leszármazottjaiban nem vizsgálja többé azt az attribútumot, ami alapján szétosztjuk a mintát. Más algoritmusok megengedik, hogy például egy A numerikus attribútum esetében először $A > x$ kérdést tegye fel a döntési fa, majd ezen csomópont valamelyik leszármazottjában a $A > y$ kérdés szerepeljen.

A rekurziót akkor szakítjuk meg valamelyik ágban, ha a következő feltételek közül teljesül valamelyik:

- Nincs több attribútum, ami alapján az elemeket tovább oszthatnánk. A csomóponthoz tartozó osztály ekkor az lesz, amelyikhez a legtöbb tanítópont tartozik.
- A fa mélysége elért egy előre megadott korlátot.

- Nincs olyan vágás, amely javítani tudna az aktuális osztályzáson. A vágás jóságáról az ID3 algoritmus részletes leírásánál valamint a 4.6.5. fejezetben szólunk.

Minden levélhez hozzá kell rendelnünk a magyarázandó változó egy értékét, a döntést. Ez általában az ún. többségi szavazás elve alapján történik: az lesz a döntés, amely kategóriába a legtöbb tanítóminta tartozik. Hasonló módon belső csomópontokhoz is rendelhetünk döntést. Ez akkor hasznos, ha olyan döntési fát kívánunk készíteni, amely nagyon gyorsan képes egy (nem feltétlenül optimális) döntést hozni: ha nagyon kevés idő áll rendelkezésre egy-egy döntéshez, csak az első néhány kérdést tesszük fel. Ha viszont bőségesen áll rendelkezésünkre idő, akkor feltesszük a további kérdéseket is. Az angol irodalomban ezt a technikát *anytime decision tree*-nek nevezik, az olyan osztályozókat, amelyek képesek nagyon rövid idő alatt egy – az optimálistól akár jelentősen eltérő – döntést hozni, majd ezt a döntést a rendelkezésre álló idő függvényében folyamatosan pontosítani, *anytime classificator*-oknak hívják.

A döntési fák előállítására a következő három fő algoritmus család ismert:

1. Interative Dichotomizer 3 (ID3) család, jelenlegi változat C5.0[10]
2. Classification and Regression Trees (CART)[11]
3. Chi-squared Automatic Interaction Detection (CHAID)[12]

4.6.3. Az ID3 algoritmus

Az ID3 [Quinlan, 1986] az egyik legismertebb osztályzó algoritmus. A döntési fák kontextusában egy A attribútumot egy adott csomóponthoz tartozó tesztattribútumnak nevezük, ha az adott csomópontban a fa az A-ra vonatkozó kérdést tesz fel.[13] Az ID3 algoritmus a tesztattribútum kiválasztásához az entrópia csökkenését alkalmazza.

Ha Y egy valószínűségi változó, amely ℓ darab lehetséges értéket p_i ($i = 1, \ldots, \ell$) valószínűséggel vesz fel, akkor Y Shannon-féle entrópiáján a

$$H(Y) = H(p_1, \ldots, p_k) = -\sum_{j=1}^{l} p_j \log_2 p_j$$

[10]Magyarul: Interaktív tagoló / felosztó

[11]Klasszifikáló és regressziós fák

[12]Khi-négyzet (χ^2) alapú automatikus interakció felismerés

[13]Figyeljünk a tesztattribútum és a tesztadatok megnevezések közti különbségre: míg a *tesztadatok* a rendelkezésünkre álló adatok egy, a tanító adatoktól független részhalmaza, addig *tesztattribútum* potenciálisan bármelyik attribútum lehet.

értéket értjük.[14] Az entrópia az információ-elmélet (lásd [Cover, 1991]) központi fogalma. Az entrópia az Y változó értékével kapcsolatos bizonytalanságunkat fejezi ki. Ha egy X változót megfigyelünk és azt tapasztaljuk, hogy értéke x_i, akkor Y-nal kapcsolatos bizonytalanságunk

$$H\left(Y|X=x_i\right) = -\sum_{j=1}^{k} \mathbb{P}\left(Y=y_j|X=x_i\right) \log_2 \mathbb{P}\left(Y=y_j|X=x_i\right)$$

nagyságú. Így ha lehetőségünk van X-et megfigyelni, akkor a várható bizonytalanságunk

$$H\left(Y|X\right) = \sum_{i=1} \mathbb{P}\left(X=x_i\right) H\left(Y|X=x_i\right)$$

Eszerint X megfigyelésének lehetősége a bizonytalanság

$$I\left(Y,X\right) = H\left(Y\right) - H\left(Y|X\right)$$

csökkenését eredményezi, azaz X ennyi információt hordoz Y-ról. Az ID3 az Y attribútum szerinti klasszifikálásakor olyan X attribútum értékei szerint ágazik szét, amelyre $I\left(Y,X\right)$ maximális, azaz $H\left(Y|X\right)$ minimális.

Az előbbiekben látott $H(Y|X)$ feltételes entrópia azokat az attribútumokat „kedveli", amelyek sokféle értéket vesznek fel és így sokfelé ágazik a fa [Quinlan, 1986]. Ez terebélyes fákat eredményez. Tegyük fel, hogy adatbázisbeli minden objektum egy egyedi azonosítószámmal rendelkezik. Ha a vizsgált attribútumok közé bevesszük ezt az azonosító kódot, akkor az 0 feltételes entrópiát fog produkálni, így az algoritmus azt választaná. Egy lehetséges megoldás a nyereségarány mutató (gain ratio) használata [Quinlan, 1993], amelyre mint normált kölcsönös információ tekintünk. Ez a mutató figyelembe veszi a gyerek csomópontokba kerülő tanítópontok számát és „bünteti" azokat az attribútumokat, amelyek túl sok gyereket hoznak létre. A nyereségarányt úgy kapjuk meg, hogy az $I(Y,X)$ kölcsönös információt elosztjuk az adott attribútum entrópiájával:

$$\text{gain\_ratio}(X) = \frac{I(Y,X)}{H(X)}.$$

Sajnos a nyereségarány sok esetben „túlkompenzál" és olyan attribútumokat részesít előnyben, amelynek az entrópiája kicsi. Egy általános gyakorlat, hogy azt az attribútumot választják, amelyik a legnagyobb nyereségarányt adja, azon attribútumok közül, amelyekhez tartozó kölcsönös információ legalább akkora mint az összes vizsgált attribútumhoz tartozó kölcsönös információk átlaga.

[14] Az entrópia képletében $0 \cdot \infty = 0$-val számolunk, mert $\lim_{p \to 0} p \log p = 0$.

4.6.4. Feltételek a csomópontokban

Az ID3 algoritmus kiválasztja a minimális feltételes entrópiával rendelkező attribútumot és annyi gyerekcsomópontot hoz létre, amennyi értéket felvesz az attribútum. Leállási feltételként szerepel, hogy egy ágat nem vágunk tovább, ha nincs több vizsgálható attribútum, azaz a fa maximális mélysége megegyezik az attribútumok számával. Az ID3 algoritmus nem feltétlenül bináris fát állít elő.

Ha bináris fa előállítása a cél vagy az intervallum típusú attribútum szofisztikáltabb kezelése, akkor a magyarázó X attribútum típusától függően kétféle feltételt szokás létrehozni. Sorrend típus esetében $X \geq c$, ahol c egy olyan érték, amelyet az X felvesz valamelyik tanítópont esetén. Intervallum típusú attribútumoknál a c két szomszédos tanítóérték átlaga. Kategória típus esetében $X \subseteq K$, ahol K az X értékkészletének egy részhalmaza. Az első esetben, azaz $X \geq c$ típusú feltételek esetén X felvett értékeinek számával lineárisan súlyozott feltételes entrópiát kell számítani:

$$H(Y|X) = \mathbb{P}(X \geq c)H(Y|X \geq c) + \mathbb{P}(X < c)H(Y|X < c),$$

amelynek számításakor a $\mathbb{P}(X \geq c)$-t és $\mathbb{P}(X < c)$ az $X \geq c$ és $X < c$ esetek relatív gyakoriságával becsülhető. A második esetben, $X \subseteq K$ feltételek esetében, az X által felvett értékek számával exponenciálisan súlyozott feltételes entrópiát számítunk, mivel egy n elemű halmaznak 2^n darab részhalmaza van.

Sok esetben akkor kapunk jó bináris döntési fát, ha egy gyökérből levélig vezető úton egy attribútumot többször is vizsgálunk (különböző konstansokkal illetve az attribútum értékkészletének különböző részhalmazaival). A fa mélysége ekkor az attribútumok számánál jóval nagyobb is lehet.

4.6.5. Vágási függvények

Miért pont a kölcsönös információt használja az ID3 algoritmus? Milyen jó tulajdonsággal rendelkezik a kölcsönös információ? Léteznek-e további vágási függvények, amelyek rendelkeznek ezekkel a jó tulajdonságokkal? A válaszok kulcsa a *Taylor-Silverman elvárások* (impurity-based criteria) és a *vágások jóságának* fogalma.

4.6.1. Definíció *Legyen X egy olyan diszkrét valószínűségi változó, amely k-értéket vehet fel. Az eloszlásfüggvény értékei legyenek $P = (p_1, p_2, \ldots, p_k)$. A Φ vágási függvény a $p_1, p_2, \ldots, p_k$ értékekhez rendel egy valós számot, amint látni fogjuk, Φ segítségével a vágás jóságát jellemezhetjük számszerűen. A $\Phi : [0,1]^k \longmapsto R$ vágási függvénnyel szemben támasztott Taylor-Silverman elvárások a kővetkezők:*

1. $\Phi(P) \geq 0$
2. $\Phi(P)$ *akkor veszi fel a minimumát, ha* $\exists j : p_j = 1$
3. $\Phi(P)$ *akkor veszi fel a maximumát, ha* $\forall j : p_j = 1/k$
4. $\Phi(P)$ *a* P *komponenseire nézve szimmetrikus, tehát a* $p_1, p_2, \ldots, p_k$ *értékek tetszőleges permutációjára ugyanazt az értéket adja.*
5. $\Phi(P)$ *differenciálható az értelmezési tartományában mindenhol*

Adott $\mathcal{T}$ tanítóminta esetén a vágási függvény számításakor a p_j valószínűséget nem ismerjük, így a relatív gyakorisággal közelítjük azaz, ha a j-edik osztályba tartozó tanítópontok halmazát $\mathcal{T}^j$-vel jelöljük, akkor $p_j = \frac{|\mathcal{T}^j|}{|\mathcal{T}|}$. A valószínűségvektor empirikus megfelelőjét $P(\mathcal{T})$-vel jelöljük:

$$P(\mathcal{T}) = (\frac{|\mathcal{T}^1|}{|\mathcal{T}|}, \frac{|\mathcal{T}^2|}{|\mathcal{T}|}, \ldots, \frac{|\mathcal{T}^\ell|}{|\mathcal{T}|}).$$

4.6.2. Definíció *Az olyan* V *vágás jósága, amely során a* $\mathcal{T}$ *tanítópontokat* $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_\ell$ *diszjunkt tanítóhalmazba osztjuk szét, megegyezik a*

$$\Delta\Phi(V, \mathcal{T}) = \Phi(P(\mathcal{T})) - \sum_{i=1}^{\ell} \frac{\mathcal{T}_i}{\mathcal{T}} \cdot \Phi(P(\mathcal{T}_i))$$

értékkel.

Minél nagyobb $\Delta\Phi(V, \mathcal{T})$, annál jobb a vágás. Adott Φ vágási függvény és tanítóponthalmaz esetén célunk megtalálni azt a vágást, amely a $\Delta\Phi(V, \mathcal{T})$ maximális értékét eredményezi. Mivel a $\Phi(P(\mathcal{T}))$ adott tanítóhalmaz esetén rögzített, ezért elég a $\sum_{i=1}^{\ell} \frac{\mathcal{T}_i}{\mathcal{T}} \cdot \Phi(P(\mathcal{T}_i))$ érték minimumát megtalálni.

Mivel Φ a vágási függvény csak az osztályok relatív gyakoriságát veszi figyelembe, a vágás jósága, $\Delta\Phi(V, \mathcal{T})$, nulla lesz abban az esetben, ha az osztályok eloszlása a gyerekekben megegyezik a szülőben található osztályeloszlással. Ez megfelel elvárásainknak, hiszen nem nyerünk semmit az olyan vágással, amely során az egyes osztályba tartozó pontok relatív száma egymáshoz viszonyítva mit sem változik.

Most már látható Taylor és Silverman miért fogalmazta meg az elvárásait. Tekintsük a második és a harmadik elvárást. Azt szeretnénk, hogy a gyermekekben található tanítóminták minél homogénebbek legyenek. Ideális esetben olyan gyerekek jönnek létre, amelyekhez tartozó tanítópontok egy osztályba tartoznak. Ehhez az osztályhoz tartozó relatív gyakoriság 1, a többi osztályé 0 és a Φ vágási függvény a minimumát veszi fel. A legrosszabb esetben az összes

osztály relatív gyakorisága megegyezik, azaz a vágás során olyan gyerek jött létre, amelyben az osztályattribútum teljesen megjósolhatatlan. A harmadik elvárás szerint ezt az esetet büntetni kell, pontosabban: a Φ vágási függvény ekkor vegye fel a maxmimumát. Értelemszerűen a minimum és a maximum között a vágási függvény „normális és kezelhető" legyen, azaz legyen deriválható legalábbis minden pontban.

Nem meglepő, hogy az entrópia teljesíti az öt feltételt.

4.6.3. Lemma *Az entrópia, mint vágási függvény, megfelel a Taylor-Silverman elvárásoknak [Quinlan, 1987].*

Különböző kutatók különböző vágási függvényeket vezettek be. Például a CART algoritmusban a Gini indexet [Breiman és tsa., 1984, Gelfand és tsa., 1991] használták:

$$Gini(\mathcal{P}) = 1 - \sum_{j=1}^{k} p_j^2.$$

A DKM vágási függvényt [Dietterich és tsa., 1996][Kearns és Mansour, 1996] bináris osztályozás esetén ajánlják:

$$DKM(\mathcal{P}) = 2 \cdot \sqrt{p_1 p_2}$$

4.6.4. Lemma *A Gini és a DKM vágási függvények megfelelnek a Taylor-Silverman elvárásoknak.*

Elméletileg bizonyították [Kearns és Mansour, 1996], hogy a DKM vágási függvény ugyanakkora hiba mellett kisebb döntési fákat állít elő, mintha entrópia vagy Gini index alapján választanánk ki a vágást.

Itt szeretnénk visszautalni az ID3 algoritmus ismertetése végén leírtakra. Az entrópia alapú vágási függvények azokat a vágásokat részesítik előnyben, amelyek sokfelé vágnak, azaz sok gyereket hoznak létre. Általában is igaz, hogy ha a vágás jóságát a fenti módon definiáljuk és a vágási függvény kielégíti a Taylor-Silverman elvárásokat, akkor olyan vágások jönnek létre, amelyekhez sok gyerek tartozik. Természetesen ez a probléma nem jelentkezik bináris döntési fák esetében. Ott minden belső csúcsnak pontosan két gyereke van.

A megoldást a vágás jóságának normalizálása jelenti. Például az információnyereség helyett, amint korábban említettük, a nyereségarányt (gain ratio) célszerű használni, amelyet megkapunk, ha az információnyereséget elosztjuk az entrópiával. Általános esetben is hasonlót teszünk, De Mántaras szerint [De Mántaras, 1991] a vágás jóságának normáját a következőképpen célszerű képezni:

$$||\Delta\Phi(V, \mathcal{T})|| = \frac{\Delta\Phi(V, \mathcal{T})}{-\sum_{j=1}^{k}\sum_{i=1}^{\ell} p_{i,j} \log p_{i,j}},$$

ahol k a gyermekek számát, ℓ az osztályok számát, $p_{i,j} = |\mathcal{T}_{i,j}|/|\mathcal{T}|$, és a $\mathcal{T}_j^i$ a j-edik gyermek i-dik osztályba tartozó tanítópontjainak halmazát jelöli.

4.6.6. Továbbfejlesztések

Míg az ID3 családba tartozó fák csak klasszifikációra, addig a CHAID és a CART klasszifikációra és regresszióra is alkalmazhatók. A C4.5 (amelynek kereskedelmi, javított változata a C5.0) és a CHAID fák kizárólag egyetlen attribútumra vonatkozó egyenlő, kisebb, nagyobb teszteket használnák a csomópontokban a döntésekhez (egyváltozós fák), azaz a jellemzők terét téglatestekre vágják fel. A CART fák ferdén is tudnak vágni, attribútumok lineáris kombinációját is tesztelik (relációs fák). Míg a CART eljárás mindig bináris döntéseket használ a csomópontokban, addig egy nominális attribútumra egy C4.5 fa annyi felé ágazik, ahány lehetséges értéket az attribútum felvehet.

Talán a leglényegesebb különbség a különböző fák között, hogy mit tekintenek jó vágásnak. Nominális magyarázott változó esetén a CHAID eljárás – nevének megfelelően – a χ^2-tesztet használja. A CART metodológia a Gini-indexet minimalizálja. A Gini-index alapján mindig olyan attribútumot keresünk, amely alapján a legnagyobb homogén osztályt tudjuk leválasztani.

Ha a magyarázandó Y változó intervallum skálán mért, akkor a CART eljárás egyszerűen a varianciájának csökkentésére törekszik, a CHAID pedig F-tesztet használ.

A CHAID konzervatív eljárás, csak addig növeli a fát, amíg a csúcsban alkalmazható legjobb szétvágás χ^2-, vagy F-teszt szerinti szignifikanciája meghalad egy előre adott küszöböt. A CART és C4.5 eljárások nagyméretű fát építenek, akár olyat is, amelyik tökéletesen működik a tanuló adatbázison vagy olyan heurisztikus leállási szabályokat alkalmaznak, hogy a fa nem lehet egy előre adott korlátnál mélyebb, vagy hogy egy csúcsot nem szabad már szétvágni, ha egy korlátnál kevesebb eset tartozik bele. Mindenesetre a kialakuló fa nagy és terebélyes lesz, túl speciális, amely nem csak az alappopuláció jellemzőit, hanem a mintában előforduló véletlen sajátosságokat is modellezi. Ezért a fát felépítése után egy ellenőrző adatbázist használva meg szokták metszeni (pruning) és elhagyják a felesleges döntéseket. A metszéssel a 4.6.8. fejezetben foglalkozunk részletesebben.

Tanácsos megvizsgálni, hogy nem fordul-e elő, hogy a generált C5.0 vagy CHAID fa egymás után ismételten kevés (két-három) attribútum értékét teszteli. Ez arra utalhat, hogy az attribútumok valamely függvénye (pl.: hányadosa - egy főre eső jövedelem) bír magyarázó erővel és a fa ezt a kapcsolatot próbálja ismételt vagdosással közelíteni. Ha feltételezzük, hogy a két (három) változó milyen kombinációja lehet releváns az adott osztályozási vagy feladatban, képezhetünk ennek megfelelően egy új attribútumot a két (három) változóból.

Láttuk, hogy többféle mutatószám létezik a vágási kritérium kiválasztására. Ezek között nem létezik univerzálisan legjobb: bármelyikhez lehet készíteni olyan adatbázist, amelyet rosszul osztályoz a vágási kritériumot használó algoritmus. A következőkben néhány ismert vágási függvény egységes leírását mutatjuk be.

4.6.7. Súlyozott divergenciafüggvények alapján definiált vágási függvények

Bináris vágási függvények esetén a szülő csomópont N tanító pontját osztjuk szét úgy, hogy a bal oldali gyerekbe N_B tanító pont jut, a jobboldaliba pedig N_J. Az $N_i, i \in \{B, J\}$ pontból $N_{j,i}$ tartozik a j-edik osztályba. Legyen $\pi_i = N_i/N$ és $p_{j,i} = N_{j,i}/N$. A j-edik osztály gyakoriságát a szülőben p_j-vel jelöljük. A fenti jelölésekkel a χ^2 statisztika átírható az alábbi formába:

$$\chi^2/N = \pi_B \sum_{j=1}^{k} p_{jB}(p_{jB}/p_j - 1) + \pi_J \sum_{j=1}^{k} p_{jJ}(p_{jJ}/p_j - 1)$$

Legyen $\vec{u} = (u_1, u_2, \ldots, u_k)$ és $\vec{v} = (v_1, v_2, \ldots, v_k)$ két diszkrét eloszlásfüggvény. Amennyiben a divergencia-függvényüket az alábbi módon definiáljuk

$$d(\vec{u} : \vec{v}) = \sum_{j=1}^{k} u_j(u_j/v_j - 1),$$

akkor a χ^2 statisztika átírható a kővetkezőképpen (alkalmazzuk a $u_j(u_j/v_j - 1) = 0$ konvenciót $u_j = v_j = 0$ esetén):

$$\chi^2 = N\big(\pi_B d(\vec{p}_B : \vec{p}) + \pi_J d(\vec{p}_J : \vec{p})\big),$$

ahol $\vec{p}_B = (p_{1,B}, p_{2,B}, \ldots, p_{\ell,B})$ illetve $\vec{p}_J = (p_{1,J}, p_{2,J}, \ldots, p_{\ell,J})$ feltéve, hogy az osztályokat $1, 2, ..., \ell$-lel sorszámoztuk. Ha a divergenciafüggvénynek a következőt használjuk

$$d(\vec{u} : \vec{v}) = 2\sum_{j=1}^{k} u_j \log(u_j/v_j),$$

akkor az entrópiához jutunk. Továbbá $d(\vec{u} : \vec{v}) = 2\sum_{j=1}^{k}(u_j^2 - v_j^2)$ esetén a Gini index N-edrészét kapjuk.

A közös magot az *erő divergencia függvény* adja [Shih, 1999]:

$$d_\lambda(\vec{u} : \vec{v}) = \frac{1}{\lambda(\lambda+1)} \sum_{j=1}^{k} u_j\big((u_j/v_j)^\lambda - 1\big),$$

ahol $-1 < \lambda \leq \infty$. A d_λ függvény értékét a $\lambda = 0$ helyen, az ugyanitt vett határértéke adja d_λ-nak. Az erő divergencia függvény alapján definiáljuk a vágási függvények egy családját:

$$C(\lambda) = \pi_B d_\lambda(\vec{p}_B : \vec{p}) + \pi_J d_\lambda(\vec{p}_J : \vec{p})$$

Láttuk, hogy $\lambda = 1$ estén a χ^2 statisztikát kapjuk, $\lambda = 0$-nál pedig az entrópiát. További ismert, a vágás jóságát jellemző függvényeket[15] is megkaphatunk az erő divergencia függvényből. Freeman-Tuckey statisztika adódik $\lambda = -1/2$-nél és a Cressie-Read $\lambda = -2/3$-nál [Read és Cressie, 1988].

4.6.5. Tétel *A $C(\lambda)$ osztályba tartozó a vágás jóságát jellemző függvények teljesítik a Taylor-Silverman elvárásokat.*

A vágás jóságát jellemző ismert függvény az MPI index, amelyet az alábbi módon definiálnak:

$$M = \pi_B \pi_J \Big(1 - \sum_{j=1}^{k} p_{jB} \cdot p_{jJ}/p_j\Big)$$

Egy kis kézimunkával az MPI index átalakítható az alábbi formára:

$$M = \pi_B 2\pi_J^2 d_1(\vec{p}_J : \vec{p}) + \pi_J 2\pi_B^2 d_1(\vec{p}_B : \vec{p}),$$

amely a $D(\lambda) = \pi_B 2\pi_J^2 d_\lambda(\vec{p}_J : \vec{p}) + \pi_J 2\pi_B^2 d_\lambda(\vec{p}_B : \vec{p}) = \pi_B \pi_J C(\lambda)$ vágási függvényosztály tagja. Szerencsére ez a függvényosztály is rendben van az elvárásaink tekintetében:

4.6.6. Tétel *A $D(\lambda)$ vágási függvényosztályba tartozó vágási függvények teljesítik a Taylor-Silverman elvárásokat.*

4.6.8. Döntési fák metszése

A döntési fák metszése (pruning) során a felépített fát kicsit egyszerűsítjük, egyes csomópontokat eltávolítunk. Feltételezzük, hogy a felépítés során a fa túltanult, azaz megtanult olyan esetiségeket is, amelyek csak az aktuális tanítóhalmazra jellemzőek, nem általánosan érvényes szabályszerűségek. A metszést ezért egy, a $\mathcal{T}$ tanítóhalmaztól független, $\mathcal{T}_1$ címkézett adathalmaz segítségével szokás elvégezni.

[15] Vegyük észre, hogy a fenti függvények a 4.6.2. fejezetben használt jelölések szerinti $\Delta\Phi(V, \mathcal{T})$-vel analógok.

Beszélhetünk *előmetszésről* (prepruning) és *utómetszésről* (postpruning) . Az előmetszés nem más mint egy intelligens – általában statisztikai megfontolásokon alapuló – megállási feltétel, amelyet a fa építés közben alkalmazunk. Habár lenne olyan vágási függvény, amely megfelel minden feltételnek, és nem értük el az előre megadott maximális mélységet sem, mégsem osztjuk tovább a tanítópontokat egy új feltétel (elágazás) bevezetésével.

Az utómetszés során nagy fát növesztünk, majd elkezdjük azt zsugorítani. A kutatók többsége az utómetszést tartja jobb megoldásnak.

A két legismertebb útómetszési eljárás a *részfa helyettesítés* (subtree replacement) és a *részfa felhúzás* (subtree raising).

A részfa helyettesítés során egy belső pontból induló részfát egyetlen levéllel helyettesítünk, amennyiben az így kapott fa a $\mathcal{T}_1$ adathalmazon jobban osztályoz, mint az eredeti fa a $\mathcal{T}_1$ adathalmazon. Osztályozók összehasonlításáról, az osztályozók minőségének mérésére használt mérékekről a **??** részben írunk.

A részfa felhúzás során a fá valamely közbülső (nem levél) csomópontját illetve csomópontjait a leggyakrabban választott élek alkotta úttal helyettesítjük, amennyiben az így kapott fa a $\mathcal{T}_1$ adathalmazon jobban osztályoz, mint az eredeti fa a $\mathcal{T}_1$ adathalmazon.

4.6.9. Döntési fák ábrázolása

A döntési fa előállítása után két fontos kérdés szokott felmerülni. Egyrészt tudni szeretnénk, hogy melyik levélbe esik sok tanító pont, azaz melyek azok a szabályok, amelyek sok tanító pontra érvényesek. Másrészt látni szeretnénk, hogy a levelek mennyire jól osztályoznak; a tanítópontoktól (tanítóadatbázisbeli objektumoktól példányoktól) független tesztpontok közül (ha vannak tesztpontok) milyen arányban osztályozott rosszul az adott levél. Az első kérdés tehát azt vizsgálja, hogy mennyire jelentős az adott levél, a második pedig azt, hogy mennyire jó, mennyire igaz a levélhez tartozó szabály. Ezeket az értékeket azonnal látni szeretnénk, ha ránézünk egy döntési fára.

Elterjedt módszer (ezt használják például a SAS rendszerében is), hogy minden levelet egy körcikkely reprezentál. A körcikkely nagysága arányos a levélhez tartozó tanító pontokkal, a színe pedig a levélhez tartozó szabály jóságát adja meg. Például minél sötétebb a szín, annál rosszabb az adott levélre eső helyesen osztályozott pontok aránya.

4.6.10. Regressziós fák és modell fák

Amint láttuk a CART és CHAID algoritmusok esetében, döntési fákhoz hasonló modelleket használhatunk regressziós feladatokra is. Ilyenkor nyilván nem rendelhetjük a levelekhez az osztálycímkék diszkrét halmazának egy-egy

elemét, hiszen a magyarázandó változó folytonos. Egyszerű esetben az egyes levelekben a tanítóadatbázis az adott levélhez tartozó objektumok (példányok) magyarázandó változóinak átlagát vesszük. Ekkor beszélünk *regressziós fáról*.

A *modell fák* esetében a levelekhez nem egy-egy számérték tartozik, mint a magyarázandó változó fa által becsült illetve előrejelzett értéke, hanem a levélhez tartozó tanítópontok (tanítóadatbázisbeli objektumok) alapján egy-egy lineáris regessziós modellt készítünk minden egyes levélre külön-külön, és ezt használjuk a magyarázandó változó értékének becslésére. Az adattér szomszédos tartományaihoz tartozó levelek lineáris regressziós modelljei jelentősen különbözhetnek, amely a határterületeken való folytonosság durva hiányát eredményezheti a modell fa egésze, mint előrejelző/becslő modell tekintetében. Ezt olyan módon csökkenthetjük, hogy nem csak a levelekhez kapcsolódóan készítünk lineáris regressziós modelleket, hanem a közbülő csomópontokban is, és egy objektum magyarázandó változtójának becslésekor az objektumot tartalmazó levélhez vezető út mentén lévő összes lineáris regressziós modell becslését/előrejelzését kombináljuk. A hatékony implementáció érdekében a levélhez vezető út mentén lévő modelleket a levélhez tartozó modellel "egybegyúrhatjuk", így valójában csak egyetlen lineáris regressziós modellt kell használnunk a becsléshez.

Akárcsak a CART algoritmus, számos további regressziós illetve modell fát építő eljárás a fa létrehozásakor a magyarázandó változó szórásának csökkentésére törekszik, ez alapján határozza meg, hogy a fa milyen kérdések tegyen fel a csomópontokban, más szóval: milyen feltétel szerint vágjon.

4.7. Bayes-hálózatok

A Bayes-hálózatok két fontos elvre építenek. A maximum likelihood elv szerint egy objektum (elem) osztályozásánál azt az osztályt fogjuk választani, amelynek a legnagyobb a valószínűsége a megfigyelések és az adott objektum osztályattribútumtól különböző attribútumai alapján. A Bayes-tétel szerint pedig meghatározhatjuk a feltételes valószínűséget, ha ismerünk néhány másik valószínűséget.

A Bayes-tétel segítségével meghatározható az optimális (lásd 4.1.2. szakaszt) klasszifikációs szabály. Az egyszerűség kedvéért a tévedés költsége legyen minden esetben azonos. Az osztályozandó példányokat egyenként tekintjük, Y_i-vel jelöljük azt az eseményt, hogy az osztályozadó objektum az i-edik osztályba tartozik ($Y = y_i$). A korábbiakhoz hasonlóan az objektumok (példányok) megfigyelhető tulajdonságait az $\vec{X}$ írja le, $\vec{X}$ komponenseit, az egyes attribútumokat, $X_1, \ldots X_k$-val jelöljük. Az egyes attribútumok adott (osztályozandó) objektum (példány) esetén felvett konkrét értékeit $x_1, \ldots, x_k$-val jelöljük,

$\vec{x} = (x_1, ..., x_k)$. Egy ismeretlen, $\vec{x}$ tulajdonságú példányt abba az osztályba (i) érdemes sorolni, amelyikre $\mathbb{P}\left(Y_i|\vec{X} = \vec{x}\right)$ maximális. A Bayes-szabály alapján

$$\mathbb{P}\left(Y_i|\vec{X} = \vec{x}\right) = \frac{\mathbb{P}\left(\vec{X} = \vec{x}, Y_i\right)}{\mathbb{P}\left(\vec{X} = \vec{x}\right)} = \frac{\mathbb{P}\left(\vec{X} = \vec{x}|Y_i\right)\mathbb{P}\left(Y_i\right)}{\mathbb{P}\left(\vec{X} = \vec{x}\right)}.$$

Mivel $\mathbb{P}\left(\vec{X} = \vec{x}\right)$ minden i osztályra konstans, ezért $\mathbb{P}\left(Y_i|\vec{X} = \vec{x}\right)$ maximalizálásához elegendő $\mathbb{P}\left(\vec{X} = \vec{x}|Y_i\right)\mathbb{P}\left(Y_i\right)$-t maximalizálni. $\mathbb{P}\left(Y_i\right)$ vagy a priori adott, vagy pedig a mintából a relatív gyakoriságokkal egyszerűen becsülhető. Így már „csak" $\mathbb{P}\left(\vec{X} = \vec{x}|Y_i\right)$-t kell meghatározni.

Amennyiben k darab bináris magyarázó attibútumunk van, az Y pedig ℓ féle értéket vehet fel, akkor $\ell(2^k-1)$ darab $\mathbb{P}\left(\vec{X} = \vec{x}|Y_i\right)$ értéket kellene megbecsülnünk. A 3.3.6 részben láttuk, hogy egy valószínűség megbecsléséhez relatív gyakorisággal mennyi tanítópontot kell vennünk. A gyakorlati esetek többségében ennyi tanítópont nem áll rendelkezésünkre, ezért valamilyen feltétellel kell élnünk a modell kapcsán. A naív Bayes-hálók feltételezik, hogy az egyes attribútumok feltételesen függetlenek egymástól.

4.7.1. Naív Bayes-hálók

A naív Bayes-hálók olyan feltételezéssel élnek, amelynek segítségével a $\ell(2^k-1)$ darab megbecsülendő paraméter száma $\ell \cdot k$-ra csökken. Eszerint az osztályattribútum adott értéke mellett az $\vec{X} = (X_1, \ldots, X_k)$ attribútumok feltételesen függetlenek egymástól (lásd a 2.2.1. fejezetet). Ekkor a $\mathbb{P}\left(\vec{X} = \vec{x}|Y_i\right)$ valószínűség kifejezhető a $\mathbb{P}\left(X_j|Y\right)$ valószínűségek szorzataként, hiszen

$$\mathbb{P}\left(X_1, X_2|Y_i\right) = \mathbb{P}\left(X_1|X_2, Y_i\right)\mathbb{P}\left(X_2|Y_i\right) = \mathbb{P}\left(X_1|Y_i\right)\mathbb{P}\left(X_2|Y_i\right)$$

Az első egyenlőtlenségnél a valószínűségek általános tulajdonságát használtuk fel, a másodiknál pedig a feltételes függetlenséget. Könnyű belátni, hogy k magyarázó változó esetén a következőt kapjuk

$$\mathbb{P}\left(\vec{X} = \vec{x}|Y_i\right) = \mathbb{P}\left((X_1, X_2, \ldots, X_k) = (x_1, x_2, \ldots, x_k)\,|Y_i\right) = \prod_{j=1}^{k}\mathbb{P}\left(X_j = x_j|Y_i\right).$$

A $\mathbb{P}\left(X_j = x_j|Y_i\right)$ valószínűségeket a mintából becsülhetjük. Vegyük észre, hogy ha minden osztályhoz tartozik elegendő objektum a $\mathcal{T}$ tanítóadatbázisban, a

$\mathbb{P}(X_j = x_j|Y_i)$ alakú, egyetlen X_j magyarázó változót (attribútumot) tartalmazó feltételes valószínűségek sokkal jobban becsülhetők a tanítóadatbázis, mint minta alapján, mintha $\mathbb{P}((X_1, X_2, \ldots, X_k) = (x_1, x_2, \ldots, x_k)|Y_i)$-t vagy $\mathbb{P}(Y_i|(X_1, X_2, \ldots, X_k) = (x_1, x_2, \ldots, x_k))$-t becsülnénk közvetlenül.

Kategória típusú attribútum

Amennyiben az X_j kategória típusú, akkor $\mathbb{P}(X_j = x_j|Y_i)$ valószínűséget a relatív gyakorisággal közelítjük, tehát meghatározzuk az X_j attribútumában x_j értéket felvevő elemek arányát a Y_i osztályú elemek között. Ezt szemlélteti a 4.16. ábrán látható példa.

Problémát jelenthet, ha valamelyik relatív gyakoriság nulla, mert ekkor – amint a 4.16. ábra példáján is látható – a szorzat is nulla lesz a többi tagtól függetlenül. Legegyszerűbb megoldás, hogy az adott attribútum minden értékének előfordulásához hozzáadunk egyet. Ha volt elég mintánk, akkor a valószínűségek alig torzulnak, viszont sikerül kiküszöbölnünk azt, hogy a nulla tag miatt a többi relatív gyakoriságot nem vesszük figyelembe. Ha például egy adott osztályba tartozó elemek valamely attribútuma három értéket vehet fel és az előfordulások: 0, 150, 250. Akkor 0, 150/400, 250/400 helyett 1/403, 151/403, 251/403 értékeket használunk. Erre a technikára az irodalomban, mint *Laplace estimation* hivatkoznak. Egy kifinomultabb módszer, ha egy helyett p_k-t adunk a relatív gyakorisághoz, ahol p_k-val jelöljük a k-adik attribútumérték relatív gyakoriságát a *teljes* tanítóhalmazban (tehát nem csak a Y_i osztályba tartozó objektumok között).

Szám típusú attribútum

Amennyiben X_j szám típusú és tudjuk a $\mathbb{P}(X_j|Y_i)$ eloszlásának típusát, akkor a keresett valószínűséghez szükséges eloszlásparamétereket statisztikai módszerrel becsüljük. Ha például normális eloszlással van dolgunk, akkor elég meghatároznunk a várható értéket és a szórást osztályonként, ezekből tetszőleges értékhez tartozó valószínűség a sűrűségfüggvényből közvetlenül adódik. Az következő képletekben $|Y_i|$-vel az i-dik osztályba tartozó példányok számát, $x_{i,j}^k$-vel az i-edik osztályba tartozó k-dik példány j-edik attribútumának értékét, $\mu_{i,j}$-vel illetve $\sigma_{i,j}^*$-gal az i-edik osztályba tartozó elemek j-edik attribútumának mintaátlagát illetve empirikus szórását jelöljük. A várható értéket a mintaátlaggal (empirikus közép : $\mu_{i,j} = \sum_{k=1}^{|Y_i|} x_{i,j}^k/|Y_i|$), a szórásnégyzetet a korrigált empirikus szorásnégyzettel ($\sigma_{i,j}^{*2} = \sum_{k=1}^{|Y_i|}(x_{i,j}^k - \mu_{i,j})^2/(|Y_i| - 1)$) becsüljük. A számítások során úgy tekinthetjük, hogy a keresett valószínűséget

a

$$\mathbb{P}\left(X_j = x_j | Y_i\right) = \frac{1}{\sigma^*_{i,j}\sqrt{2\pi}} e^{(x_j - \mu_{i,j})^2 / 2\sigma^{*2}_{i,j}}$$

képlet adja. Megjegyezzük, hogy a folytonos esetben valójában nincs sok értelme annak a kérdésnek, hogy mi a valószínűsége annak, hogy egy X változó értéke pontosan egyenlő x-szel. Ehelyett azt a kérdést szokás feltenni, hogy mi a valószínűsége annak, hogy egy X változó értéke egy adott $[x' - \epsilon ... x' + \epsilon]$ intervallumba esik, ezért ahhoz hogy *valószínűségről* beszélhessünk, valójában integrálnunk kellene a fenti sűrűségfüggvényt egy rövid $[x' - \epsilon ... x' + \epsilon]$ intervallumra. A gyakorlatban ugyanakkor nyugodtan használhatjuk a fenti képlet által adott számokat a naív Bayes osztályozóban az egyes osztályok valószínűségeinek számításakor.

A naív Bayes osztályozó hátrányra, hogy feltételes függetlenséget feltételez, és azt, hogy az attribútumok egyenlően fontosak az osztályozás során. Sokat javíthatunk a naív Bayes osztályozók pontosságán, ha előfeldolgozás során meghatározzuk a fontos attribútumokat, tehát azokat, amelyekről úgy gondoljuk, hogy nem függetlenek az osztályattribútumtól. Több kutató arról számol be, hogy a megfelelő attribútumkiválasztással párosított naív Bayes osztályozó felveszi a versenyt a bonyolultabb, újabb módszerekkel.

4.7.2. Naív Bayes-hálók és a logisztikus regresszió kapcsolata

Ebben a részben belátjuk, hogy amennyiben minden magyarázó attribútum valós típusú, akkor a normális eloszlást feltételező naív Bayes osztályozó (Gaussian Naive Bayes, röviden: GNB) egy lineáris osztályozó, amely nagyon hasonlít a logisztikus regresszióra.

Foglaljuk össze milyen feltételezésekkel él a GNB:

- Az Y bináris valószínűségi változó, melynek eloszlása p_Y paraméterű binomiális eloszlás.
- Minden X_j magyarázó változó valós típusú.
- $X_j|Y_i$ feltételes valószínűségi változó $\mu_{i,j}, \sigma_{i,j}$ paraméterű normális eloszlással írható le, tehát $\mathbb{P}\left(X_j = x_j | Y_i\right) = \frac{1}{\sigma_{i,j}\sqrt{2\pi}} e^{(x_j - \mu_{i,j})^2 / 2\sigma^2_{i,j}}$
- a magyarázó változók adott Y esetén feltételesen függetlenek egymástól.

Vegyük észre, hogy az $X_j|Y_i$ feltételes valószínűségi változó szórása, $\sigma_{i,j}$, attribútumról attribútumra más lehet. Feltételezzük továbbá, hogy a szórás nem függ Y-tól: $\sigma_{0,j} = \sigma_{1,j} = \sigma_j$.

Célunk belátni, hogy ezek a feltevések hasonló alakú $\mathbb{P}(Y|X)$-t adnak, mint azt a logisztikus regresszió teszi (emlékeztetőként: $\mathbb{P}(Y=1|X) = \frac{1}{1+e^{-\vec{x}^T\vec{w}}}$). Induljunk ki a Bayes-szabályból

$$\begin{aligned}\mathbb{P}(Y=1|X) &= \frac{P(Y=1)P(X|Y=1)}{P(Y=1)P(X|Y=1)+P(Y=0)P(X|Y=0)} \\ &= \frac{1}{1+\frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}} = \frac{1}{1+\exp\ln\frac{P(Y=0)P(X|Y=0)}{P(Y=1)P(X|Y=1)}}\end{aligned}$$

most használjuk ki a feltételes függetlenséget:

$$\begin{aligned}\mathbb{P}(Y=1|X) &= \frac{1}{1+\exp\ln\frac{P(Y=0)}{P(Y=0)}+\sum_j \ln\frac{P(X_j|Y=0)}{P(X_j|Y=1)}} \\ &= \frac{1}{1+\exp\ln\frac{1-p_Y}{p_Y}+\sum_j \ln\frac{P(X_j|Y=0)}{P(X_j|Y=1)}}\end{aligned} \tag{4.11}$$

Vizsgáljuk meg közelebbről a szummában szereplő tagot:

$$\begin{aligned}\ln\frac{P(X_j|Y=0)}{P(X_j|Y=1)} &= \ln\frac{\frac{1}{\sqrt{2\pi\sigma_j^2}}\exp-\frac{(X_j-\mu_{j,0})^2}{2\sigma_j^2}}{\frac{1}{\sqrt{2\pi\sigma_j^2}}\exp-\frac{(X_j-\mu_{j,1})^2}{2\sigma_j^2}} = \ln\exp\frac{(X_j-\mu_{j,1})^2-(X_j-\mu_{j,0})^2}{2\sigma_j^2} \\ &= \frac{(2X_j(\mu_{j,0}-\mu_{j,1})+\mu_{j,1}^2-\mu_{j,0}^2}{2\sigma_j^2} = \frac{\mu_{j,0}-\mu_{j,1}}{\sigma_j^2}X_j + \frac{\mu_{j,1}^2-\mu_{j,0}^2}{2\sigma_j^2}\end{aligned}$$

Ha ezt visszahelyettesítjük a 4.11 egyenletbe, akkor látható, hogy $\mathbb{P}(Y=1|X)$ tényleg az X_j attribútumok súlyozott összegének nemlineáris függvényeként adódik:

$$\mathbb{P}(Y=1|X) = \frac{1}{1+e^{w_0+\vec{x}^T\vec{w}}},$$

ahol a súlyok

$$w_j = \frac{\mu_{j,0}-\mu_{j,1}}{\sigma_j^2},$$

a torzítás pedig:

$$w_0 = \ln\frac{1-p_Y}{p_Y} + \sum_j \frac{\mu_{j,1}^2-\mu_{j,0}^2}{2\sigma_j^2}$$

Összegezzük a hasonlóságokat és a különbségeket a GNB és a logisztikus regresszió között. Legfőbb hasonlóság, hogy mind a két módszer lineáris szeparálást végez, azaz az osztályozáshoz a magyarázó attribútumok súlyozott összegét veszi alapul. Különbség van azonban a súlyok meghatározásában. A

logisztikus regresszió közvetlenül becsli a súlyokat, míg a GNB normális eloszlást feltételezve megbecsli a várható értéket és a szórást, majd ez alapján számít egy súlyt. A logisztikus regresszió tehát közvetlenül becsli $\mathbb{P}(Y|X)$-et, míg a Bayes osztályozó ezt közvetve teszi, $\mathbb{P}(Y)$ és $\mathbb{P}(X|Y)$ becslésével. Be lehet látni, hogy amennyiben fennáll a normalitásra tett feltétele a GNB-nek, akkor a GNB és a logisztikus regresszió ugyanazt azt osztályozót (azaz ugyanazokat a súlyokat) eredményezik.

A logisztikus regresszió – mivel nem él semmilyen feltételezéssel az adatra vonatkozóan – egy általánosabb módszernek tekinthető, mint a GNB. Ha nem teljesül a normalitásra tett feltétel, akkor a GNB torz ereményt ad, míg a logisztikus regresszió „adaptálódik a helyzethet".

Az általános módszerek egyik hátránya, hogy jóval több tanítópontra van szükségük, mint azoknak, amelyek valamilyen feltételezéssel élnek a háttérben megbújó modellel kapcsolatban. Nem meglepő ezért, hogy különbség van a tanulás konvergenciájának sebességében: a logisztikus regressziónak $O(n)$ a Bayes hálónak csak $O(\log n)$ tanítópontra van szüksége ugyanaakkora pontosság eléréséhez (amennyiben a normalitásra tett feltétel teljesül).

Amint láttuk, a naív Bayes osztályozó akkor is boldogul, ha az attribútumok közt szám típusú és kategóris típusú attribútum egyaránt előfordul, míg a logisztikus regresszió csak szám típusú attribútumokat képes kezelni.

4.7.3. Bayes hihetőségi hálók

A Bayes hihetőségi hálók (Bayesian belief networks, más névan Bayes-hálók, Bayesian networks) a függetlenségre tett feltételt enyhítik. Lehetővé teszik az adatbányásznak, hogy egy irányított, körmentes gráf segítségével a változók közötti függőségi struktúrát előre megadja. A gráf csomópontjai megfigyelhető és nem megfigyelhető, de feltételezett (rejtett) változók lehetnek. Úgy gondoljuk, hogy a gráf a függőségeket jól leírja, azaz

$$\mathbb{P}\left((Z_1, Z_2, \ldots, Z_s) = (z_1, z_2, \ldots, z_s)\right) = \prod_{j=1}^{s} \mathbb{P}\left(Z_j = z_j | par\left(Z_j\right)\right)$$

teljesül, ahol $par\left(Z_j\right)$ a Z_j csúcs szüleit (a gráfban közvetlenül belemutató csúcsok halmazát jelöli). A fenti képletbeli Z egyaránt lehet magyarázott változó (osztályattribútum, Y) és magyarázó változó $(X_1, ..., X_k)$ is.

Mivel a háló struktúrája a teljes eloszlást leírja, ezért általánosságban tetszőleges Z_j csúcsokat kijelölhetünk outputnak / előrejelzendőnek. Amikor a Bayes-hálókat osztályozásra használjuk, az osztályattribútumnak megfelelő csúcsot választjuk outputnak.

Ha nincsenek rejtett változók, akkor a szükséges

$$\mathbb{P}\left(Z_j = z_j | par\left(Z_j\right)\right)$$

valószínűségek közvetlen becsülhetők a mintából. Ha a háló rejtett változókat is tartalmaz, akkor a gradiens módszer egy változata alkalmazható. Végül megemlítjük, hogy olyan eljárások is ismertek, amelyek segítségével a hálózat topológiája a tanuló példákból kialakítható, nem feltétlenül szükséges azt előre megadni.

4.8. Szupport Vektor Gépek (SVM-ek)

A szupport vektor gépek (support vector machines, SVM-ek) alapesetben bináris osztályozási feladatok megoldására használhatók abban az esetben, amikor az adatbázisbeli objektumok attribútumai szám típusúak, így az egyes objektumokat egy sokdimenziós tér pontjaiként képzelhetük el. A szupport vektor gépek lineáris szeparációt végeznek, azaz a Perceptron algoritmushoz hasonlóan egy elválasztó hipersíkot keresnek a két osztály pontjai között. Egész pontosan azt az elválasztó hipersíkot keresik, amely a lehető legjobban szeparálja a két osztály pontjait, abban az értelemben, hogy az elválasztó hipersíkhoz legközelebbi pontok távolsága a lehető legnagyobb legyen. Az ilyen hipersíkot *maximal margin hyperplane*-nek szokták nevezni. Ezt szemlélteti a 4.17. ábrán látható kétdimenziós példa. Az elválasztó hipersík dimenziószáma az adattér dimenziószámánál egyel kisebb, ezért a 4.17. ábrán látható példában az elválasztó hipersík egy egyenes lesz.

A 4.17. ábra középső részén teli karikával és teli háromszögekkel jelzett pontokat *szupport vektoroknak* nevezzük. A szupport vektorok a vastag vonallal jelölt elválasztó hipersíkhoz legközelebb eső, az elválasztó hipersíkkal párhuzamos vékony vonalakkal jelzett hipersíkokat érintő tanítópontok. Könnyen látható, hogy az elválasztó hipersík megadható a szupport vektorok segítségével, a többi pontra nincs is szükség az osztályozáshoz.

A kérdés, hogy mit tesznek a szupport vektor gépek, ha nem létezik megfelelő, a két osztályt elválasztó hipersík? Az SVM-ek egyrészt megengednek egy "kis hibát", azt, hogy némely pont az elválasztó hipersík rossz oldalára kerüljön, másrészt lehet, hogy az adatot egy másik, általában magasabb dimenziószámú térbe vetítve már található megfelelő elválasztó hipersík. Az előbbi mondatbeli "kis hiba" úgy értendő, hogy minél távolabb van a rossz oldalra került pont az elválasztó hipersíktól, annál nagyobb a hibája. Ha az adatok nem szeparálhatók lineárisan, akkor olyan elválasztó hipersíkot keresünk, amelynek az összesített hibája kicsi, és emellett a "jó" olalra került pontoktól lehetőleg

távol halad. A szupport vektor gépek C (complexity) paraméterével állíthatjuk be, hogy mekkora súllyal számítson a hipersík hibája a legjobb hipersík keresésekor.

Az, hogy az adatpontok az eredeti térben nem szeparálhatók lineárisan, nem jelenti, hogy egy magasabb dimenziószámú térbe vetítve sem található megfelelő elválasztó hipersík. Ezt szemlélteti a 4.18. ábra. A szupport vektor gépek hatékonyságának egyik kulcsa, hogy ezt a magas (akár végtelen) dimenziószámú térbe történő vetítést valójában nem végzik el: belátható, hogy a magasabb dimenziós térbeli elválasztó hipersík kereséséhez elegendő, ha a pontok közti magasabb dimenziós térbeli távolságokat ki tudjuk számolni. Ehhez pedig nem feltétlenül szükséges a pontok vetítése a magasabb dimenzíoszámú térbe.

A következőkben egy példával szemléltetjük, hogy a magasabb dimenziószámú térbeli távolságokat az eredeti adatpontok alapján is ki lehet számolni egy megfelelően definiált távolságfüggvény, ún. *kernel*, segítségével. Tegyük fel, hogy az (x_1, x_2) attribútumokkal (koordinátákkal) adott kétdimenzíos adatpontokat úgy vetítjük egy magasabb dimenziószámú térbe, hogy egy harmadik attribútumot veszünk fel, amely a két korábbi attribútum szorzata: $(x_1, x_2) \to (x_1, x_2, x_1x_2)$. Két pont új térbeli (euklidészi vagy más távolságmérték szerinti) távolsága nyilván kifejezhető a két pont x_1 és x_2 koordinátáinak függvényeként. Ilyen módon tehát, képletesen szólva, beépítjük a vetítést egy, az eredeti pontokon értelmezett távolság függvénybe, kernelbe.

A C mellett tehát a szupport vektor gépek másik alapvető paramétere a magasabb dimenziószámú térbe történő vetítést helyettesítő távolságfüggvény, más néven kernel. Az irodalomban gyakran *kernel trükk* néven hivatkoznak arra a technikára, hogy a magasabb dimenziószámú térbe történő vetítés nélkül számoljuk ki az egyes pontok magasabb dimenziószámú térbeli távolságát.

4.9. Ensemble modellek

Amint a korábbi fejezetekben láttuk, az osztályozási feladat rengeteg, különféle módon megoldható, a különböző osztályozó algoritmusok, eljárások más-más megközelítéssel élnek, ezért azt várjuk, hogy különböző példányok osztályozásánál fognak hibázni. Adódik az ötlet, hogy kombináljuk a különböző osztályozó modelleket[16], és ezáltal ún. *ensemble* modelleket hozzunk létre.[17] Sok osz-

[16] Az osztályozó modell alatt a tanulási fázis végére létrehozott osztályozó függvényt értjük, amely egy-egy objektumot az objektum attribútumai alapján az osztályok valamelyikébe besorol.

[17] Az ensemble szó eredetileg együtteset, zenekart jelent, amelyben sok zenész játszik együtt különböző hangszereken.

tályozó modellt egymással kombinálva általában jobb osztályozási eljáráshoz jutunk, mint a legjobb modell önmagában.

Tekintsük példaként azt, hogy adott egy bináris (kétosztályos) osztályozási feladat, és van 100 darab osztályozó modellünk (esetlegesen ugyanazon algoritmus többször is előfordulhat köztük különböző változatban, különböző paraméterekkel, pl. neurális hálók különböző topológiákkal, SVM-ek különböző kernelekkel), és tegyük fel, hogy ezek egymástól *függetlenül* hibáznak, egyenként mindegyik 0.4 valószínűséggel. Ekkor annak a valószínűsége, hogy egy adott objektum osztályozásakor az osztályozók legalább fele hibázik mindössze

$$\text{Többségi hiba}=\sum_{i=50}^{100}\binom{100}{i}0.4^i(1-0.4)^{100-i}\approx 0.027=27\%.$$

Ha tehát egyszerű többségi szavazással azt az osztályt választjuk, amelybe az osztályozó modellek többsége szerint tartozik az osztályozandó objektum, akkor az esetek több, mint 95%-ában helyes osztályt kapjuk, miközben az osztályozó modellek egyenként csak az esetek 60%-ában osztályoznak helyesen.

Túl szép ahhoz, hogy igaz legyen! Sajnos a gyakorlatban nem is sikerül ennyire drasztikus javulást elérnünk, mert realisztikus esetekben szinte kizárt, hogy a rendelkezésünkre álló modellek hibakarakterisztikája valóban *teljesen független* legyen egymástól, a hibák egyáltalán ne korreláljanak egymással. Ha ennyire jól nem is működnek az ensemble technikák, sok osztályozó modell kombinálása általában szignifikánsan javít az osztályozás minőségén a legjobb egyéni osztályozó modellhez viszonyítva: az olyan adatbányászati versenyeket, ahol az osztályozás minősége számít elsődlegesen, rendszerint valamilyen ensemble modell segítségével nyerik meg.

A következőkben Dietterich cikke nyomán az ensemble modellek elméleti hátterével foglalkozunk [Dietterich, 2000], majd három, széles körben alkalmazott, általános ensemble technikát mutatunk be, amelyeket tetszőleges osztályozó algoritmusok esetében használhatunk.

4.9.1. Dietterich elmélete

Dietterich (2000) azt a kérdést teszi fel, vajon miért lehetséges, hogy az ensemble modellek jól működnek a *gyakorlatban*. Egyrészt az egymástól többé-kevésbé függetlenül hibázó osztályozó algoritmusok "kiolthatják" egymás hibáit, még ha a gyakorlatban nem is olyan drasztikus sikerrel, mint az előző fejezetben látott bevezető példában, másrészt a szerző az ensemble módszerek sikerének három további alapvető okát tárgyalja.

1. Az első ok *statisztikai*: az osztályozó algoritmusok tanulási fázisát úgy tekinthetjük, hogy a legjobb f leképezést keressük a magyarázó változók

és a magyarázott változó (osztályattribútum) között. Tipikusan *nagyon sok* (végtelen) lehetőség közül keressük az optimális f leképezést. Gondoljunk példaként a lineáris vagy logisztikus regresszióra: mivel a w_i súlyok tetszőleges valós számok lehetnek, a teljes keresési tér végtelenül nagy. Még nagy adatbázisok esetén is igaz, hogy a tanítóadatbázisbeli objektumok száma gyakran túlságosan kevés a keresési tér méretéhez képest, ahhoz, hogy vizsgált leképezések közül az *optimálisat* megtaláljuk. Várhatóan csak egy közel-optimálisat találunk. Több különböző osztályozó algoritmus kölönböző közel-optimális megoldást találhat, melyeket kombinálva a végső modell jobb lehet, mint az egyenkénti modellek. Modellek kombinációjának egyszerű módja regesszió esetén a magyarázandó változóra adott becsült/előrejelzett értékeket átlagolása, illetve osztályozásnál a már említett többségi szavazás.

2. A második ok *számítási*: még ha rendelkezésre is áll egy kellőképpen nagy tanítóadatbázis (és ezért a statisztikai probléma nem jelentkezik), az osztályozó algoritmus könnyen elvétheti az optimális megoldást, hiszen a legtöbb algoritmus valamilyen lokális keresést végez: gondoljunk csak a döntési fák építéskor használt algoritmusokra, amelyek mohón mindig az adott pillanatban legjobb attribútum szerinti vágást választják vagy arra, hogy – amint említettük – a neurális hálók súlyainak meghatározásakor a gradiens módszer egyik változatát használják. Valójában mind a döntési fák építése, mind a neurális hálók súlyainak meghatározása NP-nehéz feladat [Hyafil és Rivest, 1976, Blum és Rivest, 1988]. Bár néhány kivételes osztályozó modellnél a lokális kereséssel megtaláljuk a globális optimumot, a lokális keresési eljárások gyakran olyan lokális optimumra vezetnek, amely nem esik egybe a globális optimummal. Ezért a különböző kezdőpontokból indított lokális keresések eredményeiként kapott osztályozó modellek kombinálása jobb modellhez vezet, mint bármelyik eredeti modell önmagában véve.

3. A harmadik ok *reprezentációs*: számos esetben a "tökéletes" osztályozó modell nem található meg az osztályozó tanítása során vizsgált osztályozó modellek halmazában. A különböző modellek kombinálása (pl. regresszió esetén súlyozott átlagolása) révén kiterjeszthetjük a reprezentálható osztályozó modellek körét. A reprezentációs indok némiképp ellentmondásos, hiszen számos osztályozó algoritmus, elvben, az összes lehetséges osztályozó modell közül keresi a legjobbat. Mind a neurális hálók, mind a döntési fák rendkívül rugalmas osztályozó algoritmusok. Elegendően sok adat esetén az összes lehetséges osztályozó modellt tekintik, ezzel kapcsolatos tételek bizonyíthatóak [Hornik és tsa., 1990]. A valóságban

azonban véges tanítóhalmaz mellett az osztályozó algoritmusok tanítási fázisában csak végesen sok lehetséges osztályozó modellt tekintünk, a reprezentált osztályozó modellek halmazának bővítése erre a véges halmazra vonatkozik. Ebből adódik az is, hogy a reprezentációs ok összefügg a statisztikai okkal.

4.9.2. Boosting

A boosting eljárások során osztályozó modellek sorozatát hozzuk létre. Az osztályozó modellek rendszerint ugyanolyan típusúak, például mindegyik modell egy-egy döntési fa. Legtöbbször a modellt előállító algoritmus paramétereit sem változtatjuk, azaz, az előző példát folytatva: mindegyik döntési fa esetében ugyanazt a vágási függvényt használjuk, ugyanolyan metszési eljárást alkalmazunk, stb. A modellek közti diverzitás abból adódik, hogy a tanítóhalmazbeli objektumokat más-más súllyal vesszük figyelembe a modellek létrehozása során.

Az első modell létrehozásakor minden objektumot azonos (például egy) súllyal veszünk figyelembe, ahogy tettük ezt eddig mindig, valahányszor csak osztályozó algoritmusok tanításáról volt szó. Ezt követően megnézzük, hogy a tanítóadatbázis mely objektumait osztályozza helyesen a létrehozott modell, és melyeket nem. A helyesen osztályozott példányok súlyait csökkentjük, a helytelenül osztályozottakét pedig növeljük, és az új súlyokat figyelembe véve[18] egy újabb osztályozót hozunk létre. Abban bízunk, hogy ez az új osztályozó helyesen fogja osztályozni a korábbi osztályozó által tévesen osztályozott objektumokat. Az eddigiekhez hasonlóan csökkentjük azon példányok súlyait, amelyeket a második osztályozó helyesen osztályoz, és növeljük azokét, amelyeket helytelenül, majd az új súlyokat figyelembe véve egy harmadik osztályozó modellt hozunk létre, és így tovább, egészen addig, amíg a kíván számú osztályozó modellt létre nem hoztuk.

Egy ismeretlen osztályba tartozó, új objektum (példány) osztályozása során az egyes osztályozók (súlyozott) többségi szavazatát illetve regresszió esetén a modellek kimeneteinek (súlyozott) átlagát vesszük. Az egyes osztályozó modelleket súlyozhatjuk például aszerint, hogy milyen teljesítményt értek el a teljes tanítóadatbázison.

Több változata is létezik a fenti eljárásnak attól függően, hogy a modellek létrehozása során konkrétan hogyan változtatjuk az egyes példányok súlyait, és az ismeretlen osztályba tartozó objektumok osztályozása során hogyan

[18]Ha például egy objektum (példány) súlya 3, akkor az osztályozó tanítása során úgy tekintjük, mintha 3 darab ugyanolyan objektum fordulna elő az adatbáyisban. Habár nemegész súlyok esetén nem ennyire szemléletes a helyzet, a legtöbb eddig látott osztályozó algoritmus könnyen adaptálható arra az esetre is, ha az objektumok súlyai nem egész számok.

súlyozzuk az egyes modelleket. Az egyik legismertebb változat az AdaBoost [Freund és Schapire, 1994, Rätsch és tsa., 2001].

4.9.3. Bagging és Stacking

A bagging eljárás során az ensemble-ben részt vevő modellek diverzitása abból adódik, hogy a modelleket különböző attribútumhalmazokat használva hozzuk létre. Példaként tegyük fel, hogy 20 magyarázó változónk van egy adatbázisban. Válasszunk ki véletlenszerűen 10-et közülük és ezek segítségével hozzunk létre egy osztályozó modellt. Majd válasszunk ki a 20 attribútum közül véletlenszerűen újra 10-et (nem baj, ha a két halmaz nem diszjunkt), és hozzunk létre egy újabb osztályozó modellt, és így tovább. Nem szükséges, hogy az összes osztályozóhoz 10 attribútumot használjunk.

Az egyes osztályozók kimeneteit (felismert osztálycímkéket vagy regresszió során becsült mennyiségeket) a boosting-nál látottakhoz hasonlóan kombináljuk: (súlyozott) többségi szavazással illetve súlyozott átlaggal.

Ennél valamivel kifinomultabb eljárás, ha egy ún. *metamodellt* hozunk létre.

4.9.1. Definíció Metamodellnek *nevezünk egy m osztályozó vagy regressziós modellt, ha m* bemeneteként *más osztályozó vagy regressziós modellek* kimeneteit *használjuk. Ebben a kontextusban* elemi modellnek *nevezzük azokat az osztályozó vagy regressziós modelleket, amelyek bementei az eredeti adatbázis attribútumai vagy azok valamely részhalmaza.*

A bagging metamodelles változata esetében valamilyen osztályozó algoritmust használunk arra, hogy a sok különböző elemi osztályozó kimenete alapján felismerje a helyes osztálycímkét.

A metamodell tanításához a következő eljárás javasolt. A rendelkezésünkre álló $\mathcal{T}$ tanítóhalmazt felosztjuk két diszjunkt halmazra, $\mathcal{T}_1$-re és $\mathcal{T}_2$-re. A $\mathcal{T}_1$ halmazt használjuk az elemi modellek tanítására, majd a $\mathcal{T}_2$-beli objektumok osztálycímkéit felismertetjük az elemi modellekkel. A metamodell tanítóadatbázisának magyarázó változóinak értékei az elemi modellek által felismert (becsült) osztálycímkék lesznek, a magyarázott változó értéke pedig a $\mathcal{T}_2$-beli valódi osztálycímké lesz. Az így kapott adatbázist használjuk a metamodell tanítására. Az eredeti tanítóadatok $\mathcal{T}_1$-re és $\mathcal{T}_2$-re való felosztása a korábban már említett (és a későbbiekben részletesen tárgyalásra kerülő) túltanulás miatt lényeges: ellenkező esetben a metamodell nem tudná megtanulni, hogy milyen kapcsolat van az elemi modellek által az új objektumokra adott kimenet és a valódi kimenet között, hanem csak azt, hogy a korábban látott objektumokat melyik modell milyen jól osztályozza (ami nem feltétlenül azonos azzal, hogy új objektumok osztályozása esetén hogy teljesít a modell).

A stacking annyiban különbözik a bagging metamodelles változatától, hogy (i) elemi modellnek különböző típusú modelleket használ (például neurális hálót, Bayes-osztályozót és döntési fát) vagy ugyanazon modell különböző változatait, és (ii) nem feltétlen választja ki az attribútumok különböző részhalmazait a különböző elemi modellekhez: az összes elemi modellhez az összes attribútumot használhatja.

4.10. Tanuló algoritmusok értékelése

Az osztályozók témakörének elején írtuk, hogy az osztályozó algoritmusok két fázisban dolgoznak: az első, tanítási fázisban egy $\mathcal{T}$ tanítóadatbázis segítségével létrehoznak egy modellt, amely képes arra, hogy új objektumokat osztályozzon, olyan objektumokat, amelyekről nem ismert, hogy milyen osztályba tartoznak. Az osztályozó (és regressziós) algoritmusok tárgyalásánál mindeddig az első fázisra összpontosítottunk. Felmerül a kérdés, hogy ha túl vagyunk a tanításon, vajon hogyan tudjuk mérni, hogy a kapott osztályozó modell mennyire jól működik.

Az első ötletünk az lehet, hogy az osztályozó modell tanítása után a tanításhoz korábban használt $\mathcal{T}$ *tanítóadatbázist* használjuk a modell értékelésére. Ekkor azt tekintjük, hogy az elkészült modell mennyire jól osztályozza a $\mathcal{T}$ elemeit: az egyes objektumok tényleges osztályát összehasonlítjuk a modell kimenetével és megnézzük, hogy az esetek mekkora részében egyezik a modell kimenete a tényleges osztálycímkével. Az ilyen módon, a tanítóhalmazon számított hibát *resubstitution error*, azaz *visszahelyettesítéses hibának* nevezzük.

A visszahelyettesítéses hiba legtöbbször túlságosan is optimista abban az értelemben, hogy a visszahelyettesítéses hiba gyakran jóval kisebb annál, mint amivel egy valós alkalmazásban szembesülnénk. Például egy $k = 1$ legközelebbi szomszédot figyelembe vevő legközelebbi szomszéd osztályozó teljesítményét a tanítóhalmazon mérve azt kapjuk, hogy az osztályozó tökéletesen osztályoz: amikor egy $x \in \mathcal{T}$ objektumot szeretnénk osztályozni, az x példány a tanítóhalmaz összes példánya közül nyilván saját magához van a legközelebb, tehát osztályozáskor tényleg a saját osztálycímkéjét kapja. Nem biztos ugyanakkor, hogy új, még nem látott objektumokra is ilyen jól működik az osztályozó.

A valóságban egy osztályozó illetve regressziós modellt *új* objektumok osztályozására illetve *új* objektumok valamely nem mérhető tulajdonságának előrejelzésére szeretnénk haszálni. Egy hitelbírálati rendszerben például nem az a kérdés, hogy a múltbeli ügyfelekről vissza tudjuk-e keresni, hogy azok késedelmesen fizették-e vissza a hitelüket, hanem az, hogy az új ügyfelekről mennyire nagy biztonsággal tudjuk előrejelezni, hogy késnek-e a visszafizetéssel. Az osztályozók kiértékelése során egy ilyen gyakorlati szituációt szeretnénk szimulálni.

A kiértékelés alapgondolata tehát az, hogy egy osztályozó algoritmus kiértékeléséhez a tanítóobjektumoktól független, új tesztobjektumokat kell használnunk, amelyeket korábban *soha*(!) nem látott az osztályozó algoritmus. Ezért a gyakorlatban legtöbbször azt az elvet követik, hogy a rendelkezésre álló címkézett adatokat (amelyek esetében ismert, hogy melyik példány melyik osztályba tartozik) két diszjunkt részre osztják, egy $\mathcal{T}$ tanító és egy $\mathcal{T}_1$ tesztadatbázisra, és a rendszer minőségét a tesztadatbázison mérik.

A következőkben a korábban már többször említett túltanulás (túlilleszkedés, overfitting) jelenségével foglalkozunk részletesebben, majd pedig az osztályozó és regressziós algoritmusok kiértékelésére használt protokollokkal (4.10.2. fejezet). Bár a helyesen osztályozott objektumok aránya intuitív mérőszám lehet egy osztályozó minőségének mérésére, számos esetben (például ha az osztályok eloszlása kiegyensúlyozatlan) félrevezető lehet. Az osztályók és regressziós modellek minőségének mérésére használt legfontosabb mérőszámokat a . fejezetben foglaljuk össze. Végezetül azzal a kérdéssel foglalkozunk, hogy mikor mondhatjuk nagy bizonyossággal, hogy az egyik osztályozó modell jobban teljesít, mint a másik (. fejezet).

4.10.1. Túltanulás

A túltanulás (túlilleszkedés, overfitting) jelensége arra vonatkozik, amikor egy osztályozó algoritmus a tanulási fázisban a $\mathcal{T}$ tanítóadatok egyedi, speciális tulajdonságait tanulja meg, ezek alapján osztályoz, ahelyett, hogy az adott területre jellemző általános szabályszerűségeket tárna fel és használna a későbbiekben az új, ismeretlen osztályba tartozó objekutomok (példányok) osztályozására.

A következőkben két példát mutatunk a túltanulásra. Tegyük fel, hogy egy döntési fa építésekor korlátozzuk a döntési fa csúcsainak számát n-ben, $n = 1$ esetében csak egyetlen csúcsot engedünk meg, ez nyilván egy levél lesz, amely minden objektumot (példányt) a többségi osztályba sorol. Az ilyen, szélsőségesen egyszerű döntési fa sem a $\mathcal{T}$ tanítóadatbázison mért visszahelyettesítéses hiba szerint, sem az attól független $\mathcal{T}_1$ tesztadatbázison mért hiba szerint nem fog kimondottan jól teljesíteni. Ha több csúcsot engedünk meg, akkor esélyt adunk a döntési fának, hogy valamilyen nem-triviális összefüggést találjon a magyarázó változók és magyarázandó változó (osztályattribútum) között. Ezáltal az osztályozás minősége javul, a hiba csökken a tanítóadatbázison és a tesztadatbázison is. Ha túl nagyra növeljük a csúcsok számát, azt tapasztaljuk, hogy az osztályozás minősége csak a tanítóadatbázison javul, a tesztadatbázison nem. Ekkor a döntési fa a tanítóadatbázisra jellemző egyedi, speciális sajátosságokat is megtanulja, ekkor beszélünk túltanulásról. Ahogy említettük, döntési fáknál a metszés segíthet a túltanulás kiküszöbölésében.

Ahhoz, hogy a modell kiértékelése fair legyen, figyelnünk kell az adatok megfelelő felosztására, ha a döntési fa építése után metszést is végzünk: tegyük fel, hogy a $\mathcal{T}$ tanítóadatbázis segítségével felépítünk egy döntési fát, majd ennek minőségét a $\mathcal{T}$-től független $\mathcal{T}_1$ adatbázis segítségével mérjük és azt tapasztaljuk, hogy a döntési fa túltanult. A korábbiakban látott módon ezt a problémát metszéssel orvosuljuk, egyes részfákat levelekkel illetve utakkal helyettesítünk, ha a helyettesítés után a $\mathcal{T}_1$ adatbázison javul az osztályozás minősége. Vajon mérhetjük-e ezek után a döntési fa minőségét a $\mathcal{T}_1$ adatbázis segítségével? Nem, hiszen a $\mathcal{T}_1$-t felhasználtuk a végső modell kialakításához, a $\mathcal{T}_1$ segítségével döntöttünk arról, hogy a fa mely részeit akarjuk kimetszeni. Emlékezzünk: ahhoz, hogy fair módon értékeljünk ki egy osztályozó algoritmust, olyan adatokra van szükégünk, amelyet soha nem látott korábban az algoritmus, most tehát szükségünk lesz egy $\mathcal{T}_2$ adathalmazra, amely $\mathcal{T}$-től és $\mathcal{T}_1$-től egyaránt független.

A túltanulást szemléltető második példáként tekintsük a polinomiális regressziót. Tegyük fel, hogy a magyarázandó Y attribútumon kívül mindössze egy magyarázó attribútum, X van az adatbázisban. Vegyük észre, hogy a korábbiakban látott lineáris regressziós algoritmus alkalmazható magasabb fokú polinomiális regresszióra is, ehhez mindösszesen annyit kell tennünk, hogy előfeldolgozási lépésként további attribútumként felvesszük X^2-t, X^3-t, ..., attól függően, hogy milyen fokú polinommal kívánjuk közelíteni Y-t. A 4.19. ábrán egy példát láthatunk a túltanulásra: a jobboldali ábrán látható görbe, 5-dik fokú polinom, minden pontra tökéletesen illeszkedik ugyan, mégsem mondhatjuk, hogy jobban megragadja az általános trendet, mint a baloldali ábrán látható egyenes.

A túltanulás mindkét látott példában a modell bonyolúltságával van összefüggésben: a tanulás során túlságosan bonyolult modellt (túl nagy döntési fát, túl magas fokú polinomot) használtunk.

A túltanulást fel tudjuk ismerni például abból, hogy a modell lényegesen különböző sikerrel osztályoz a független teszthalmazon, mint a tanítóhalmazon. Az egyes modellek esetében gyakran tudunk is tenni valamit a túltanulás ellen: metszés a döntési fáknál, egyszerűbb struktúra illetőleg kevesebb belső neuron használata neurális hálóknál, vagy az olyan esetekben amikor az osztályozó illetve regressziós algoritmus tanulása egy célfüggvény optimumának közvetlen keresését jelenti, mint például a mátrix faktorizációs eljárásoknál, beépíthetünk egy "büntetőtagot" a célfüggvénybe, amely a túl bonyolult modelleket bünteti és ezáltal nem engedi, hogy a tanulás eredményeként kapott modell túl bonyolult legyen. A túltanulás jelenségének megértése a modellek kiértékelése szempontjából azért lényeges, mert aláhúzza azt az alapmegállapításunkat, hogy a modell fair módon történő kiértékeléséhez új, korábban a modell által sohasem látott adatokat kell használnunk.

A korábbiakhoz képest kevésbé intuitív, hogy a túltanulás nem csak az egyes modellek szintjén léphet fel, hanem "magasabb szinten" is. Ezt is egy példával szemléltetjük: egy ügyfél azzal bízott meg bennünket, hogy készítsünk számára egy osztályozó modellt. Sok különböző eljárást kipróbáltunk: döntési fákat, neurális hálókat, Bayes osztályozókat, szupport vektor gépeket, stb. Az egyes osztályozó algoritmusok tanítása során a $\mathcal{T}$ tanítóadatbázist használtuk, az algoritmusok kiértékeléséhez pedig a $\mathcal{T}$-től független $\mathcal{T}_1$ adatbázist. Az osztályozási feladat megoldásként nyilván az általunk vizsgált sok, különféle modell közül a legjobbat szállítjuk az ügyfélnek. Vajon mit mondhatunk ezen legjobb modell teljesítményéről? Mondhatjuk-e, hogy a modell hibája (közelítőleg) akkora, amekkorának a $\mathcal{T}_1$ halmazon mértünk? Elsőre azt gondolnánk, hogy igen, hiszen a modellt készítése során a $\mathcal{T}_1$-től független $\mathcal{T}$-t használtuk tanítóadatként. A végső modell megalkotása szempontjából azonban a legjobb modell kiválasztása is a tanulási folyamat részének tekinthető: előforulhat, hogy az általunk legjobbnak választott modell pusztán a $\mathcal{T}_1$ halmaz valamilyen specialitása miatt bizonyult a legjobbnak és egy másik adathalmaz mellett már nem lenne jobb a többi vizsgált modellnél. Ahhoz tehát, hogy a kiválasztott legjobb modell hibáját fair módon becsüljük, egy újabb, $\mathcal{T}$-től és $\mathcal{T}_1$-től egyaránt független $\mathcal{T}_2$ adathalmazra van szükségünk.

4.10.2. Kiértékelési protokollok

Általában feltételezzük, hogy az osztályozó algoritmus tanítása során használt tanítóadatok reprezentatívak, a későbbiekben osztályozandó adatok ugyanolyan eloszlásból kerülnek ki. Ez ugyan a gyakorlatban nem mindig teljesül, de a feltételezés hiányában aligha lehetne bármilyen osztályozót is tanítani. Így tehát az osztályozók kiértékelésekor is azt tételezzük fel, hogy a tanítóadatok reprezentatívak.

Honnan tudjuk eldőnteni, hogy az adathalmazunk egy része reprezentatíve? Általánosan sehogy. Van azonban egy egyszerű vizsgálat, amelyet érdemes elvégezni. A tanító és a teszt adathalmazban az egyes osztályok eloszlása nagyjából meg kell, hogy egyezzen. Nem várhatunk jó osztályozást, ha a tanítóhalmazba nem került valamely osztályból egyetlen elem sem.

4.10.1. Definíció *Az eredeti adathalmaz olyan particionálását (tanító és teszthalmazra), amelyre teljesül, hogy az osztályok relatív előfordulásai a tanítóhalmazban és a teszthalmazban nagyjából megegyeznek,* rétegzett (stratified) particionálásnak vagy mintavételezésnek *hívjuk.*

A következőkben a leggyakrabban használt kiértékelési protokollokat tekintjük át.

Ismételt mintavételezésen alapuló kiértékelés

Az eredeti adathalmaz nagyobb részét (általában kétharmadát) válasszuk tanítóhalmaznak, a maradékon határozzuk meg a modell hibáját. Ismételjük többször az eljárást különbőző, véletlenszerűen választott tanítóhalmazokon. Az osztályozás végső hibáját az egyes felosztásokon mért hibák átlagaként adjuk meg.

Kereszt-validáció és a leave-one-out

Osszuk fel a tanítóhalmazt N részre. Az adott osztályozó módszerrel N különböző tanítást fogunk végezni. Minden tanításnál egy rész lesz a tesztelőhalmaz a többi uniója pedig a tanítóhalmaz. Minden tanításnál más tesztelőhalmazt választunk. A végső hibát az egyes hibák átlaga adja. Igen elterjedt, hogy N értékének 10-et adnak meg.

A kereszt-validáció egy speciális esete, amikor a N értéke megegyezik a tanítópontok számával, azaz csak egyetlen objektumból (példányból) áll a tesztadatbázis. Ezt a módszert *leave-one-out*-nak (egy kimarad) hívják. A módszer előnye, hogy teljesen determinisztikus, továbbá a tanításhoz a lehető legtöbb információt használja. Hátrány ugyanakkor, hogy a tanítást sokszor kell elvégezni, ami nagyon költséges lehet, továbbá a teszteléshez használt adathalmaz biztos, hogy nem rétegzett.

Egyes kutatók úgy vélik, hogy a kereszt-validáció jelentősége túl van értékelve, hiszen elméletileg nem lehet bizonyítani, hogy megbízhatóbb erdeményt szolgál, mint az egyszerű oszd ketté (taníts, majd tesztelj) módszer.

4.10.3. Mérőszámok

A legfontosabb mutatószám az osztályozó *pontossága* (accuracy), amely a jól osztályozott objektumok számának arányát adja meg az összes objektum számához viszonyítva. A pontossághoz nagyon hasonló mérőszám a hibaarány (misclassification ratio, error rate), amely a helytelenül osztályozott objektumok aránya. Nyilván:

$$\text{hibaarány} = 1 - \text{pontosság}.$$

A pontosság és hibaarány megtévesztő lehet. A magas pontosság (illetve alacsony hibaarány) nem biztos, hogy a módszerünk minőségének az eredménye. Ha például bináris osztályzás esetében az egyik osztály előfordulásának valószínűsége 90%, akkor egy 88% pontosságú osztályozó rossz osztályozó, hiszen pontossága rosszabb, mint azon naív osztályozóé, amely mindig a gyakoribb osztályra tippel. Egy még naívabb osztályozó a véletlen osztályozó, amely

a C osztályt p_c valószínűséggel választja, ahol p_c a C osztály előfordulásának valószínűsége. A valószínűséget relatív gyakorisággal közelítik. A véletlen osztályozó várható pontossága az előbbi példában: $0.9 * 0.9 + 0.1 * 0.1 = 82\%$.

Egy osztályozó *kappa statisztikája* az osztályozó pontosságát a véletlen osztályozóhoz hasonlítja. Tegyük fel, hogy a tanítóhalmazon az egyes osztályok relatív gyakoriságai $p_1, p_2, \ldots, p_k$ és a tanítóhalmazon az osztályok előfordulása $n_1, n_2, \ldots, n_k$. Legyen $N = \sum_{i=1}^{k} n_i$ és $M = \sum_{i=1}^{k} n_i p_i$. A kappa statisztikát ekkor a

$$\frac{T - M}{N - M}$$

adja, ahol T-vel a helyesen osztályzott pontokat jelöljük. A véletlen osztályozó kappa statisztikája nulla, a tökéletes osztályozóé pedig egy.

A pontosság (és hibaarány) nem csak azért lehet félrevezető, mert a naív illetve véletlen modellek pontossága nagy (hibaaránya kicsi) lehet és ezekhez kell viszonyítanunk. Kiegyensúlyozatlan osztályeloszlás esetén sem célszerű a pontosság (és hibaarány) kasználata. Képzeljük el, hogy egy ritka betegség diagnosztizálására valamilyen osztályozó algoritmust használunk. A ritka betegség a népesség mindössze $0,1\%$-át érinti. Vajon melyik osztályozó a jobb:
(i) amelyik mindenkit egészségesnek osztályoz, vagy
(ii) amelyik az esetek 5%-ában téved ugyan, de a betegek nagy részét felismeri?
Az első modell, nyilván, teljesen használhatatlan, míg a második sokat segíthet a betegség diagnosztikájában, még akkor is, ha nem tökéletes. Ezzel szemben az első modell pontossága mégis magasabb, mint a másodiké.

Az előbbi mérőszámoknál részletesebben írja le egy osztályozó teljesítményét az ún. *keveredési mátrix* (*confusion matrix*), amely annyi sorból és oszlopból áll, amennyi az osztályok száma. Az i-edik sor j-edik eleme adja meg azoknak a pontoknak a számát, amelyeket az osztályozó a j-edik osztályba sorol, holott azok az i-edik osztályba tartoznak. A főátlón található elemek adják meg a helyesen osztályozott pontok számát. Alább egy keverési mátrixot láthatunk:

		Felismert (előrejelzett) osztály			
		a	b	c	$\sum$
	a	88	10	2	100
Tényleges	b	14	40	6	60
osztály	c	18	10	12	40
	$\sum$	120	60	20	

Bináris osztályozás esetére, amikor az osztályozó kimenete nulla vagy egy (igaz/hamis, vagy pozitív/negatív) további fogalmakat definiálunk. Többosz-

tályos feladat esetén kijelölhetünk egy kitüntetett (pozitív) osztályt, és minden egyéb osztályt összevonhatunk negatív osztályként, és ekkor a bináris esethez hasonlóan használhatjuk az alábbi megnevezéseket. A jól osztályozott objektumok számát TP-vel (True Positiv) és TN-nel (True Negative) jelöljük attól függően, hogy melyik osztályba tartoznak. A rosszul osztályozott objektumok jelölése FP, FN (False Positive, False Negative). A következő keveredési mátrix összefoglalja a jelöléseket:

		Felismert (előrejelzett) osztály	
		+	−
Tényleges osztály	+	TP	FN
	−	FP	TN

A *felidézést* vagy *megbízhatóságot* (angolul recall vagy true positive rate), amelyet bináris osztályozásnál érzékenységnek (sensitivity) is hívnak az

$$R = \frac{TP}{TP + FN}$$

hányados adja. A *precisiont* [19] a következőképpen számolhatjuk:

$$P = \frac{TP}{TP + FP}.$$

E két érték parametrikus harmonikus közepét F-mértéknek (F-measure) nevezzük:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}}.$$

A leggyakrabban, amikor ennek ellenkezőjét nem jelezzük, $\alpha = 0.5$ mellett számítjuk az F-measure-t. A $\frac{FP}{FP+TN}$ hányadost selejtnek (fallout, false positive rate) is nevezik. A korábban már tárgyalt *pontosság* (accuracy) is definiálható a TP-k és TN-k segítségével: $\frac{TP+TN}{N}$.

Tekintsünk egy olyan osztályozó modellt, amely nem csak egy diszkrét döntést ad eredményül, hogy egy adott teszthalmazbeli objektum (példány) a pozitív vagy negatív osztályba tartozik, hanem egy folytonos kimenetet eredményez, amely annál nagyobb, minél több eséllyel tartozik (a modell szerint) egy adott objektum (példány) a pozitív osztályba. A TP-k, TN-k, FP-k és FN-k

[19] Magyarra mind a *precision*-t, mind az *accuracy*-t pontosságnak fordíthatjuk. Vegyük azonban észre, hogy a precision és accuracy nem azonos. A jegyzetben a precision-ra angol nevével hivatkozunk, csak az accuracy-t fordítjuk magyarra.

száma ekkor annak függvénye, hogy milyen θ küszöbérték felett tekintjük a modell kimenetét pozitívnak. A true positive rate-t (recall, felidézés, megbízhatóság) jelöljük TPR-rel: $TPR = TP/(TP + FN)$. Ehhez hasonlóan jelöljük FPR-rel (false positive rate) a FP-k arányát az összes negatív osztályba tartozó objektumhoz képest: $FPR = FP/(FP + TN)$. Nyilván TPR és FPR is a θ küszöbérték függvény. A TPR-t ábrázolhatjuk FPR függvényében. Az így kapott görbét nevezik *Receiver-Operator Curve*-nek vagy röviden ROC görbének. Az AUC (Area Under the Curve) az adatbányászatban (hacsak ennek ellenkezőjét nem jelzik) a ROC görbe alatti területre vonatkozik. Tökéletes osztályozó modell esetében, amikor található olyan küszöbszám, amely mellett a modell kimenete tökéletesen megegyezik a tényleges osztályokkal, az AUC értéke 1, véletlenszerű kimenetet adó modell esetében pedig 0.5. Az AUC-ra mutat példát a 4.20. ábra.

Hiba mérése valószínűségi döntési rendszerek esetén

Valószínűségi döntési rendszerek esetén a kimenet egy valószínűségi eloszlás, nem pedig egy konkrét osztály. Nem azt mondjuk, hogy adott attribútumértékekkel rendelkező ügyfél kockázatos, hanem azt, hogy 80%-ot adunk annak valószínűségére, hogy kockázatos és 20%-at arra, hogy nem. Ha az osztályok száma k, akkor az osztályozás eredménye egy k dimenziós valószínűségi vektor, ezen valószínűségi vektor elemeinek összege 1. Hogyan határozzuk meg a hibát ilyen esetben?

Négyzetes veszteségfüggvény – Tetszőleges elem konkrét osztályát is leírhatjuk egy valószínűségi vektorral. Ha az elem a j-edik osztályba tartozik, akkor a valószínűségi vektor j-edik eleme legyen 1, a többi pedig nulla. Az osztályozás hibája ekkor az elem osztályához tartozó vektor és az osztályozás eredményeként kapott vektor különbségének normája lesz. Általában az euklideszi normát használjuk és a négyzetgyök számításától eltekintünk:

$$Er(\mathbf{p}, \mathbf{a}) = \sum_{i=1}^{k} (p_i - a_i)^2,$$

ahol $\mathbf{p}$ a valószínűségi döntési rendszer kimenete, az $\mathbf{a}$ pedig a tényleges osztályt reprezentáló vektor, p_i illetve a_i ezen vektorok komponensei. Mivel az a_i-k közül egyetlen érték 1, a többi nulla, a négyzetes veszteségfüggvény átírható $1 - 2p_j \sum_{i=1}^{k} p_i^2$, ahol j-vel az osztály sorszámát jelöltük.

Ha az osztályattribútum teljesen független a többi attribútumtól, akkor a négyzetes veszteségfüggvény azokat az osztályozásokat fogja jutalmazni, amelyek a bemenettől függetlenül olyan valószínűségi vektorokat állítanak elő,

amely megfelel az osztályattribútum eloszlásfüggvényének, azaz a kimeneti vektor i-edik eleme adja meg az i-edik osztály előfordulásának valószínűségét. Nem nehéz ezt az állítást belátni. Jelöljük az i-edik osztály előfordulásának valószínűségét p_i^*-vel. A várható értéke a négyzetes veszteségfüggvénynek egy adott tesztelem esetén:

$$\mathbb{E}\big[\sum_{i=1}^{k}(p_i - a_i)^2\big] = \sum_{i=1}^{k}\big(\mathbb{E}[p_i^2] - 2\mathbb{E}[p_i a_i] + \mathbb{E}[a_i^2]\big) = \sum_{i=1}^{k}(p_i^2 - 2p_i p_i^* + p_i^*) =$$

$$= \sum_{i=1}^{k}\big((p_i - p_i^*)^2 + p_i^*(1 - p_i^*)\big).$$

Felhasználtuk, hogy az a_i várható értéke p_i^*, továbbá, hogy $a_i^2 = a_i$ hiszen a_i értéke csak egy vagy nulla lehet. A végső képletből látszik, hogy a várható érték akkor lesz minimális, ha $p_i = p_i^*$ minden i-re.

Hiba mérése regresszió esetében

Amikor a magyarázandó attribútum szám típusú, akkor a leggyakrabban használt hiba a négyzetes hibaátlag (vagy annak gyöke). Az elterjedt használat oka, hogy a négyzetes hibaösszeg könnyen kezelhető matematikailag – gondoljuk csak a lineáris regresszióra, amely sok regressziós módszer kiindulópontjaként szolgál. Ha csökkenteni szeretnék a különc pontok által okozott hiba mértékét, akkor használhatunk átlagos hibakülönbséget is.

Többször láttuk, hogy nem az abszolút hiba érdekel minket, hanem a relatív hiba. Azt gondoljuk, hogy ugyanakkora hibát vétünk, ha 200 helyett 220-at jósolunk, mint amikor 1 helyet 1.1-et. A fenti hibamértékek és azok relatív változatainak pontos képlete a következő táblázatban látható. A képletekben a teszthalmaz (amely származhat például kereszt-validációból) az i-edik objektumának magyarázandó változójának tényleges értékét y_i-vel, ugyanezen objektum magyarázandó változójának a modell által becsült értékét $\widehat{y}_i$-vel jelöljük.

hibamérték	képlet
átlagos négyzetes hiba	$\frac{1}{n}\sum_i \left((y_i - \widehat{y}_i)^2\right)$
átlagos négyzetes hibagyök	$\sqrt{\frac{\sum_i \left((y_i-\widehat{y}_i)^2\right)}{n}}$
abszolút hibaátlag	$\frac{\sum_i \lvert y_i-\widehat{y}_i\rvert}{n}$
relatív négyzetes hiba	$\frac{\sum_i \left((y_i-\widehat{y}_i)^2\right)}{\sum_i \left((y_i-\bar{y})^2\right)}$
relatív négyzetes hibagyök	$\sqrt{\frac{\sum_i \left((y_i-\widehat{y}_i)^2\right)}{\sum_i \left((y_i-\bar{y})^2\right)}}$
relatív abszolút hiba	$\frac{\sum_i \lvert y_i-\widehat{y}_i\rvert}{\sum_i \lvert y_i-\bar{y}\rvert}$
korrelációs együttható	$\frac{\sum_i \left((y_i-\bar{y})(\widehat{y}_i-\bar{\widehat{y}})\right)}{\sqrt{\left(\sum_i \left((y_i-\bar{y})^2\right)\right)\left(\sum_i \left((\widehat{y}_i-\bar{\widehat{y}})^2\right)\right)}}$

A korrelációs együttható (amely mínusz egy és plusz egy közé esik) kilóg a sorból két dolog miatt. Egyrészt ez a mérték skála invariáns, azaz, ha minden jósolt értéket megszorzunk egy adott konstanssal, akkor a korrelációs együttható nem változik. Másrészt minél jobb az osztályozó módszer, annál közelebb lesz az együttható egyhez. A többi mérték értéke 0 lesz a tökéletes osztályozó estében.

4.10.4. Osztályozók összehasonlítása

Ebben a részben azzal foglalkozunk, hogy mikor mondhatjuk, hogy az egyik osztályozó (regressziós modell) jobb a másiknál? Ha valamilyen mérték szerint jobbnak találjuk az egyik modellt, honnan tudjuk, hogy az valóban jobb, a különbség nem csak a véletlen műve?

Alsó és felső korlát egy osztályozó teljesítményére

Tegyük fel, hogy a bináris osztályozónk p valószínűséggel ad helyes eredményt, tehát a pontossága p. Adott N tesztpont mellett a helyesen osztályozott pontok számát jelöljük N^*-gal. A helyesen osztályozott pontok száma egy N, p paraméterű, binomiális eloszlású valószínűségi változó. Tetszőleges α értékhez ($1-\alpha$ a statisztikai próba szintje) meg tudjuk határozni az elfogadási tartományt a helyesen osztályozott pontok számára vonatkozóan (ezt hívják bináris tesztnek). Határozzuk meg azt az N^*/N-nél kisebb, legkisebb p-t (jelöljük p_l-el), amelyre N^* az elfogadási tartományba esik. Határozzuk meg ezenkívül azt

az N^*/N-nél nagyobb, legnagyobb p-t (jelöljük p_u-val), amelyre f az elfogadási tartományba esik. A tesztadatbázis objektumai alapján csak azt tudjuk elmondani, hogy az igazi p (nagy valószínűséggel) a $[p_l, p_u]$ intervallumba esik.

A fenti módszer meglehetősen számításigényes. A p_l, p_u értékek meghatározásának nehézsége abból adódik, hogy a p a valós számok halmazából kerül ki, a valószínűségi események (és így N^* és az elfogadási tartományok korlátai is) azonban egész számok. A pontosság rovására a $[p_l, p_u]$ intervallumok meghatározása közvetlen számítható, amennyiben a binomiális eloszlást $\mu = Np, \sigma = Np(1-p)$ paraméterű normális eloszlással közelítjük.

Student-féle t-próbán alapuló összehasonlítás

A következő részben legyenek adottak B és L osztályozók (gondolhatjuk, hogy a B egy Bayes-osztályozóra, a L pedig egy logisztikus regresszión alapuló módszerre utal). Legyen adott N darab teszthalmaz. A két osztályozó i-edik teszthalmazon mért pontosságát jelöljük b_i és l_i-vel, legyen $d_i = b_i - l_i$.

Tegyük fel, hogy egy ν pontosságú osztályozó pontossága egy adott tesztalmazon megegyezik egy ν várható értékű és ismeretlen szórású normális eloszlású valószínűségi változó egy megfigyelésével. Ezek a megfigyelt pontosságok rendelkezésünkre állnak, azt kell eldönteni, hogy az eredeti pontosságok statisztikailag eltérnek-e egymástól. Nullhipotézisünk tehát az, hogy a két osztályozó teljesítménye valójában nem tér el egymástól, a megfigyelt különbség mindössze a véletlen műve.

A Student-féle[20] t-próba egy ismeretlen várható értékű és szórású normális eloszlású valószínűségi változó várható értékére tett feltételt próbál eldönteni: azt vizsgálja, hogy a várhatóérték szignifikánsan eltér-e egy adott e értéktől. Mivel a nullhipotézis szerint a két osztályozó teljesítménye nem tér el, ezért azt vizsgáljuk, hogy a különbségek várható értéke $e = 0$-tól eltér-e. Jelöljük $\bar{d}$-gal a mintaátlagot (d_i-k átlagát) és a σ_d^*-gal az empirikus korrigált szórást (d_i-k empirikus korrigált szórását). Legyen

$$E = \frac{\bar{d} - 0}{\sqrt{\sigma_d^{*2}/N}.}$$

Mivel E folytonos, nincs értelme azt kérdeznünk, hogy mi annak a valószínűsége, hogy E egy konkrét értéket vesz fel. Azt a kérdést viszont feltehetjük, hogy mi annak a valószínűsége, hogy E egy adott intervallumba esik. Tudjuk továbbá, hogy a nullhipotézis teljesülése esetén E mennyiség $N-1$ szabadságfokú t-eloszlású. Ha tehát azt kívánjuk, hogy a legfeljebb α legyen a valószínűsége

[20]Érdekességként jegyezzük meg, hogy az eljárás megalkotójának valódi neve Gosset, a módszer elnevezése onnan származik, hogy Student álnéven publikálta.

annak, hogy a nullhipotézist elutasítjuk, miközben a nullhipotézis valójában igaz, akkor olyan intervallumot keresünk, amelybe E a nullhipotézis teljesülése esetén $1-\alpha$ valószínűséggel esik. Az alábbi tábázat N és α néhány értéke mellett mutatja az intervallumhatárokat. Ha E nem esik az adott intervallumba, elutasítjuk a nullhipotézist, tehát arra a következtetünk, hogy a megfigyelt különbség nem csak a véletlen műve.

	$\alpha = 0.01$	$\alpha = 0.05$	$\alpha = 0.10$
$N = 5$	$-4.60 \ldots + 4.60$	$-2.78 \ldots + 2.78$	$-2.13 \ldots + 2.13$
$N = 10$	$-3.25 \ldots + 3.25$	$-2.26 \ldots + 2.26$	$-1.83 \ldots + 1.83$
$N = 20$	$-2.86 \ldots + 2.86$	$-2.09 \ldots + 2.09$	$-1.73 \ldots + 1.73$
$N = 50$	$-2.68 \ldots + 2.68$	$-2.01 \ldots + 2.01$	$-1.68 \ldots + 1.68$
$N = 100$	$-2.63 \ldots + 2.63$	$-1.98 \ldots + 1.98$	$-1.66 \ldots + 1.66$

Vegyük észre, hogy a táblázatban szereplő intervallumok szimmetrikusak a nullára. Ez azzal függ össze, hogy a t-eloszlás szimmetrikus. A nullhipotézis teljesülése mellett tehát nem csak az igaz, hogy α valószínűséggel esik E az intervallumon kívülre, hanem az is, hogy E éppen $\alpha/2$ valószínűséggel esik az intervallum alsó határa alá, és ugyanekkor eséllyel lesz E nagyobb, mint az intervallum felső határa. A statisztikai eloszásfüggvények tábázataiban általában azt adják meg, hogy egy adott eloszlású változó mely küszöbértéknél lesz kisebb egy adott valószínűséggel. A fenti intervallumhatárokhoz tartozó küszöbértékeket tehát a t-eloszlásfüggvény táblázatában $1-\alpha/2$ valószínűségnél kell keresnünk.

Példa – 5-szöros keresztvalidáció során az egyik osztályozó pontosságát 0.8-nak, 0.7-nek, 0.75-nek, 0.6-nak és 0.8-nak mértük, míg egy másik osztályozó pontosságát ugyanezen teszthalmazokon 0.95-nek, 0.7-nek, 0.75-nek, 0.9-nek és 0.85-nek mértük. Kijelenthetjük-e legalább 95%-os biztonsággal, hogy a második osztályozó jobb, mint az első?

Nem, hiszen $d_1 = 0.8 - 0.95 = -0.15, d_2 = 0.7 - 0.7 = 0 \ldots, \bar{d} = -0.1, E = -1.96$, ami beleesik a $-2.78 \ldots + 2.78$ intervallumba, tehát nem tudjuk elutasítani a nullhipotézisünket, amely szerint a megfigyelt különbség csak a véletlennek tudható be.

4.11. További osztályozási protokollok

Eddig azt feltételeztük, hogy az osztályozó illetve regressziós algoritmusunkat egyszer tanítjuk, majd az új objektumokat *egyesével* dolgozzuk fel: külön-külön kell az új példányokat osztályoznunk, illetve a magyarázandó változó értékét külön-külön kell minden egyes példányra becsülnünk. A legtöbb gyakorlati

alkalmazáshoz jól illeszkedik ez a feltételezés. Ha például egy új ügyfél érkezik egy biztosító társasághoz, az ügyfél elvárhatja, hogy kvázi-azonnal kapjon egy ajánlatot, hogy mennyibe kerülne, ha biztosítást kötne. Ilyen esetben az új ügyfelet rögtön, a többi új ügyféltől függetlenül kell valamelyik kockázati osztályba sorolni. Egy jelbeszédi jeleket felismerő rendszer esetében szintén elvárjuk, hogy képes legyen egyesével felismerni a jeleket: ha például egy néma ember ilyen módon kíván kommunikálni velünk, azt szeretnénk, hogy a rendszer folyamatosan tolmácsoljon, ne csak sok száz vagy sok ezer jel elmutogatása után. A példák sorát hosszasan lehetne folytatni. Vannak azonban ezektől eltérő gyakorlati alkalmazások, amelyekben sok, ismeretlen osztályba tartozó objektumot kell osztályoznunk kvázi-egyidejűleg, azaz nem számít, hogy milyen sorrendben dolgozza fel a rendszer az ismeretlen osztályba tartozó objektumokat. Ilyen esetekben az adatok strukturájának jobb feltárását az ismeretlen osztályokba tartozó objektumok is segíthetik.

4.11.1. Semi-supervised osztályozás

A semi-supervised (részben felügyelt, félig felügyelt) tanítás – röviden: SSL – esetén adott egy $\mathcal{T}$ tanítóhalmaz, és tudjuk, hogy a tanítóhalmazbeli objektumok (példányok) mely osztályokba tartoznak. Azt feltételezzük, hogy egyszerre *nem csak egyetlen*, hanem *sok* címkézetlen objektummal van dolgunk, amelyekről el kell eldöntenünk, hogy melyik osztályba tartoznak. Az osztályozó legelőször azt az objektumot osztályozza, amelynek osztályozásában a leginkább biztos. Egy $k = 5$ legközelebbi szomszédot figyelembe vevő legközelebbi szomszéd osztályozó például *biztosabb* egy olyan objektum osztályozásában, amelynél mind az 5 legközelebbi szomszéd egyazon osztályba tartozik és a legközelebbi szomszédok nagyon közel vannak az osztályozandó objektumhoz, mint egy olyan objektum osztályozásában, amelynek a 5 legközelebbi szomszédja közül 2 az egyik osztályba tartozik, 3 a másikba, és ráadásul a szomszédok kicsit távolabb is vannak. Miután az osztályozó osztályozta azt az objektumot, amelynek osztályozásában leginkább biztos, ezt az objektumot, az osztályozó által megállapított osztálycímkével, hozzávesszük a tanító halmazhoz, újratanítjuk a modellt (vagy módosítjuk a modellt az új tanítóobjektumnak megfelelően), és a maradék címkézetlen objektumok közük ismét azt osztályozzuk, amelyiknek az osztályozásában leginkább biztos a modell.

Az SSL számításigénye, a konvencionális osztályozáshoz képest, általában jelentősen nagyobb. Ugyanakkor a semi-supervised osztályozás során az osztályozó algoritmus képes a címkézetlen adatokban rejlő struktúrát is figyelembe venni, ami különösen akkor lényeges, ha kevés a címkézett tanítóadat, vagy a tanítóadatbázis nem reprezentatív. Erre láthatunk egy példát a 4.21. ábrán.

A 4.21. ábran az adatbázisbeli objektumok egy kétdimenziós tér pontja-

inak felelnek meg (két szám típutú magyarázó attribútummal rendelkeznek). Minden objektum két osztály valamelyikébe tartozik: az egyik osztályba tartozó objektumokat háromszögekkel, a másikba tartozókat karikákkal jelöltük. A teli háromszögek illetve teli karikák jelzik a tanítóhalmazba tartozó, címkézett objektumokat, x-szel jelöltük azokat az objektumokat amelyekről az algoritmus adott lépésben még nem döntötte el, hogy mely osztályba tartoznak. Üres háromszögek és karikák az algoritmus döntéseit jelölik. A tanítóhalmazt az (a) alábra mutatja. A (b) alábrán azt látjuk, hogy egy konvencionális protokoll szerinti osztályozó, amely egyenként tekinti az osztályozandó objektumokat, a kőrív mentén alul elhelyezkedő karikákat minden bizonnyal a háromszögek közé sorolná, hiszen ezekhez jóval közelebb van a tanítóhalmazbeli hároszög, mint a tanítóhalmazbeli kör. Ezzel szemben a semi-supervised protokoll szerinti osztályozás során a kőrív mentén végighaladva, minden iterációban a már felismert karikákhoz legközelebbi karikát ismernénk fel karikaként és ezáltal a semi-supervised protokoll szerinti osztályozó képes helyesen osztályozni az objektumokat. A (c)-(e) alábrák a semi-supervised osztályozás első, második, illetve harmadik iterációit mutatják, az (f) alábra pedig a semi-supervised osztályozás végerdményét.

Az SSL osztályozásnak számos változata létezik attól függően, hogy miként formalizáljuk azt, hogy egy osztályozó mennyire biztos egy objektum osztályozásában.

4.11.2. Active Learning

A semi-supervised learning-hez hasonlóan az active learning esetén is abból indulunk ki, hogy nem egyesével osztályozzuk az objektumokat, hanem egyszerre osztályozunk sok objektumot. A semi-supervised esethez hasonlóan adott néhány címkézett objektum, egy viszonylag kicsit tanítóadatbázis, amely nem feltétlenül reprezentatív. Miközben az algoritmus folyamatosan osztályozza az objektumokat, néhány kérdést feltehet a felhasználónak: néhány objektum tényleges osztálycímkéjére rákérdezhet. Nyilván azokra érdemes rákérdezni, amelyek osztályozásában a leginkább bizonytalan az algoritmus, amelyek tényleges osztálycímkéjének ismerete sokat javíthat a modell pontosságán.

A semi-supervised learning-hez hasonlóan az active learning-nek is számos változata ismert. Az active learning eljárások kiértékelése némiképp eltér a konvencionális osztályozók kiértékelésétől. Active learning eljárások által adott osztályozás minőségét általában a felhasználó felé feltett kérdések számának függvényében szokták ábrázolni. Akkor mondjuk, hogy az egyik active learning eljárás jobb a másiknál, ha ugyanannyi kérdést feltéve a felhasználónak az első eljárás tendenciózusan jobban osztályoz, mint a második.

4.11.3. Transfer learning

Néhány esetben egy-egy terület, amelyen osztályozókat kívánunk használni olyannyira új, hogy még nem állnak rendelkezésre megfelelő, címkézett adatok, amelyeket tanítóadatként használhatnánk. Szerencsés esetben azonban vannak címkézett adataink egy hasonló területről, amely alapján készíthetünk egy osztályozót, amelyet adaptálhatunk a célterületre. Erre példa lehet az, amikor egy bank egy új országban első fiókjait nyitja meg: ekkor még nem áll rendelkezésére elegendő adat az adott országból, amely alapján egy hitelbírálati osztályozó algoritmust készíthessen, ugyanakkor más országokból, ahol korábban már végzett tevékenységet, bőségesen állhat rendelkezésre adat, amely alapján egy osztályozó algoritmust lehet tanítani, majd az elkészül osztályozó modellt a helyi igényekre lehet szabni. Az ilyen eljárást nevezik transfer learning-nek: a célterülethez képesti rokonterületen tanított osztályozó modellt viszünk át ("transfer") egy új területre.

4.11.4. Többosztályos és többcímkés osztályozás

Az osztályok száma és viszonya szerinti legegyszerűbb eset a már említett *bináris osztályozás*, amikor két osztály adott, és egy-egy objektum (pédány) vagy az egyik vagy a másik osztályba tartozik (de nem mindkettőbe egyszerre!). Többosztályos, *multiclass* problémáról akkor beszéltünk, amikor kettő helyett több osztályunk volt, és egy-egy példány pontosan egy osztályba tartozott.

A többcímkés, *multilabel*, osztályozási feladatok annyiban különböznek a multiclass problémáktól, hogy egy objektum *egyidejűleg több osztályba* is tartozhat. Tekintsük pédaként azt az esetet, amikor embereket osztályozunk és az osztályok különböző betegségek szerinti kockázati csoportoknak felelnek meg. Egy ember egyidejűleg több betegség szerint is tartozhat a veszélyeztetettek csoportjába, így tehát egy objektum (egy ember példában) több osztályba is tartozhat egyidejűleg.

A multilabel osztályozási feladatokat a legegyszerűbb esetben az osztályonkénti bináris osztályozásra vezethetjük vissza: az első bináris osztályozó eldönti, hogy egy adott ember az első betegség szerint veszélyeztetett-e, a második első bináris osztályozó eldönti, hogy egy adott ember a második betegség szerint veszélyeztetett-e, stb. Az egymástól független bináris osztályozásokra való visszavezetés azonban általában nem optimális: nem veszi figyelembe, hogy az osztálycímkék legtöbbször nem függetlenek egymástól, hanem – valamilyen formában – korrelálnak egymással.

4.12. Ajánlórendszerek és ritka mátrixok faktorizációja

A ritka mátrixok faktorizációjára szolgáló algoritmusok (lásd pl. [Karimi és tsa., 2012], [Symeonidis és tsa., 2010], [Salakhutdinov és Mnih, 2008], [Kurucz és tsa., 2007]) első sorban a 1.2. részben már említett ajánlórendszerek alapjául szolgálnak [Koren, 2009]. Kimenetüket tekintve a regressziós algoritmusokhoz hasonlóak (folytonos skálán értelmezett számértéket adnak eredményül), ezért tárgyaljuk a mátrix faktorizációs algoritmusokat az osztályozó és regressziós algoritmusokkal egy fejezetben.

Az egyszerűség kedvéért a mátrixfaktorizációs algoritmusokat a filmajánlókon keresztül mutatjuk be, de hangsúlyozzuk, hogy számos további esetben használhatóak, amikor kettő (esetleg több) különböző típusú objektum adott (filmajánlók esetében: *felhasználók* és filmek) és az objektumok közti ismert kapcsolatokból újabb kapcsolatokat szeretnénk kikövetkeztetni. Ilyen eset lehet például a gének, betegségek és gyógyszerek közti potenciális új kapcsolatok feltárása, a szemantikus világháló RDF-hármasain végzett valószínűségi következtetés [Drumond, 2012], vagy annak előrejelzése, hogy várhatóan mely felhasználó mely blogot fogja kommentezni [Buza és Galambos, 2013].

A ritka mátrixok faktorizációja esetén abból indulunk ki, hogy adott egy mátrix. A mátrix sorai felelnek meg az egyik típusú objektumnak (felhasználóknak), oszlopai a másik típusú objektumnak (filmeknek). Ha az adott alkalmazásban kettőnél több típusú objektum van, akkor kétdimenziós mátrixok helyett magasabb dimenziószámú tenzorokkal dolgozunk és *ritka tenzorok faktorizációjáról* vagy egyszerűbben: *tenzor faktorizációról* beszélünk. Néhány objektum közti összefüggés adott: néhány (u, v) párra tudjuk, hogy az u jelű felhasználónak mennyire tetszett a v jelű videófilm (hány pontra értékelte). A mátrix celláinak legnagyobb része azonban üres. A gyakorlati alkalmazásokban (pl. filmajánlók) az összes cellának 99%-a, vagy akár ennél is nagyobb része lehet üres. Lásd a 4.22. ábrát.

A feladat a hiányzó cellák értékeinek kitöltése. Feltesszük, hogy van valamilyen összefüggés a megadott és hiányzó cellaértékek között, a megadott értékekből lehet következtetni a hiányzó értékekre: ha például egy felhasználónak tetszett egy film, akkor ugyanazon film egy hozzá hasonló ízlésű felhasználónak is várhatóan tetszeni fog.

4.12.1. Collaborative filtering

A hiányzó értékek becslésére számos eljárás ismert. A legegyszerűbbek egyike a *collaborative filtering*. A *user-based collaborative filtering* (felhasználó alapú

collaborative filtering) alapgondolata a követő: ha a hiányzó értékű cella az u felhasználóhoz és v videófilmhez tartozó sorban és oszlopban van, akkor az u felhasználó által korábban megadott, ismert értékelések alapján keresünk néhány u'_1, u'_2... felhasználót, amelyek hasonlóak u-hoz és amelyek értékelték a v videófilmet. Ezen értékelések alapján (például átlagolva ezeket, vagy súlyozott átlagukat számítva) becsüljük a hiányzó cella értékét.

Mivel a felhasználók és filmek szerepe felcserélhető, ezért hasonló felhasználók helyett kereshetünk hasonló filmeket is: olyan filmeket, amelyeket hasonlóan értékeltek a felhasználók. Ekkor *item-based collaborative filteringről* beszélünk. A két eljárást kombinálva kapjuk a *hybrid collaborative filtering* eljárásokat: a legegyszerűbb esetben kiszámolunk egy-egy számot user-based és item-based módon, és ezek átlaga lesz a hybrid eljárás által adott becslés.

A fent említett, nagyon egyszerű eljáráson kívül használhajtuk a korábban tárgyal SVD (singular value decomposition) egy változatát is a hiányzó értékek becsléséhez. Szintén nagyon népszerű a gradiens módszeren alapuló mátrix faktorizációs eljárás, amelyek a következő részben mutatunk be.

4.12.2. Gradiens módszeren alapuló mátrix faktorizáció

A ritka mátixok gradiens módszeren alapuló faktorizációja során az eredeti, hiányos értékeket tartalmazó $\mathbf{M}$ mátrixot két kisebb mátrix $\mathbf{U}$ és $\mathbf{V}$ szorzatára bontjuk, pontosabban: olyan $\mathbf{U}$ és $\mathbf{V}$ mátrixokat keresünk, amelyek szorzata jól közelíti $\mathbf{M}$ ismert értékeit. Az $\mathbf{U}$ sorainak száma megegyezik a felhasználók számával, míg $\mathbf{V}$ oszlopainak száma a videófilmek száma. $\mathbf{U}$ oszlopainak számát – amely megegyezik $\mathbf{V}$ sorainak számával, hiszen így lesz a két mátrix összeszorozható – k-val jelöljük, k az eljárás egyik paramétere, amelyet *rejtett faktorok számának* nevezhetünk. Az egyik legeszerűbb esetben a közelítés négyzetes hibája és a túl bonyolúlt modelleket büntető tag súlyozott összegéből álló célfüggvény minimumát keressük:

$$\sum_{i,j} \big(m_{i,j} - \sum_{k=0}^{K} u_{i,k} v_{k,j}\big)^2 + \lambda\big(\sum_{i,j} u_{i,j}^2 + \sum_{i,j} v_{i,j}^2\big) \tag{4.12}$$

A korábban említett k mellett λ is az eljárás paramétere.

A (4.12) függvény minimumát gradiens módszerrel keressük, eközben határozzuk meg $\mathbf{U}$ és $\mathbf{V}$ celláinak értékeit is.

Vegyük észre, hogy azzal az implicit feltételezéssel élünk, hogy a felhasználók és filmek leképezhetők egy k dimenziószámú vektortérbe, hiszen az $\mathbf{U}$ egyes sorai a felhasználóknak felelnek meg, $\mathbf{V}$ oszlopai pedig a videófilmeknek. A feltevéshez hozzá tartozik, hogy a leképkezés olyan, hogy a felhasználóknak és videófilmeknek megfelelő k dimenziós vektorok skaláris szorzata jellegzetes

arra nézve, hogy milyen erős a kapcsolat a két objektum között (mennyire szereti az adott felhasználó az adott filmet).

A következőkben azzal foglalkozunk, hogyan határozzuk meg $\mathbf{U}$ és $\mathbf{V}$ értékeit. Az alapötlet az, hogy kezdetben véletlenszerűen inicializáljuk a két mátrix elemeit, majd apró lépésekben egészen addig "igazítjuk" azokat, amíg $\mathbf{U} \times \mathbf{V}$ elég jól nem közelíti $\mathbf{M}$ ismert értékeit. Az ilyen javítási lépésekhez sorra vesszük $\mathbf{M}$ ismert értékeit (akár többször is végighaladunk $\mathbf{M}$ ismert értékein), és valahányszor egy olyan értékhez érünk, amely nem egyezik meg a hozzátartozó sorvektor és oszlopvektor szorzatával, "igazítunk" kicsit a sor- és oszlopvektorokon, kicsit csökkentjük vagy növeljük a megfelelő sor és oszlopvektor értékeit úgy, hogy szorzatuk *közelebb* legyen az $\mathbf{M}$ mátrix aktuálisan tekintett értékéhez. Ezt szemlélteti a 4.23. ábra.

A kérdés az, hogy milyen egy jó javítási lépés?

1. A javítás, a vektorok érékein végzett változtatás legyen arányos a hibával, legyen a hiba ϵ-szorosa. A 4.23. ábra példáján a hiba: $9 - 4 = 5$, ha $\epsilon = 0.1$, akkor épp az ábrán látható, $0.1 \times 5 = 0.5$-tel történő változtatást végezzük. Az ϵ az eljárás paramétere.

2. Vegyük észre, hogy a vektorok különböző komponensei más-más mértékben járulnak hozzá hibához. A 4.23. ábra példáján a sorvektor első komponensét (3-at) 1-gyel szoroztuk (az oszlopvektor első komponensével), a sorvektor második komponensét (2-t) pedig 3-mal szoroztuk (az oszlopvektor második komponensével). Látható, hogy a második komponens nagyobb mértékben járul hozzá a hibához. Logikus tehát, hogy a második komponenst nagyobb mértékben csökkentsük. A leggyakrabban alkalmazott eljárás a következő: miután az előző pontban látott módon kiszámoltuk, hogy mennyivel változzon a sorvektor egy komponensének értéke, ezt a számot még megszorozzuk az oszlopvektor hozzátartozó komponensének értékével. Így tehát a példában a sorvektor első komponensét, a 3-at, $(9-4) \times 0.1 \times 1 = 0.5$-del csökkentjük, a második komponenst, a 2-t pedig $(9 - 4) \times 0.1 \times 3 = 1.5$-del csökkentjük ($\epsilon = 0.1$ mellett). Hasonlóképpen vesszük figyelembe a sorvektor egyes komponenseit az oszlopvektor frissítésénél: az első komponenst, 1-t, $(9 - 4) \times 0.1 \times 3 = 1.5$-del csökkentjük, míg a másodikat $(9 - 4) \times 0.1 \times 2 = 1$-gyel csökkentjük ($\epsilon = 0.1$ mellett). Ezt a módosítási lépést intuíción alapulva vezettük be, azonban matematikailag igazolható, hogy ezen a módosítási lépés mellett az algoritmus a négyzetes hibát fogja minimalizálni (azaz az $\mathbf{U} \times \mathbf{V}$ által becsült értékek és $\mathbf{M}$ tényleges értékei közti négyzetes eltérést minimalizáljuk ilyen frissítési lépés mellett).

3. Ha az $\mathbf{U}$ illetve $\mathbf{V}$ mátrixbeli számok nagyon nagyok lesznek, az könnyen a

túltanulás egy formája lehet. Ezt úgy akadályozzuk meg, hogy az értékek frissítése során, az eddigiek mellett kivonjuk az eredeti érték λ-szorosát. Az ábrán jelölt sorvektor első komponensének (3-nak) új értéke tehát: $3-\big((9-4)\times\epsilon\times 1\big)-\big(\lambda\times 3\big)$ lesz. Belátható hogy így a négyzetes eltérés plussz az $\mathbf{U}$ és $\mathbf{V}$ mátrixok elemeinek négyzetösszegét, mint célfüggvényt optimalizáljuk. Az $\mathbf{U}$ és $\mathbf{V}$ mátrixok elemeinek négyzetösszegét gyakran regularizációs tényezőnek is nevezik.

Az $\mathbf{M}$ mátrix ismert elemeit általában nem elég egyetlen egyszer tekintenünk. Ezért a következőképpen járunk el: végigmegyünk az $\mathbf{M}$ mátrix ismert elemein, és elvégezzük a nekik megfelelő javítási lépéseket. Majd a továbbiakban iteratíve újra és újra végigmegyünk $\mathbf{M}$ ismert értékein és közben újabb javításokat végzünk. Az eljárás paraméterei tehát: ϵ, λ és az iterációk száma.

A fentiekben az egyik leginkább elterjedt mátrix faktorizációs eljárást írtuk le. A fenti eljárás rengeteg változata ismert attól függően, hogy milyen célfüggvényt optimalizálunk, és ezzel összefüggően milyen javítási lépéseket végzünk.

Ahogy emlt́ettük, a gyakorlati alkalmazásokban az $\mathbf{M}$ mátrix elemeinek túlnyomó többsége ismeretlen. Ezért az eljárás memória-hatékony implementációja során csak $\mathbf{M}$ ismert értékeit tároljuk például (sor, oszlop, érték) hármasok formájában. Az $\mathbf{U}$ és $\mathbf{V}$ mátrixok meghatározása után az ismeretlen értékek egyenként becsülhetők, a becslések egyenként kiírhatók a kimenetre, így valójában sohasem szükséges teljes $\mathbf{M}$ mátrix tárolása a memóriában, amely a legtöbb alkalmazásban hatalmas méretű.

4.13. További gyakorlati, alkalmazási problémák

Habár végig törekedtünk gyakorlat-közeli szempontból bemutatni az osztályozó és regessziós algoritmusokat, az osztályozók gyakorlati alkalmazásával kapcsolatos néhány kérdést még nem tárgyaltunk kellő részletességgel. Ebben a fejezetben ezt pótoljuk.

4.13.1. Többosztályos osztályozási feladatok visszavezetése bináris osztályozásra

Amint láttuk, számos osztályozó algoritmus, mint pédául a naive Bayes, döntési fák vagy a legközelebbi szomszéd osztályozó, könnyedén elboldogulnak többosztályos osztályozási feladatokkal. Más algoritmusokat viszont, úgy mint a szupport vektor gépeket vagy a logisztikus regressziót, alapvetően bináris (kétosztályos) osztályozási feladatok megoldására tervezték.

Szerencsére rendelkezésünkre állnak technikák arra, hogy egy többosztályos osztályozási feladatot bináris osztályozásra vezessünk vissza. Az egyik

ilyen technikával, a *one-versus-all*-lal már megismerkedtünk a logisztikus regresszióná a 4.3.5. fejezetben. Most további két technikát tekintünk át.

Az osztályok számát n-nel jelölve, az *all-versus-all* technika $\frac{1}{2}n(n-1)$ darab bináris osztályozásra vezeti vissza az adott többosztályos osztályozási feladatot. Ha például négy osztály adott, a, b, c, d, akkor az első osztályozó azt dönti el, hogy inkább a-ba tartozik-e egy objektum (példány), mint b-be, a második azt, hogy inkább a-ba tartozik-e egy objekum, mint c-be, stb. Általánosan: az osztályok minden párjának megfelel egy-egy bináris osztályozó. Az all-versus-all technika hátránya, hogy számításigényes és, hogy a páronkénti döntések inkonzisztensek lehetnek, a páronkénti döntéseket egyesítő, végső osztályt meghatározó algoritmus nem-triviális. Előny viszont, hogy a páronkénti döntéseket megfelelő módon egyesítő algoritmus mellett a végső döntés még akkor is helyes lehet, ha az $\frac{1}{2}n(n-1)$ osztályozó közül néhány hibás döntést hoz.

A többosztályos osztályozási feladatok bináris osztályozásra való visszavezetésének egy kifinomultabb módja a *hibajavító kódokon* alapul [Radovanović, 2011, Witten és tsa., 2011]. Az előző példát folytatva ezt is egy négyosztályos osztályozási feladat kontextusában mutatjuk be. Feleltessünk meg egy-egy bináris vektort a négy osztálynak az alábbiak szerint:

Osztály	Vektor
a	1111111
b	0000111
c	0011001
d	0101010

Mivel a vektorok kódolására hibajavító kódokat használtunk, ha valamelyik bit hibás, még mindig ki tudjuk következtetni, hogy melyik osztály kódjáról van szó: bármely egybites eltérés esetén egyértelmű, hogy melyik a leginkább hasonló kód.

Bináris osztályozókat a bináris vektorok egyes komponenseinek fogunk megfeleltetni: az első bináris osztályozó tehát azt hivatott eldönteni, hogy az osztályozandó objektum a osztályba tartozik-e vagy sem, a második bináris osztályozó azt dönti el, hogy az osztályozandó objektum a és d osztályok valamelyikébe *vagy* b és c osztályok valamelyikébe tartozik-e, stb. Ha a hét osztályozó közül legfeljebb egy hibázik, a végső osztályozás helyes lesz.

4.13.2. Kategorikus attribútumok kezelése

Egyes osztályozó algoritmusok, pl. naive Bayes vagy egyes döntési fák, egyaránt gond nélkül kezelik a szám típusú és a diszkrét típusú attribútumokat.

Ha egy algoritmus csak diszkrét attribútumokkal képes dolgozni, azaz nem képes a szám típusú attribútumok kezelésésre, a szám típusú attribútumainkat diszkretizálhatjuk az *Előfeldolgozási* fejezetben látott módon. Mit tehetünk azonban olyan esetben, amikor egy algoritmus csak szám típusú attribútumokat kezel (pl. szupport vektor gépek, logisztikus regresszió, egyes neurális hálók), és nem szeretnénk az adatbázisbeli diszkrét attribútumokat eldobni?

Ha az attribútum ordinális, azaz a sorrendiség értelmezett az értékei között, pédául: kicsi $<$ közepes $<$ nagy, megpróbálhatunk számként tekinteni az attribútumra és az egyes értékeket számokkal kódolni: -1 =kicsi, 0 =közepes, $+1$ =nagy.

Mit tehetünk azonban olyan esetben, amikor az attribútum nominális és nincs semmilyen sorrendezés az értékei között? Ilyen példa lehet az, amikor az attribútum értékei élelmiszereket (sajt, kenyét, tej, kolbász...), vagy országokat írnak le. Az egyes értékeket számoknak megfelelteni és képletekben azokkal számolni túlságos "merészség" lenne, a modell nagymértékben függhetne attól, hogy milyen értéket milyen számmal kódoltunk.

A leggyakrabban követett eljárás az előfeldolgozási lépések közt tárgyalt új attribútumok bevezetésének egy speciális esete. Ha egy nominális attribútum n darab különféle értéket vehet fel, akkor ezen attribútumot n darab bináris (0 vagy 1 értékű) attribútummal helyettesítjük: az olyan esetekben, amikor a nominális attribútum értékkészletének első értékét veszi fel, az első bináris attribútum értéke lesz 1, a többi pedig 0; az olyan esetekben, amikor a nominális attribútum értékkészletének második értékét veszi fel, a második bináris attribútum értéke lesz 1, a többié pedig 0; stb.

Alternatív megoldásként a nominális attribútum-értéket helyettesíthetjük annak valamilyen, az adott alkalmazásban releváns, számmal kifejezhető tulajdonságával: például egy élelmiszert a kalóriatartalamával vagy árával, országot a GDP-jének, lakosságának, területének nagyságával.

4.13.3. Feature extraction

Eddig végig azzal az implicit feltételezéssel éltünk, hogy adataink egy táblázatként képzelhetők el: a táblázat sorai felelnek meg az objektumoknak (példányoknak), oszlopai pedig az attribútumoknak. Másképp megközelítve: azt feltételeztük, hogy adataink *strukturáltak* abban az értelemben, hogy minden objektum ugyazon attribútumokkal rendelkezik, csak az attribútumok értékei különböznek objektumról objektumra.

Rengeteg valós alkalmazás esetében azonban az adataink *strukturálatlanok*, csak *részben strukturáltak* (semi-structured) vagy valamilyen más módon strukturáltak, pl. idősorok esetében világos, hogy az egymást követő időpontokban végzett mérések eredményei egyfajta struktúrát hordoznak, hiszen nem cserél-

hetők fel tetszőlegesen az idősorok elemei, ugyanakkor – különösen abban az esetben ha adatbázisunk különböző hosszúságú idősorokból áll – az idősorokat nehezen tudjuk "beleerőltetni" a táblázatos modellbe. Az idősorok elemzésére speciális technikák léteznek, amelyekkel külön fejezetben foglalkozunk majd. Mit tehetünk azonban (általánosan) ha egy alkalmazásban strukturálatlan vagy nem megfelelő módon strukturált adatok (szövegek, képek, videók) keletkeznek és ezeket szeretnénk elemezni?

Az általánosan követett eljárás szerint megpróbáljuk az adatokat valamilyen módon egy megfelelő táblázattá konvertálni. Az attribútumokat, a táblázat oszlopait más néven feature-öknek, tulajdonságoknak, jellemzőnek illetve leírónak is nevezik. Összegyűjtjük tehát az objektumok néhány (vagy akár sok) olyan tulajdonságát, amely az adott alkalmazás szempontjából releváns lehet, és amelyek értékeit a rendelkezésünkre álló adatok alapján meg tudjuk határozni. Képadatbázisok bányászata esetén egy tulajdonság lehet például a kék színű pixelek darabszáma: nyilván az adatbázis minden egyes képére (külön-külön) meg tudjuk határozni, hogy hány kék színű pixelt tartalmaz.

Túlmutat a jelen jegyzet keretein, hogy milyen feature-öket érdemes használnunk az egyes alkalmazásokban, de hangsúlyozzuk, hogy a megoldásunk sikere, minősége, pontossága nagyban múlhat azon, hogy milyen feature-öket választottunk, egy gyakorlati alkalmazában figyelmet kell szentelnünk ennek a kérdésnek.

4.13.4. Paraméterkeresés

Bár viszonylag egyszerű, fontossága miatt külön írunk a *paraméterkeresés* technikájáról.

A fejezetben tárgyalt osztályozó, regressziós és mátrix faktorizációs eljárások nagy része bőségesen rendelkezik a szakértő (felhasználó) által beállítandó paraméterekkel. Mátrix-faktorizációnál ilyen a λ, ϵ és az iterációk száma; neurális hálózatoknál ilyen a tanulási ráta, a hálózat struktúrája (rétegek száma, belső neuronok száma az egyes rétegekben), a tanítási iterációk száma; döntési fáknál az alkalmazott vágási függvény vagy az egy levélre eső objektumok (példányok) elvárt száma, stb.

Ritkán van elképzelésünk arról, hogy az adott alkalmazásban hogy érdemes a modell paramétereit megválasztanunk. Még ritkább az, amikor matematikailag is be tudjuk bizonyítani, hogy az alkalmazásra jellemző körülmények (pl. adatok eloszlása) mellett mi egy paraméter optimális értéke. Az esetek legnagyobb részében azonban egy igen egyszerű eljárást követünk: kipróbáljuk a modellt a paraméterek valamilyen értéke mellett, majd megnézzük, hogy a modell jól működik-e vagy sem.

Az egyik kérdés, ami adódik, hogy milyen stratégia szerint vegyük sorra

a paraméterek lehetséges értékeit? Ha kevés paraméterünk van (pl. 2 vagy 3 szám), akkor egy általunk választott intervallumon belül, adott felbontás mellett megvizsgálhatjuk az összes lehetséges kombinációt. Ha például 3 darab szám típusú paraméter értékére egyenként 10-10 lehetőséget vizsgálunk, az összesen $10 \times 10 \times 10 = 1000$ kombináció kipróbálását jelenti. Az ilyen, összes lehetséges kombináció kipróbálásán alapuló keresést nevezik *kimerítő keresésnek*-nek. Nem szükséges, hogy a vizsgált paraméterértékek egyenletesen legyenek elosztva a vizsgált tartományon belül: a szupport vektor gépek C paraméterének értékét vizsgálhatjuk például a $2^{-10}...2^{10}$ intervallumban, úgy hogy az alábbi értékeket tekintjük: $2^{-10}, 2^{-9}, 2^{-8}, ...$

Ha sok paraméterrel dolgozunk, a minden kombináció vizsgálata lehetetlen a gyakorlatban. Ilyenkor valamilyen lokális keresést használhatunk: kipróbáljuk az eljárást valamilyen paraméterértékekre, majd egyik-másik értéket kicsit csökkentjük/növeljük, és megnézzük, hogy javult-e a modell.

Adódik a kérdés, hogy a túltanulás jelenségének ismeretében hogyan osszuk fel a rendelkezésünkre álló címkézett adatokat a paraméterkereséshez? Azt javasoljuk, hogy három független részre osszuk a címkézett adatokat, ezeket jelöljük rendre $\mathcal{T}_1$-gyel, $\mathcal{T}_2$-vel és $\mathcal{T}_3$-mal. A $\mathcal{T}_1$-et használjuk a modell tanításához egy adott paraméterkombináció mellett. Ebben a kontextusban mindent olyan lépést a tanítás részének tekintünk, ami a modell (adott paraméterek melletti) létrehozásához vezet, így pédául a döntési fák metszését is a tanítás részének tekintjük. (Ha például döntési fát metszeni is szeretnénk, akkor $\mathcal{T}_1$-et tovább osztjuk $\mathcal{T}_{1,A}$ és $\mathcal{T}_{1,B}$ diszjunkt halmazokra.) Adott paraméterértékek mellett a $\mathcal{T}_1$ halmazon elkészített modellt a $\mathcal{T}_2$ halmaz segítségével kiértékeljük. Ez minden vizsgált paraméterkombinációra elvégezzük, így végül – $\mathcal{T}_2$ segítségével – kiválasztjuk a legjobb paraméterkombinációt. Az így kiválasztott modellt $\mathcal{T}_3$ halmazon kiértékeljük, ezáltal egy független halmaz segítségével becsüljük a modell minőségét.[21]

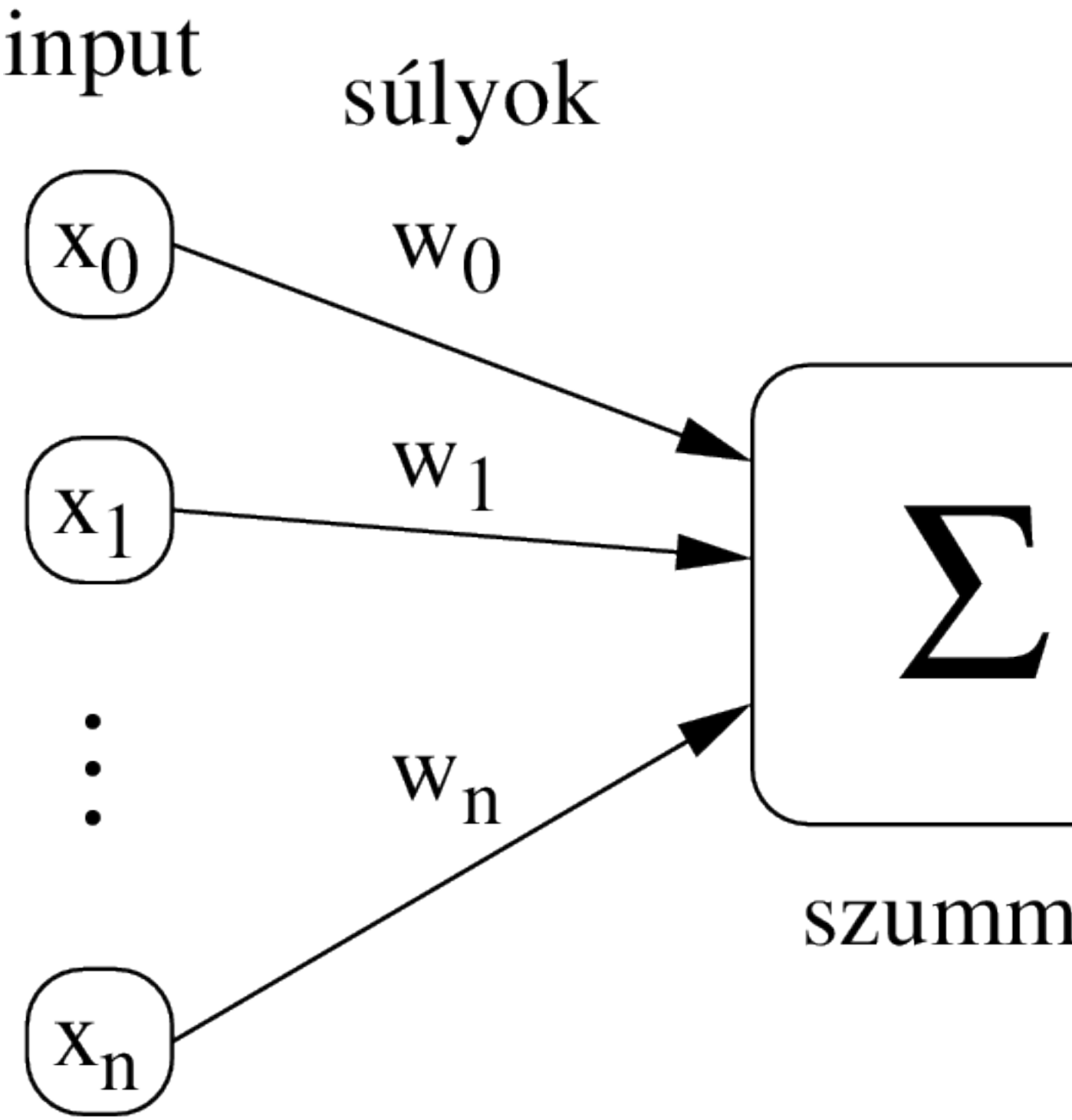
4.13.5. Mit tegyünk, ha az osztályozónk nem (elég) jó?

Egy tanítóadatbázis segítségével létrehozunk egy osztályozó (vagy regressziós) modellt, és, a korábban tárgyalt módon egy független adathalmaz segítségével leteszteljük, hogy mennyire jó. Mit tegyünk, ha a modell minősége alulmarad az elvárásainkhoz képest vagy, amire az adott alkalmazásban szükség van? Mikor érdemes azzal próbálkoznunk, hogy több adatot gyűjtünk? Előrelátható-e, hogy a több adat úgysem fog segíteni? Mikor érdemes bonyolultabb algoritmust használnunk és mikor felesleges?

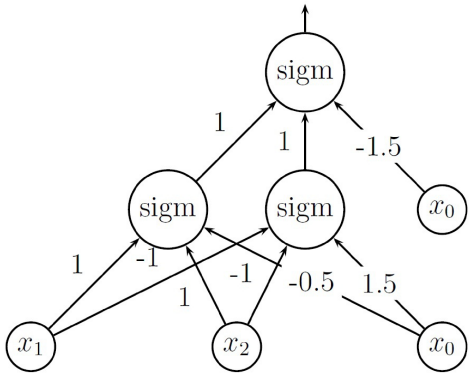
[21] A legjobb paraméterértékek kiválasztása után, a végső modellt $\mathcal{T}_1 \cup \mathcal{T}_2$-n taníthatjuk és ebben az esetben ezt a végső modellt értékeljük ki $\mathcal{T}_3$-n.

A válaszokhoz a korábban tárgyalt túltanulás jelenségén keresztül jutunk. Próbáljuk megmérni, vajon fellépett-e a túltanulás: mérjük a modell minőségét, pontosságát a tesztadatok mellett a tanítóadatokon is. Ha a tanítóadatokat jelentősen jobban osztályozza a modell, akkor egy túltanított modellünk van. A túltanuláson segíthet valamennyit, ha további adatot gyűjtünk, hiszen nagyobb adatbázisban "világosabban" megmutatkozhatnak az általános szabályszerűségek, nagyobb adatbázis mellett nagyobb az esélyünk, hogy az adatokban látható szabályszerűségek valóban általánosak, nem csak a véletlennek köszönhetően figyelhetők meg egy konkrét adatbázisban. Mivel a túltanulás azt jelenti, hogy túl speciális modellt hoztunk lére, a túltanulás ellen elsődlegesen regularizációval védekezhetünk: úgy modosítjuk az osztályozó algoritmust vagy annak paramétereit, hogy az osztályozó algoritmus a tanulási fázis végére egy egyszerűbb modellt hozzon létre. Egy neurális háló esetén csökkenthetjük a belső neuronok számát, döntési fáknál előírhatjuk, hogy a fa megengedett maximális mélysége kisebb legyen a fa korábbi mélységénél, mátrix faktorizációs algoritmusoknál nagyobbra állíthatjuk λ értékét, stb. Az is lehetséges, hogy egy másik algoritmust választunk, amelyről tudjuk, hogy egyszerűbb modellt hoz létre.

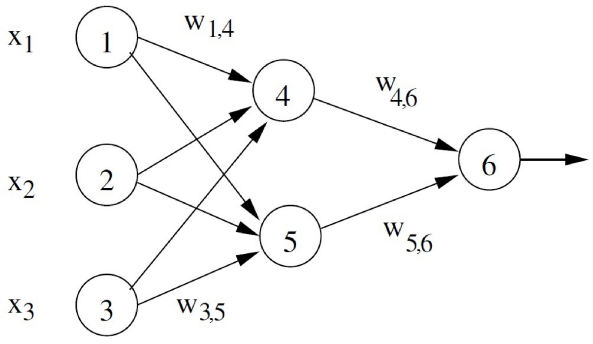
A túltanulással ellentétes eset, amikor a modell azért nem teljesít optimálisan, mert az osztályozó algoritmus túl egyszerű modellt hozott létre, amely nem volt képes megradagni minden szabályszerűséget. Ilyen esetben nincs sok értelme további adatot gyűjteni, ugyanakkor paraméterezhetjük (módosíthajtuk) úgy az osztályozó algoritmust, hogy a tanulási fázisban kifinomultabb, valamivel bonyolultabb modellt hozzon létre, illetve választhatunk egy másik algoritmust, amely bonyolultabb modellt hoz létre.



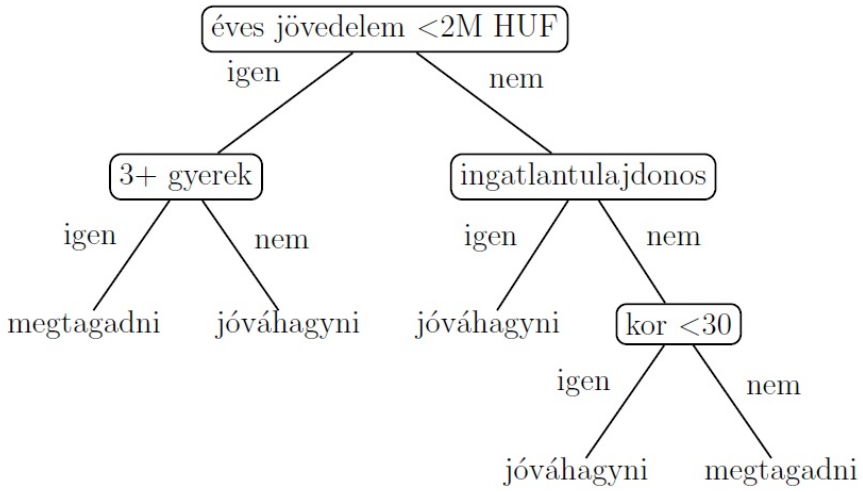
4.9. ábra. Logisztikus regresszió sémája



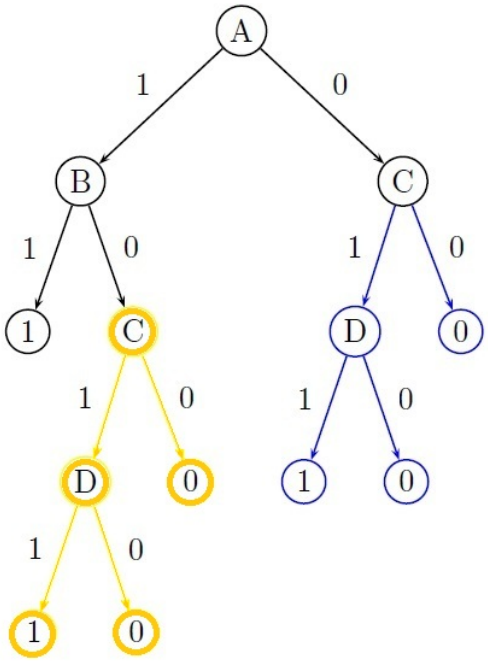
4.10. ábra. Az XOR (kizáró vagy) függvény logisztikus regressziók összekapcsolásával. A *sigm* rövidítés a szigmoidfüggvényt jelenti.



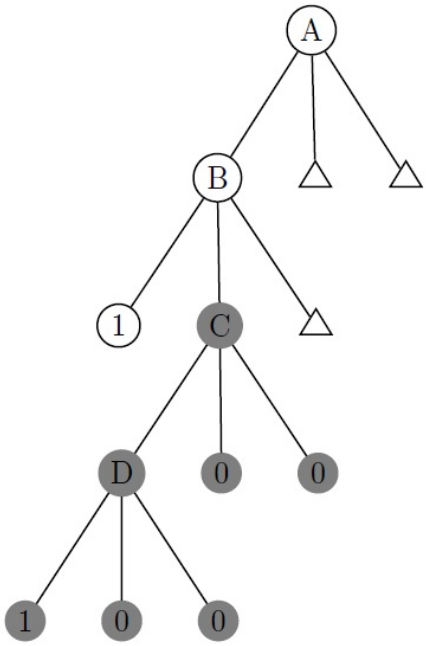
4.11. ábra. Többrétegű előrecsatolt neurális hálózat



4.12. ábra. Példa: döntési fa hitelbírálatra



4.13. ábra. Példa egy adott döntési sorozattal ekvivalens döntési fa

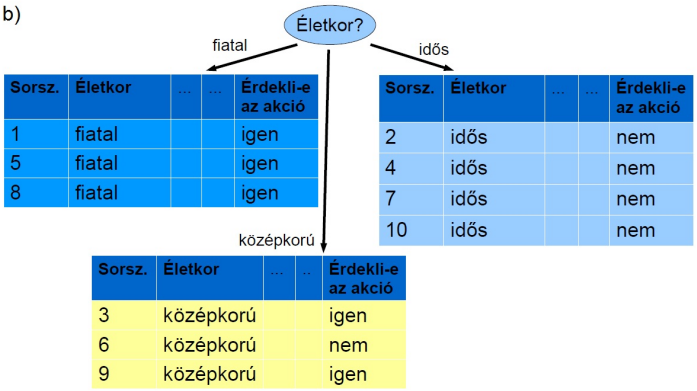


4.14. ábra. Az ismétlődő részfaprobléma szemléltetése

a)

Sorsz.	Életkor	Testsúly	Sportol	Érdekli-e az akció
1	fiatal	alacsony	igen	igen
2	idős	közepes	nem	nem
3	középkorú	magas	nem	igen
4	idős	közepes	igen	nem
5	fiatal	magas	nem	igen
6	középkorú	alacsony	nem	nem
7	idős	alacsony	nem	nem
8	fiatal	közepes	nem	igen
9	középkorú	magas	igen	igen
10	idős	közepes	igen	nem

b)

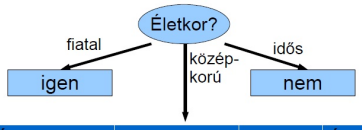


Sorsz.	Életkor	...	...	Érdekli-e az akció
1	fiatal			igen
5	fiatal			igen
8	fiatal			igen

Sorsz.	Életkor	...	...	Érdekli-e az akció
2	idős			nem
4	idős			nem
7	idős			nem
10	idős			nem

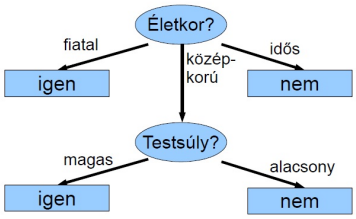
Sorsz.	Életkor	...	..	Érdekli-e az akció
3	középkorú			igen
6	középkorú			nem
9	középkorú			igen

c)



Sorsz.	Életkor	Testsúly	Sportol	Érdekli-e az akció
3	középkorú	magas	nem	igen
6	középkorú	alacsony	nem	nem
9	középkorú	magas	igen	igen

d)



4.15. ábra. Döntési fa rekurzív előállítása.

a) A tanítóadatbázis

Sorsz.	Életkor	Testsúly	Sportol	Érdekli-e az akció
1	fiatal	alacsony	igen	igen
2	idős	közepes	nem	nem
3	középkorú	magas	nem	igen
4	idős	közepes	igen	nem
5	fiatal	magas	nem	igen
6	középkorú	alacsony	nem	nem
7	idős	alacsony	nem	nem
8	fiatal	közepes	nem	igen
9	középkorú	magas	igen	igen
10	idős	közepes	igen	nem

b) A tanítóadatbázis alapján becsült feltételes valószínűségek és az osztályok apriori valószínűségei

Jelölések:

Magyarázó attribútumok:
X: életkor
Y: testsúly
Z: sportol

Magyarázandó attribútum (osztályváltozó):
C: érdekli-e az akció?

P(X=fiatal | C=igen) = 3/5
P(X=idős | C=igen) = 0
P(X=középk. | C=igen) =2/5

P(X=fiatal | C=nem) = 0
P(X=idős | C=nem) = 4/5
P(X=középk. | C=nem) = 1/5

P(Y=alacsony | C=igen) = 1/5
P(Y=közepes | C=igen) = 1/5
P(Y=magas | C=igen) =3/5

P(Z=igen | C=igen) = 2/5
P(Z=nem | C=igen) = 3/5

P(Z=igen | C=nem) = 2/5
P(Z=nem | C=nem) = 3/5

P(C=igen) = 5/10
P(C=nem) = 5/10

c) Új (ismeretlen osztályba tartozó) példány osztályozása

Sorsz.	X	Y	Z	C
11	fiatal	közepes	igen	?

$$P(C=igen \mid X=fiatal, Y=közepes, Z=igen) =$$

$$= \frac{P(C=igen)*P(X=fiatal|C=igen)*P(Y=közepes|C=igen)*P(Z=igen|C=igen)}{P(X=fiatal, Y=közepes, Z=igen)} =$$

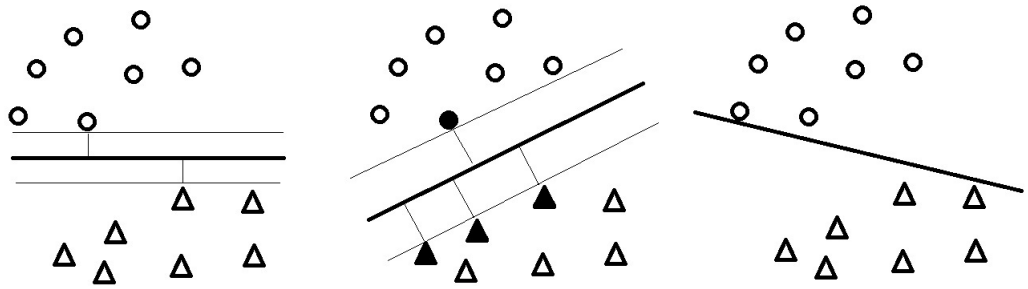
$$= \frac{5/10 * 3/5 * 1/5 * 2/5}{P(X=fiatal, Y=közepes, Z=igen)} = \frac{0{,}024}{P(X=fiatal, Y=közepes, Z=igen)}$$

Hasonlóképpen: $P(C=nem \mid X=fiatal, Y=közepes, Z=igen) =$

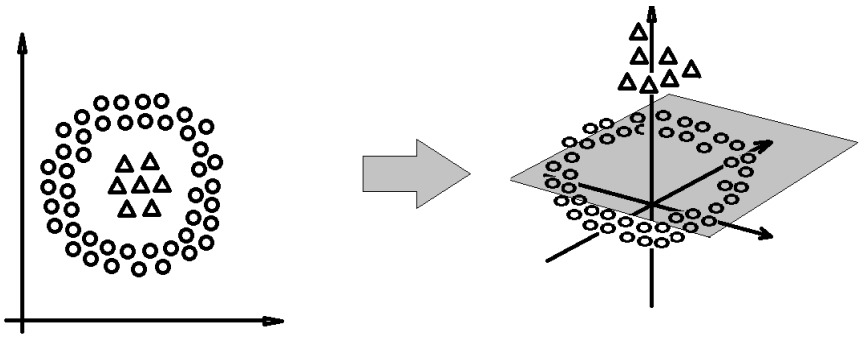
$$= \frac{5/10 * 0 * 3/5 * 2/5}{P(X=fiatal, Y=közepes, Z=igen)} = \frac{0}{P(X=fiatal, Y=közepes, Z=igen)}$$

Az "igen" osztály valószínűsége nagyobb, ezért az "igen" osztályba soroljuk.

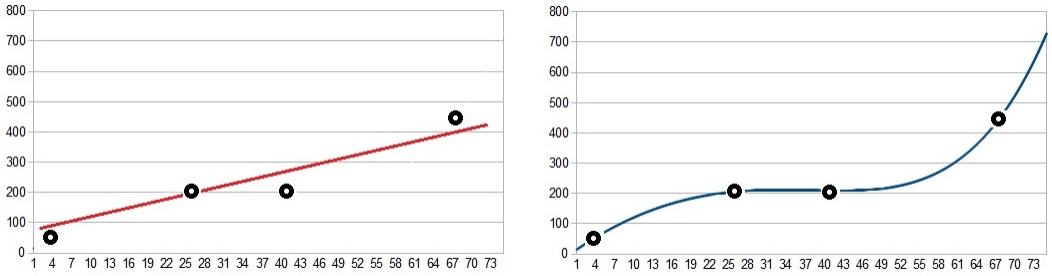
4.16. ábra. Példa: naív Bayes osztályozó.



4.17. ábra. A szupport vektor gépek olyan elválasztó egyenest keresnek, amely legjobban szeparálja a két osztály pontjait – a két osztályt háromszögekkel és körökkel jelöltük. A vastag vonallal jelzett három lehetséges elválasztó egyenes közül a szupport vektor gépek a középső ábrán láthatót választják. Vékony segédvonalakkal jeleztük az elválasztó egyenes és hozzá legközelebbi pontok távolságát, valamint az ezen pontokon átmenő, az elválsztó egyenessel párhuzamos egyeneseket. A vékony segédvonalakkal jelzett egyenesek közti távolság a margó.

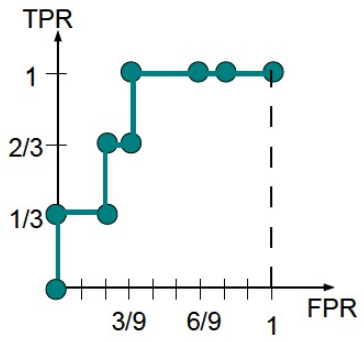


4.18. ábra. Ha az osztályok nem szeparálhatók lineárisan, az adatpontokat magasabb dimenziós térbe vetíthetjük, ahol már lineárisan szeparálhatóak lesznek az osztályok. A két osztályt háromszögekkel és körökkel jelöltük. Az ábra bal oldalán az eredeti adatpontok, jobb oldalon pedig ezek magasabb dimenziószámú térbeli képe látható. A jobb oldali ábra úgy értendő, hogy a vetítés hatására a háromszögek a szürkével jelölt sík fölé, a körök pedig az alá kerültek. A magasabb dimenziós térbeli elválasztó hipersík nem-lineáris szeparációnak felel meg az eredeti térben.

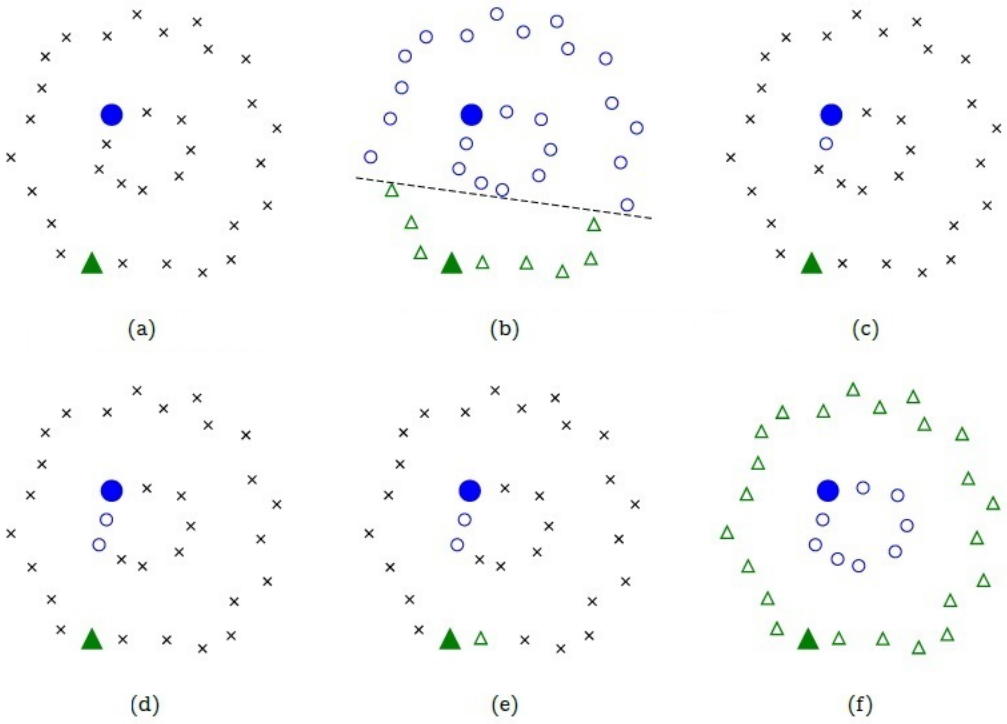


4.19. ábra. Példa a túltanulásra: a jobboldali ábrán látható görbe, 5-dik fokú polinom, minden pontra tökéletesen illeszkedik ugyan, mégsem mondhatjuk, hogy jobban megragadja az általános trendet, mint a jobboldali ábrán látható egyenes.

Valóság	-	-	-	-	-	-	+	-	+	-	-	+	
Model	0.12	0.28	0.37	0.39	0.41	0.46	0.49	0.51	0.53	0.54	0.6	0.8	
TP	3	3	3	3	3	3	3	2	2	1	1	1	0
FP	9	8	7	6	5	4	3	3	2	2	1	0	0
TN	0	1	2	3	4	5	6	6	7	7	8	9	9
FN	0	0	0	0	0	0	0	1	1	2	2	2	3
TPR	1	1	1	1	1	1	1	2/3	2/3	1/3	1/3	1/3	0
FPR	1	8/9	7/9	6/9	5/9	4/9	3/9	3/9	2/9	2/9	1/9	0	0

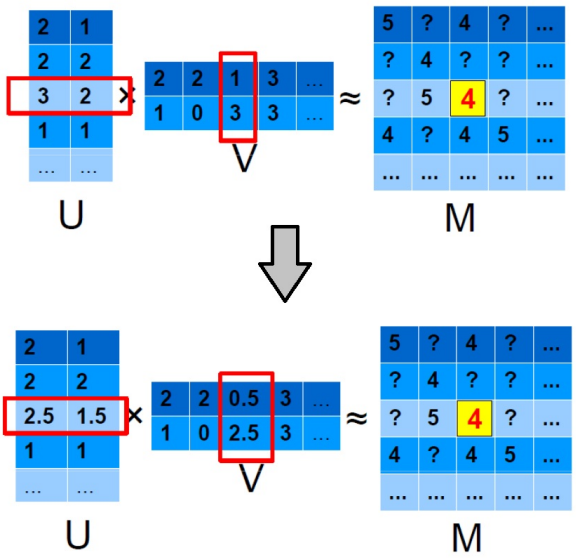


4.20. ábra. Egy folytonos kimenetet adó modell és AUC-ja. A tényleges osztálycímkék a táblázat *Valóság* megnevezésű sorában láthatók, a modell kimenetét a *Modell* sor mutatja. Minél nagyobb a kimenet értéke, a modell szerint annál nagyobb eséllyel tartozik az adott objektum a pozitív osztályba. A táblázatban a TP-k arányát (TPR) és FP-k arányát (FPR) számoltuk ki különböző θ küszöbértékekre: első oszlop: a modell az összes objektumot (példányt) pozitívnak nyilvánítja, második oszlop: a legkisebb kimenettel rendelkő objektumot (példányt) kivéve minden objektumot pozitívnak nyilvánít a model, stb.



4.21. ábra. Példa egy olyan esetre, amikor a semi-supervised protokoll szerinti osztályozás jobban teljesít, mint a konvencionális protokoll szerinti osztályozás. [Marussy és Buza, 2013] Az ábra részletes magyarázatát lásd a szövegben.

4.22. ábra. Az ajánlórendszerek hátterében álló adatok egy ritka mátrix elemei. A mátrix sorai a felhasználóknak felelnek meg, oszlopai az egyes termékeknek (filmeknek). Feltehetjük, hogy néhány terméket a felhasználók 1-től 5-ig terjedő skálán értékeltek. A termékek nagyrészéről azonban nem tudjuk, hogy egy-egy felhasználónak tetszenek-e vagy sem, ezeket az esetekel jelöltük kérdőjelekkel. A feladat az, hogy eldöntsük, mely termékek várhatóan mely felhasználóknak fognak fognak tetszeni.



4.23. ábra. A gradiens alapú mátrixfaktorizációs algoritmus egy javítólépésének szemléltetése: mivel $3 \times 1 + 2 \times 3 = 9$, ami nagyobb 4-nél, ezért az **M** mátrix jelölt cellájához tartozó **U**-beli és **V**-beli sor- illetve oszlopvektorban szereplő számokat csökkentjük, azért, hogy szorzatuk közelebb kerüljön 4-hez.

5. fejezet

Gyakori mintázatok és asszociációs szabályok

Különböző típusú gyakori mintázatok kinyerése az adatbányászat eltulajdoníthatatlan területe. A feladat kezdetben a vásárlói szokások kinyerésénél merült fel részfeladatként. A gyakran együtt vásárolt termékek, termékhalmazok ismerete potenciálisan profit-növekedést jelenthet, így ezek kinyerése jelentette az első lépést vásárlói szokások feltárásánál.

Egyes alkalmazásokban a gyakori minták meghatározásánál elemhalmazok helyett sorozatokat, gyökeres fákat, címkézett gráfokat vagy bool-formulákat keresnek. Az 5.3 fejezetben bemutatjuk a gyakori minták bányászatának absztrakt modelljét, majd egyesével vesszük a különböző típusú mintákat és megvizsgáljuk, hogy milyen technikák alkalmazhatók.

5.1. Gyakori elemhalmazok

A gyakori elemhalmazok bányászata nagyon népszerű kutatási terület volt az 1990-es évek végén és az új évezred első évtizedében. Mára valamit veszített ugyan népszerűségéből, de az új eljárások kutatása és a meglévő új helyzetre történő adaptációja folyamatos. A publikált algoritmusokkal könyveket lehetne megtölteni. Ezért ebben a jegyzetben csak a leghíresebb algoritmusokat és ötleteket ismertetjük.

5.1.1. A gyakori elemhalmaz fogalma

Legyen $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ elemek halmaza és $\mathcal{T} = \langle t_1, \ldots, t_n \rangle$ az $\mathcal{I}$ hatványhalmaza felett értelmezett sorozat, azaz $t_j \subseteq \mathcal{I}$. A $\mathcal{T}$ sorozatot *bemeneti sorozatnak* hívjuk, amelynek t_j elemei a *tranzakciók*. Az $I \subseteq \mathcal{I}$ elemhalmaz

fedése megegyezik azon tranzakciók sorozatával, amelyeknek részhalmaza az I. Az I elemhalmaz *támogatottsága* a fedésének elemszámával egyezik meg (jelölésben $supp(I)$). Az I *gyakori*, amennyiben támogatottsága nem kisebb egy előre megadott konstansnál, amelyet hagyományosan *min_supp*-pal jelölünk, és *támogatottsági küszöbnek* hívunk. A gyakori elemhalmazok keresése során adott egy $\mathcal{I}$ elemhalmaz, $\mathcal{T}$ bemeneti sorozat, *min_supp* támogatottsági küszöb, feladatunk meghatározni a gyakori elemhalmazokat és azok támogatottságát. Az egyszerűség kedvéért a halmazt jelölő kapcsos zárójeleket (sőt az elemek határoló vesszőt) gyakran elhagyjuk, tehát például az

$$\langle \{A, C, D\}, \{B, C, E\}, \{A, B, C, E\}, \{B, E\}, \{A, B, C, E\} \rangle$$

sorozatot

$$\langle ACD, BCE, ABCE, BE, ABCE \rangle$$

formában írjuk.

Az általánosság megsértése nélkül feltehetjük, hogy az $\mathcal{I}$ elemein tudunk egy rendezést definiálni, és a minták illetve a tranzakciók elemeit minden esetben nagyság szerint növő sorrendben tároljuk. Ezen rendezés szerinti lexikografikusan tudjuk rendezni az azonos méretű halmazokat.

A keresési teret úgy képzelhetjük el, mint egy irányított gráfot, amelynek csúcsai az elemhalmazok, és az I_1-ből él indul I_2-be, amennyiben $I_1 \subset I_2$, és $|I_1| + 1 = |I_2|$. A keresési tér bejárásán mindig ezen gráf egy részének bejárását fogjuk érteni. Tehát például a keresési tér szélességi bejárása ezen gráf szélességi bejárását jelenti.

Elterjedt, hogy a támogatottság helyett *gyakoriságot,* a támogatottsági küszöb helyett *gyakorisági küszöböt* használnak, melyeket $freq(I)$-vel, illetve *min_freq*-kel jelölnek. Az I elemhalmaz gyakoriságán a $supp(I)/|\mathcal{T}|$ hányadost értjük.

A gyakorlatban előforduló adatbázisokban nem ritka, hogy az elemek száma $10^5 - 10^6$, a tranzakcióké pedig $10^9 - 10^{10}$. Elméletileg már az eredmény kiírása is az $\mathcal{I}$ elemszámában exponenciális lehet, hiszen előfordulhat, hogy $\mathcal{I}$ minden részhalmaza gyakori. A gyakorlatban a maximális méretű gyakori elemhalmaz mérete $|\mathcal{I}|$-nél jóval kisebb (legfeljebb 20-30). Ezen kívül minden tranzakció viszonylag kicsi, azaz $|t_j| \ll |\mathcal{I}|$. A keresési tér tehát nagy, ami azt jelenti, hogy az egyszerű nyers erő módszerek (határozzuk meg minden elemhalmaz támogatottságát, majd válogassuk ki a gyakoriakat) elfogadhatatlanul lassan futnának.

A későbbiekben gyakran használjuk majd tranzakciók esetén a „szűrt" jelzőt. Egy tranzakció szűrt tranzakcióját úgy kaphatjuk meg, ha töröljük belőle a ritka elemeket. A szűrt tranzakciók minden információt tartalmaznak a gyakori elemhalmazok kinyeréséhez, ezért a legtöbb algoritmus első lépése a gyakori

elemek meghatározása, majd a szűrt tranzakciók előállítása. Ezután az eredeti adatbázist nem használják többé.

A bemenetet illetően három adattárolási módot szoktak elkülöníteni. *Horizontális adatbázis*ról beszélünk, ha a tranzakciókat azonosítóval látjuk el, és minden azonosítóhoz tároljuk a tranzakcióban található elemeket. *Vertikális adatbázisnál* minden elemhez tároljuk az elemet tartalmazó tranzakciók azonosítóit (sorszámát). A vertikális tárolás nagy előnye, hogy gyorsan megkaphatjuk egy elemhalmaz fedését (az elemekhez tartozó kosarak metszetét kell képezni), amiből közvetlen adódik a támogatottság. Mind a horizontális, mind a vertikális ábrázolási módnál használhatunk az elemek vagy tranzakciók felsorolása helyett rögzített szélességű bitvektorokat. Az i-edik elem (tranzakció) meglétét az i-edik pozícióban szereplő 1-es jelzi.

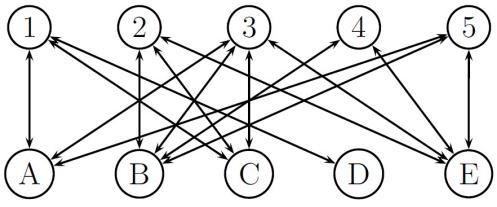
tranzakció	elemhalmaz
1	C
2	A,B,C

elem	tranzakcióhalmaz
A	2
B	2
C	1,2

tranzakció	elem
1	C
2	A
2	B
2	C

5.1. táblázat. Horizontális-, vertikális- és relációs tárolási mód

Tudjuk, hogy egy tranzakcióban változó számú elem lehet (és fordítva: egy elem változó számú tranzakcióban szerepelhet). A legtöbb mai adatbázis *relációs táblák* formájában van elmentve, amelyekben csak rögzített számú attribútum szerepelhet. A valóságban ezért a tranzakciók két attribútummal rendelkező relációs tábla formájában találhatók, ahol az első attribútum a tranzakciót, a második pedig az elemet adja meg (pontosabban a tranzakciók és az elemek azonosítóit). A három tárolási módszerre mutatnak példát a 5.1 táblázatok.



5.1. ábra. Gráfos ábrázolási mód

A bemenetet elemhalmazok sorozataként definiáltuk. Ábrázoljuk ezt, mint $G = (I, T, R)$ irányítatlan, páros gráf, vagy mint B bináris mátrix. Ha a t

tranzakció tartalmazza az i elemet, akkor és csak akkor az (i, t) eleme R-nek. Vagy mátrix esetén a t sorának i eleme 1 (különben 0). A

$$\langle ACD, BCE, ABCE, BE, ABCE \rangle$$

bemenethez tartozó gráf és bináris mátrix az 5.1 és az 5.2 ábrán látható.

	A	B	C	D	E
1	1		1	1	
2		1	1		1
3	1	1	1		1
4		1			1
5	1	1	1		1

5.2. ábra. Bináris mátrixos ábrázolási mód

A bemeneti adatot szokták a *sűrű* (dense) illetve a *ritka* (sparse) jelzővel illetni, amellyel a bináris mátrixban található 1-esek számára utalnak. Vásárlói kosarakat ábrázoló mátrix tipikusan ritka[1], ugyanis a kosarakban általában jóval kevesebb termék van (50-100), mint az összes termék száma (10 000-100 000).

A tranzakciók száma általában nagy, de a mai tárolókapacitások mellett, még egészen nagy adatbázisok is elférnek a memóriában: egy 10^{10} tranzakciót tartalmazó adatbázis csak kb. 1 GB helyet kíván, amennyiben a tranzakciók átlagos mérete 6 elem. Csak extrém nagy adathalmazok esetén nem alkalmazhatók azok az algoritmusok, amelyek feltételezik, hogy a bemenet (vagy a szűrt tranzakciók) elférnek a memóriában.

Mielőtt bemutatjuk az APRIORI módszert elemhalmazok esetén, gondolkozzunk el azon, vajon működne-e az alábbi egyszerű algoritmus a gyakorlatban. Olvassuk be a háttértárolóból az adatbázis első blokkját, és vizsgáljuk meg az első tranzakciót. Ennek a t_1 tranzakciónak az összes részhalmazát tároljuk el a memóriában és mindegyikhez rendeljünk egy számlálót 1 kezdeti értékkel. Az I elemhalmazhoz rendelt számláló fogja tárolni I támogatottságát. Az első tranzakció feldolgozása után vizsgáljuk meg sorban a többit: a t_i tranzakció minden részelemhalmazának számlálóját növeljük eggyel, vagy vegyük fel a memóriába egy új számlálóval, amennyiben az eddigi feldolgozott

[1] A ritka mátrix a gyakori mintázatok kontextusban kicsit másként értendő, mint a korábban látott ajánlórendszereknél: ott a mátrix abban az értelemben volt ritka, hogy a cellák nagyrészének értéke *ismeretlen* volt, itt viszont *tudjuk*, hogy a cellák nagyrésze azt kódolja, hogy az adott kosárban az adott termék *nem* fordul elő, nem pedig azt, hogy nem ismerjük, hogy a termék előfordul-e vagy sem.

tranzakciókban még nem fordult elő. Az adatbázis teljes végigolvasása után az összes – valahol előforduló – elemhalmaz támogatottsága rendelkezésünkre áll, amiből könnyen megkaphatjuk a gyakoriakat.

Látható, hogy ennél az egyszerű algoritmusnál lemezhozzáférés szempontjából gyorsabbat nem lehet találni, mert az adatbázis egyszeri végigolvasása mindenképpen szükséges a támogatottság meghatározásához és ennél az algoritmusnál elég is. A gyakorlatban mégsem használják ezt a gyors és egyszerű algoritmust. Ennek oka, hogy valós adatbázisokban nem ritka, hogy valamelyik tranzakció sok elemet tartalmaz. Egy átlagos szupermarketben mindennapos, hogy valaki 60 különböző elemet vásárol. Ekkor csak a számlálók mintegy 16 ezer TB-ot foglalnának, amennyiben a számlálók 4 byte-osak. A számlálókat mindenképpen a memóriában szeretnénk tartani, hiszen egy új tranzakció vizsgálatánál nem tudjuk előre, hogy melyik számlálót kell növelni.

Abban az esetben, ha biztosan tudjuk, hogy a tranzakciók egyike sem tartalmaz sok elemet, vagy az adatbázis bináris értékeket tartalmazó mátrix formájában adott, ahol az oszlopok (attribútumok) száma kicsi, akkor a fenti algoritmus hatékonyan használható.

Amir és szerzőtársai [Amir és tsa., 1997] a fenti algoritmus kis módosítását javasolták. Egyrészt csak olyan elemhalmazokat vizsgáltak, amelyek mérete nem halad meg egy előre megadott korlátot, másrészről a vizsgált elemhalmazokat és számlálóikat – a gyors visszakeresés érdekében – szófában tárolták. A módszernek két súlyos hátránya van: nem teljes (az algoritmus nem találja meg azokat az elemhalmazokat, amelyek mérete nagyobb az előre megadott küszöbnél), továbbá túlságosan nagy a memóriaigénye (sok lehet a hamis jelölt).

Amennyiben az adatbázisunk kicsi, akkor még a fenti algoritmusokat sem kell implementálnunk, mert egy szabványos adatbázis-lekérdező nyelv segítségével megkaphatjuk a gyakori elemhalmazokat. Az alábbi SQL parancs a gyakori elempárokat adja eredményül:

```
SELECT I.elem,J.elem, COUNT(I.tranzakció)
FROM tranzakciók I, tranzakciók J
WHERE I.tranzakció=J.tranzakció AND I.elem<J.elem
GROUP BY I.elem, J.elem
HAVING COUNT(I.tranzakció) >= min_supp
```

Látnunk kell, hogy a fenti parancs az összekapcsolás (FROM mezőben két tábla) művelet miatt nem fog működni, ha az adatbázis mérete túl nagy.

A következőkben bemutatjuk a három leghíresebb gyakori elemhalmazokat kinyerő (GYEK) algoritmust. Mindhárman az üres mintából indulnak ki. Az algoritmusok egy adott fázisában *jelöltnek* hívjuk azokat az elemhalmazokat, amelyek támogatottságát meg akarjuk határozni. Az algoritmus akkor *teljes,*

ha minden gyakori elemhalmazt megtalál és *helyes,* ha csak a gyakoriakat találja meg.

Mindhárom algoritmus három lépést ismétel. Először jelölteket állítanak elő, majd meghatározzák a jelöltek támogatottságát, végül kiválogatják a jelöltek közül a gyakoriakat. Természetesen az egyes algoritmusok különböző módon járják be a keresési teret (az összes lehetséges elemhalmazt), állítják elő a jelölteket, és különböző módon határozzák meg a támogatottságokat.

Az általánosság megsértése nélkül feltehetjük, hogy az $\mathcal{I}$ elemein tudunk definiálni egy teljes rendezést, és a jelöltek, illetve a tranzakciók elemeit ezen rendezés szerint tároljuk. Más szóval az elemhalmazokat sorozatokká alakítjuk. Egy sorozat ℓ-elemű *prefixén* a sorozat első ℓ eleméből képzett részsorozatát értjük. A példákban majd, amennyiben a rendezésre nem térünk ki külön, az ábécé szerinti sorrendet használjuk. A GYEK algoritmusok általában érzékenyek a használt rendezésre. Ezért minden algoritmusnál megvizsgáljuk, hogy milyen rendezést célszerű használni annak érdekében, hogy a futási idő, vagy a memóriaszükséglet a lehető legkisebb legyen.

A jelölt-előállítás *ismétlés nélküli,* amennyiben nem állítja elő ugyanazt a jelöltet többféle módon. Ez a hatékonyság miatt fontos, ugyanis ismétléses jelölt-előállítás esetében minden jelölt előállítása után ellenőrizni kellene, hogy nem állítottuk-e elő már korábban. Ha ezt nem tesszük, akkor feleslegesen kötünk le erőforrásokat a támogatottság ismételt meghatározásánál. Mindhárom ismertetett algoritmusban a jelöltek előállítása ismétlés nélküli lesz, amit a rendezéssel tudunk garantálni.

Az algoritmusok pszeudokódjaiban GY-vel jelöljük a gyakori elemhalmazok halmazát, J-vel a jelöltekét és j.számláló-val a j jelölt számlálóját. Az olvashatóbb kódok érdekében feltesszük, hogy minden számláló kezdeti értéke nulla, és az olyan halmazok, amelyeknek nem adunk kezdeti értéket (például GY), nem tartalmaznak kezdetben egyetlen elemet sem.

5.1.2. Az Apriori algoritmus

Az Apriori algoritmus az egyik legkorábbi GYEK algoritmus. Szélességi bejárást valósít meg, ami azt jelenti, hogy a legkisebb mintából (ami az üres halmaz) kiindulva szintenként halad előre a nagyobb méretű gyakori elemhalmazok meghatározásához. A következő szinten (iterációban) az eggyel nagyobb méretű elemhalmazokkal foglalkozik. Az iterációk száma legfeljebb eggyel több, mint a legnagyobb gyakori elemhalmaz mérete.

A jelöltek definiálásánál a következő egyszerű tényt használja fel: *Gyakori elemhalmaz minden részhalmaza gyakori.* Az állítást indirekten nézve elmondhatjuk, hogy egy elemhalmaz biztosan nem gyakori, ha van ritka részhalmaza. Ennek alapján ne legyen jelölt azon elemhalmaz, amelynek van ritka részhal-

maza. Az APRIORI algoritmus ezért építkezik lentről. Egy adott iterációban csak olyan jelöltet veszünk fel, amelynek összes valódi részhalmazáról tudjuk, hogy gyakori. Az algoritmus onnan kapta a nevét, hogy az ℓ-elemű jelölteket a bemeneti sorozat ℓ-edik átolvasásának megkezdése előtt (a priori) állítja elő. Az ℓ-elemű jelöltek halmazát J_ℓ-lel, az ℓ-elemű gyakori elemhalmazokat pedig GY_ℓ-lel jelöljük.

Algoritmus APRIORI

Require: $\mathcal{T}$: tranzakciók sorozata,
min_supp: támogatottsági küszöb,

$\ell \leftarrow 0$
$J_\ell \leftarrow \{\emptyset\}$
while $|J_\ell| \neq 0$ **do**
  támogatottság_meghatározás($\mathcal{T}, J_\ell$);
  $GY_\ell \leftarrow$ gyakoriak_kiválogatása(J_ℓ, min_supp);
  $J_{\ell+1} \leftarrow$ jelölt_előállítás(GY_ℓ);
  $\ell \leftarrow \ell + 1$;
end while
return GY

A kezdeti értékek beállítása után egy ciklus következik, amely akkor ér véget, ha nincsen egyetlen ℓ-elemű jelölt sem. A cikluson belül először meghatározzuk a jelöltek támogatottságát. Ehhez egyesével vesszük a tranzakciókat, és azon jelöltek számlálóját növeljük eggyel, amelyeket tartalmaz a vizsgált tranzakció. Ha rendelkezésre állnak a támogatottságok, akkor a jelöltek közül kiválogathatjuk a gyakoriakat.

Jelöltek előállítása

A JELÖLT-ELŐÁLLÍTÁS függvény az ℓ-elemű gyakori elemhalmazokból $(\ell+1)$-elemű jelölteket állít elő. Azok és csak azok az elemhalmazok lesznek jelöltek, amelyek minden részhalmaza gyakori.

A jelöltek előállítása során olyan ℓ-elemű, gyakori I_1, I_2 elemhalmaz párokat keresünk, amelyekre igaz, hogy

- I_1 lexikografikusan megelőzi I_2-t,
- I_1-ből a legnagyobb elem törlésével ugyanazt az elemhalmazt kapjuk, mintha az I_2-ből törölnénk a legnagyobb elemet.

Ha a feltételeknek megfelelő párt találunk, akkor képezzük a pár unióját, majd ellenőrizzük, hogy a kapott elemhalmaznak minden valódi részhalmaza gyakori-e. A támogatottság anti-monotonitása miatt szükségtelen az összes valódi

részhalmazt megvizsgálni; ha mind az $\ell + 1$ darab ℓ-elemű részhalmaz gyakori, akkor az összes valódi részhalmaz is gyakori. Az I_1, I_2 halmazokat a jelölt *generátorainak* szokás hívni.
Példa – Legyenek a 3-elemű gyakori elemhalmazok a következők: $GY_3 = \{ABC, ABD, ACD, ACE, BCD\}$. Az ABC és ABD elemhalmazok megfelelnek a feltételnek, ezért képezzük az uniójukat. Mivel $ABCD$ minden háromelemű részhalmaza a GY_3-nak is eleme, az $ABCD$ jelölt lesz. Az ACD, ACE pár is megfelel a két feltételnek, de uniójuknak van olyan részhalmaza (ADE), amely nem gyakori. Az APRIORI a következő iterációban tehát már csak egyetlen jelölt támogatottságát határozza meg.

A fenti módszer csak akkor alkalmazható, ha $\ell > 0$. Az egyelemű jelöltek előállítása egyszerű: minden egyelemű halmaz jelölt, amennyiben az üres elemhalmaz gyakori ($|\mathcal{T}| \geq min\_supp$). Ez összhangban áll azzal, hogy akkor lehet egy elemhalmaz jelölt, ha minden részhalmaza gyakori.

Jelöltek támogatottságának meghatározása

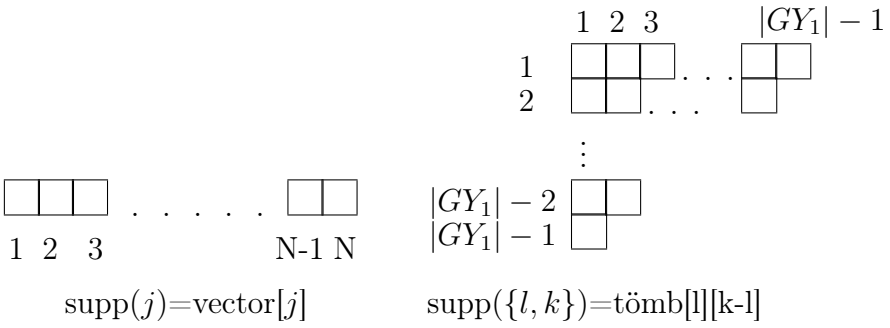
A jelöltek előfordulásait össze kell számolni. Ehhez egyesével vizsgáljuk a kosarakat, és azon jelöltek számlálóit növeljük eggyel, amelyeket tartalmaz a kosár.

1- és 2-elemű jelöltek támogatottsága – Könnyű dolgunk van, amennyiben a jelöltek mérete 1 vagy 2. A feladatot megoldhatjuk egy olyan lista, illetve féltömb segítségével, amelyekben a számlálókat tároljuk. Az elemek támogatottságának meghatározásánál a lista j-edik eleme tárolja a j-edik elem számlálóját. A tranzakciók feldolgozásánál végigmegyünk a tranzakció elemein és növeljük a megfelelő cellákban található számlálókat.

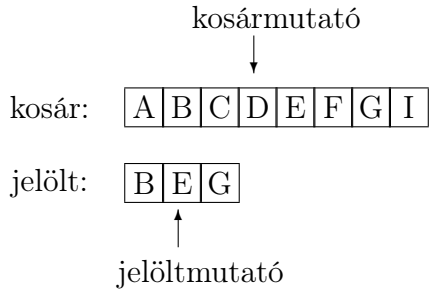
Az első végigolvasás után kiválogathatjuk a gyakori elemeket. A továbbiakban már csak ezekkel az elemekkel dolgozunk, így új sorszámokat adhatunk nekik a $[1..|GY_1|]$ intervallumból (GY_j-vel jelöljük a j-elemű gyakori mintákat). Az l és k-adik elemekből álló pár támogatottságát a tömb l-edik sorának $(k-l)$-edik eleme tárolja (az általánosság megsértése nélkül feltehetjük, hogy $l < k$).

Ha egy számláló 4 byte-ot foglal, akkor a tömb helyigénye nagyjából $4 \cdot \binom{|GY_1|}{2}$ byte. Azon elempárokhoz tartozó tömbelem értéke, amelyek sosem fordulnak elő együtt, 0 lesz. Helyet takaríthatunk meg, hogy ha csak akkor vesszük fel egy jelöltpár számlálóját, ha a párt legalább egy tranzakció tartalmazza [Ozden és tsa., 1998]. A párok támogatottságának meghatározása kevesebb memóriát fog igényelni, de ezzel együtt lassabb is lesz.

Nagyobb elemhalmazok támogatottsága – Vizsgáljuk meg részletesebben



5.3. ábra. Adatstruktúrák az 1- és 2-elemű jelöltek támogatottságának meghatározásához.

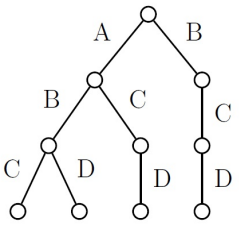


5.4. ábra. Jelöltmutató és kosármutató

az 5. sort. Adott egy tranzakció és ℓ-méretű jelöltek egy halmaza. Feladatunk meghatározni azon jelölteket, amelyek a tranzakció részhalmazai. Megoldhatjuk ezt egyszerűen úgy, hogy a jelölteket egyesével vesszük, és eldöntjük, hogy tartalmazza-e őket a tranzakció. Rendezett halmazban rendezett részhalmaz keresése elemi feladat.

Vegyünk fel két mutatót, amelyek a kosár, illetve a jelölt elemein fognak végighaladni. Kezdetben mutasson mindkét mutató az elemhalmazok első elemeire. Amennyiben a két mutató által mutatott elemek megegyeznek, akkor léptessük mindkét mutatót a következő elemre. Ha a tranzakcióban található elem kisebb sorszámú, akkor csak a kosár mutatóját léptessük, ellenkező esetben pedig álljunk meg; ekkor a kosár biztosan nem tartalmazza a jelöltet. Ha a jelölt utolsó eleme is megegyezik a kosár valamelyik elemével, akkor a kosár tartalmazza a jelöltet.

Ennek az egyszerű módszernek a hátránya, hogy sok jelölt esetén lassú,



5.5. ábra. Az ABC, ABD, ACD, BCD jelölteket tároló szófa.

hiszen annyiszor kell a tranzakció elemein végighaladni, amennyi a jelöltek száma. A gyorsabb működés érdekében a jelölteket szófában vagy hash-fában (hash-tree) célszerű tárolni. A szófát szokás prefix-fának vagy lexikografikus fának is hívni [Agrawal és tsa., 2001]. Az eredeti Apriori implementációban hash-fát alkalmaztak, azonban tesztek bizonyítják, hogy a szófa gyorsabb működést eredményez, mint a hash-fa. A hash-fa szófával való helyettesítéséről már a [Mueller, 1995]-ban írtak, ahol a szófát alkalmazó Apriori algoritmust SEAR-nek nevezték el. A továbbiakban a szófában való keresést ismertetjük (a szófák felépítéséről és típusairól az alapfogalmak 2.5.1 részében már szóltunk).

A *szófa* éleinek címkéi elemek lesznek. Minden csúcs egy elemhalmazt reprezentál, amelynek elemei a gyökérből a csúcsig vezető út éleinek címkéivel egyeznek meg. Feltehetjük, hogy az egy csúcsból induló élek, továbbá az egy úton található élek címkék szerint rendezve vannak (pl. legnagyobb elem az első helyen). A jelöltek számlálóit a jelöltet reprezentáló levélhez rendeljük. A 5.5. ábrán egy szófát láthatunk.

A t tranzakcióban az ℓ-elemű jelölteket úgy találjuk meg, hogy a jelölteket leíró fa gyökeréből kiindulva, rekurzív módon bejárunk bizonyos részfákat. Ha egy d szintű belső csúcshoz a tranzakció j-edik elemén keresztül jutunk, akkor azon élein keresztül lépünk eggyel mélyebb szintre, amelyeknek címkéje megegyezik a tranzakció j'-edik elemével, ahol $j < j' \leq |t| - \ell + d$ (ugyanis $\ell - d$ elemre még szükség van ahhoz, hogy levélbe érjünk). Ha ily módon eljutunk egy ℓ szintű csúcshoz, az azt jelenti, hogy a csúcs által reprezentált elemhalmazt tartalmazza t, így ennek a levélnek a számlálóját kell növelnünk eggyel.

A szófát prefix fának is szokták hívni, ami arra utal, hogy a közös prefixeket csak egyszer tárolja. Ettől lesz gyorsabb a szófás támogatottság-meghatározás a naiv módszernél. A közös prefixeket összevonjuk, és csak egyszer foglalkozunk velük.

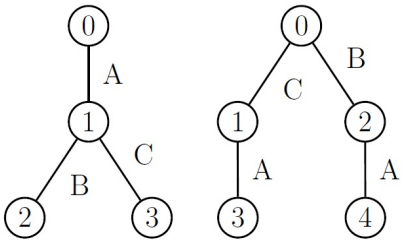
A szófa nagy előnye a gyors támogatottság-meghatározás mellett, hogy a jelölt-előállítást is támogatja. Tudjuk, hogy két gyakori elemhalmaz akkor lesz generátor, ha a legnagyobb sorszámú elemük elhagyásával ugyanazt az elem-

halmazt kapjuk, vagy más szavakkal, a két gyakori elemhalmaz $\ell - 1$ hosszú prefixei megegyeznek. A támogatottság-meghatározásában használt szófát felhasználhatjuk a következő iterációs lépés jelöltjeinek az előállítására, hiszen a szófa tárolja a jelölt-előállításhoz szükséges gyakori elemhalmazokat.

Az egész algoritmus alatt tehát egyetlen szófát tartunk karban, amely az algoritmus kezdetekor csak egy csúcsból áll (ez reprezentálja az üres halmazt). A támogatottság-meghatározás után töröljük azon leveleket, amelyek számlálója kisebb *min_supp*-nál. Az iterációs lépés végére kialakuló szófa alapján előállítjuk a jelölteket, amely során a szófa új, eggyel mélyebb szinten lévő levelekkel bővül. A jelölt-előállítás során arra is lehetőségünk van, hogy az előző iterációban gyakorinak talált elemhalmazokat és azok számlálóit kiírjuk (a kimenetre vagy a háttértárolóra).

A rendezés hatása az Apriori algoritmusra – Amennyiben a szófa hatékony adatstruktúra sorozatok tárolására, és gyors visszakeresésére, akkor ugyanez mondható el elemhalmazok esetére is. Ha tehát elemhalmazok adottak és az a feladat, hogy gyorsan megállapítsuk, hogy egy elemhalmaz szerepel-e a megadottak között, akkor elég definiálnunk az elemeken egy teljes rendezést, ami alapján a halmazokat sorozatokká alakíthatjuk.

Különböző rendezések különböző sorozatokat állítanak elő, amelyek különböző szófákat eredményeznek. Erre mutat példát a következő ábra, ahol két olyan szófát láthatunk, amelyek a AB, AC elemhalmazokat tárolják. Az első szófa az ABC szerint csökkenő sorrendet használ ($C \prec B \prec A$), míg a második ennek ellenkezőjét.



5.6. ábra. Példa: különböző rendezést használó szófák

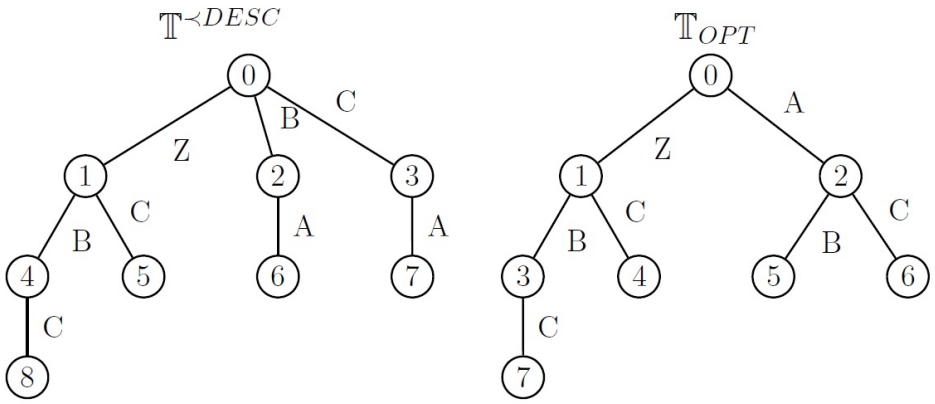
Egy szófa memóriaigénye arányos a szófa pontjainak számával, így jogos az az igény, hogy azt a teljes rendezést válasszuk, amely a legkevesebb pontú, azaz minimális méretű szófát adja. Ez az ún. minimális szófa előállításának feladata. Sajnos ez egy nehéz feladat.

5.1.1. Tétel *A minimális szófa probléma NP-nehéz .*

Eredetileg a feladatot n-esekre bizonyították, de ebből következik, hogy halmazokra is érvényes. Legyen ugyanis az alaphalmaz $\mathcal{I}$. Ekkor minden halmazt felfoghatunk, mint egy $|\mathcal{I}|$ hosszú bináris értékeket tartalmazó vektort.

A fenti példát szemlélve az embernek az az érzése támad, hogy az a rendezés adja a legkevesebb csúcsú szófát, amelyeben az elemek a halmazokban való előfordulások számának arányában csökkenő sorba vannak rendezve. Ugyanis a gyakori elemek fognak a halmazok kapott sorozatok elejére kerülni, és ezek az elemek, mivel gyakoriak sok sorozat elején lesznek megtalálhatók. A szófa a közös prefixeket csak egyszer tárolja, így akkor lesz a szófa mérete várhatóan a legkisebb, ha minél több sorozatnak van közös prefixe. Az előző ábra is ezt sugallta.

Sajnos a fenti módon kapott szófa nem feltétlenül adja a legkevesebb csúcsot tartalmazó szófát. Ezt a legegyszerűbben egy ellenpéldával tudjuk bizonyítani. Legyenek a halmazaink a következőek: AB, AC, CZ, BCZ, BZ, Z. A Z elem gyakorisága 4, a B, C-é 3 és az A elemé 2. Ha felrajzoljuk ezen gyakoriságok alapján kapott rendezés ($Z > B > C > A$, de a C, B elemek sorrende tetszőleges lehet) szerinti szófát, akkor a bal oldali szófát kapjuk. Ha az A és B, C elemek sorrendjét felcseréljük, akkor eggyel kevesebb pontot tartalmazó szófát kapunk (jobb oldal).



5.7. ábra. Ellenpélda arra, hogy az előfordulás szerinti csökkenő sorrend adja a minimális méretű szófát

Tapasztalatok alapján gyakoriság szerint csökkenő rendezés kisebb szófát eredményez, mint a gyakoriság szerint növekvő rendezés, vagy más véletlenszerűen megválasztott rendezések. Ennek ellenére olyan szófát célszerű alkalmazni, amelyben az elemeken értelmezett rendezés a gyakoriság szerint növekvő sorrendnek felel meg. Ennek ugyanis két előnye van. Egyrészről a szófa pontjai kisebbek lesznek (kevesebb él indul ki belőlük), de ami még fontosabb, hogy

a ritka elemek lesznek közel a gyökérhez. A ritka elemekkel kevesebb kosárbeli elem fog egyezni, ezáltal a szófa kisebb részét járjuk be a támogatott jelöltek meghatározása során.

A továbbiakban bemutatunk néhány gyorsítási ötletet, amelynek segítségével nagymértékben lecsökkenthető a szófa alapú Apriori algoritmus futási ideje és memóriaigénye.

Ritka jelöltek törlése

Ritka jelöltek törléséhez és a gyakori jelöltek kiírásához be kell járnunk a szófát és amikor levélbe érünk, akkor össze kell hasonlítani a levél számlálóját a támogatottsági küszöbbel. Ha a számláló nagyobb, akkor az eredményfájlba írjuk a levél által reprezentált halmazt és a számlálót. Ellenkező esetben töröljük a levelet.

A bejárást megtakaríthatjuk, ha a fenti műveletet a jelöltek előállításával együtt tesszük meg. A jelöltek előállításánál is be kell járni a szófát. A testvér levelek közül töröljük a ritka jelölteket, majd a megmaradtakból generáljunk új, eggyel nagyobb méretű jelölteket.

Zsákutca nyesés

Szükségtelen tárolni azon csúcsokat, amelyekből az összes elérhető levelet töröltük. Ezek ugyanis lassítják a támogatottságok meghatározását (miközben szerepet nem játszanak benne) és feleslegesen foglalják a memóriát.

Nem mindegy azonban, hogy mikor távolítjuk el a zsákutcákat. Ha például a AB, AC, BC két-elemű jelölt lett gyakori, akkor a BC levélből (de az AC-ból sem) nem fogunk új levelet felvenni, azaz a BC levél zsákutca lesz. Ezt a levelet azonban nem törölhetjük az ABC új jelölt felvétele előtt, hiszen a BC halmaz az ABC-nek valódi részhalmaza, így szükséges, hogy szerepeljen a fában.

Könnyű belátni, hogy tetszőleges I halmaz nem generátor, eggyel kisebb méretű, valódi részhalmaza lexikografikus rendezés szerint I után következik. Ezért, ha preorder bejárást használunk a jelöltek előállítása során, akkor egy levelet azonnal törölhetünk, ha belőle nem tudtunk új levelet felvenni. Garantált, hogy egyetlen részhalmazt sem töröltünk még, hiszen a valódi, nem generátor részhalmazokat csak később fogjuk meglátogatni a preorder bejárás szerint.

A bemenet tárolása

Amikor megvizsgálunk egy kosarat annak érdekében, hogy eldöntsük, mely jelölteket tartalmazza, akkor az operációs rendszer a háttértárolóból bemásolja

a tranzakciót a memóriába. Ha van elég hely a memóriában, akkor a tranzakció ott is marad, és amikor ismét szükség van rá, nem kell lassú IO műveletet végeznünk. A bemenetet tehát szükségtelen explicit eltárolnunk a memóriában, hiszen az operációs rendszer ezt megteszi helyettünk. Sőt, ha a program eltárolja a bemeneti adatot (például egy listában), akkor a valóságban duplán lesz eltárolva.

A bemenet tárolásának vannak előnyei is. Például összegyűjthetjük az azonos tranzakciókat és ahelyett, hogy többször hajtanánk végre ugyanazon a tranzakción a támogatott jelöltek meghatározását, ezt egyszer tesszük meg. Szükségtelen az eredeti tranzakciókat tárolni. Az első végigolvasás után rendelkezésre állnak a gyakori elemek. A ritka elemek úgysem játszanak szerepet, ezért elég a tranzakcióknak csak a gyakori elemeit tárolni. Ennek további előnye, hogy sokkal több azonos „szűrt" tranzakció lehet, ezáltal tovább csökken a támogatott jelölteket kereső eljárás meghívásának száma. Ráadásul az ℓ-edik végigolvasás során törölhetjük azokat a szűrt tranzakciókat, amelyek nem tartalmaznak egyetlen ℓ-elemű jelöltet sem.

A szűrt tranzakciókat célszerű olyan adatstruktúrában tárolni, amit gyorsan fel lehet építeni (azaz gyorsan tudjuk beszúrni a szűrt tranzakciókat) és gyorsan végig tudunk menni a beszúrt elemeken. Alkalmazhatunk erre a célra egy szófát, de tesztek azt mutatják, hogy egy piros-fekete fa (kiegyensúlyozott bináris fa), amelynek csúcsaiban egy-egy szűrt tranzakció található, még jobb megoldás, mert jóval kisebb a memóriaigénye.

Tranzakciók szűrése

A feldolgozás során a tranzakciókat módosíthatjuk/törölhetjük annak érdekében, hogy az Apriori még hatékonyabb legyen. A tranzakció szűrése alatt a tranzakció olyan elemeinek törlését értjük, amelyek nem játszanak szerepet az algoritmus kimenetének előállításában. A nem fontos elemek lassítják az algoritmust, gondoljunk itt a támogatottság meghatározásának módjára. A szófa egy belső csomópontjánál meg kell határoznunk a közös elemeket az élek címkéinek és a tranzakció elemeinek halmazában. Minél több elem van a tranzakcióban, annál tovább tart ez a művelet.

Szűrésnek tekinthetjük az első iteráció után végrehajtott lépést:

1. szűrés – Minden tranzakcióból töröljük a ritka elemeket.

Egyszerű szűrőötletek a következők:

2. szűrés – Az ℓ-edik iterációban a t tranzakció feldolgozása után töröljük a t-t, amennyiben a t elemeinek száma nem nagyobb, mint ℓ. Nyilvánvaló, hogy

ez a tranzakció nem tartalmaz olyan elemhalmazt, amely a későbbi iterációban lesz jelölt.

3. szűrés – Töröljük a tranzakciót, amennyiben az nem tartalmaz jelöltet.

Ennek az ötletnek a javított változata:

4. szűrés – Töröljük a tranzakció azon elemeit, amelyek nem elemei egyetlen olyan jelöltnek sem, amelyet tartalmaz a tranzakció.

Amennyiben az így keletkezett tranzakció mérete ℓ, akkor töröljük teljesen a tranzakciót. Például, ha a háromelemű jelöltek halmaza $\{ABC, ABD, BCD, FGH\}$ és $t = ABCDH$, akkor a H elemet törölhetjük a tranzakcióból. $t' = ABCGH$ esetében a teljes tranzakciót töröljük.

Az előző szűrőötletet tovább szigoríthatjuk. Mi kell ahhoz, hogy egy elem eleme legyen majd egy olyan $\ell + 1$-elemű j jelöltnek a következő iterációban, amelyet tartalmaz az aktuális jelölt. Szükséges feltétel, hogy a j minden ℓ-elemű részhalmazát tartalmazza a tranzakció. A j egy eleme pontosan ℓ darab részhalmaznak az eleme. Ez alapján:

5. szűrés – Töröljük a tranzakció azon elemeit, amelyek nem elemei ℓ darab olyan jelöltnek, amelyet tartalmaz a tranzakció.

Természetesen most is igaz, hogy ez után a szűrés után alkalmazzuk a második szűrő ötletet, ha ez lehetséges. A fenti példában a $t'' = ABCFGH$ tranzakciót ez a szűrés teljes egészében törli.

Equisupport nyesés

Az egyenlő támogatottságú elemhalmazok alapján történő, ún. equisupport nyesés talán a legelterjedtebb trükk a gyakori elemhalmazok kinyerésének meggyorsítására. A nyesés az equisupport tulajdonság egy következményét használja ki. A támogatottság meghatározásánál kihagyhatjuk azokat a halmazokat, amelyeknek van olyan ℓ-elemű valódi részhalmazuk, amelyek támogatottsága egyenlő egy (ℓ-1)-elemű részhalmazukéval.

Equisupport tulajdonság – Legyen $X \subset Y \subseteq \mathcal{I}$. Ha $supp(X) = supp(Y)$, akkor $supp(X \cup Z) = supp(Y \cup Z)$ teljesül minden $Z \subseteq \mathcal{I}$-re.

Ez az állítás minden $Z \subseteq \mathcal{I}$ elemhalmazra igaz, de nekünk elég lesz csak a $Z \subseteq \mathcal{I} \setminus Y$ halmazokra koncentrálnunk.

Az equisupport nyesés és a zárt elemhalmazok közötti összefüggés egyértelmű. Az X elemhalmaz nem zárt, és lezártja Y, amennyiben $X \subset Y$, $supp(X) = supp(Y)$, továbbá nem létezik olyan elemhalmaz, amelynek Y valódi részhalmaza, és támogatottsága megegyezik Y támogatottságával. Egy X elemhalmaz akkor, és csak akkor lehet egy egzakt (100% bizonyosságú) asszociációs szabály feltétel része, ha X nem zárt elemhalmaz. Az X elemhalmaz *kulcs minta* [Bastide és tsa., 2000], ha nincs vele egyenlő támogatottságú valódi részhalmaza.

Ha az Y jelöltnek a támogatottsága megegyezik az X-el jelölt prefixe támogatottságával, akkor felesleges az Y-t tartalmazó $Y \cup Z$ halmazokat mint új jelölteket előállítani, az equisupport tulajdonság alapján ezek támogatottsága $X \cup Z$ részhalmazukból közvetlenül számítható [Goethals, 2002].

Az alulról építkező algoritmusoknál (ilyen a már ismertetett Apriori, mellett a következő fejezetekben tárgyalt Eclat és Fp-growth is) a prefixek támogatottsága mindig elérhető, így a *prefix equisupport nyesést* (ahol az X az Y prefixe és $|X| + 1 = |Y|$) bármikor alkalmazhatjuk. A prefix equisupport nyesés a következőképpen működik: miután kiszámoltuk a P elemhalmaz gyerekeinek támogatottságát, a ritka elemek elhagyásakor ellenőrizzük, hogy a támogatottságuk egyenlő-e a szülő támogatottságával, azaz $supp(P)$-vel. Az ezt teljesítő elemeket nem kell figyelembe vennünk mint generátorokat a következő jelölt-előállítás során. Ezen jelölteket töröljük és az utolsó elemeiket egy halmazban tároljuk el, amit equisupport halmaznak hívunk és P-hez rendeljük. Vegyük észre, hogy az elemhalmazháló prefix bejárásnak köszönhetően a jelölt-előállítás során az $X \setminus Y \prec z$ minden $z \in Z$, ahol $\prec$ az elemhalmaz bejárásánál használt rendezés.

Amikor kiírjuk a GY gyakori elemhalmazt, vele együtt kiírjuk minden $E' \subseteq E$ halmazokkal vett unióját is, ahol E a GY prefixeinek equisuporthalmazainak uniója.

Példa – Legyenek a kételemű, A prefixű gyakori elemhalmazok a következők: $\{AB, AC, AD\}$ és $supp(A) = supp(AB) = supp(AC) = 4$ továbbá $supp(AD) = 3$. A többi A prefixű jelölt előállításához egyedül az AD elemhalmazt kell figyelembe vennünk. Azonban egy jelölt létrehozásához mind az Apriori, az Eclat- és az Fp-growth algoritmusnál legalább két elemhalmaz szükséges, így itt véget is ér az A prefixű halmazok feldolgozása. Az AD és A elemhalmazok kiírásakor BC minden részhalmazát is hozzájuk kell venni, így végül az AD, ABD, ACD, $ABCD$, valamint az A, AB, AC, ABC halmazok kerülnek kiírásra; az előbbiek támogatottsága 3, utóbbiaké 4 lesz.

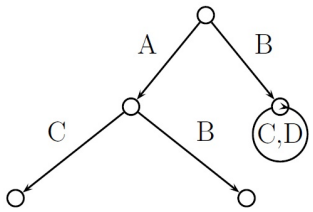
Ha az adatbázis csak zárt elemhalmazokat tartalmaz, akkor nem tudjuk ezt a nyesést alkalmazni, a támogatottságok egyenlőségének vizsgálata viszont lelassítja az algoritmust. A tapasztalat azonban azt mutatja, hogy az ellenőrzés

gyors (például az APRIORI algoritmusnál nem kell újra bejárni a szófát), és nem okoz cache miss-t. A kevés nemzárt elemhalmazt tartalmazó adatbázisoknál elenyésző a futásiidő növekedése. Az equisupport nyesés ezért biztonságos gyorsítási trükknek tekinthető.

A fenti leírásban nem használtuk ki az APRIORI algoritmus sajátosságait, csak azt, hogy az algoritmus alulról építkező és az elemhalmaz bejárás során definiálva van egy rendezés és így a prefix is. A továbbiakban jobban a részletekbe mélyedünk és megnézzük, hogy mit kell tennünk az APRIORI algoritmusban, ha a prefix equisupport nyesést kívánjuk alkalmazni.

Az APRIORI algoritmus szófás megközelítése esetén minden csúcshoz egy listát kell hozzávennünk, mely az equisupport halmaz elemeit tartalmazza. A ritkának bizonyuló jelöltek eltávolításakor ellenőrizzük, hogy a levél támogatottsága megegyezik-e prefixének támogatottságával. Ha igen, a levelet törölhetjük a szófából, és az éle címkéjét hozzáírjuk a szülő equisupport halmazához. Minden i elem egy equisupport halmazban tekinthető egy i címkéjű hurokélnek. A hurokéleket nem kell figyelembe venni a támogatottság meghatározásakor, de a jelölt-előállításnál igen.

Példa – Legyenek AB, AC, BC, BD a gyakori párok. $supp(AB) \neq supp(A) \neq supp(AC)$ és $supp(B) = supp(BC) = supp(BD)$. A 5.8 ábra a szófa ritka jelöltek eltávolítása utáni állapotát mutatja. Vegyük észre, hogy ha a hurokéleket figyelmen kívül hagytuk volna a jelöltgenerálás során, akkor az ABC elemhalmazt nem állítottuk volna elő mint jelölt, holott minden részhalmaza gyakori.



5.8. ábra. Példa: equisupport levelek eltávolítása

Ez a példa az equisupport nyesés és a zsákutcanyesés közti összefüggésre is felhívja a figyelmet. Láttuk, hogy a B csomópont nem vezet 2 mélységű levélbe, így a zsákutcanyesés törölte volna ezt a csúcsot, és nem lett volna jelölt az ABC elemhalmaz. Újra kell értelmeznünk a csomópontok mélységét a zsákutcanyesésnél azért, hogy ne töröljön olyan leveleket, amikre szükség lehet a jelölt-előállítás során. Az X elemhalmaz támogatottsága megegyezik az X olyan bővítésének támogatottságával, ahol a hozzáadott elem az X valamely prefixéhez tartozó equisupport halmaz egy eleme. Így amikor figyelembe

vesszük az X csomópont mélységét a zsákutcanyesés során, hozzá kell adnunk X aktuális mélységéhez a gyökérből az X-be vezető pontok equisupport halmazainak összméretét. Például az 5.8 ábrán látható szófán a B mélysége 1 helyett 3.

A szófa hatékony megvalósításának részleteit és további gyorsítási ötleteket a [Bodon, 2003a, Bodon, 2003b] és [Borgelt és Kruse, 2002] jelű cikkekben találhatunk. Egy olyan programcsomag, amely szófa alapú Apriori implementációt tartalmaz (továbbá hatékony Eclat és Fp-growth implementációt) és kutatási célokra szabadon letölthető a

`http://www.cs.bme.hu/~bodon/en/fim_env`

oldalról.

Borgelt-féle támogatottság-meghatározás

Ha a tranzakciókat szófában vagy Patrícia-fában (lásd 2.5.1. fejezet) tároljuk, akkor egy másik technikát is használhatunk a támogatottságok meghatározására. Ezt a módszert alkalmazza Christian Borgelt a világhírű Apriori implementációja utolsó változataiban [Borgelt, 2003, Borgelt és Kruse, 2002].

Az a megfigyelés áll az ötlet mögött, hogy két tranzakció a közös prefixig ugyanazt a programfutást eredményezi a támogatottság meghatározásakor (ugyanazt a szófarészt járjuk be). Ha szófában tároljuk a tranzakciókat, akkor rendelkezésre áll minden szükséges információ a közös prefixekről. Megoldható, hogy ugyanazokat a prefixeket csak egyszer dolgozzuk fel, és ne annyiszor, ahányszor előfordulnak.

A tranzakciófába minden csomóponthoz egy számlálót rendelünk. Az I elemhalmaz számlálója azoknak a tranzakcióknak a számát tárolja, amelyek prefixe I. Ebből a szempontból ez a megoldás eltér a bemenet tárolásánál bemutatott (lásd 5.1.2-es rész) szófa alapú megoldástól. A tranzakció szófánál és a jelölt szófánál használt rendezésnek meg kell egyeznie. Ez hátrány, mivel az egyes szófákhoz más-más rendezés lenne optimális.

Az algoritmus a következőkezőkben leírt módon működik [Borgelt, 2003]. Párhuzamosan bejárjuk a jelölt- és a tranzakció szófát duplán rekurzív módon. Két mutatót használunk, melyek kezdetben az egyes gyökerekre mutatnak. Ezután végigmegyünk mindkét csúcs élein. Ha a tranzakciószófa aktuális címkéje kisebb vagy egyenlő a másik címkénél, akkor rekurzívan továbblépünk a tranzakciószófában a gyerekcsomópontra (az szófa aktuális csomópontmutatója nem változik). Amennyiben a két címke egyenlő, a rekurziót azokkal a gyerekekkel folytatjuk, amelyekre a mutatók által mutatott élek mutatnak. A pszeudó-kód a Borgelt-féle támogatottság-meghatározás egy tovább optimalizált változatát adja meg.

Algoritmus BORGELT_SUPPCOUNT

Require: n_c: a szófa aktuális csomópontja,
n_t: a tranzakciófa aktuális csomópontja,
ℓ: az n_c-ből levélbe vezető út hossza,
i: az n_c legkisebb olyan élének indexe, amely címkéje nagyobb, mint az n_t-be vezető él címkéje

```
if ℓ = 0 then
  n_c.számláló ← n_c.számláló + n_t.számláló
else
  for j = 0 to n_t.élszám − 1 do
    while i < n_c.élszám AND n_c.él[i].címke < n_t.él[j].címke do
      i ← i + 1
    end while
    if i < n_c.élszám AND n_c.él[i].címke ≥ n_t.él[j].címke then
      BORGELT_SUPPCOUNT(n_c, n_t.él[j].gyermek, ℓ, i)
      if n_c.él[i].címke = n_t.él[j].címke then
        BORGELT_SUPPCOUNT(n_c.él[i].gyermek, n_t.él[j].gyermek, ℓ − 1, 0)
        i ← i + 1
      end if
    else
      break
    end if
  end for
end if
```

A fenti megoldásnak hátránya, hogy sok olyan utat jár be a jelölt szófában, amelyet az eredeti támogatottság meghatározó módszer nem tenne, mert nem vezet levélbe. A módszer nem veszi figyelembe, hogy a tranzakciónak csak egy részét kell kiértékelnünk. Megoldhatjuk a problémát, ha hozzárendelünk egy számlálót a tranzakció szófa minden pontjához. A számláló adja meg a pontból kiinduló leghosszabb út hosszát. A támogatottság meghatározása során nem vesszük figyelembe azokat a csomópontokat, melyek számlálója kisebb, mint $\ell - 1$, ahol ℓ azon lépések számát adja, amelyeket meg kell még tenni a jelölt szófa aktuális pontjából, hogy levélbe jussunk. Az algoritmus gyorsítható, ha a tranzakció szűrésének ötletét (lásd 5.1.2-ös rész) is alkalmazzuk. További részletek tudhatunk meg a [Borgelt, 2003] tanulmányból.

Futási idő és memóriaigény

A gyakori elemhalmaz keresési feladat megadásakor elmondtuk, hogy már az eredmény kiírása – ami a futási időnek a része – az $|I|$-ben exponenciális lehet. A memóriaigényről is hasonló mondható el. Az $(\ell + 1)$-elemű jelöltek előállí-

tásához szükségünk van az összes ℓ-elemű jelöltre, amelyek száma akár $\binom{|I|}{|I|/2}$ is lehet. Ezek a felső korlátok élesek is, hiszen $min\_supp = 0$-nál minden elemhalmaz gyakori.

Borgelt algoritmusának indítása előtt tehát nem sokat tudunk mondani a futási időről. A futás során, azonban egyre több információt gyűjtünk, így felmerül a kérdés, hogy ezt fel tudjuk-e használni az algoritmus maradék futási idejének jóslására. Például, ha a gyakori elemek száma négy, akkor tudjuk, hogy a legnagyobb gyakori elemhalmaz mérete legfeljebb négy (azaz még legfeljebb háromszor olvassuk végig az adatbázist), az összes jelölt maximális száma pedig $\binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 11$. A következőkben megvizsgáljuk, hogy mit tudunk elmondani a jelöltek számáról és a maximális jelöltek méretéről, ha adottak az ℓ-elemű gyakori elemhalmazok (GY_ℓ).

A következő rész fontos fogalma a kanonikus reprezentáció lesz.

5.1.2. Lemma *Adott n és ℓ pozitív egészek esetében a következő felírás egyértelmű:*

$$n = \binom{m_\ell}{\ell} + \binom{m_{\ell-1}}{\ell-1} + \cdots + \binom{m_r}{r},$$

ahol $r \geq 1$, $m_\ell > m_{\ell-1} > \cdots > m_r$ és $m_j \geq j$ minden $j = r, r+1, \ldots, \ell$ számra.

Ezt a reprezentációt hívják *ℓ-kanonikus reprezentációnak*. Meghatározása nagyon egyszerű: m_ℓ-nek ki kell elégítenie a $\binom{m_\ell}{\ell} \leq n < \binom{m_\ell+1}{\ell}$ feltételt, $m_{\ell-1}$-nek a $\binom{m_{\ell-1}}{\ell-1} \leq n - \binom{m_\ell}{\ell} < \binom{m_{\ell-1}+1}{\ell-1}$ feltételt, és így tovább, amíg $n - \binom{m_\ell}{\ell} - \binom{m_{\ell-1}}{\ell-1} - \cdots - \binom{m_r}{r}$ nulla nem lesz.

Legyen $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ elemek halmaza és GY_ℓ egy olyan $\mathcal{I}$ feletti halmazcsalád[2], amelynek minden eleme ℓ-elemű. Az ℓ-nél nagyobb méretű $I \subseteq \mathcal{I}$ halmaz *fedi* a GY_ℓ-et, ha I minden ℓ-elemű részhalmaza eleme GY_ℓ-nek. Az összes lehetséges $(\ell+p)$-méretű GY_ℓ-et fedő halmazokból alkotott halmazcsaládot $J_{\ell+p}(GY_\ell)$-lel jelöljük. Nem véletlen, hogy ezt a halmazt ugyanúgy jelöltük, mint az Apriori algoritmus jelöltjeit, ugyanis az $(\ell+p)$-méretű jelöltek ezen halmazcsaládnak az elemei, és ha az algoritmus során minden jelölt gyakori, akkor az $(\ell+p)$-méretű jelöltek halmaza megegyezik $J_{\ell+p}(GY_\ell)$-lel.

A következő tétel megadja, hogy adott GY_ℓ esetén legfeljebb mennyi lehet a $J_{\ell+p}(GY_\ell)$ elemeinek száma.

5.1.3. Tétel *Ha*

$$|GY_\ell| = \binom{m_\ell}{\ell} + \binom{m_{\ell-1}}{\ell-1} + \cdots + \binom{m_r}{r}$$

[2]A H-t az $\mathcal{I}$ feletti *halmazcsaládnak* nevezzük, amennyiben $H \subseteq 2^{\mathcal{I}}$.

ℓ-kanonikus reprezentáció, akkor

$$|J_{\ell+p}(GY_\ell)| \leq \binom{m_\ell}{\ell+p} + \binom{m_{\ell-1}}{\ell-1+p} + \cdots + \binom{m_s}{s+p},$$

ahol s a legkisebb olyan egész, amelyre $m_s < s+p$. Ha nincs ilyen egész szám, akkor a jobb oldal 0.

A fenti tétel a Kruskal–Katona tétel következménye, ezért a tételben szereplő felső korlátot a továbbiakban $KK_\ell^{\ell+p}(|GY_\ell|)$-el jelöljük.

5.1.4. Tétel *A 5.1.3.. tételben szereplő felső korlát éles, azaz adott n, ℓ, p számokhoz mindig létezik GY_ℓ, amelyre $|GY_\ell| = n$, és $|J_{\ell+p}(GY_\ell)| = KK_\ell^{\ell+p}(|GY_\ell|)$.*

A kanonikus reprezentáció segítségével egyszerű éles felső becslést tudunk adni a legnagyobb jelölt méretére (jelölésben $maxsize(GY_\ell)$) is. Tudjuk, hogy $|GY_\ell| < \binom{m_\ell+1}{\ell}$, ami azt jelenti, hogy nem létezhet olyan jelölt, amelynek mérete nagyobb m_ℓ-nél.
Következmény – Amennyiben a $|GY_\ell|$ számnak az ℓ-kanonikus reprezentációjában szereplő első tag $\binom{m_\ell}{\ell}$, akkor $maxsize(GY_\ell) \leq m_\ell$.

Az m_ℓ számot a továbbiakban $\mu_\ell(|GY_\ell|)$-el jelöljük. Ez az érték azt is megmondja, hogy mekkora jelöltméretnél válik nullává a felső korlát, azaz:
Következmény – $\mu_\ell(|GY_\ell|) = \ell + \min\{p | KK_\ell^{\ell+p}(|GY_\ell|) = 0\} - 1$

A maradék futási idő jóslására a következő állítás nyújt segítséget.
Következmény – Az összes lehetséges ℓ-nél nagyobb méretű jelölt száma legfeljebb

$$KK_\ell^{\text{összes}}(|GY_\ell|) = \sum_{p=1}^{\mu_\ell(|GY_\ell|)} KK_\ell^{\ell+p}(|GY_\ell|).$$

A fenti korlátok szépek és egyszerűek, mivel csak két paramétert használnak: az ℓ aktuális méretet és az ℓ-elemű gyakori elemhalmazok számát ($|GY_\ell|$). Ennél jóval többet tudunk. Nem csak a gyakori elemhalmazok számát ismerjük, hanem már pontosan meghatároztuk őket magukat is! Az új információ segítségével számos esetben jobb felső korlátot adhatunk. Például, ha a GY_ℓ-ben csak páronként diszjunkt elemhalmazok vannak, akkor nem állítunk elő jelölteket. A 5.1.3.. tételben szereplő felső korlát azonban jóval nagyobb lehet nullánál. A következőkben bemutatjuk, hogyan lehet a meglévő felső korlátot az ℓ méretű gyakori elemhalmazok *struktúrájára* rekurzívan alkalmazni. Ehhez feltesszük, hogy egy teljes rendezést tudunk definiálni az $\mathcal{I}$ elemein, ami

alapján tetszőleges elemhalmaznak meg tudjuk határozni a legkisebb elemét. Vezessük be a következő két jelölést:

$$GY_\ell^i = \{I - \{i\} | I \in GY_\ell, i = \min I\},$$

A GY_ℓ^i halmazt úgy kapjuk GY_ℓ-ből, hogy vesszük azon halmazokat, amelyek legkisebb eleme i, majd töröljük ezekből az i elemet.

Ezek után definiálhatjuk a következő rekurzív függvényt tetszőleges $p > 0$-ra:

$$KK^*_{\ell,p}(GY_\ell) = \begin{cases} \binom{|GY_\ell|}{p+1} & , \text{ ha } \ell = 1 \\ \min\{KK_\ell^{\ell+p}(|GY_\ell|), \sum_{i\in\mathcal{I}} KK^*_{\ell-1,p}(GY_\ell^i)\} & , \text{ ha } \ell > 1. \end{cases}$$

A definícióból következi, hogy $KK^*_{\ell,p}(GY_\ell) \leq KK_\ell^{\ell+p}(|GY_\ell|)$, továbbá

5.1.5. Tétel $|J_{\ell+p}(GY_\ell)| \leq KK^*_{\ell,p}(GY_\ell)$.

Bizonyítás – A bizonyítás teljes indukción alapul, az $\ell = 1$ eset triviális. Tulajdonképpen csak azt kell belátni, hogy

$$|J_{\ell+p}(GY_\ell)| \leq \sum_{i\in\mathcal{I}} KK^*_{\ell-1,p}(GY_\ell^i)$$

Az egyszerűség kedvéért vezessük be a következő jelölést: $H \cup i = \{I \cup \{i\} | I \in H\}$, ahol H egy $\mathcal{I}$ feletti halmazcsalád. Vegyük észre, hogy $GY_\ell = \sum_{i\in\mathcal{I}} GY_\ell^i \cup i$ és $GY_\ell^i \cap GY_\ell^j = \emptyset$ minden $i \neq j$ elempárra. Azaz a GY_ℓ halmazcsalád egy partícióját képeztük.

Amennyiben $I \in J_{\ell+p}(GY_\ell)$, és I-nek legkisebb eleme i, akkor $I \setminus \{i\} \in J_{\ell-1+p}(GY_\ell^i)$, hiszen $I \setminus \{i\}$ minden $(\ell-1)$-elemű részhalmaza GY_ℓ^i-beli. Ebből következik, hogy

$$J_{\ell+p}(GY_\ell) \subseteq \bigcup_{i\in\mathcal{I}} J_{\ell-1+p}(GY_\ell^i) \cup i.$$

Abból, hogy az GY_ℓ^i halmazcsaládok páronként diszjunktak következik, hogy $J_{\ell-1+p}(GY_\ell^i) \cup i$ is páronként diszjunkt halmazcsaládok. Ebből következik az állítás, hiszen:

$$\begin{aligned} |J_{\ell+p}(GY_\ell)| &\leq |\bigcup_{i\in\mathcal{I}} J_{\ell-1+p}(GY_\ell^i) \cup i| \\ &= \sum_{i\in\mathcal{I}} |J_{\ell-1+p}(GY_\ell^i) \cup i| \\ &= \sum_{i\in\mathcal{I}} |J_{\ell-1+p}(GY_\ell^i)| \\ &\leq \sum_{i\in\mathcal{I}} KK^*_{\ell-1,p}(GY_\ell^i), \end{aligned}$$

ahol az utolsó egyenlőtlenségnél az indukciós feltevést használtuk.

A páronként diszjunkt halmazok esete jó példa arra, hogy a minimum kifejezésben szereplő második tag kisebb lehet az elsőnél. Előfordulhat azonban az ellenkező eset is. Például legyen $GY_2 = \{AB, AC\}$. Könnyű ellenőrizni, hogy $KK_2^3(|GY_2|) = 0$, ugyanakkor a második tagban szereplő összeg 1-et ad. Nem tudhatjuk, hogy melyik érték a kisebb, így jogos a két érték minimumát venni.

Javíthatjuk a legnagyobb jelölt méretére, illetve az összes jelölt számára vonatkozó felső korlátokon is. Legyen $\mu_\ell^*(GY_\ell) = \ell + \min\{p|KK_{\ell+p}^*(GY_\ell) = 0\} - 1$ és

$$KK_{\text{összes}}^*(GY_\ell) = \sum_{p=1}^{\mu_\ell^*(GY_\ell)} KK_{\ell+p}^*(GY_\ell).$$

Következmény – $maxsize(GY_\ell) \leq \mu_\ell^*(GY_\ell) \leq \mu_\ell(|GY_\ell|)$.

Következmény – Az összes lehetséges ℓ-nél nagyobb méretű jelölt száma legfeljebb $KK_{\text{összes}}^*(GY_\ell)$ lehet, és $KK_{\text{összes}}^*(GY_\ell) \leq KK_\ell^{\text{összes}}(|GY_\ell|)$.

A KK^* érték függ a rendezéstől. Például a $KK_{2,1}^*(\{AB, AC\})$ értéke 1, amennyiben a rendezés szerinti legkisebb elem A, és 0 bármely más esetben. Elméletileg meghatározhatjuk az összes rendezés szerinti felső korlátot, és kiválaszthatjuk azt, amelyik a legkisebb értéket adja. Ez a megoldás azonban túl sok időbe telne. A szófa által használt rendezés szerinti felső korlátot viszonylag könnyen meghatározhatjuk. Ehhez azt kell látnunk, hogy a gyökér i címkéjű éléhez tartozó részfa levelei reprezentálják a GY_ℓ^i elemeit. A szófa egyetlen bejárásával egy egyszerű rekurzív módszer segítségével minden csúcshoz kiszámíthatjuk a $KK_{\ell-d,p}^*(GY_{\ell-d}^I)$ és $KK_{\ell-d}^{\ell-d+p}(|GY_{\ell-d}^I|)$ értékeket, ahol d a csúcs mélységét jelöli, $GY_{\ell-d}^I$ pedig az adott csúcshoz tartozó részfa által reprezentált elemhalmazokat. A gyökérhez kiszámított két érték adja meg a KK és KK^* korlátokat.

Ha a maradék futási idő becslésére kívánjuk használni a fenti felső korlátot, akkor tudnunk kell, hogy a jelöltek támogatottságának meghatározása függ az Apriori algoritmusban felhasznált adatstruktúrától. Szófa esetében például egy jelölt előfordulásának meghatározásához el kell jutnunk a jelöltet reprezentáló levélhez, ami a jelölt méretével arányos lépésszámú műveletet igényel. A maradék futási idő pontosabb felső becsléséhez a $KK_{\ell+p}^*(GY_\ell)$ értékeket súlyozni kell $(\ell + p)$-vel.

5.1.3. Az Eclat algoritmus

Az Eclat az üres mintából indulva egy rekurzív, mélységi jellegű bejárást valósít meg. A rekurzió mélysége legfeljebb eggyel több, mint a legnagyobb gyakori elemhalmaz mérete. Az Apriori-val szemben mindig egyetlen jelöltet állít elő, majd ennek azonnal meghatározza a támogatottságát. Az $(\ell+1)$-elemű, P prefixű jelölteket, ahol $|P| = \ell - 1$ az ℓ-elemű, P prefixű gyakori elemhalmazokból állítja elő egyszerű páronkénti unióképzéssel.

Az algoritmus központi fogalma az ún. TID-halmaz. Egy elemhalmaz *TID-halmazának* (Transaction IDentifier) elemei azon bemeneti sorozatok azonosítói (sorszámai), amelyek tartalmazzák az adott elemhalmazt. Más szóval egy TID-halmaz a vertikális adatbázis egy megfelelő sora. Például az

$$\langle AD, AC, ABCD, B, AD, ABD, D \rangle$$

bemenet esetén az $\{A, C\}$ elemhalmaz TID-halmaza $\{1, 2\}$, amennyiben egy tranzakció azonosítója megegyezik a bemeneti sorozatban elfoglalt helyével, és a helyek számozását nullától kezdjük.

A TID-halmaz két fontos tulajdonsággal bír:

1. Az I elemhalmaz TID-halmazának mérete megadja az I támogatottságát.
2. Egy jelölt TID-halmazát megkaphatjuk a generátorainak TID-halmazaiból egy egyszerű metszetképzéssel.

Algoritmus Eclat

Require: $\mathcal{T}$: tranzakciók sorozata,
min_supp: támogatottsági küszöb,

támogatottság_meghatározás($\mathcal{T}, J_1$);
$GY_1 \leftarrow$ gyakoriak_kiválogatása(J_1, min_supp);
for $i \leftarrow 1$ to $|\mathcal{T}|$ **do**
  for all $j \in t_i \cap GY_1$ **do**
    $j.TID \leftarrow j.TID \cup \{i\}$
  end for
end for
return $GY_1 \cup$ Eclat-segéd$(\emptyset, GY_1, min_supp)$

Először meghatározzuk a gyakori elemeket, majd felépítjük a gyakori elemek TID-halmazait. A későbbiekben nem használjuk a bemenetet, csak a TID-halmazokat. Az algoritmus lényege a Eclat-segéd rekurziós eljárás. Jelöljük a P prefixű, P-nél eggyel nagyobb méretű gyakori elemhalmazokból alkotott halmazcsaládot GY^P-vel. Nyilvánvaló, hogy $GY^{\emptyset} = GY_1$.

Algoritmus ECLAT-SEGÉD

Require: P: prefix elemhalmaz,
GY^P: P prefixű, P-nél eggyel nagyobb méretű gyakori elemhalmazokból alkotott halmazcsalád,
min_supp: támogatottsági küszöb.

for all $gy \in GY^P$ **do**
  for all $gy' \in GY^P, gy \prec gy'$ **do**
    $j \leftarrow gy \cup gy'$
    $j.TID \leftarrow gy.TID \cap gy'.TID$
    if $|j.TID| \geq min\_supp$ **then**
      $GY^{gy} \leftarrow GY^{gy} \cup \{j\}$
    end if
  end for
  if $|GY^{gy}| \geq 2$ **then**
    $GY \leftarrow GY \cup GY^{gy} \cup$ ECLAT-SEGÉD(gy, GY^{gy}, min_supp)
  else
    $GY \leftarrow GY \cup GY^{gy}$
  end if
end for
return GY

Az ECLAT jelölt-előállítása megegyezik az APRIORI jelölt-előállításával, azzal a különbséggel, hogy nem ellenőrizzük az unióképzéssel kapott halmaznak minden részhalmazára, hogy gyakori-e (a mélységi bejárás miatt ez az információ nem is áll rendelkezésünkre). Látható, hogy az ECLAT abban is különbözik az APRIORI-tól, hogy egy jelölt előállítása után azonnal meghatározza a támogatottságát, mielőtt újabb jelöltet állítana elő. Nézzünk egy példát a keresési tér bejárására.

Példa – Legyen $\mathcal{T} = \langle ACDE, ACG, AFGM, DM \rangle$ és $min\_supp = 2$. Első lépésben meghatározzuk a gyakori elemeket: A, C, D, G, M, ami nem más, mint $GY^{\emptyset}$. Ezután előállítjuk és azonnal meg is határozzuk az (A, C), (A, D), (A, G), (A, M) párok unióját. Ezek közül csak az AC, AG halmazok gyakoriak. A következő rekurziós lépésben ennek a két halmaznak vesszük az unióját, állítjuk elő a TID-halmazát, amely alapján kiderül, hogy az ACG ritka, és a rekurzió ezen ága véget ér. Ezután a C elemnek vesszük az unióját a sorban utána következő elemekkel egyesével és így tovább.

Látnunk kell, hogy az ECLAT legalább annyi jelöltet állít elő, mint az APRIORI. A mélységi bejárás miatt ugyanis egy jelölt előállításánál nem áll rendelkezésünkre az összes részhalmaz. Az előző példa esetében például az {A,C,G} támogatottságát hamarabb vizsgálja, mint a {C,G} halmazét, holott ez utóbbi

akár ritka is lehet. Ebben a tekintetben tehát az ECLAT rosszabb az APRIORI-nál, ugyanis több lesz a ritka jelölt.

Az ECLAT igazi ereje a jelöltek támogatottságának meghatározásában van. A jelöltek TID-halmazainak előállítása egy rendkívül egyszerű és nagyon gyors művelet lesz. Emellett ahogy haladunk egyre mélyebbre a mélységi bejárás során, úgy csökken a TID-halmazok mérete, és ezzel a támogatottság meghatározásának ideje is. Ezzel szemben az APRIORI-nál ahogy haladunk az egyre nagyobb méretű jelöltek felé, úgy nő a szófa mélysége, és lesz egyre lassabb minden egyes jelölt támogatottságának meghatározása (persze a zsákutca nyesés segít ezen egy kicsit).

A keresési tér bejárása függ a prefix definíciójától, amit az elemeken definiált rendezés határoz meg. Melyek lesznek azok a jelöltek, amelyek az APRIORI-ban nem lennének jelöltek (tehát biztosan ritkák), illetve várhatóan melyik az a rendezés, amely a legkevesebb ilyen tulajdonságú halmazt adja? Ha egy elemhalmaz jelölt az ECLAT algoritmusban, de az APRIORI-ban nem, akkor van olyan részhalmaza, amely ritka. Amennyiben feltételezzük, hogy az elemek függetlenek, akkor azon részhalmaz előfordulásának lesz legkisebb a valószínűsége (és ezzel együtt az esélye annak, hogy ritka), amely a leggyakoribb elemet nem tartalmazza. A jelölt prefixe generátor, tehát gyakori, így akkor lesz a legnagyobb esélye annak, hogy minden részhalmaz gyakori, ha a prefix a leggyakoribb elemet nem tartalmazza. Az ECLAT algoritmusnál a legkevesebb ritka jelöltet és így a legjobb futási időt tehát a gyakoriság szerint növekvő rendezéstől várhatjuk.

Példa – Ennek a gondolatmenetnek az illusztrálására nézzük a következő példát. Legyenek gyakori halmazok a következők: A, B, C, D, AB, AC, BC, AD, ABC, továbbá $supp(D) \prec supp(C) \prec supp(B) \prec supp(A)$. Amennyiben az ECLAT algoritmus a gyakoriság szerint csökkenő sorrendet használja, akkor az előállítás sorrendjében a következő halmazok lesznek jelöltek: A, B, C, D, AB, AC, AD, ABC, ABD, ACD, BC, BD, CD. Ugyanez gyakoriság szerint növekvő sorrendnél D, C, B, A, DC, DB, DA, CB, CA, CBA, BA. Az utóbbi esetben tehát négy ritka jelölt helyett (ABD, ACD, BD, CD) csak kettő lesz (CD, BD). Megjegyezzük, hogy ez a két elemhalmaz az APRIORI esetében is jelölt lesz. A gyakoriság szerint csökkenő esetben egyszer állítunk elő olyan háromelemű jelöltet, amelynek van olyan kételemű részhalmaza, amelyet nem vizsgáltunk. Ez a jelölt a CBA és a nem megvizsgált részhalmaz a BA. Mivel a részhalmaz éppen a leggyakoribb elemeket tárolja, ezért van nagy esélye annak, hogy gyakori (főleg ha hozzávesszük, hogy a jelölt két generátora, CB és CA is gyakori).

Javíthatunk az algoritmus hatékonyságán, ha nem a jelöltek TID-listáit tároljuk, hanem a jelölt és prefixe TID-listájának különbségét. A prefix tá-

mogatottságából és a TID listák különbségéből a támogatottság egyértelműen megadható. A különbségi listák akár nagyobbak is lehetnek az eredeti TID-listáknál (például, ha a I támogatottsága kicsi, de a prefixének támogatottsága nagy), így a legjobb megoldást a két technika ötvözése adhatja (például 4-nél kisebb elemszámnál TID lista, utána különbségi listák) [Zaki és Gouda, 2003]. A különbségi listát használó algoritmusok nagy fölénnyel verik a többi algoritmust, amennyiben a bemenet sűrű, és nagy méretű gyakori minták is vannak.

5.1.4. Az FP-GROWTH algoritmus

Az FP-GROWTH algoritmus[3][Han és tsa., 2000] egy mélységi jellegű, rekurzív gyakori elemhalmaz kereső algoritmus, a keresési tér bejárása tekintetében megegyezik az ECLAT-tal. A támogatottságok meghatározását az egyelemű gyakori halmazok meghatározásával, majd a bemenet *szűrésével* és *vetítésével* valósítja meg rekurzív módon. A bemenet szűrése azt jelenti, hogy az egyes tranzakciókból töröljük a bennük előforduló ritka elemeket. A $\mathcal{T}$ elemhalmaz P elemhalmazra vetítését (jelölésben $\mathcal{T}|P$) pedig úgy kapjuk, hogy vesszük a P-t tartalmazó tranzakciókat, majd töröljük belőlük a P-t. Például $\langle ACD, BCE, ABCE, BE, ABCE\rangle|B = \langle CE, ACE, E, ACE\rangle$. Az algoritmus pszeudokódja a következőkben olvasható.

Algoritmus FP-GROWTH

Require: $\mathcal{T}$: tranzakciók sorozata,
min_supp: támogatottsági küszöb,

FP-GROWTH-SEGÉD$(\mathcal{T}, min_supp, \emptyset)$

A segédeljárás harmadik paramétere (P) egy prefix elemhalmaz, az első paraméter pedig az eredeti bemenet P-re vetítése. Az eredeti bemenet $\emptyset$-ra vetítése megegyezik önmagával.

[3]Az FP a Frequent Pattern rövidítése, ami miatt az algoritmust *mintanövelő* algoritmusnak is hívják. Ez az elnevezés azonban félrevezető, ugyanis szinte az összes gyakori elemhalmaz kereső algoritmus mintanövelő abban az értelemben, hogy egy új jelölt a generátorainak egyelemű bővítése, vagy más szóval növelése. Az FP-GROWTH sajátsága nem a jelöltek előállítása, hanem a jelöltek támogatottság-meghatározásának módja.

Algoritmus FP-GROWTH-SEGÉD

Require: $\mathcal{T}$: vetített bemenet,
min_supp: támogatottsági küszöb,
P: prefix elemhalmaz,
támogatottság_meghatározás($\mathcal{T}, J_1$);
$GY_1 \leftarrow$ gyakoriak_kiválogatása(J_1, min_supp);
$T^* \leftarrow$ SZŰRÉS(T, GY_1)
for all $gy \in GY_1$ **do**
  $T^*|gy \leftarrow$ VETÍTÉS(T^*, gy)
  $GY \leftarrow GY \cup \{P \cup \{gy\}\}$ FP-GROWTH-SEGÉD$(T^*|gy, min_supp, P \cup \{gy\})$
  $T^* \leftarrow$ TÖRLÉS(T^*, gy)
end for
return GY

Egy rekurziós lépés három fő lépésből áll. Először meghatározzuk azon elemek támogatottságát, amelyek előfordulnak valamelyik tranzakcióban. Ezekből kiválasztjuk a gyakoriakat. Ezután minden gy gyakori elemet egyesével veszünk. Meghatározzuk a gy-hez tartozó vetített bemenetet, majd meghívjuk az algoritmust rekurzívan a $\mathcal{T}|gy$ bemenetre. Törölnünk kell a gy elemet a $\mathcal{T}^*$-beli tranzakciók elemei közül annak érdekében, hogy egy jelöltet csak egyszer állítsunk elő.

A jelöltek előállításának tekintetében az FP-GROWTH algoritmus a legegyszerűbb. Ha az I elemhalmaz gyakori, akkor a következő rekurziós szinten azon $I \cup j$ halmazok lesznek a jelöltek, ahol j az I-re vetített bemenetben előforduló elem és $I \cup j$ nem volt jelölt korábban. Tulajdonképpen az FP-GROWTH a nagy elemszámú jelöltek támogatottságának meghatározását visszavezeti három egyszerű műveletre: egyelemű gyakori elemhalmazok kiválogatása, szűrés és vetített bemenet előállítása.

A szűrés után egyesével vesszük a gyakori elemeket. Ezt valamilyen rendezés szerint kell tennünk és ez a rendezés határozza meg, hogy milyen sorban járjuk be a keresési teret, milyen vetített bemeneteket állítunk elő és mely elemhalmazok lesznek a hamis jelöltek. Az ECLAT-nál elmondottak itt is élnek; várhatóan abban az esetben lesz a hamis jelöltek száma minimális, amennyiben a prefixben a legritkább elemek vannak, azaz a 9. sorban gyakoriság szerint növekvő sorban vesszük az elemeket.

Az FP-GROWTH algoritmus szerves része az *FP-fa,* amelyben a szűrt bemenetet tároljuk. Az FP-fa segítségével könnyen előállíthatjuk a vetített bemeneteket, azokban könnyen meghatározhatjuk az elemek támogatottságát, amiből előállíthatjuk a vetített, majd szűrt bemenetet. Ezt a vetített és szűrt bemenetet szintén egy FP-fában tároljuk, amelyet *vetített FP-fának* hívunk.

Az FP-fa egy keresztélekkel és egy fejléc táblával kibővített szófa. Az élek

címkéi gyakori elemek. Az egyszerűbb leírás kedvéért egy (nemgyökér) cs csúcs címkéjén a gyökértől a cs-ig vezető út utolsó (cs-hez legközelebbi) élének címkéjét értjük. Minden csúcs egy elemhalmazt reprezentál, amelynek elemei a gyökérből a csúcsig vezető út csúcsainak címkéivel egyeznek meg. Minden csúcshoz egy számlálót rendelünk. Ez a számláló adja meg, hogy a csúcs által reprezentált halmaz mennyi bemeneti (vagy vetített) elemhalmaznak a prefixe. Az azonos címkéjű csúcsok láncolt listaszerűen össze vannak kötve keresztirányú élekkel. A lánc legelső elemére mutat a fejléctáblának az adott eleméhez tartozó mutatója.

Példa – Tegyük fel, hogy bemenetként a

$$\langle ACDFMQ, ABCFMO, BFO, BCKSQ, ACFMQ, CS, DFJ, FHI\rangle$$

sorozat van adva, és $min\_supp = 3$. A gyakori elemek: A, B, C, F, M, Q, amelyek támogatottsága rendre 3, 3, 5, 6, 3, 3. Ekkor a szűrt bemenetet ($\langle ACFMQ$, $ABCFM$, BF, BCQ, $ACFMQ$, C, F, $F\rangle$) reprezentáló FP-fa, amely gyakoriság szerint csökkenő sorrendet ($F \succ C \succ A \succ B \succ M \succ Q$) használ, a 5.9. ábrán látható

Egy FP-fát hasonló módon építünk fel, mint egy szófát. Különbség, hogy egy I elemhalmaz beszúrásánál nem csak az I-t reprezentáló levélnek a számlálóját növeljük eggyel, hanem minden olyan csúcsét, amelyet érintünk a beszúrás során (hiszen ezen csúcsokat reprezentáló halmazok az I prefixei). A keresztirányú éleket és a fejléctáblát is egyszerűen megkaphatjuk. Legyen a fejléctábla mutatóinak kezdeti értéke NIL. Amikor beszúrunk egy új, i címkéjű csúcsot, akkor két dolgot kell tennünk. Az új csúcs keresztél mutatója felveszi a fejléctábla i-hez tartozó bejegyzését, majd ezt a bejegyzést az új csúcs címére cseréljük. Ezzel tulajdonképpen olyan láncot készítünk, amelyben a csúcsok a beszúrási idejük szerint csökkenően vannak rendezve (az először beszúrt elem van leghátul) és a lista a fejléctáblában kezdődik.

A fejléc mutatókból kiindulva és a keresztéleket követve megkaphatjuk a vetített bemenetet és meghatározhatjuk a vetített bemenetben gyakori elemeket. Az adott tranzakciók előfordulása megegyezik a keresztélek által mutatott pontok számlálójával. Ezek alapján a vetített bemenetet szűrhetjük és belőle egy újabb FP-fát építhetünk fel. Ezt a fát vetített FP-fának hívjuk. A következő ábrán az M elemhez tartozó vetített és szűrt bemenet FP-fáját láthatjuk (amelyet a Q elem feldolgozása után kapunk).

Az FP-fa mérete – hasonlóan a szófa méretéhez – függ az elemeken definiált rendezéstől. Az FP-GROWTH algoritmus akkor lesz hatékony, ha a fa elfér a memóriában, ezért fontos lenne azt a rendezést használni, ami várhatóan a legkisebb fát eredményezi. Az APRIORI esetében már elmondtuk, hogy az a

heurisztika, amely az elemek gyakoriság szerint csökkenő rendezését használja, általában kis méretű fát eredményez.

Egyszerű lesz a vetített bemenet előállítása és a szűrt bemenetből egy elem törlése, amennyiben a legritkább gyakori elemet (gy_r) vesszük először. Ez összhangban áll azzal, hogy a pszeudokód 9. sorában az elemeket gyakoriság szerint növekvő sorrendben vesszük. A gy_r csak levél címkéje lehet. Mivel a fából törölni fogjuk a gy_r címkéjű csúcsokat a rekurziós művelet után (13. sor), a következő elem is csak levél címkéje lesz.

Nézzük most meg, hogy amennyiben a szűrt bemenet egy FP-fában van tárolva, akkor hogyan kaphatjuk meg a gy_r elemre vett vetítésben az elemek támogatottságát. A fejléctábla gy_r eleméhez tartozó mutatóból kiindulva a keresztélek alkotta láncban pontosan azok a csúcsok vannak, amelyek gy_r-t tartalmazó bemeneti elemet reprezentálnak. Az egyes elemhalmazok előfordulását a gy_r címkéjű csúcsokhoz rendelt számláló adja meg, az elemeiket pedig a gyökérig felsétálva kaphatjuk. A lista utolsó csúcsának feldolgozása után rendelkezésünkre állnak a gy_r elemhez tartozó vetített bemenetben valahol előforduló elemek támogatottságai, amely alapján kiválogathatjuk a vetített bemenetben gyakori elemeket.

Ugyanilyen bejárással kaphatjuk meg a vetített, majd szűrt bemenetet tartalmazó FP-fát. A fejléctáblából kiindulva végigmegyünk a láncolt lista elemein. A csúcs által reprezentált elemhalmazból töröljük a ritka elemeket, majd a kapott elemhalmazt beszúrjuk az új FP-fába. A kis memóriaigény érdekében a gyakoriság szerint csökkenő sorrendet használjuk. Ezt a sorrendet a vetített bemenet alapján állítjuk fel (lévén az új fa a vetített és szűrt bemenetet fogja tárolni), ami különbözhet az eredeti FP-fában alkalmazott rendezéstől.

Példa – Folytassuk az előző példát és állítsuk elő a legritkább gyakori elemhez (Q) tartozó vetített és szűrt bemenetet. A fejléctábla Q eleméhez tartozó mutatóból kiindulva mindössze két csúcsot látogatunk meg, ami azt jelenti, hogy a vetített bemenet két különböző elemhalmazt tartalmaz: az $FCAM$-et kétszer, a CB-t egyszer. Ez alapján a vetített bemenetben egyetlen gyakori elem van, C. Ez a rekurziós ág nem folytatódik, hanem visszatér a QC gyakori elemhalmazzal. Az FP-fából törölhetjük a fejléctábla Q bejegyzéséhez tartozó mutatóból, keresztirányú élek segítségével elérhető csúcsokat. A következő vizsgált elem az M. Az M vetített bemenetében három gyakori elem van, és a vetített szűrt bemenet az FCA elemhalmazt tartalmazza háromszor. Ezt a vetített, szűrt bemenetet egy egyetlen útból álló FP-fa fogja reprezentálni. A többi FP-fa ugyanilyen egyszerűen megkapható.

Hatékonysági szempontból rendkívül fontos, hogy a rekurziót ne folytassuk, ha a vizsgált FP-fa egyetlen útból áll. A rekurzió helyett képezzük inkább az út által reprezentált elemhalmaz minden részhalmazát. A részhalmaz támoga-

tottságát annak a csúcsnak a számlálója adja meg, amely a legmélyebben van a részhalmazt meghatározó csúcsok között.

Az FP-growth* algoritmus

2003 novemberében megszervezték az első gyakori elemhalmaz-kinyerő algoritmusok versenyét [Goethals és Zaki, 2003]. Bárki benevezhetett egy általa készített programot. Ezeket központilag tesztelték különböző adatbázisokon, különböző támogatottsági küszöbökkel. Nem volt olyan implementáció, amely minden esetben a legjobban szerepelt, de ki lehet emelni néhány olyat, amelyek szinte mindig az elsők között végeztek. A fődíjat (egy sört és egy pelenkát!) a szervezők végül az FP-growth* algoritmus készítőinek adták [Grahne és Zhu, 2003].

Az FP-growth* algoritmus az FP-growth módosítása. Előnye, hogy gyorsabban állítja elő a vetített fát, amiért viszont memóriával fizet. Nézzük meg, hogy pontosan mi történik egy rekurziós lépésben:

Először ellenőrizzük, hogy a fa egyetlen útból áll-e. Ha nem, akkor a legritkább elemből kiindulva előállítjuk a vetített fákat, és rekurzívan meghívjuk az algoritmust. A vetített fában első lépésként meg kell határozni a vetített bemenetben szereplő elemek támogatottságát, második lépésként pedig előállítjuk a vetített FP-fát. Ez tulajdonképpen az aktuális fa adott elemhez tartozó ágainak kétszeri bejárását jelenti. Az első bejárást lehet meggyorsítani egy segédtömb használatával.

Az FP-fa építésénél töltsünk fel egy, kezdetben 0 értékeket tartalmazó tömböt is. Amikor beszúrunk egy t (akár vetített) tranzakciót az (akár vetített) FP-fába, növeljük eggyel a tömb (i, j)-edik celláját, amennyiben az i és j elemei t-nek. A fa felépítése után rendelkezésünkre áll egy tömb, ami tartalmazza az elempárok előfordulását. Ha ezek után egy vetített fát akarunk készíteni, akkor szükségtelen időt töltenünk az első lépéssel, hiszen a tömb megfelelő sorából közvetlen megkaphatjuk a támogatottságokat. Összességében az első lépés gyorsabb (nem kell a fában bolyonganunk, csak a tömb elemeit kiolvasni), a második lassabb (a tömböt is fel kell tölteni), a memóriafogyasztás pedig nagyobb (a tömb méretével).

5.1.5. További technikák

1993 óta több száz cikk jelent meg gyakori elemhalmazokat kinyerő algoritmusok témájában. Legtöbb cikk egy új gyorsítási trükköt vagy egy új módszert mutatott be. A korábban már említett, 2003-ban és 2004-ben megrendezett gyakori elemhalmazokat kinyerő algoritmusok versenyén sebesség tekintetében az Eclat és az FP-growth különböző módosításai bizonyul-

tak a legjobbnak, a memóriahasználat terén pedig az APRIORI volt kiemelkedő. További technikák közül az érdeklődő olvasók figyelmébe ajánljuk a SETM [Houtsma és Swami, 1993], APRIORI-TID [Agrawal és Srikant, 1994], APRIORI-HYBRID [Agrawal és Srikant, 1994], DHP [Park és tsa., 1995], DIC, Patrícia, Tree projection és DF-apriori [Pijls és Bioch, 1999] algoritmusokat.

5.1.6. Mintavételező algoritmus elemzése

Az egyszerű mintavételező algoritmust bemutatunk az 5.3.5 részben. Itt azt vizsgáljuk, hogy mekkora mintát célszerű venni annak érdekében, hogy az algoritmus minden gyakori elemhalmazt megtaláljon.

Mintavétel nagysága

Mintavételezésen alapuló eljárásoknál a minta mérete központi kérdés. Ha a minta túl kicsi, akkor a mintából nyert információ távol állhat a teljes adatbázisban található globális „helyzettől". Mivel fölöslegesen nagy minta lassú algoritmusokat eredményez, ezért fontos egy kicsi, de már pontos képet adó mintaméret meghatározása. A 3.3.6 részben megadtuk, hogy mekkora mintát kell választani, ha azt akarjuk, hogy a relatív gyakoriságok megegyezzenek az előfordulások valószínűségével. Használjuk most is a A 3.3.6 részben bevezetett elnevezéseket és jelöléseket.

Nézzük, hogy mennyivel kell csökkenteni a gyakorisági küszöböt (min_freq') ahhoz, hogy kicsi legyen annak valószínűsége, hogy tetszőleges gyakori elem mintához tartozó gyakorisága kisebb a csökkentett küszöbnél, tehát:

$$\mathbb{P}(\text{gyakoriság}(x,m) < min\_freq') = \mathbb{P}\Big(\frac{Y}{m} < min\_freq'\Big)$$

egy adott küszöbnél (δ') kisebb kell legyen és tudjuk, hogy

$$p > min\_freq$$

A fenti egyenletre alkalmazva a Hoeffding-korlátot azt kapjuk, hogy

$$\mathbb{P}\big(\frac{Y}{m} < min\_freq'\big) =$$

$$\mathbb{P}\big(\frac{Y}{m} - p < min\_freq' - p\big) <$$

$$\mathbb{P}\big(\frac{Y}{m} - p < min\_freq' - min\_freq\big)$$

$$\leq e^{-2(\text{min\_freq'-min\_freq})^2 m}$$

tehát ahhoz, hogy a hibázás valószínűsége kisebb legyen δ'-nél teljesülnie kell, hogy

$$min\_freq' < min\_freq - \sqrt{\frac{1}{2m} \ln \frac{1}{\delta'}}$$

A 5.2 táblázat azt mutatja, hogy rögzített hibakorlát mellett ($\delta' = 0.001$) adott mintaérethez mennyi legyen a csökkentett küszöb.

	Minta mérete			
min_freq (%)	20000	40000	60000	80000
0.25	0.13	0.17	0.18	0.19
0.50	0.34	0.38	0.40	0.41
0.75	0.55	0.61	0.63	0.65
1.00	0.77	0.83	0.86	0.88
1.50	1.22	1.30	1.33	1.35
2.00	1.67	1.77	1.81	1.84

5.2. táblázat. A küszöb csökkentése adott mintaméretekre rögzített $\delta = 0.001$ mellett

5.1.7. Elemhalmazok Galois lezártja

Egy minta zárt, ha nincs vele egyező támogatottságú bővebb minta. Esetünkben ez azt jelenti, hogy ha egy elemhalmaz nem zárt, akkor pontosan azokban a bemeneti elemekben fordul elő, amelyekben a lezártja. Ha például az A elem lezártja az AB halmaz, akkor tudjuk, hogy az A halmaz soha nem fordul elő a bemeneti elemekben a B elem nélkül.

Ebben a részben a lezárt további tulajdonságait fogjuk megismerni. Azért illetjük a lezártat a Galois jelzővel, mert teljesülni fog a lezárás operátorra a Galois elméletből jól ismert 3 tulajdonság. Mielőtt erre rátérünk nézzük meg, hogy az elemhalmazokat tartalmazó mintakörnyezet egyértelmű-e a zártságra nézve.

5.1.6. Lemma *Az elemhalmazokat tartalmazó mintakörnyezet a zártságra nézve egyértelmű.*

Bizonyítás – Indirekt tegyük fel, hogy az I elemhalmaznak létezik két lezártja, azaz létezik I', I'' különböző elemhalmazok, amelyekre a minimalitás mellett teljesülnek a $I \subset I', I \subset I'', |I'| = |I''|, supp(I') = supp(I'')$ feltételek. Ez azt jelenti, ahogy azon tranzakciók, amelyek I-t tartalmazzák, tartalmazzák az $I' \setminus I$ és az $I'' \setminus I$ halmazokat is. De ebből következik, hogy ezek a tranzakciók

$I' \cup I''$ is tartalmazzák, azaz $I' \cup I''$ is lezártja I-nek, így sem I' sem I'' nem lehet minimális.

A fentiek miatt a gyakori zárt elemhalmazokból és azok támogatottságaiból egyértelműen meg tudjuk határozni a gyakori elemhalmazokat és azok támogatottságát. A gyakori zárt minták tehát a zárt minták egy veszteségmentes tömörítése, érdemes csak ezeket meghatározni és eltárolni [Pasquier és tsa., 1999a, Pasquier és tsa., 1999b, Pasquier és tsa., 1998, Zaki és Ogihara, 1998].

A zárt elemhalmazok fogalma

Az I elemhalmaz zárt, amennyiben nincs nála bővebb halmaz, amelynek támogatottsága megegyezik I támogatottságával. Jelöljük *cover*-rel azt a függvényt, amely egy elemhalmazhoz az azt tartalmazó tranzakciók halmazát adja meg.

A zárt elemhalmazokra adhatunk egy másik definíciót is. Vezessük, be a *cover′* függvényt:

5.1.7. Definíció *Legyen $\mathcal{T} = \langle t_1, \ldots, t_n \rangle$ tranzakciók sorozata, amelynek minden eleme az $\mathcal{I}$-nek egy részhalmaza. Definiáljuk a $cover' : 2^{\mathbb{N}} \to 2^{\mathcal{I}}$ függvényt a következőképpen*

$$cover'(T) = \{i \in \mathcal{I} | \forall j \in T, i \in cover(t_j)\} = \bigcap_{t \in T} cover(t)$$

Tehát $cover'(T)$ megadja azon közös elemeket, amelyeket minden olyan tranzakció tartalmaz, amelynek sorszáma T-beli.

A $(cover, cover')$ függvénypárt az $\mathcal{T}$ és $\mathcal{I}$ hatványhalmazai közötti Galois-kapcsolatnak hívjuk. Legyen a példaadatbázisunk a következő:

$$\langle ACD, BCE, ABCE, BE, ABCE \rangle.$$

Ekkor: $cover(\{A, C\}) = \{1, 3, 5\}, cover(\emptyset) = \{1, 2, 3, 4, 5\}$, $cover'(\{1, 2, 3\}) = \{C\}, cover'(\{1, 4\}) = \emptyset$.

Az alábbi tulajdonságok igazak tetszőleges $t, t_1, t_2 \subseteq \mathcal{T}$ és $I, I_1, I_2 \subseteq \mathcal{I}$ halmazokra:

(1) $I_1 \subseteq I_2 \Rightarrow cover(I_1) \supseteq cover(I_2)$ (1′) $T_1 \subseteq T_2 \Rightarrow cover'(T_1) \supseteq cover'(T_2)$
(2) $T \subseteq cover(I) \Longleftrightarrow I \subseteq cover'(T)$

5.1.8. Definíció *A $h = cover' \circ cover$ (vagy $h' = cover \circ cover'$) operátort Galois-lezárás operátornak hívjuk.*

Belátható, hogy tetszőleges halmaznak a lezártja tartalmazza magát a halmazt, továbbá a Galois-lezárás operátora idempotens és monoton, tehát

$(I)\ I \subseteq h(I)$	$(I')\ T \subseteq h'(T)$
$(II)\ h(h(I)) = h(I)$	$(II')\ h'(h'(T)) = h'(T)$
$(II)\ I_1 \subseteq I_2 \Rightarrow h(I_1) \subseteq h(I_2)$	$(III')\ T_1 \subseteq T_2 \Rightarrow h'(T_1) \subseteq h'(T_2)$

5.1.9. Definíció (zárt elemhalmaz) *I elemhalmaz zárt, amennyiben $I = h(I)$.*

Tetszőleges elemhalmazt (I) tartalmazó minimális elemszámú zárt elemhalmazt a lezárás operátor alkalmazásával kaphatunk meg; ez éppen $h(I)$ lesz. A példaadatbázisban található zárt elemhalmazok alábbiak:

zárt elemhalmazok
{∅}, {C}, {B,E}, {B,C,E}, {A,C}, {A,B,C,E}, {A,C,D}, {A,B,C,D,E}

Adósok vagyunk még annak bizonyításával, hogy a két definíció ekvivalens, azaz, ha $h(C) = C$, akkor C-nél nincs bővebb halmaz, amely támogatottsága megegyezne C támogatottságával, illetve fordítva. A két állítás közvetlen adódik a következő tételből.

5.1.10. Tétel *Minden elem támogatottsága megegyezik lezártjának támogatottságával, tehát*

$$supp(I) = supp(h(I))$$

Bizonyítás – A lezárás (1) tulajdonsága miatt $supp(I) \geq supp(h(I))$. Ugyanakkor

$$supp(h(I)) = |cover(h(I))| = |cover(cover'(cover(I)))| =$$
$$= |h'(cover(I))| \leq supp(I)$$

a (III') miatt, amiből következik az egyenlőség.

Az 5.3.4 részben bemutatjuk, hogy a gyakori mintákból hogyan választhatjuk ki a zártakat, illetve az APRIOR-CLOSE algoritmust, ami már eleve csak a gyakori zárt mintákat állítja elő. Az APRIOR-CLOSE algoritmusnál léteznek gyorsabb algoritmusok, lásd pl. CHARM [Zaki és Hsiao, 2002], CLOSET [Pei és tsa., 2000], CLOSET+ [Wang és tsa., 2003], MAFIA [Burdick és tsa., 2001]. Ezek ismertetésétől eltekintünk.

5.1.8. Kényszerek kezelése

Ebben a részben azt a speciális feladatot nézzük meg, hogy miként lehet csökkenteni a bemenetet, ha az anti-monoton kényszerek mellett monoton kényszereket is megadunk. Már az általános mintakeresésnél megtárgyaltuk, hogy tetszőleges anti-monoton kényszer könnyűszerrel beépíthető az APRIORI algoritmusba. Most azt nézzük meg, hogy a monoton kényszerek hogyan alkalmazhatók a bemeneti tér csökkentésére.

Adott egy bemeneti sorozat, minimális támogatottsági küszöb és monoton kényszerek $\mathcal{C}$ halmaza. Feladat a bemenet csökkentése oly módon, hogy bármely teljes algoritmus a csökkentett bemeneten is teljes legyen.

ExAnte

Az ExAnte [Bonchi és tsa., 2003] algoritmus kétféle lépést ismétel egészen addig, amíg ez valamilyen változást jelent. Az első lépés azon tranzakciók törlése, amelyek nem adnak igaz értéket minden $\mathcal{C}$-beli kényszeren. Az ilyen tranzakciók csak olyan minták támogatottságát növelik, amelyek úgysem elégítik ki a kényszereket (ez következik a kényszerek monoton tulajdonságából). A második lépésben a bemenet elemei közül töröljük a ritkákat, hiszen azok úgysem játszanak szerepet a támogatottság meghatározásánál.

Látnunk kell, hogy az első lépésbeli törlés új ritka elemekhez vezethet, ami csökkenti bizonyos tranzakciók méretét, ami viszont ahhoz vezethet, hogy ezek újabb kényszereket fognak sérteni. Jogos tehát, hogy a két módszert felváltva futtassuk addig, amíg van valami változás. Az algoritmus a bemenet csökkentése mellett előállítja azon gyakori elemeket, amelyekre minden kényszer teljesül. Gyakori elemhalmaz csak ezekből az elemekből épülhetnek fel.

Példa – Az adatbázisban 8 elem és 9 tranzakció van. Legyen $min\_supp = 4$. Minden elemnek van egy ára. Az egyetlen kényszer (`sum(i.ár) > 44`) szerint a halmazban található termékek árának összege 44-nél nagyobb legyen. A következő két táblázat adja meg az adatokat.

Az első végigolvasás során meghatározzuk az elemek támogatottságát azon tranzakciókban, amelyek kielégítik a kényszert (a 4-es kivételével mindegyik). Ezután töröljük a ritka elemeket (A, E, F, H). Ismét végigmegyünk az adatbázison, de most már ezeket az elemeket nem nézzük, aminek következtében újabb tranzakciók esnek ki (2,7,9). A kiesett tranzakciók miatt csökkennek a támogatottságok, így újabb elem lesz ritka (G). Ezt így folytatjuk, amíg van változás. A 4. végigolvasás után azt kapjuk, hogy csak az 1,3,6,8 tranzakciókat és a B, C, D elemeket kell figyelembe venni.

termék	ár
A	5
B	8
C	14
D	30
E	20
F	15
G	6
H	12

TID	tranzakció	ár összeg
1	B, C, D, G	58
2	A, B, D, E	63
3	B, C, D, G, H	70
4	A, E, G	31
5	C, D, F, G	65
6	A, B, C, D, E	77
7	A, B, D, F, G, H	76
8	B, C, D	52
9	B, E, F, G	49

5.3. táblázat. Tranzakciós adatbázis a termékek áraival.

5.1.9. Többszörös támogatottsági küszöb

Az univerzális támogatottsági küszöbnek vannak előnyei és hátrányai. Előnye, hogy felhasználhatjuk azt a tényt, hogy gyakori minta minden részmintája gyakori, ami alapján hatékony algoritmusokat adhatunk. Hátránya, hogy a ritkán előforduló, de mégis fontos mintákat csak akkor tudjuk kinyerni, ha a támogatottsági küszöböt alacsonyra állítjuk. Ez viszont rengeteg gyakori mintához fog vezetni, ha egyáltalán le tud futni az algoritmus.

Különböző támogatottsági küszöbök (vagy másként támogatottsági küszöb függvényének) megadásával ez a probléma elkerülhető: a nem lényeges mintáknak legyen nagy a küszöbük, a lényegesebbeknek legyen alacsony.

Egyedi támogatottsági küszöbök bevezetésével azonban felborul eddigi kényelmes világunk, amelyet az biztosított, hogy nem lehet egy minta gyakori, ha van ritka részmintája. A részminták támogatottsági küszöbe ugyanis nagyobb lehet, így hiába nagyobb a támogatottsága, ettől még lehet ritka. A következőkben bemutatjuk a legelső és legegyszerűbb támogatottsági küszöb függvényt, majd bemutatjuk az MSApriori algoritmust, amely ezt hatékonyan kezeli.

MSApriori algoritmus

Kézzel megadni a $2^{\mathcal{I}}$ minden elemének támogatottsági küszöbét fáradságos, sőt nagy $|\mathcal{I}|$ esetén kivitelezhetetlen feladat. Az MSApriori algoritmusnál csak az egyelemű elemhalmazok támogatottsági küszöbét lehet megadni. Jelöljük az i elem küszöbét $MIS(i)$-vel. Az I elemhalmaz támogatottsági küszöbe legyen a legkisebb támogatottsági küszöbbel rendelkező elemének támogatottsági küszöbe ($MIS(I) = \min_{i \in I}\{MIS(i)\}$). Akkor gyakori az I halmaz, ha

támogatottsága nagyobb vagy egyenlő $MIS(I)$-nél.

A definícióból következik, hogy tényleg nem mondhatjuk, hogy gyakori minta minden részmintája gyakori. Például az ABC elemhalmaz BC részhalmazának nagyobb lehet MIS értéke. Ha a feladat megoldására az APRIORI algoritmust használjuk úgy, hogy csak a gyakori elemhalmazok kiválasztásának módját módosítjuk (min_supp cseréje $MIS(I)$-re), akkor nem garantált, hogy jó megoldást kapunk. Ha például a BC ritka, akkor az ABC halmaz nem lenne a jelöltek között annak ellenére, hogy akár gyakori is lehet.

Szerencsére a probléma könnyen orvosolható. Csak azt kell észrevennünk, hogy mi okozhatja a hibát. Az általánosság megsértése nélkül feltehetjük, hogy az elemek MIS értékük alapján növekvő sorba vannak rendezve. A MIS definíciójából következik, hogy tetszőleges ℓ-elemű $I = \{i_1, \ldots, i_\ell\}$ halmaz $\ell - 1$ darab $(\ell - 1)$-elemű részhalmazának MIS értéke megegyezik I MIS értékével, ami $MIS(i_1)$. Ezeknek a részhalmazoknak tehát gyakorinak kell lenniük, hiszen a támogatottság monotonsága most is fennáll. Az egyetlen részhalmaz, amely lehet ritka, az I legelső elemét nem tartalmazó részhalmaz. Ezt a részhalmazt tehát ne vizsgáljuk a jelölt előállítás második lépése során. Kivétel ez alól azon eset, amikor a második elem MIS értéke megegyezik az első elem MIS értékével, mert ekkor még ennek a részhalmaznak is gyakorinak kell lennie.

Amennyiben $\ell > 2$, akkor biztos, hogy a generátorok egyike sem egyezik meg a legkisebb elemet nem tartalmazó részhalmazzal ($\ell > 2$ esetében ugyanis a generátorok $(\ell - 2)$-elemű prefixei megegyeznek, amelyek biztos, hogy tartalmazzák a jelölt első elemét). Ez pedig garantálja, hogy az algoritmus teljes, amennyiben az összes gyakori elempárt megtaláltuk. Nézzük meg most az egy- és kételemű jelöltek esetét.

Gyakori elemek meghatározásánál a szokásos eljárást követjük: minden elem jelölt. Elempárok esetében azonban nem állíthatjuk, hogy egy pár akkor jelölt, ha mindkét eleme gyakori. Például az AB pár lehet gyakori akkor is, ha az A ritka. Ha ugyanis B-nek MIS értéke kisebb A-nak MIS értékénél, akkor az AB-nek a MIS értéke megegyezik B-nek a MIS értékével, így AB lehet gyakori. Szerencsére szükségtelen az összes elemet figyelembe venni. Ha például az A elem ritka és az A MIS értéke a legkisebb, akkor a támogatottság monotonságából következik, hogy az A-t tartalmazó halmazok ritkák. Ha tehát MIS érték szerint növekvően vannak rendezve az elemek, akkor a legkisebből kiindulva keressük meg az első gyakori elemet. Az összes utána következőt figyelembe kell venni a jelöltpárok előállításánál akkor is, ha valamelyik ritka.

5.2. Asszociációs szabályok

A gyakori elemhalmazokat felhasználhatjuk szabályok kinyerésére. Az $I_1 \to I_2$ asszociációs szabály azt állítja, hogy azon bemeneti elemek, amelyek tartalmazzák I_1-et, tartalmazzák általában I_2-t is. Például a pelenkát vásárlók sört is szoktak venni.

Mi a haszna ezeknek a szabályoknak? Például az, hogy egy szupermarket extra profithoz juthat az alábbi módon: Ha $I_1 \to I_2$ szabály igaz, akkor óriási hírverés közepette csökkentsük I_1 termékek árát (mondjuk 15%-kal). Emellett diszkréten emeljük meg I_2 termék árát (mondjuk 30%-kal) úgy, hogy az I_1 árcsökkentéséből származó profitcsökkenés kisebb legyen, mint az I_2 áremeléséből származó profitnövekedés. Az I_1 és I_2 termékek eladásai együtt mozognak, tehát az I_2 termék eladása is nőni fog, összességében a profitunk nőni fog, és a leárazás reklámnak is jó volt.

Az internetes üzletek is figyelembe vesznek ilyen szabályokat a stratégiájuk kialakítása során. Tudják, milyen terméket vásárolnak együtt. Sokszor az együtt vásárlást elő is írják azzal, hogy nem adják el önmagában az olcsó árucikket, csak akkor, ha megveszi az ügyfél a drága kiegészítőt is.

Az ilyen szabályokból nyert információt használhatják emellett áruházak terméktérképének kialakításához is. Cél a termékek olyan elrendezése, hogy a vevők elhaladjanak az őket érdekelhető termékek előtt. Gondoljuk meg, hogyan lehet kiaknázni e célból egy asszociációs szabályt.

Elemhalmazok sorozatát ábrázolhatjuk bináris értékeket tartalmazó táblával is. Ekkor az asszociációs szabályok attribútumok közötti összefüggést mutatnak: ha az I_1 attribútumok értékei 1-es, akkor nagy valószínűséggel az I_2 attribútumok értéke is az. A valószínűség értékét a szabály *bizonyossága* adja meg. Csak olyan szabályok lesznek érdekesek, amelyek bizonyossága magas. Például a házasságban élők 85%-ának van gyermekük.

Az asszociációs szabályok felhasználási területe egyre bővül. A piaci stratégia meghatározásán túl egyre fontosabb szerepet játszik a döntéstámogatás és pénzügyi előrejelzések területén is.

Nézzük most az asszociációs szabály pontos definícióját.

5.2.1. Az asszociációs szabály fogalma

Használjuk a 5.1.1 részben bevezetett definíciókat és jelöléseket (elemhalmaz, kosár, támogatottság, fedés, gyakori elemhalmaz stb.).

5.2.1. Definíció (asszociációs szabály) *Legyen $\mathcal{T}$ az $\mathcal{I}$ hatványhalmaza felett értelmezett sorozat. Az $R : I_1 \xrightarrow{c,s} I_2$ kifejezést c bizonyosságú, s támogatottságú* asszociációs szabálynak *nevezzük, ha I_1, I_2 diszjunkt elemhalmazok,*

és

$$c = \frac{supp_{\mathcal{T}}(I_1 \cup I_2)}{supp_{\mathcal{T}}(I_1)},$$
$$s = supp_{\mathcal{T}}(I_1 \cup I_2)$$

A szabály bal oldalát feltétel rész*nek, a jobb oldalát pedig* következmény rész*nek nevezzük.*

Az $R : I_1 \to I_2$ szabály bizonyosságára gyakran $conf(R)$-ként hivatkozunk.

Feladat egy adott kosársorozatban azon asszociációs szabályok megtalálása, amelyek gyakoriak (támogatottságuk legalább min_supp), és bizonyosságuk egy előre megadott korlát felett van. Jelöljük ezt a bizonyossági korlátot min_conf-fal. A feltételt kielégítő szabályokat *érvényes asszociációs szabályoknak* hívjuk, az 1 bizonyossággal rendelkezőket pedig *egzakt asszociációs szabálynak.*

5.2.2. Definíció (érvényes asszociációs szabály) *$\mathcal{T}$ kosarak sorozatában, min_supp támogatottsági és min_conf bizonyossági küszöb mellett az $I_1 \xrightarrow{c,s} I_2$ asszociációs szabály érvényes, amennyiben $I_1 \cup I_2$ gyakori elemhalmaz, és $c \geq min\_conf$*

A fenti feladatot két lépésben oldjuk meg. Először előállítjuk a gyakori elemhalmazokat, majd ezekből az érvényes asszociációs szabályokat. Az első lépésről szól az 5.1. fejezet, nézzük most a második lépést.

Minden I gyakori termékhalmazt bontsunk fel két diszjunkt nem üres részre ($I = I_1 \cup I_2$), majd ellenőrizzük, hogy teljesül-e a $\frac{\text{supp}(I)}{\text{supp}(I_1)} \geq min\_conf$ feltétel. Amennyiben igen, akkor a $I_1 \to I_2$ egy érvényes asszociációs szabály. A támogatottság anti-monoton tulajdonságát felhasználhatjuk annak érdekében, hogy ne végezzünk túl sok felesleges kettéosztást.

5.2.1 Észrevétel *Amennyiben I_1, I gyakori elemhalmazok a $\mathcal{T}$ bemeneti sorozatban, és $I_1 \subset I$, illetve $I_1 \to I \backslash I_1$ nem érvényes asszociációs szabály, akkor $I_1' \to I \backslash I_1'$ sem érvényes semmilyen $I_1' \subset I_1$-re.*

Bizonyítás – Az $I_1 \xrightarrow{c,s} I \backslash I_1$ nem érvényes szabály, tehát $c = \frac{\text{supp}(I_1 \cup (I \backslash I_1))}{\text{supp}(I_1)} = \frac{\text{supp}(I)}{\text{supp}(I_1)} < min\_conf$. Mivel a támogatottság anti-monoton, ezért $\text{supp}(I_1') \geq \text{supp}(I_1)$, amiből $\frac{1}{\text{supp}(I_1')} \leq \frac{1}{\text{supp}(I_1)}$, és ebből, ha c'-vel jelöljük az $I_1' \to I \backslash I_1'$ szabály bizonyosságát, akkor

$$c' = \frac{\text{supp}(I)}{\text{supp}(I_1')} \leq \frac{\text{supp}(I)}{\text{supp}(I_1)} < min\_conf$$

tehát $I_1' \to I \backslash I_1'$ sem érvényes asszociációs szabály.

Maximális következményű asszociációs szabály

A maximális méretű gyakori mintákból az összes gyakori mintát meghatározhatjuk. Ez abból következik, hogy gyakori minta minden részmintája gyakori. Asszociáció szabályoknál is vannak olyanok, amelyekből más szabályok levezethetők. Nézzünk két egyszerű levezetési szabályt. Tegyük fel, hogy $I_1 \to I_2$ érvényes asszociációs szabály, ekkor

- $I_1 \to I_2'$ is érvényes, minden $I_2' \subseteq I_2$-re.
- $I_1 \cup i \to I_2 \setminus \{i\}$ is érvényes minden $i \in I_2$-re. Ezek szerint a következményrészből tetszőleges elemet áttehetünk a feltételrészbe.

Mindkét állítás a támogatottság anti-monoton tulajdonságából közvetlenül adódik.

Ezek szerint minden asszociációs szabály levezethető a maximális következményrésszel rendelkező asszociációs szabályokból. Persze a levezethetőség nem a lejobb szó, ugyanis a szabályok paramétereire nem tudunk következtetni.

Egzakt asszociációs szabályok bázisa

A 100%-os bizonyossággal rendelkező asszociációs szabályokat *egzakt asszociációs szabályoknak* hívjuk. Az egzakt asszociációs szabályokra érvényes tranzitivitás is, tehát $I_1 \to I_2$ és $I_2 \to I_3$-ból következik, hogy $I_1 \to I_3$. Matematikus beállítottságú emberek agyában azonnal felmerül, hogy van-e az egzakt asszociációs szabályoknak egy minimális bázisa, amelyből minden egzakt asszociációs szabály levezethető. Ehhez a bázishoz a *pszeudó-zárt elemhalmazokon* keresztül jutunk.

5.2.3. Definíció *Az I elemhalmaz $\mathcal{T}$-re nézve* zárt*, amennyiben nem létezik olyan I' minta, amelynek I valódi részhalmaza, és I' támogatottsága megegyezik I támogatottságával ($supp(I') = supp(I)$).*

5.2.4. Definíció *$I \subseteq \mathcal{I}$ pszeudo-zárt elemhalmaz, ha nem zárt, és minden pszeudo-zárt $I' \subset I$ elemhalmazra fennáll, hogy lezártja valódi része I-nek.*

Az üres halmaz pszeudo-zárt, amennyiben az nem zárt.

A pszeudo-zárt elemhalmazok segítségével tudunk egy olyan szabálybázist megadni, amelyekből az összes egzakt asszociációs szabály megkapható.

5.2.5. Definíció *Legyen FP a pszeudo-zárt elemhalmazok halmaza $\mathcal{T}$-ben. Ekkor a Duquenne–Guigues-bázist a következőképpen definiáljuk:*

$$DG = \{r : I_1 \to h(I_1) \setminus I_1 | I_1 \in FP, I_1 \neq \emptyset\},$$

ahol az I lezártját $h(I)$-vel jelöltük.

5.2.6. Tétel *A Duquenne–Guigues-bázisból az összes egzakt szabály levezethető és a bázis minimális elemszámú, tehát az egzakt szabályoknak nincsen olyan kisebb elemszámú halmaza, amelyből az összes egzakt asszociációs szabály levezethető.*

A Duquenne–Guigues-bázis maghatározásához a pszeudo-zárt elemhalmazokra van szükség, amelyek a nem zárt gyakori elemhalmazokból kerülnek ki. A pszeudo-zártság eldöntéséhez a definícióból indulunk ki: amennyiben I nem zárt gyakori termékhalmaznak létezik olyan részhalmaza, amely lezártja tartalmazza I-t, akkor I nem pszeudo-zárt elemhalmaz. Ellenkező esetben az. Jelöljük az i-elemű gyakori, illetve gyakori zárt halmazokat GY_i és ZGY_i-vel.

Az algoritmus menete a következő: Vegyük fel az üres halmazt a pszeudo-zártak közé, amennyiben az nem zárt. Ezután vizsgáljuk $GY_1 \setminus ZGY_1$, $GY_2 \setminus ZGY_2$, $\ldots GY_m \setminus ZGY_m$ halmazokat. Az $I \in GY_i \setminus ZGY_i$ pszeudo-zártságának eldöntéséhez, az összes eddig megtalált kisebb elemszámú pszeudo-zárt elemhalmazra ellenőrizzük, hogy részhalmaza-e I-nek és ha igen akkor lezártja tartalmazza-e I-et. Amennyiben tehát létezik olyan $I' \in FP_j$ $(j < i)$, amire fennáll, hogy $I' \subset I$ és $I \subseteq h(I')$, akkor I nem pszeudo-zárt, ellenkező esetben igen. Ekkor I lezártja az I-t tartalmazó legkisebb zárt halmaz.

5.2.2. Érdekességi mutatók

Az asszociációs szabályok gyakorlati alkalmazása során az alábbi három súlyos probléma jelentkezett:

1. Az asszociációs szabályok száma túl nagy. Ha magasra állítjuk a minimális támogatottsági (min_supp) és minimális bizonyossági (min_conf) küszöbszámokat, akkor kevés szabály lesz érvényes, azonban ekkor számos – amúgy érdekes – szabály rejtve marad. Ellenkező esetben azonban rengeteg szabály jön létre, amelyek közül kézzel kiválogatni a fontosakat szinte lehetetlen feladat.

2. Az asszociációs szabályok félrevezetők lehetnek. Mivel az adatbányászat fontos stratégiai döntéseknek adhat alapot, félrevezető szabály rossz stratégiát eredményezhet. Fejtsük ki ezt egy kicsit bővebben. Egy asszociációs szabályra szoktak úgy tekinteni (helytelenül, lásd 5.2.6 rész), mint egy valószínűségi okozatiság viszonyra: adott termékhalmaz megvásárlása nagy valószínűséggel másik termékhalmaz megvásárlását „okozza". Az okozatiság valószínűségét a szabály bizonyossága adja meg. Csak ennek az értékét vizsgálni azonban nem elég!

   Képzeljünk el egy büfét, ahol az alábbiak teljesülnek. Az emberek egyharmada hamburgert vesz, egyharmada hot-dogot, egyharmada hamburgert

és hot-dogot egyszerre. Azok és csak azok vesznek majonézt, akik hamburgert esznek. Ezek szerint a „kosarak" 66%-a tartalmaz hot-dogot és a hot-dogot tartalmazó kosarak 50%-a majonézt is. Emiatt a hot-dog $\rightarrow$ majonéz érvényes asszociációs szabály lehet. Ezen asszociációs szabály alapján a hot-dogért felelős részleg vezetője [:-)] úgy dönt, hogy a nagyobb értékesítés reményében csökkenti a hot-dog árát és növeli a majonézét. A várakozásokkal ellentétben a profit csökkenni fog! Miért? Azért, mert a hamburger fogyasztók a hot-dog kedvező ára miatt inkább hot-dogot vesznek, aminek valójában semmi köze a majonézhez, azaz annak eladása nem fog nőni. Következtetésünk az, hogy egy asszociációs szabály *nem jelent okozatiságot.*

A példa jól szemlélteti, hogy a bizonyosság nem a legtökéletesebb mutató az összefüggések méréséhez. Gondoljunk arra, hogy egy szabály bizonyossága a következményrész feltételes valószínűségét próbálja becsülni, tehát $I_1 \xrightarrow{c,s} I_2$ esetén $c = p(I_2|I_1) = \frac{p(I_1,I_2)}{p(I_1)}$. Amennyiben $p(I_2|I_1)$ megegyezik $p(I_2)$-vel, akkor a szabály nem hordoz semmi többlet- hasznos információt, kivéve azt, hogy I_2 az I_1-et tartalmazó kosarakban is ugyanolyan gyakori, mint általában. Azonban rengeteg(!) ilyen szabály van.

3. A legtöbb szabály nem érdekes. Pontosabban a szabályok nagy része bizonyos más szabályoknak semmitmondó speciális esetei, apró módosításai. Szükség lenne valahogy a szabályokat fontosságuk alapján sorba rendezni, vagy minden szabályhoz egy érdekességi mutatót rendelni.

A második problémára a függetlenségi mutató bevezetése lesz a megoldás. A harmadik problémának is köze van a függetlenséghez. Érdekes szabályt, ha „felhigítunk" egy kicsit független elemekkel, akkor még kaphatunk érdekes szabályt. A felhigított szabály azonban egy extra feltételt tartalmaz így feleslegesen speciálist. Többet ér egy általános szabály, mint sok speciális szabály felsorolása.

5.2.3. Szabályok függetlensége

Az összefüggőség mérésére számos mutatószámot vezettek be, például a lift értéket, kovarianciát és korrelációt vagy a χ^2-statisztikát. A következőkben ezeket tekintjük át.

Lift érték

Egy szabály nem érdekes, ha a feltétel és a következményrészek függetlenek egymástól. Valószínűségszámításbeli ismereteinket felidézve: az X és az Y események függetlenek egymástól, ha $P(X,Y) = P(X)P(Y)$, azaz ha a $\frac{P(X,Y)}{P(X)P(Y)}$

hányados értéke 1. Minél jobban eltér a hányados egytől, annál inkább összefüggők az események. Ez alapján egy szabály *lift* értékét, amely a függetlenséget szándékozik megragadni, a következőképpen definiáljuk:

$$\text{lift}(I \to I') = \frac{freq(I \cup I')}{freq(I) \cdot freq(I')},$$

ahol $freq$ a gyakoriságot jelöli, és feltételeztük, hogy a valószínűséget a relatív gyakorisággal közelíthetjük.

Ha ezek után egy adatbázisból a rejtett összefüggéseket asszociációs szabályok formájában akarjuk kinyerni, akkor a támogatottsági és bizonyossági küszöb mellett függetlenségi küszöböt (min_lift) is megadhatunk. Például, ha $min\_lift = 1.3$, akkor azok a szabályok érdekesek, amelyekre $lift(R) \geq 1.3$ vagy $lift(R) \leq \frac{1}{1.3}$.

Gyakori termékhalmazból alkotott asszociációs szabály lift értékének meghatározásához minden adat rendelkezésünkre áll, így könnyedén megkaphatjuk az értékét.

A lift érték előnye, hogy könnyű értelmezni, még a matematika iránt kevésbé fogékonyak is megértik. Írjuk át a lift definícióját a következő alakra: $\text{lift}(I \to I') = \frac{\frac{freq(I \cup I')}{freq(I)}}{freq(I')}$. Ez az I' feltételes relatív gyakoriságának és az I' relatív gyakoriságának a hányadosa. Ha például vásárlói szokások elemzésénél a sör $\to$ pelenka szabály lift értéke 2, akkor a sört vásárlók körében a pelenkát vásárlók aránya dupla annyi, mint úgy általában a pelenkát vásárlók aránya.

Empirikus kovariancia, empirikus korreláció

A lift érték bevezetésénél használt logika alapján mondhatnánk, hogy két esemény akkor független, ha a $P(X, Y)$ és a $P(X)P(Y)$ szorzat különbsége 0. Minél jobban eltér a különbség nullától, annál nagyobb az összefüggés X és Y között. Legyen tehát a függetlenségi mutatónk

$$cov(I \to I') = freq(I \cup I') - freq(I) \cdot freq(I').$$

Relatív gyakoriságváltozás helyett abszolút gyakoriságváltozást használunk. De mi köze mindennek a címben említett empirikus kovarianciához? Egyáltalán, mi az az empirikus kovariancia?!?

„*Ausztrál kutatók állítása szerint a sok stressz elhízáshoz vezet.*"
Forrás: http://www.hirtv.hu/eletmod/?article_hid=165457

Az X és Y valószínűségi változók kovarianciája $cov(X, Y) = E[(X - \mu)(Y - \nu)] = E[X \cdot Y] - \mu \cdot \nu$, ahol μ és ν az X és Y várható értékét jelöli. Könnyű be-

látni, hogy a kovariancia nulla, amennyiben X és Y függetlenek. Ha a sűrűségfüggvényeket nem ismerjük, hanem csak megfigyelések (x_i, y_i)-k állnak rendelkezésünkre, akkor empirikus kovarianciáról beszélünk, amelynek definíciója: $\frac{1}{n}\sum_{j=1}^{n}(x_j - \bar{x})(y_j - \bar{y})$, ahol $\bar{x}$ és $\bar{y}$ a mintaátlagokat jelölik.

Az I és I' valószínűségi változók jelölhetik két termék megvételét. Az asszociációs szabályoknál bevezetett jelöléseket használva a mintaátlaga ekkor a gyakorisággal egyezik meg az i_j pedig 1, amennyiben a j-edik kosár tartalmazza az i terméket. Ekkor

$$\begin{aligned} cov(I \to I') &= \frac{1}{n}\sum_{j=1}^{n}(i_j - freq(I))(i'_j - freq(I')) \\ &= \frac{1}{n}\Big(\sum_{j=1}^{n} i_j i'_j - freq(I)\sum_{j=1}^{n} i'_j - freq(I')\sum_{j=1}^{n} i_j + n freq(I) freq(I')\Big) \\ &= freq(I \cup I') - freq(I)freq(I') - freq(I)freq(I') + freq(I)freq(I') \\ &= freq(I \cup I') - freq(I)freq(I'). \end{aligned}$$

A kovariancia normalizálásából adódik a korreláció: $corr(X, Y) = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$. A korreláció értéke mindig -1 és 1 közé esik. Számítsuk ki egy asszociációs szabály empirikus korrelációját. Mivel egynek és nullának a négyzete egy és nulla, ezért $\sigma_X^2 = E[X^2] - E^2[X] = E[X] - E^2[X]$. Ebből

$$\begin{aligned} corr(I \to I') &= \frac{Cov(I \to I')}{\sigma_I \sigma_{I'}} = \frac{freq(I \cup I') - freq(I)freq(I')}{\sqrt{E[I](1 - E[I])} \cdot \sqrt{E[I'](1 - E[I'])}} \\ &= \frac{freq(I \cup I') - freq(I)freq(I')}{\sqrt{freq(I)freq(\bar{I})freq(I')freq(\bar{I'})}}. \end{aligned}$$

A χ^2-statisztika

Valójában a lift mutató nem ragadja meg kellőképpen a két esemény (X és Y előfordulása) statisztikai függetlenségét. Tudjuk, hogy az X, Y események függetlenek, ha $P(X)P(Y) = P(X, Y)$, amelyet átírhatunk $1 = P(Y|X)/P(X)$ alakra. A jobb oldal annyiban tér el a függetlenségi mutatótól, hogy abban a valószínűségek helyén relatív gyakoriságok szerepelnek. Pusztán a relatív gyakoriságok hányadosa nem elég jó mérték a függetlenség mérésére. Nézzünk például a következő két esetet. Első esetben négy tranzakció van, $supp(I) = 2$, $c = 0.5$, amiből lift 1 adódik. A másodikban a tranzakciók száma négyezer, $supp(I) = 1992$, $c = 0.504$, amiből lift 1.012 következik. Ha csak a függetlenségi mutatókat ismernénk, akkor azt a téves következtetést vonhatnánk le, hogy az első esetben a két esemény függetlenebb, mint a második esetben. Holott érezzük, hogy az első esetben olyan kevés a tranzakció, hogy abból

nem tudunk függetlenségre vonatkozó következtetéseket levonni. Minél több tranzakció alapján állítjuk, hogy két elemhalmaz előfordulása összefüggésben van, annál jobban kizárjuk ezen állításunk véletlenségének (esetlegességének) esélyét.

A függetlenség mérésére a statisztikusok által alkalmazott eszköz az ún. χ^2 próbastatisztika. Az $A_1, A_2, \ldots, A_r$ és $B_1, B_2, \ldots, B_s$ két teljes eseményrendszer χ^2 próbastatisztikáját az alábbi képlet adja meg:

$$\chi^2 = \sum_{i=1}^{r} \sum_{j=1}^{s} \frac{\left(k_{ij} - \frac{k_{i.}k_{.j}}{n}\right)^2}{\frac{k_{i.}k_{.j}}{n}}$$

ahol k_{ij} az $A_i \cap B_j$ esemény, $k_{i.} = \sum_{j=1}^{s} k_{ij}$ az A_i esemény és $k_{.j} = \sum_{i=1}^{r} k_{ij}$ a B_j esemény bekövetkezésének számát jelöli. Minél kisebb a próbastatisztika, annál inkább függetlenek az események. A jelölést megjegyzését segítő kétszer kettes kontingenciatáblát a következő ábra mutatja.

	X	nem X	$\sum$
Y	$k_{1,1}$	$k_{1,2}$	$k_{1.}$
nem Y	$k_{2,1}$	$k_{2,2}$	$k_{2.}$
$\sum$	$k_{.1}$	$k_{.2}$	n

A mi esetünkben az egyik eseményrendszer az I elemhalmaz a másik az I' elemhalmaz előfordulásához tartozik, és mindkét eseményrendszernek két eseménye van[4] (előfordul az elemhalmaz az adott tranzakcióban, vagy sem). A következő táblázat mutatja, hogy a χ^2 próbastatisztika kiszámításához szükséges értékek közül melyek állnak rendelkezésünkre támogatottság formájában.

	I	nem I	$\sum$
I'	$supp(I \cup I')$		supp(I')
nem I'			
$\sum$	supp(I)		$\lvert\mathcal{T}\rvert$

A hiányzó értékeket a táblázat ismert értékei alapján könnyen pótolni, hiszen például $k_{2,1} = supp(I) - supp(I \cup I')$.

A χ^2 próbastatisztika helyett használhatjuk mutatószámnak a próba p-értékét. A p-érték megegyezik azzal a legnagyobb próbaszinttel, amely mellett a hipotézisünket (függetlenség) elfogadjuk.

[4]Amennyiben mindkét eseményrendszer két eseményből áll, akkor az eredeti képletet módosítani szokás a Yates-féle korrekciós együtthatóval, azaz $\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{2} \left(\left|k_{ij} - \frac{k_{i.}k_{.j}}{n}\right| - \frac{1}{2}\right)^2 / \frac{k_{i.}k_{.j}}{n}$.

A χ^2-próba közelítésen alapul ezért akkor működik jól, ha a kontingencia táblázat elemei nagyok. Kétszer kettes táblázat esetében az ökölszabály az, hogy mind a négy elem nagyobb legyen 10-nél.

Mielőtt teljes elégedettségben hátradőlnénk a karosszékünkben, mert találtunk egy tudományosan megalapozott módszert, olvassuk el a következőket.

5.2.7. Tétel *Kétszer kettes kontingenciatáblák esetében a χ^2 próbastatisztika értéke megegyezik az empirikus korreláció négyzetének n-szeresével, ahol n-nel a minták számát jelöljük.*

Bizonyítás – Írjuk fel a χ^2 próbastatisztika értékét kétszer kettes kontingenciatáblák esetére:

$$\chi^2 = \sum_{i=1}^{2}\sum_{j=1}^{2} \frac{\left(k_{ij} - \frac{k_{i.}k_{.j}}{n}\right)^2}{\frac{k_{i.}k_{.j}}{n}} = \sum_{i=1}^{2}\sum_{j=1}^{2} \frac{\left(k_{ij} - \frac{(k_{i1}+k_{i2})(k_{1j}+k_{2j})}{k_{11}+k_{12}+k_{21}+k_{22}}\right)^2}{\frac{(k_{i1}+k_{i2})(k_{1j}+k_{2j})}{n}}$$

$$= \sum_{i=1}^{2}\sum_{j=1}^{2} \frac{\frac{(k_{11}k_{22}-k_{12}k_{21})^2}{n^2}}{\frac{(k_{i1}+k_{i2})(k_{1j}+k_{2j})}{n}}$$

$$= \frac{(k_{11}k_{22}-k_{12}k_{21})^2}{n} \sum_{i=1}^{2}\sum_{j=1}^{2} \frac{1}{(k_{i1}+k_{i2})(k_{1j}+k_{2j})}$$

$$= \frac{(k_{11}k_{22}-k_{12}k_{21})^2}{n} \cdot \Big(\frac{1}{k_{11}+k_{12}} \big(\frac{1}{k_{11}+k_{13}} + \frac{1}{k_{12}+k_{22}} \big)$$

$$+ \frac{1}{k_{21}+k_{22}} \big(\frac{1}{k_{11}+k_{13}} + \frac{1}{k_{12}+k_{22}} \big) \Big)$$

$$= \frac{(k_{11}k_{22}-k_{12}k_{21})^2}{n} \cdot \Big(\big(\frac{1}{k_{11}+k_{12}} + \frac{1}{k_{21}+k_{22}} \big) \big(\frac{1}{k_{11}+k_{13}} + \frac{1}{k_{12}+k_{22}} \big) \Big)$$

$$= \frac{(k_{11}k_{22}-k_{12}k_{21})^2}{n} \cdot \Big(\frac{k_{11}+k_{12}+k_{21}+k_{22}}{(k_{11}+k_{12})(k_{21}+k_{22})} \cdot \frac{k_{11}+k_{12}+k_{21}+k_{22}}{(k_{11}+k_{21})(k_{12}+k_{22})} \Big)$$

$$= \frac{n(k_{11}k_{22}-k_{12}k_{21})^2}{k_{1.}k_{2.}k_{.1}k_{.2}} = \frac{n^3(k_{11}-\frac{k_{1.}k_{.1}}{n})^2}{k_{1.}k_{2.}k_{.1}k_{.2}} = \frac{n(f_{11}-f_{1.}f_{.1})^2}{f_{1.}f_{2.}f_{.1}f_{.2}},$$

ahol $f_{ij} = k_{ij}/n$. A bizonyítás során többször felhasználtuk, hogy $n = k_{11} + k_{12} + k_{21} + k_{22}$.

Ha a χ^2 próbastatisztika végső soron egy normalizált kovariancia, a kovariancia pedig a lift érték „testvére”. Akkor miért mond többet a χ-próbastatisztika a lift értéknél?

Egyrészről azért, mert az eredményként egy eloszlásfüggvényt kapunk, nem csak egy számot. Ez olyan, mint amikor megkérdezzük az útvonaltervező programtól, hogy mennyi időbe fog telni, hogy eljussunk A pontból B-be. Egy kezdetleges program egy konkrét számot adna eredményül. A valóságban azonban a helyes válasz egy eloszlásfüggvény, amelynek meghatározhatjuk például a várható értékét és a szórását. A szórás, amely a bizonytalanságra utal, szintén fontos paraméter.

Másrészről azért, mert figyelemebe veszi az adatbázis méretét. Nem nekünk kell meghatároznunk egy jó lift értéket, amely adatbázisonként más lesz, hanem csak a próba szintjét kell megadnunk és máris szűrhetjük ki azokat a szabályokat, amelyek feltétel- és következményrésze között nincs szignifikáns kapcsolat. Olyan, mintha a szűrésre használt küszöböt is automatikusan állítanánk elő.

A binomiális próba

A χ^2-próba és az ebből adódó p-érték nem használható, ha a 2×2-es kontingenciatáblázat valamely eleme kisebb, mint 10. Ezért visszatérünk az elemi valószínűségszámításhoz.

Induljunk ki abból, hogy az I és az I' termékek függetlenek egymástől, azaz $P(I, I') = P(I)P(I')$. Legyen $Z_j = I_j \cdot I'_j$, azaz $Z_j = 1$, amennyiben a j-edik kosárban előfordul az I és az I' termék is. A $Z = \sum_{j=1}^{n} Z_j$ binomiális eloszlású valószínűségi változó n és $P(I, I')$ paraméterekkel. A $P(I, I')$ valószínűséget a $freq(I)freq(I')$ értékkel közelítjük.

Azt kell eldöntenünk, hogy a megfigyeléseink $(z_1, \ldots z_n)$ ellentmondanake a kiindulási feltételből kapott következtetésnek. Jelöljük a próba szintjét $1 - \alpha$-val és legyen $z = \sum_{j=1}^{n} z_j$. Határozzuk meg azt a legszűkebb $[l, u]$ intervallumot, amelyre igaz, hogy $\sum_{k=l}^{u} P(Z = k) \leq 1 - \alpha$. Amennyiben z a $[l, u]$ intervallumba esik, akkor X és Y (tehát az I és I' termékhalmazok) függetlenek egymástól.

Ha ezt a megközelítést használjuk egy asszociációs szabály függetlenségének megadására, akkor legyen a függetlenségi mutató a szabály p-értéke. Határozzuk meg azt az $[l', u']$ intervallumot, amelynek minden k elemére igaz, hogy $P(Z = k) > P(Z = z)$. A p-érték ekkor $\sum_{k=l'}^{u'} P(Z = k)$.

Fisher-féle egzakt próba

A binomiális próba a $P(I, I')$ valószínűséget a $freq(I)freq(I')$ relatív gyakoriságával közelíti. A közelítés pontatlansághoz vezet. Gondoljuk meg, hogy a binomiális eloszlás nemnulla valószínűséget fog rendelni az n-nél kisebb, $\min\{supp(I), supp(I')\}$-nél nagyobb értékekhez. Azonban ezeknek a valószínűségeknek nullának kellene lenniük. Nem fordulhat az elő, hogy az I-nél na-

gyobb, I-t részhalmazként tartalmazó halmaznak $supp(I)$-nél nagyobb legyen a támogatottsága. Hasonló mondható el az $n - supp(I) - supp(I')$ értékekre, amennyiben $n - supp(I) - supp(I') > 0$. A Fisher-féle egzakt próba a közelítés helyett a pontos valószínűségeket használja.

Tegyük fel, hogy a kontingenciatáblázat ún. marginális értékei ($k_{1.}$, $k_{2.}$, $k_{.1}$, $k_{.2}$) és így a minták száma is adva vannak. Ez az asszociációs szabályoknál azt jelenti, hogy a kosarak száma, $supp(I) = k_{1.}$ és $supp(I') = k_{.1}$ rögzítettek. A kérdés a következő: Ha tudjuk, hogy a $k_{1.}$ darab I termék és a $k_{.1}$ darab I' termék egyenletes eloszlás szerint véletlenszerűen van szétszórva az n kosárban, akkor mennyi az esélye annak, hogy az I'-t tartalmazó kosarakból X darabban lesz I. Elvonatkoztatva a részletektől ez ugyanaz a kérdés, mint amelyet a hipergeometrikus eloszlás bemutatásakor tettünk fel (lásd a 2.2.2 rész). Ezek szerint

$$P(X, n, k_{1.}, k_{.1}) = \frac{\binom{k_{1.}}{X}\binom{k_{2.}}{k_{.1}-X}}{\binom{n}{k_{.1}}}.$$

Ez a valószínűség már önmagában egy jó mutatószám. Minél nagyobb az értéke, annál függetlenebbek az I és az I' termékek. Ha a χ^2 statisztikához hasonló p-értéket szeretnénk kapni, akkor ki kell számolni az összes olyan X'-re a $P(X', n, k_{1.}, k_{.1})$ valószínűséget, amelyre $P(X', n, k_{1.}, k_{.1}) \leq P(X, n, k_{1.}, k_{.1})$. Ezeket az X' értékeket hívjuk *extrémebb*, azaz kisebb valószínűségű értékeknek. A p-érték ezen extrém értékhez rendelt valószínűségek összege Formálisan:

$$p_{\text{Fisher}}(I \to I') = \sum_{X' \in \mathcal{X}} \mathbb{P}(X', n, supp(I), supp(I')),$$

ahol $\mathcal{X} = \{X' : P(X', n, supp(I), supp(I')) \leq P(supp(I \cup I'), n, supp(I), supp(I'))\}$.

A Fisher-próbát nem csak kis értékeknél használhatjuk, hanem függetlenség eldöntésére is ez a módszer adja a legjobb eredményt. Hátránya, hogy nagy $n, k_{1.}, k_{.1}$ értékeknél nehéz a valószínűségeket kiszámítani. Így jutunk el a χ^2 próbához. Amennyiben $k_{1.} \ll N$, akkor a hipergeometrikus eloszlást közelíthetjük az $k_{1.}, k_{.1}/n$ paraméterű binomiális eloszlással. A binomiális eloszlást pedig a normális eloszlással közelíthetjük. Standard normális eloszlású valószínűségi változók négyzetének összege pedig olyan valószínűségi változót ad, amelynek eloszlása a χ^2 eloszlás.

Értékinvariancia – Egy függetlenségi mutatót értékinvariánsnak hívunk, amennyiben a kontingencia-táblázat tetszőleges sorait vagy oszlopait felcserélve ugyanazt a kimenetet (p-értéket) kapjuk. Bináris esetre gondolva ez azt jelenti, hogy X és Y függetlensége esetén, X és $\bar{Y}$ (továbbá $\bar{X}$,Y és $\bar{X}$,$\bar{Y}$) is az. Ha például megállapítjuk, hogy a tejvásárlás és kenyérvásárlás függetlenek egymástól, akkor tejvásárlás, nem kenyérvásárlás is függetlenek.

Könnyű belátni, hogy a Fisher-féle egzakt próba és a χ^2 próba megfelel a fenti elvárásnak, de a binomiális próba nem. A Fisher-féle egzakt próbához csak azt kell meggondolnunk, hogy

$$P(X, n, k_{1.}, k_{.1}) = \frac{\binom{k_{1.}}{X}\binom{k_{2.}}{k_{.1}-X}}{\binom{n}{k_{.1}}} = \frac{\binom{n-k_{1.}}{k_{.1}-X}\binom{n-k_{2.}}{X}}{\binom{n}{k_{.1}}} = P(k_{.1} - X, n, n - k_{1.}, k_{.1}),$$

tehát attól, hogy a két sort (vagy a két oszlopot) felcseréljük még ugyanazt a hipergeometrikus eloszlást kapjuk. A χ^2 próbára vonatkozó állítás közvetlen adódik a χ^2 statisztika definíciójából.

A binomiális próba esetét egy példával vizsgáljuk. Induljunk ki a bal oldali kontingenciatáblából majd cseréljük fel a két sorát.

	X	nem X	$\sum$
Y	2	0	2
nem Y	0	1	1
$\sum$	2	1	3

	X	nem X	$\sum$
Y	0	1	1
nem Y	2	0	2
$\sum$	2	1	3

A bal oldali kontingenciatáblához (3, 4/9) paraméterű binomiális eloszlás tartozik. A kettőhöz nagyobb valószínűség tartozik, mint a nullához és a háromhoz, ezért a p-érték $1 - 3 \cdot \frac{4}{9} \cdot \frac{5^2}{9^2} = 0.588$. A jobb oldali kontingenciatábla binomiális eloszlásához tartozó valószínűség 2/9. A legnagyobb valószínűséget a (3, 2/9) paraméterű binomiális eloszlás nullánál veszi fel a maximumát ezért a p-érték egy.

Érdekesség – Most, hogy tudjuk hogyan kell függetlenséget meghatározni, feltehetjük azt a kérdést, hogy legalább hány megfigyelésnek kell rendelkezésünkre állnia ahhoz, hogy összefüggést állapítsunk meg.

Adott $1 - \alpha$ próbaszint mellett csak akkor tudunk összefüggést megállapítani (függetlenséget elutasítani), ha az elfogadási tartományon kívül van olyan pont, amelyet felvehet azoknak a megfigyeléseknek a száma, amelyre mindkét vizsgált tulajdonság fenáll. Az elfogadási tartományba a legnagyobb valószínűséggel rendelkező pontok esnek. Amennyiben a legkisebb valószínűségű pont valószínűsége kisebb α-nál, akkor ez a pont nem esik az elfogadási tartományba. Kétoldali próbánál két legkisebb valószínűségi pont is lehet, így ezen valószínűségek összege kell α-nál kisebbnek lennie. Ha n páratlan, akkor csak egy legkisebb valószínűségi pont lehet, éljünk ezért ezzel a feltétellel. Az általánosság megsértése nélkül feltehetjük, hogy $k_{1.} \leq k_{.1}$ és a hipergeometrikus eloszlás módusza ($\lfloor \frac{(k_{1.}+1)(k_{.1}+1)}{n+2} \rfloor$) nem nagyobb, mint az értelmezése tartomány ($[\max(0, n - k_{1.} - k_{.1}), \min(k_{1.}, k_{.1})]$) felezőpontja. A legkisebb valószínűségi

pont ekkor a $k_{1.}$, amelynek valószínűsége

$$P(k_{1.}, n, k_{1.}, k_{.1}) = \frac{\binom{k_{1.}}{k_{1.}}\binom{k_{2.}}{k_{.1}-k_{1.}}}{\binom{n}{k_{.1}}} = \frac{\frac{(n-k_{1.})!}{(k_{.1}-k_{1.})!(n-k_{.1})!}}{\frac{n!}{(n-k_{.1})!k_{.1}!}}$$
$$= \frac{(n-k_{1.})(n-k_{1.}-1)\cdots(k_{.1}-k_{1.}+1)}{n(n-1)\cdots(k_{.1}+1)}$$
$$= \prod_{i=0}^{n-k_{.1}+1}(1-\frac{k_{1.}}{n-i})$$

A fenti valószínűség rögzített n esetén akkor lesz a legnagyobb, ha $k_{1.}$ minél nagyobb, tehát $k_{1.} = k_{.1}$. Ekkor viszont

$$P(k_{1.}, n, k_{1.}, k_{1.}) = \frac{1}{\binom{n}{k_{1.}}},$$

amely $k_{1.} = \lfloor n/2 \rfloor$-nél és $k_{1.} = \lceil n/2 \rceil$ veszi fel a minimumát. Az 5.4 táblázat második oszlopa megadja a legkisebb valószínűséget néhány n-re Ezek szerint

n	$P(\lfloor n/2 \rfloor, n, \lfloor n/2 \rfloor, \lfloor n/2 \rfloor)$	p-érték	
		binom	χ^2
3	33.33%	29.76 %	8.32%
5	10.00%	18.34 %	2.53%
7	2.85%	12.11 %	0.81%
9	0.79%	8.24 %	0.27%
11	0.21%	5.70 %	0.09%
13	0.06%	4.00 %	0.03%
15	0.02%	2.82 %	0.01%

5.4. táblázat. p-értékek extrém kontingencai-táblázat esetén

97%-os bizonyossággal már hét megfigyelésből összefüggőséget állapíthatunk meg. Ehhez a legextrémebb eseménynek kell bekövetkeznie, nevezetesen, 7 megfigyelésből háromra teljesül egy tulajdonság (X) és csak erre a három megfigyelésre egy másik tulajdonság (Y) is teljesül. Tehát a kontingenciatáblázat:

	X	nem X	$\sum$
Y	3	0	3
nem Y	0	4	4
$\sum$	3	4	7

	p-érték					
	$k_{1,1} = k_{1.} - 1$			$k_{1,1} = k_{1.} - 2$		
n	fisher	binom	χ^2	fisher	binom	χ^2
5	100%	58.17%	70.9%	40%	100%	13.6%
7	48.57%	61.95%	27.0%	100%	100%	65.9%
9	20.62%	39.33%	9.89%	100%	69.4%	76.4%
11	8.00%	25.45%	3.56%	56.71%	70.75%	37.6%
13	2.91%	16.75%	1.27%	28.61%	49.42%	16.9%
15	1.01%	11.19%	0.45%	13.19%	34.30%	7.21%
17	0.35%	7.59%	0.16%	5.67%	23.74%	2.95%
19	0.11%	5.21%	0.05%	2.30%	16.44%	1.17%

5.5. táblázat. A Fisher-féle próba p-értékei

A $P(\lfloor n/2 \rfloor, n, \lfloor n/2 \rfloor, \lfloor n/2 \rfloor)$ érték egyben annak a tesztnek a p-próbája, amelyben a megfigyelések száma n és $k_{11} = k_{1.} = k_{.1} = \lfloor n/2 \rfloor$. Ha a próba szintje ennél az értéknél nagyobb, akkor elutasítjuk a függetlenségre tett feltételt, ellenkező esetben elfogadjuk. A függetlenség eldöntésére használhatnánk más próbát is. Az 5.4 táblázat harmadik és negyedik oszlopa a megfigyelés p-értékét adja meg binomiális és χ^2 próba esetén. Láthatjuk, hogy a binomiális próba jóval nagyobb p-értékeket ad ugyanarra a megfigyelésre, azaz a binomiális próba „függetlenség felé húz". Például $n = 11$ és $\alpha = 5\%$ estén a Fisher próba elutasítja a függetlenséget a binomiális próba pedig elfogadja azt.

Ha megszorítkozunk olyan kontingenciatáblákra, amelyeknél $k_{1,1} = k_{1.} - 1$, tehát nem a legextrémebb eset következik be, akkor a Fisher-féle próba p-értékei a 5.5. táblázat szerint. A 97%-os bizonyosság megtartásához most már 13 megfigyelés kell (binomiális próba szerint 21, χ^2-próba szerint is 13). Ha hat-hat megfigyelésnél teljesül az X, Y tulajdonságok, akkor abban az esetben állapítunk meg összefüggést, ha az X, Y tulajdonsággal együtt rendelkező megfigyelések száma 0, 5 vagy 6.

További mutatószámok

A lift, χ-statisztika, vagy p-érték mellett még számos elterjedt mutatószám létezik a függetlenség mérésére. A teljesség igénye nélkül felsorolunk néhányat a 5.6. táblázatban.

Név	Jelölés	Képlet	Megjegyzés
empirikus kovariancia	ϕ	$freq(I \cup I') - freq(I)freq(I')$	Az általános képlet átírásából adódik, felhasználva, hogy $\overline{I} = freq(I)$ és $\sum_{j=1}^{n} I_j = supp(I)$
empirikus korreláció		$\frac{freq(I \cup I') - freq(I)freq(I')}{\sqrt{freq(I)freq(\overline{I})}\sqrt{freq(I')freq(\overline{I'})}}$	Az általános képlet átírásából adódik, a fentiek mellett felhasználva, hogy $I_j^2 = I_j$.
esélyhányados	α	$\frac{freq(I \cup I') \cdot freq(\overline{I}, \overline{I'})}{freq(I, \overline{I'}) \cdot freq(\overline{I}, I')}$	odds ratio, cross-product ratio
Yule féle Q érték	Q	$\frac{\alpha - 1}{\alpha + 1}$	
Yule féle Y érték	Y	$\frac{\sqrt{\alpha} - 1}{\sqrt{\alpha} + 1}$	*measure of colligation*
conviction	V	$\frac{freq(I)freq(\overline{I'})}{freq(I, \overline{(I')})}$	Az $I \to I'$ implikáció logikai megfelelője alapján definiálják.
conviction*	V*	$\max\{V(I, I'), V(I', I)\}$	
Jaccard-koefficiens	ς	$\frac{freq(I \cup I')}{freq(I) + freq(I') - freq(I \cup I')}$	
koszinusz mérték	cos	$\arccos(\frac{freq(I \cup I')}{\sqrt{freq(I)freq(I')}})$	
normált kölcsönös entrópia	H^n	$\frac{H(I'\|I)}{H(I)}$	

5.6. táblázat. Néhány további mutatószám asszociációs szabályok függetlenségének mérésére

Asszociációs szabályok rangsora

Az asszociációs szabályok kinyerésének feladatában adott bemeneti sorozat és küszönszámok mellett célunk volt meghatározni az asszociációs szabályokat. Ennyi. Aztán mindenki kezdjen a szabályokkal, amit akar.

A gyakorlatban általában sok érvényes asszociációs szabályt találunk, hasznos lenne őket sorba rendezni. Ha a három paraméterhez (támogatottság/gyakoriság, bizonyosság, függetlenségi mutató) tudnánk súlyt rendelni fontosságuk szerint, akkor az alapján sorrendet tudnánk felállítani. A marketinges a támogatottságot részesítené előnyben a statisztikus a függetlenségi mutatót. Elvégre kit érdekel a két termékhalmaz támogatottsága, ha a két termékhalmaz független egymástól.

Függetlenség kifejezésére több mutatószámot adtunk meg: lift érték, empirikus kovariancia, empirikus korreláció, χ^2-statisztika, p-érték. Ráadásul χ^2-statisztika helyett használhatunk hipergeometrikus (vagy binomiális) eloszlás alapján definiált p-értéket is. Matematikusokban azonban felmerül a kérdés, hogy ugyanazt a sorrendet adják-e az egyes függetlenségi mutatók.

A χ^2-statisztika és az ebből származtatott p-érték ugyanazt a sorrendet fogja adni, hiszen a p-érték a χ^2-statisztika szigorúan monoton függvénye. A χ^2-statisztika és az empirikus korreláció között teremt szigoruan monoton kapcsolatot az 5.2.7. állítás.

Az empirikus korreláció és az empirikus kovariancia adhat különböző sorrendet. A korreláció a kovariancia normált változata. Ha két asszociációs szabály közül az elsőnek nagyobb a kovarianciája, attól még lehet kisebb a korrelációja, amennyiben az első szabályhoz tartozó két binomiális valószínűségi változó szórásának szorzata, mint a második szabályhoz tartozó két változó szórásának szorzata.

A lift érték és az empirikus kovariancia között nincs monoton kapcsolat, azaz a két mutató alapján különböző sorrendet kaphatunk. Ehhez csak azt kell meggondolnunk, hogy a, b, c, d nulla és egy közötti számokra sem igaz, hogy

$$\frac{a}{b} < \frac{c}{d} \not\Longleftrightarrow a - b < c - d.$$

5.2.4. Általánosság, specialitás

A lift mutató gyengéje, hogy ha találunk egy érdekes szabályt, akkor „az mögé elbújva" sok érdektelen szabály átmegy a szűrésen, azaz érdekesnek bizonyul. Szemléltetésképpen nézzünk egy példát. Legyen az $I_1 \to I_2$ érvényes és érdekes asszociációs szabály, továbbá I_3 egy olyan gyakori termékhalmaz, amely független I_1 és I_2-tól ($supp(I_1 \cup I_3) = freq(I_1) \cdot freq(I_3), freq(I_2 \cup I_3) = freq(I_2) \cdot freq(I_3)$) és támogatottsága olyan nagy, hogy még a $supp(I_1 \cup I_2 \cup I_3) \geq$

min_supp egyenlőtlenség is fennáll. Könnyű belátni, hogy ekkor az $I_1I_3 \to I_2$ is érvényes és érdekes asszociációs szabályok, hiszen

$$\begin{aligned}\text{lift}(I_1I_3 \to I_2) =& \frac{supp(I_1 \cup I_2 \cup I_3)}{supp(I_1 \cup I_3)supp(I_2)} = \frac{supp(I_1 \cup I_2)supp(I_3)}{supp(I_1)supp(I_2)supp(I_3)} = \\ =& \text{lift}(I_1 \to I_2) \geq \text{min\_lift},\end{aligned}$$

.

$$\text{conf}(I_1I_3 \to I_2) = \frac{supp(I_1 \cup I_2 \cup I_3)}{supp(I_1 \cup I_3)} = \frac{supp(I_1 \cup I_2)supp(I_3)}{supp(I_1)supp(I_3)} \geq \text{min\_conf}$$

Ezek alapján, egy adatbázisból kinyert érdekes asszociációs szabályok között a többség haszontalan, amennyiben sok a nagy támogatottságú, más termékektől független termék. Ha a valóságban n darab érdekes szabályunk van, de az adatbázis tartalmaz c darab a fenti tulajdonsággal rendelkező gyakori elemet, akkor az érdekességi mutató alapú szűrésen $n2^c$ szabály fog átcsúszni a fenti módon.

A fenti problémát kiküszöbölhetjük, ha a feltételrész minden elemét megnézzük független-e a feltételrész többi elemének uniójától. Ha független, akkor dobjuk ki az elemet, csak feleslegesen bonyolítja életünket. Sőt, az egész szabályt kidobhatjuk. Az eredményként kapott szabály ugyanis ott kell legyen az érvényes szabályok között, hiszen a független elem törlése esetén a függetlenségi mutató és a bizonyosság nem változik a támogatottság pedig nő.

5.2.5. Asszociációs szabályok általánosítása

Az eddig tárgyalt asszociációs szabályok számos általánosítását vezették be a kutatók. Ebben a részben ezekből szemezgetünk.

A vásárlási szokások elemzése közben a marketingesek új igénnyel álltak elő. Olyan szabályokat is ki szerettek volna nyerni, amelyek termékkategóriák között mondanak ki összefüggéseket. Például a sört vásárlók 70%-ban valami chips félét is vesznek. Lehet, hogy egyetlen sör és chips közötti asszociációs szabályt nem nyerünk ki, amennyiben sokfajta sör és chips létezik, ugyanis ezen termékek között a támogatottság „elaprózódik”. Például a sör $\to$ chips támogatottsága lehet 5000, ami érvényes asszociációs szabály lehet $min\_supp = 2000$ mellett, de ha ötféle sör létezik, akkor az egyes termékek szintjén könnyen lehet, hogy mindegyik sört tartalmazó asszociációs szabály támogatottsága 1500 alatti lesz és nem lesz érvényes.

Hierarchikus asszociációs szabályok

Gyakorlati elvárásként jelentkezett a hierarchikus asszociációs szabályok kinyerése [Srikant és Agrawal, 1995, Han és Fu, 1995, Fu, 1996, Fortin és Liu, 1996,

Thomas és Sarawagi, 1998, Shen és Shen, 1998]. Egy üzletnek a kategória szintű asszociációs szabályok legalább annyira fontosak lehetnek, mint a termékeken értelmezett szabályok (például akciót hirdetünk: '17"-os monitorok óriási árengedményekkel', miközben más számítástechnikai alkatrészek – például monitorvezérlő kártya – árait megemeljük).

Ahhoz, hogy kategóriák is szerepelhessenek asszociációs szabályokban, ismernünk kell az elemek kategóriákba, a kategóriák alkategóriákba sorolását, azaz ismernünk kell az elemek *taxonómiáját*, közgazdász nyelven szólva az elemek *nomenklatúráját*. A termék-taxonómia nem más, mint egy gyökeres címkézett fa, vagy fák sorozata. A fa leveleiben találhatók az egyes termékek, a belső csomópontokban pedig a kategóriák. Egy képzeletbeli büfé termék-taxonómiája az alábbi ábrán látható.

Ha a kategóriák halmazát $\hat{\mathcal{I}}$-vel jelöljük, akkor a bemenet továbbra is az $\mathcal{I}$ felett értelmezett sorozat, a mintatér elemei azonban $\mathcal{I} \cup \hat{\mathcal{I}}$ részhalmazai lesznek. Azt mondjuk, hogy az I kosár tartalmazza I' elemhalmazt, ha minden $i \in I'$ -re vagy $i \in I$, vagy $\exists i' \in I$, hogy $i \in \text{ős}(i')$[5]. Tehát egy kosár tartalmaz egy elemhalmazt, ha annak minden elemét, vagy annak leszármazottját tartalmazza. Nyilvánvaló, hogy ha a taxonómia egyetlen fenyőből áll, akkor a gyökérben található kategóriát minden nem üres kosár tartalmazza.

Hasonlóan módosítanunk kell az asszociációs szabályok definícióját, hiszen a 242. oldalon található definíció szerint minden $X \xrightarrow{100\%,s} \hat{X}$ szabály érvényes lenne, ha $\hat{X} \subseteq \text{ős}(X)$, és X gyakori termékhalmaz.

5.2.8. Definíció (hierarchikus asszociációs szabály) *Adott a termékek taxonómiája. A benne található termékeket és kategóriákat reprezentáló levelek, illetve belső csomópontok halmazát jelöljük $\mathcal{I}$-vel. $I_1 \xrightarrow{c,s} I_2$-t hierarchikus asszociációs szabálynak nevezzük, ha I_1, I_2 diszjunkt részhalmazai $\mathcal{I}$-nek, továbbá egyetlen $i \in I_2$ sem őse egyetlen $i' \in I_1$-nek sem.*

A támogatottság (s), és bizonyosság (c) definíciója megegyezik az 5.2.1. részben megadottal.

Hierarchikus asszociációs szabályok kinyerése csöppnyit sem bonyolultabb a hagyományos asszociációs szabályok kinyerésénél. Amikor a gyakori elemhalmazokat nyerjük ki (pl.: az Apriori módszerrel), akkor képzeletben töltsük fel a kosarakat a kosarakban található elemek ősével. Természetesen nem kell valóban előállítani egy olyan adatbázist, ami a feltöltött kosarakat tartalmazza, elég akkor előállítani ezt a kosarat, amikor a tartalmát vizsgáljuk.

Ha nem akarunk kinyerni olyan asszociációs szabályokat, amelyben bárhogyan elosztva egy elem és őse is szerepel, akkor szükségtelen az is, hogy ilyen

[5]Gyökeres gráfoknál definiálhatjuk a szülő, gyermek, ős, leszármazott fogalmakat. Ezt az alapfogalmak gráfelmélet részében megtettük.

elemhalmazokkal foglalkozzunk. Ne állítsunk elő olyan jelöltet, amely ilyen tulajdonságú [Srikant és Agrawal, 1995].

Han és Fu a fentitől különböző megközelítést javasoltak [Han és Fu, 1995, Fu, 1996]. Az algoritmus azt az észrevételt használja ki, hogy ha egy tetszőleges kategória ritka, akkor annak minden leszármazottja is ritka. Éppen ezért, az adatbázis első végigolvasása során csak a fenyők gyökerében (első szinten) található kategóriák lesznek a jelöltek. A második végigolvasásnál a gyakorinak talált elemek gyerekei, a harmadik végigolvasásnál pedig a második olvasásból kikerült gyakori elemek gyerekei, és így tovább. Akkor nincs szükség további olvasásra, ha vagy egyetlen elem sem lett gyakori, vagy a jelöltek között csak levélelemek voltak.

A gyakori elempárok meghatározásához először ismét csak a gyökerekben található kategóriákat vizsgáljuk, természetesen csak azokat, amelyeknek mindkét eleme gyakori. A következő lépésben a pár egyik tagjának a második szinten kell lennie, és hasonlóan: az i-edik végigolvasásnál a jelöltpárosok egyik tagja i-edik szintbeli.

A fenti eljárást könnyű általánosítani gyakori elemhármasok és nagyobb méretű gyakori termékhalmazok megtalálására. A leállási feltétel hasonló az Apriori algoritmuséhoz: ha a jelöltek közül senki sem gyakori, akkor minden gyakori hierarchikus termékhalmazt megtaláltunk. A továbbiakban az algoritmust nem tárgyaljuk, a részletek és futási eredmények Han és Fu cikkében találhatók meg [Han és Fu, 1995].

Hierarchikus szabály „érdekessége” – Kategóriák bevezetésével az érvényes asszociációk száma nagymértékben nőhet. Ennek oka az, hogy a kategóriák támogatottsága mindig nagyobb, mint a bennük szereplő termékeké, így sokszor szerepelnek majd gyakori termékhalmazokban, amelyekből az érvényes asszociációs szabályokat kinyerjük. A szabályok között sok semmitmondó is lesz, amelyek csökkentik az áttekinthetőséget, és a tényleg fontos szabályok megtalálását. Egy ilyen semmitmondó szabályt az alábbi példa szemléltet:

Egy élelmiszerüzletben háromféle tejet lehet kapni: zsírszegényt, félzsírosat, és normált. Az emberek egynegyede zsírszegény tejet iszik. Hierarchikus szabályok kinyerése során többek között az alábbi két érvényes szabály is szerepel (a szabály harmadik paramétere a lift értéket adja):

$$\text{tej} \xrightarrow{80\%,4.8\%,2} \text{zabpehely}$$
$$\text{zsírszegény tej} \xrightarrow{80\%,1.2\%,2} \text{zabpehely}$$

Látható, hogy a második szabály kevésbé általános, mint az első és nem hordoz semmi többletinformációt. Jogos tehát az a kijelentés, hogy egy szabály

nem érdekes, ha annak bizonyossága és támogatottsága nem tér el a nála általánosabb szabály paraméterei alapján becsült értékektől. A pontos definíciók magadásával nem terheljük az olvasót.

Kategória asszociációs szabályok

Az asszociációs szabályok kinyerésénél a bemenet elemhalmazok sorozataként van megadva (plussz néhány küszöbszám). Ábrázolhatjuk a bemenetet, mint bináris mátrix, amelynek az i-edik sor j-edik eleme egy, ha szerepel az i-edik tranzakcióban a j-edik elem, különben nulla. Tetszőleges bináris relációs táblát is választhatunk bemenetként, ekkor például (*nem* = *férfi*) $\rightarrow$ (*tájékozódási készség* = *jó*) jellegű szabályokat nyerünk ki.

Könnyen kaphatunk kategória asszociációs szabályt a meglévő módszereinkkel. Minden olyan A attribútumot, amely k különböző értéket vehet fel ($k > 2$), helyettesítsünk k darab bináris attribútummal. Ha egy elem A attribútumának értéke az i-edik attribútumérték volt, akkor csak i-edik új attribútum értéke legyen egy, a többié pedig nulla. Az így kapott bináris táblán már futtathatjuk a kedvenc asszociációs szabályokat kinyerő algoritmusunkat.

5.2.6. A korreláció nem jelent ok-okozati kapcsolatot

Szemben a bizonyossággal, a támogatottság és a számos elterjedt érdekességi mutató (χ^2-próbastatisztika, p-érték) szimmetrikus függvény, nem veszi figyelembe, hogy melyik termékhalmaz szerepel a szabály feltételrészében és melyik a szabály következményrészében. Az asszociációs szabályokban a nyilat használjuk az irány kijelölésére, amely a felületes szemlélő számára megtévesztő lehet.

Ha megvizsgáljuk az asszociációs szabályok három paraméterét (támogatottság, bizonyosság, lift érték), akkor rájöhetünk, hogy egyik paraméter sem jelent okozatiságot. A függetlenségi paraméter csak azt mondja meg, hogy a feltételrész nem független a következményrésztől. Okozatiságról szó sincs. Biztosan csak azt állíthatjuk, hogy nincs okozatiségi viszony olyan jelenségek között, amelyek között korreláció sem áll fenn (azaz függetlenek). Fordítva azonban nem igaz az állítás. A korreláció és az okozatiság összekeverése nagyon gyakori hiba, amelyre a latin *cum hoc ergo propter hoc* (magyarul: vele, ezért miatta) kifejezéssel hivatkoznak.

Ha A és B között korreláció van, akkor lehet, hogy A okozza B-t, de lehet, hogy másféle kapcsolat áll fenn köztük. Az is lehet, hogy

1. B okozza A-t;

2. egy harmadik C jelenség okozza A-t és B-t is, vagy az okozatisági viszonyuk ennél bonyolultabb;

3. a megfigyelt korrelációt véletlenek különös együttállása okozza: emlékezzünk, hogy a statisztikai tesztek sosem mondanak teljes bizonyossággal semmit, az elsőfajú hiba adja meg annak esélyét, hogy mi azt állítjuk, hogy két esemény között összefüggés áll fenn, holott azok függetlenek egymástól;

4. A és B egymást is okozhatják kölcsönösen megerősítő módon.

Nézzünk néhány példát:

- *Az egy cipőben alvás erős összefüggésben áll a fejfájással ébredéssel. Tehát a cipőben alvás fejfájást okoz.* Nyilvánvalóan hibás ez a következtetés. Sokkal kézenfekvőbb az a magyarázat, hogy az ittas állapot okozza mindkét dolgot.

- A következő állítás egy magyar kereskedelmi rádióban hangzott el. Forrásnak amerikai kutatókat neveztek meg. *A magassarkú cipő skizofréniát okoz.* Minden bizonnyal csak véletlenek különös együttállásának köszönhető ez a mefigyelés.

- Az alábbi állítás viszont a Nature magazinban hangzott el 1993-ban. *Valószínűbb, hogy rövidlátók lesznek azok a gyerekek, akik égő lámpa mellett alszanak.* Későbbi kutatások kimutatták, hogy valójában a szülők rövidlátása és a gyerekek rövidlátása között van összefüggés továbbá a rövidlátó szülők hajlamosabbak a lámpát égve hagyni, innen ered a korreláció.

Ha vásárlói kosarak elemzéséhez kanyarodunk vissza, akkor ezek szerint $I \to I'$ nem az jelenti, hogy az I termék az I' termék megvásárlását okozza. Ha mind I, mind I' megvételét egy harmadik I'' terméknek köszönhetjük, akkor csak pénzt veszítenénk, ha az I termék árát csökkentenék a I'-ét pedig növelnénk. Az I eladásának növekedése ugyanis nem okozza az I' eladását, tehát nem nyernénk vissza az I'-vel az I árcsökkenéséből származó profitkiesést.

A valóságban a termékek csoportokat alkotnak, amelyekben a termékek eladása kölcsönösen megerősítik egymást. Például, ha veszünk egy fényképezőgépet, akkor sokan memóriakártyát és tokot is vesznek. Ha okozati kapcsolatok csak a fényképező $\to$ memóriakártya és a fényképező $\to$ tok lennének, akkor matematikailag a fényképező és a memóriakártya eladásának nem kéne változnia, ha a tok árusítását megszüntetnénk. Legtöbbünknek azonban igenis számít, hogy egy helyen lehet megvásárolni mindhárom terméket, ezért az eladások igenis csökkennének. A fényképezőgép, memóriakártya és tok termékhalmaz egy olyan halmaz, amelynek elemei kölcsönösen megerősítik egymás eladását.

A Simpson-paradoxon

A helyzet valójában az eddigiekben bemutatottnál is bonyolultabb: az, hogy két termék eladásai közt korrelációt mutatunk ki az eladási adataink segítségével, nem csak azt nem jelenti, hogy az egyik a másik eladásának *ok-okozati* értelemben vett *oka*, de sajnos az is előfordulhat, hogy a két termék egymás eladását egyenesen gátolja. Ezt szemléltetjük egy példával [Tan és tsa., 2005] nyomán.

Tekintsük az A és B termékek eladásait:

	B-t vásárol	B-t nem vásárol	Σ
A-t vásárol	99	81	180
A-t nem vásárol	54	66	120
Σ	153	147	

Ezek alapján arra következtetünk, hogy az A és B termékek vásárlásai korrelálnak. Az A-t vásárlók nagyobb része vásárol B-t is ($99/180 \approx 0.55$), míg az A-t nem vásárlók nagyobbik része B-t sem vásárol. A B oldaláról névze pedig a B-t vásárlók majdnem kétharmada vásárol A-t is, míg a B-t nem vásárlóknak jóval kevesebb, mint kétharmada, 55%-a vásárol csak A-t.

Egy részletesebb elemzés során két fogyasztói csoportot külön-külön vizsgáltunk, és a következő eredményeket kaptuk. Az első fogyasztói csoportban:

	B-t vásárol	B-t nem vásárol	Σ
A-t vásárol	1	9	10
A-t nem vásárol	4	30	34
Σ	5	39	

A második fogyasztói csoportban:

	B-t vásárol	B-t nem vásárol	Σ
A-t vásárol	98	72	170
A-t nem vásárol	50	36	86
Σ	148	108	

Vegyük észre, hogy mindkét fogyasztói csoportban igaz, hogy az A-t nem vásárlók nagyobb arányban vásárolták a B-t, mint az A termék vásárlói. A B termék felől nézve az is igaz, hogy a B-t nem vásárlók nagyobb arányban vásárolták az A-t, mint a B termék vásárlói.

A paradoxon feloldása az, hogy az A és B termékek eladása valójában nem egymással korrelál, hanem mind a két termék a második fogyasztói csoportra

jellemző.

5.2.7. Asszociációs szabályok és az osztályozás

A következő részben az osztályozással és a regresszióval fogunk foglalkozni. Mik a hasonlóságok és mik a különbségek az asszociációs szabályok kinyerése és az osztályozás között? Mindkét feladatban attribútumok közötti összefüggéseket tárunk fel.

Az asszociációs szabályok előnye, hogy tetszőleges két attribútumhalmaz között találhat összefüggést. Ezzel szemben osztályozásnál kijelölünk egy attribútumot és csak azt vizsgáljuk, hogy ezt az attribútumot hogyan határozzák meg a többi attribútumok. Asszociációs szabályok jellemző alkalmazási területe a vásárlási szokások elemzése, ahol minden termékösszefüggés érdekes lehet.

Asszociációs szabályoknál bináris attribútumokkal dolgozik. Ha a feltételrészben szereplő attribútumok értéke egy, akkor a következményrészben szereplő attribútum is egy lesz. Ha a feltételrész értéke nulla, akkor nem tudunk semmilyen megállapítást tenni a következményrészre vonatkozóan. Osztályozásnál ilyen nincs, ha tudjuk a magyarázó attribútumok értékét, akkor tudjuk a magyarázandóét is. Az attribútumtípusokra annyi megkötés van, hogy a magyarázandó attribútum kategória típusú legyen (regressziónál numerikus).

Más az egyes területek tudományos cikkeinek témája is. Az asszociációs szabályokról szóló cikkek nagy része gyakori elemhalmazok kinyeréséről szól. A fő cél az, hogy minél gyorsabb algoritmust adjunk erre az adott feladatra. A feladat értlemét nem vonják kétségbe (sem azt, hogy tényleg szükség van-e olyan gyors algoritmusokra, amelyek gigabájt méretű adatokat tudnak feldolgozni másodpercek alatt és gigabájt méretű kimenetet generálnak). A cikkekben algoritmikus és adatstruktúrális megoldásokat mutatnak be, implementációs és párhuzamosíthatósági kérdéseket vizsgánk, nem ritkán egy módszer elemzésénél a hardver tulajdonságait is számításba veszik.

Ezzel szemben osztályozásnál az osztályozás pontosságának javítása a fő cél, a hatékonyságbeli kérdések csak másodlagosak. Az osztályozás kutatói általában jóval komolyabb statisztikai tudással rendelkeznek.

5.3. Gyakori minták kinyerése

A fejlett társadalmakra jellemző, hogy számos, a mindennapi életünk során gyakran használt terméket és szolgáltatást nélkülözhetetlennek tartunk. Minél sokszínűbb a felhasználói csoport, annál nehezebb egy olyan üzenetet eljuttatni részükre, ami mindenki számára egyértelmű, ám ha valakinek ez sikerül,

az nagy haszonnal járhat, hiszen pár százalékpontos növekedés is szignifikáns a nagy volumenben értékesített termékeknél. A piaci stratégiák kialakításánál is elsősorban a sokaságra, illetve a sokaság jellemzőire vagyunk kíváncsiak. Egyedi, különc elemek akkor érdekesek, ha például csalásokat akarunk felderíteni. Fenti eseteken kívül vizsgálhatjuk a gyakori balesetet okozó helyzeteket, a számítógépes hálózatban gyakran előforduló, riasztással végződő eseménysorozatokat, vagy pl. azt, hogy az egyes nyomtatott médiumoknak milyen az olvasói összetétele, és amennyiben több magazinnak, újságnak hasonló a célcsoportja, érdemes üzenetünket több helyen is elhelyezni, hogy hatékonyabban ösztönözzük meglevő és potenciális vásárlóinkat.

Oldalakon keresztül lehetne sorolni azon példákat, amikor a gyakran előforduló „dolgok" értékes információt rejtenek magukban. A szakirodalomban a dolgokat mintáknak nevezzük, és *gyakori minták kinyeréséről* beszélünk.

A minta típusa többféle lehet. Vásárlói szokások felderítésénél gyakori *elemhalmazokat* kerestünk, ahol az elemek a termékeknek felelnek meg. Utazásokkal kapcsolatos szokásoknál a gyakran igénybe vett, költséges szolgáltatások sorrendje is fontos, így gyakori *sorozatokat* keresünk. Telekommunikációs hálózatokban olyan feltételek (predikátumok) gyakori fennállását keressük, amelyek gyakran eredményeznek riasztást. Ezeket a gyakori *bool formulákat* megvizsgálva kaphatjuk meg például a gyakori téves riasztások okait. A böngészési szokások alapján fejleszthetjük oldalaink struktúráját, linkjeit, így a látogatók még gyorsabban és hatékonyabban találják meg a keresett információkat. A böngészés folyamatát *címkézett gyökeres fákkal* jellemezhetjük. Gyakori mintákat kinyerő algoritmusokat a rákkutatásban is alkalmaztak. Azt vizsgálták, hogy a rákkeltő anyagokban vannak-e gyakran előforduló molekula-struktúrák. Ezeket a struktúrákat címkézett gráfokkal írjuk le.

A példákból következik, hogy a minta típusa sokféle lehet. Sejthetjük, hogy más technikákat kell majd alkalmazni pl. címkézett gráfok keresésénél, mintha csak egyszerű elemhalmazokat keresünk. Ebben a részben egy általános leírást adunk, egy egységes matematikai keretbe helyezzük a gyakori minta kinyerésének feladatát. Emellett ismertetjük a legfontosabb módszerek általános – a minta típusától független – leírását.

5.3.1. A gyakori minta definíciója

E rész megértéséhez feltételezzük, hogy az olvasó tisztában van a 2.1 részben definiált fogalmakkal (rendezések, korlát, valódi korlát, maximális korlát, predikátum).

5.3.1. Definíció *A H halmaz a $\preceq$ rendezésre nézve* lokálisan véges*, ha minden $x, y \in H$ elemhez, ahol $x \preceq y$,véges számú olyan z elem létezik, amelyre $x \preceq$*

$z \preceq y$.

5.3.2. Definíció *Az $\mathcal{MK} = (\mathcal{M}, \preceq)$ párost, ahol $\mathcal{M}$ egy alaphalmaz, $\preceq$ az $\mathcal{M}$-en értelmezett részben rendezés,* mintakörnyezetnek *nevezzük, amennyiben $\mathcal{M}$-nek pontosan egy minimális eleme van, $\mathcal{M}$ halmaz a $\preceq$ rendezésre nézve lokálisan véges és* rangszámozott *(graded), azaz létezik a $|\ |\ : \mathcal{M} \to \mathbb{Z}$ ún. méretfüggvény , amire $|m| = |m'| + 1$, ha m-nek maximális valódi alsó korlátja m'. Az $\mathcal{M}$ elemeit* mintáknak *(pattern) nevezzük és $\mathcal{M}$-re, mint* mintahalmaz *vagy* mintatér *hivatkozunk.*

Az $m' \preceq m$ esetén azt mondjuk, hogy m' az m *részmintája*, ha $m' \prec m$, akkor *valódi részmintáról* beszélünk. A $\preceq$-t *tartalmazási relációnak* is hívjuk. Az általánosság megsértése nélkül feltehetjük, hogy a minimális méretű minta mérete 0. Ezt a mintát *üres mintának* hívjuk.

Íme az egyik legegyszerűbb példa mintakörnyezetre, amelyet vásárlói szokások feltárása során alkalmaztak először. Legyen $\mathcal{I}$ véges halmaz. Gyakori *elemhalmazok* keresésénél a $(2^{\mathcal{I}}, \subseteq)$ lesz a mintakörnyezet, ahol $\subseteq$ a halmazok tartalmazási relációját jelöli. A méretfüggvény egy halmazhoz az elemszámát rendeli. Az elemhalmazokon túl kereshetünk gyakori *sorozatokat*, *epizódokat* (véges halmazon értelmezett részben rendezéseket), *bool formulákat*, *címkézett gyökeres fákat* vagy *általános gráfokat.* Ezen mintakörnyezetek pontos definícióját a következő fejezetekben találjuk.

5.3.3. Definíció *Legyen $(H_1, \preceq_1)$ $(H_2, \preceq_2)$ két részben rendezett halmaz. Az $f : H_1 \to H_2$ függvény* rendezés váltó *vagy más szóval* anti-monoton*, amennyiben tetszőleges $x, y \in H_1$, $x \preceq_1 y$ elemekre $f(y) \preceq_2 f(x)$.*

5.3.4. Definíció *A* gyakori minta kinyerésnek feladatában *adott egy $\mathcal{B}$ bemeneti (vagy feldolgozandó) adathalmaz, $\mathcal{MK} = (\mathcal{M}, \preceq)$ mintakörnyezet, egy $supp_{\mathcal{B}} : \mathcal{M} \to \mathbb{N}$ anti-monoton függvény és egy $min\_supp \in \mathbb{N}$ küszöbszám. Feladat, hogy megkeressük azon mintákat, amelyekre a supp függvény min_supp-nál nagyobb vagy egyenlő értéket ad:*

$$GY = \{gy : gy \in \mathcal{M}, supp_{\mathcal{B}}(gy) \geq min\_supp\}.$$

A $supp_{\mathcal{B}}$ függvényt *támogatottsági függvénynek* (support function), min_supp-ot *támogatottsági küszöbnek*, a GY elemeit pedig *gyakori mintáknak* hívjuk. A nem gyakori mintákat *ritkáknak* nevezzük. Az érthetőség kedvéért a $\mathcal{B}$ tagot gyakran elhagyjuk, továbbá a $supp(m)$-re mint a minta támogatottsága hivatkozunk. A támogatottsági függvény értéke adja meg, hogy egy minta mennyire gyakori a bemenetben.

Az elemhalmazok példájánál maradva a bemenet lehet például elemhalmazok sorozata. Ekkor egy H halmaz támogatottságát úgy értelmezhetjük,

mint a sorozat azon elemeinek száma, amelyek tartalmazzák H-t. Például az $\langle\{A,D\},\{A,C\},\{A,B,C,D\},\{B\},\{A,D\},\{A,B,D\},\{D\}\rangle$ bemenet esetén $supp(\{A,D\}) = 4$. Ha min_supp-nak 4-et adunk meg, akkor $GY = \{\{A\},\{D\},\{A,D\}\}$.

A támogatottság anti-monotonitásából következik az alábbi egyszerű tulajdonság.

Részminta Antimonotonitási Tulajdonság – Gyakori minta minden részmintája gyakori.

„*Amerikai kutatás során megállapították, hogy 1 óra tévézés hatására 200 dollárral többet költünk a reklámok miatt.*" Forrás: Sláger rádió, 2007. október 25., 17 óra 48 perc

A mintákat elemhalmazok, sorozatok, gráfok, stb. formájában fogjuk keresni, azaz a minták mindig valamilyen alaphalmazon definiált struktúrák lesznek. Ha az alaphalmazon definiálunk egy teljes rendezést, akkor az alapján – könnyebben vagy nehezebben – a mintákon is tudunk teljes rendezést adni. Ezt például elemhalmazok esetében a lexikografikus rendezés, gráfok esetében a kanonikus címkézés segítségével fogjuk megtenni. A mintákon értelmezett teljes rendezés egyes algoritmusoknál (pl.: APRIORI) a hatékonyság növelésére használható, másoknak pedig alapfeltétele (pl.: Zaki). Sokszor fog felbukkanni a *prefix* fogalma is, amihez szintén egy teljes rendezésre lesz szükség.

5.3.5. Definíció *Legyen $\preceq$ a H halmazon értelmezett részben rendezés. A $\prec'$ teljes rendezést a $\prec$* lineáris kiterjesztésének *hívjuk, ha minden $x \prec y$ párra $x \prec' y$ teljesül.*

A lineáris kiterjesztéseknek azon csoportja érdekes számunkra, amelyek *mérettartóak*. Ez azt jelenti, hogy $|x| < |y|$ esetén a $x \prec' y$ feltételnek is fenn kell állnia. Amikor tehát a $\mathcal{MK} = (\mathcal{M}, \preceq)$ mintakörnyezet $\preceq$ tagjának egy mérettartó lineáris kiterjesztését akarjuk megadni, akkor az azonos méretű elemek között definiálunk egy sorrendet. A továbbiakban a mérettartó jelzőt elhagyjuk, és minden lineáris kiterjesztés alatt mérettartó lineáris kiterjesztést értünk.

5.3.6. Definíció *Legyen $\mathcal{MK} = (\mathcal{M}, \preceq)$ mintakörnyezet és $\preceq'$ a $\preceq$ egy lineáris kiterjesztése. Az m minta ℓ-elemű részmintái közül az $\preceq'$ szerinti legelsőt hívjuk az m minta ℓ-elemű prefixének.*

Például, ha $\mathcal{I} = \{A,B,C,D,E\}$, és az azonos méretű mintákon az abc rendezés szerinti lexikografikus rendezést vesszük a teljes rendezésnek, akkor például az $\{A,C,D,E\}$ minta 2-elemű prefixe az $\{A,C\}$ halmaz.

Hatékonysági kérdések

A bemeneti adat és a minták halmaza általában nagy. Például bemeneti sorozatok esetében nem ritkák a 10^9 nagyságrendű sorozatok, a mintatér pedig általában 10^5 nagyságrendű halmazok hatványhalmaza. Ilyen méretek mellett a naív algoritmusok (például határozzuk meg a mintahalmaz minden elemének támogatottságát, majd válogassuk ki a gyakoriakat) túl sok ideig futnának, vagy túl nagy lenne a memóriaigényük. Hatékony, kifinomult algoritmusokra van szükség, amelyek speciális adatstruktúrákat használnak.

Egy algoritmus hatékonyságát a futási idővel (ami arányos az elemi lépések számával) és a felhasznált memóriával jellemezzük. Például megmondhatjuk, hogy adott méretű bemenet esetén átlagosan, vagy legrosszabb esetben mennyi elemi lépést (összehasonlítás, értékadás), illetve memóriát használ. Sajnos a gyakori mintát kinyerő algoritmusok mindegyike legrosszabb esetben a teljes mintateret megvizsgálja, ugyanis a támogatottsági küszöb függvényében a mintatér minden eleme gyakori lehet.

A gyakori minta-kinyerés korszakának első 10-15 évében az algoritmusok hatékonyságát – elméleti elemzések híján – minden esetben teszteredményekkel igazolták. Szinte minden algoritmushoz lehet találni olyan bemeneti adatot, amit az algoritmus nagyon hatékonyan képes feldolgozni. Ennek eredményeként például, csak a gyakori elemhalmazokat kinyerő algoritmusok száma meghaladja a 150-et, és a mai napig nem tudunk olyan algoritmusról, amelyik az összes többit legyőzné futási idő vagy memóriafogyasztás tekintetében.

A jövő feladata ennek a káosznak a tisztázása. Ehhez a legfontosabb lépés a bemeneti adat karakterisztikájának formális leírása lenne. Sejtjük, hogy legjobb gyakori mintakinyerő algoritmus nem létezik, de talán van esélyünk értelmes megállapításokra, ha a bemenetre vonatkozóan különböző feltételezésekkel élünk (szokásos feltétel például az, hogy a bemenet olyan sorozat, melynek elemei kis méretű halmazok vagy az, hogy csak nagyon kevés magas támogatottságú minta van) és ezekhez próbáljuk megtalálni az ideális algoritmust.

5.3.2. További feladatok

A gyakori mintakinyerés egyik nagy kritikája, hogy sokszor túl nagy a kinyert minták száma. Vannak olyan feladatok, ahol nem az összes gyakori mintát kívánjuk kinyerni, hanem csak egy részüket. Erre példa az ún. *top-k* mintakinyerés, melynek során a k legnagyobb támogatottságú mintát keressük. Emellett az alábbi feladatok léteznek.

Nem bővíthető és zárt minták

5.3.7. Definíció *Az m gyakori minta $\mathcal{B}$-re nézve* nem bővíthető *(maximal), ha nem létezik olyan m' gyakori minta $\mathcal{B}$-ben, amelynek m valódi részmintája.*

5.3.8. Definíció *Az m minta $\mathcal{B}$-re nézve zárt, amennyiben nem létezik olyan m' minta $\mathcal{B}$-ben, amelynek m valódi részmintája, és m' támogatottsága megegyezik m támogatottságával ($supp(m') = supp(m)$).*

Az ember azonnal láthatja, hogy mi értelme van annak, hogy csak a nem bővíthető mintákat keressük meg: egyértelműen meghatározzák a gyakori mintákat és számuk kevesebb. Sajnos a nem bővíthető minták alapján csak azt tudjuk megmondani, hogy egy minta gyakori-e, a támogatottságot nem tudjuk megadni (legfeljebb egy alsó korlátot).

Nem ilyen triviális, hogy mi értelme van a gyakori zárt mintáknak. Azt látjuk, hogy a zárt gyakori minták a gyakori minták részhalmazai, és a zárt minták részhalmaza a nem bővíthető minták, hiszen

Tulajdonság – Minden nem bővíthető minta zárt.

Mégis mi célt szolgálnak a gyakori zárt minták? Ennek tisztázásához két új fogalmat kell bevezetnünk.

5.3.9. Definíció *Az m' minta az m minta* lezártja*, ha $m \preceq m', supp(m) = supp(m')$ és nincs $m'' : m' \prec m''$, melyre $supp(m') = supp(m'')$.*

Nyilvánvaló, ha m zárt, akkor lezártja megegyezik önmagával.

5.3.10. Definíció *Az $\mathcal{MK} = (\mathcal{M}, \preceq)$ mintakörnyezet a* zártságra nézve egyértelmű*, amennyiben minden $m \in \mathcal{M}$ minta lezártja egyértelmű.*

Látni fogjuk, hogy sorozat típusú bemenet esetén például az elemhalmazokat tartalmazó mintakörnyezet zártságra nézve egyértelmű, míg a sorozatokat tartalmazó nem az. A zártságra nézve egyértelmű mintakörnyezetekben a zárt minták jelentősége abban áll, hogy ezek ismeretében tetszőleges mintáról el tudjuk dönteni, hogy gyakori-e, és ha igen, meg tudjuk pontosan mondani támogatottságát. Szükségtelen tárolni az összes gyakori mintát, hiszen a zárt mintákból ezek egyértelműen meghatározhatók. Az m minta gyakori, ha része valamely gyakori zárt mintának, és m támogatottsága megegyezik a legkisebb olyan zárt minta támogatottságával, amelynek része m (ez ugyanis az m lezártja).

Kényszerek kezelése

Nem mindig érdekes az összes gyakori minta. Előfordulhat, hogy például a nagy méretű, vagy bizonyos mintákat tartalmazó, vagy nem tartalmazó, stb. gyakori minták nem fontosak. Általánosíthatjuk a feladatot úgy, hogy a felhasználó kényszereket, predikátumokat ad meg, és azokat a mintákat kell meghatároznunk, amelyek kielégítik az összes kényszert.

A feladat egyszerű megoldása lenne, hogy – mint utófeldolgozás – a gyakori mintákat egyesével megvizsgálva törölnénk azokat, amelyek nem elégítenek minden kényszert. Ez a megoldás nem túl hatékony. Jobb lenne, ha a kényszereket minél „mélyebbre" tudnánk helyezni a gyakori mintákat kinyerő algoritmusokban. Ez bizonyos kényszereknél megtehető, másoknál nem. Nézzük, milyen osztályokba sorolhatjuk a kényszereket.

Tulajdonképpen az is egy kényszer, hogy gyakori mintákat keresünk. A gyakoriságra vonatkozó predikátum igaz, ha a minta gyakori, ellenkező esetben hamis. Ez a predikátum anti-monoton:

5.3.11. Definíció *Legyen $(H, \preceq)$ egy részben rendezett halmaz. A $p : H \to \{igaz, hamis\}$ predikátum* anti-monoton*, amennyiben tetszőleges $x \in H$ elem esetén, ha $p(x) = igaz$, akkor $p(y)$ is igazat ad minden $y \preceq x$ elemre.*

Ha a fenti definícióba $y \preceq x$ helyett $x \preceq y$ írunk, akkor a *monoton* predikátumok definícióját kapjuk. Egy predikátum akkor és csak akkor monoton és anti-monoton egyben, ha a mintatér minden eleméhez igaz (vagy hamis) értéket rendel. Az ilyen predikátumot *triviális predikátumnak* hívjuk.

5.3.12. Definíció *Legyen $(H, \preceq)$ egy részben rendezett halmaz. A $p : H \to \{igaz, hamis\}$ predikátum* prefix anti-monoton*, amennyiben megadható a $\prec$-nek egy olyan $\preceq'$ lineáris kiterjesztése amire, ha $p(m) = igaz$, akkor p az m minden prefixén is igaz.*

5.3.13. Definíció *Legyen $(H, \preceq)$ egy részben rendezett halmaz. A $p : H \to \{igaz, hamis\}$ predikátum* prefix monoton*, amennyiben megadható a $\prec$-nek egy olyan $\preceq'$ lineáris kiterjesztése amely, ha $p(m) = igaz$, és az m' mintának m prefixe. akkor $p(m')$ is igaz.*

Minden anti-monoton (monoton) predikátum egyben prefix anti-monoton (prefix monoton) is.

5.3.14. Definíció *A p predikátum* erősen átalakítható*, amennyiben egyszerre prefix anti-monoton és prefix monoton.*

Az 5.12 ábrán látható a kényszerek kapcsolata [Pei és tsa., 2001].

Sejthetjük, hogy az anti-monoton predikátumok lesznek a legegyszerűbben kezelhetők. Ilyen anti-monoton predikátumok például a következők:

- A minta mérete ne legyen nagyobb egy adott küszöbnél.
- A mintának legyen része egy rögzített minta.

Vásárlói szokások vizsgálatánál – amikor a vásárlói kosarakban gyakran előforduló termékhalmazokat keressük – monoton kényszer például az, hogy a termékhalmazban lévő elemek profitjának összértéke (vagy minimuma, maximuma) legyen nagyobb egy adott konstansnál.

Prefix monoton predikátum például, hogy a termékhalmazban található termékek árának átlaga nagyobb-e egy rögzített konstansnál. Rendezzük a termékeket áruk szerint növekvő sorrendbe. Ezen rendezés szerinti lexikografikus rendezés legyen a teljes rendezés. Nyilvánvaló, hogy ekkor a prefixben található termékek árai nagyobbak, mint a prefixben nem szereplő termékei árai. Ez a kényszer prefix monoton, hiszen a prefix a legolcsóbb termékeket nem tartalmazza, így átlaga nem lehet kisebb. Érdemes átgondolni, hogy ez a predikátum ráadásul erősen átalakítható.

Többszörös támogatottsági küszöb

Vannak olyan alkalmazások, amelyekben a gyakoriság egyetlen, univerzális támogatottsági küszöb alapján történő definiálása nem megfelelő. Ha például vásárlási szokások elemzésére gondolunk, akkor a nagy értékű termékekkel kapcsolatos tudás legalább annyira fontos, mint a nagy mennyiségben értékesített, de kis haszonnal járó termékekkel kapcsolatos információ. Kézenfekvő megoldás, hogy annyira lecsökkentjük a támogatottsági küszöböt, hogy ezek a ritka elemek is gyakoriak legyenek, ami azzal a veszéllyel jár, hogy (ezen fontos elemek mellett) a mintatér nagy része gyakorivá válik. *Többszörös támogatottsági küszöbnél* a mintatér minden eleméhez egyedileg megadhatunk egy támogatottsági küszöböt, azaz létezik egy $min\_supp : \mathcal{M} \to \mathbb{N}$ függvény, és az m akkor gyakori, ha $supp(m) \geq min\_supp(m)$.

Többszörös támogatottsági küszöb esetén nem igaz a részminta antimonotonitási tulajdonság. Hiába nagyobb ugyanis egy részminta támogatottsága, a részmintához tartozó támogatottsági küszöb még nagyobb lehet, és így a részminta nem feltétlenül gyakori.

Dinamikus gyakori mintakinyerés

Egyre népszerűbb adatbányászati feladat a gyakori minták ún. dinamikus kinyerése. Adott egy kiindulási $\mathcal{B}$ bemenet a hozzá tartozó gyakori mintákkal és

támogatottságokkal és egy másik $\mathcal{B}'$ bemenet. Általában a $\mathcal{B}'$-t valami apró módosítással kapjuk $\mathcal{B}$-ből. Feladat, hogy minél hatékonyabban találjuk meg a $\mathcal{B}'$-ben gyakori mintákat, azaz minél jobban használjuk fel a meglévő tudást (a $\mathcal{B}$-ben gyakori mintákat). Gondolhatunk itt egy on-line áruházra, ahol kezdetben rendelkezésünkre állnak az elmúlt havi vásárlásokhoz tartozó gyakori termékhalmazok, miközben folyamatosan érkeznek az új vásárlások adatai. Hasznos, ha az újonnan felbukkanó gyakori mintákat minél hamarabb felfedezzük, anélkül, hogy a bővített adatbázisban off-line módon lefuttatnánk egy gyakori mintákat kinyerő algoritmust.

5.3.3. Az algoritmusok jellemzői

Helyes vagy *helyesen működő* jelzővel illetjük azokat az algoritmusokat, amelyek nem hibáznak, tehát csak gyakori mintákat nyernek ki és azok támogatottságát jól határozzák meg. *Teljes* egy algoritmus, ha be lehet bizonyítani, hogy az összes gyakori mintát és támogatottságaikat meghatározza. Helyesen működő és teljes algoritmusokról fogunk beszélni, de szó lesz olyan algoritmusokról is, amelyekről csak azt tudjuk, hogy (bizonyos feltételezésekkel élve) kicsi annak a valószínűsége, hogy nem talál meg minden gyakori mintát.

„*Távol-keleti felmérések szerint azoknak az iskolásgyerekeknek, akik napi egy csésze cukor nélküli zöld teát isznak, feleannyi odvas foguk van, mint az átlagnak.*" Forrás: http://www.terebess.hu/szorolapok/zoldtea.html

Szélességi bejárást valósítanak meg azok az algoritmusok[6], amelyek a legkisebb mintákból kiindulva egyre nagyobb méretű gyakori mintákat nyernek ki. Egy ilyen algoritmusra igaz, hogy az ℓ-elemű gyakori mintákat hamarabb találja meg, mint az ℓ-nél nagyobb elemű mintákat. *Mélységi bejárást* megvalósító algoritmusokra ez nem igaz; ezek minél gyorsabban próbálnak eljutni a nem bővíthető mintához. Ha ez sikerül, akkor egy újabb, nem bővíthető mintát vesznek célba.

A következőkben ismertetjük a három legfontosabb gyakori mintákat kinyerő módszert az APRIORI-t, Zaki módszerét és a mintanövelő módszert. Ennek a három algoritmusnak a szerepe abban áll, hogy szinte az összes többi algoritmus ezeknek a továbbfejlesztése, vagy ezen algoritmusok keveréke. Jelentőségüket tovább növeli az a tény, hogy ezek a módszerek alkalmazhatóak akármilyen típusú mintákat keresünk, legyenek azok elemhalmazok, sorozatok vagy gráfok. Nem pontos algoritmusokat adunk, hanem csak egy általános módszerleírást. Egyes lépéseket csak a minta típusának ismeretében lehet pontosan megadni.

[6] A szélességi bejárást megvalósító algoritmusokat szintenként haladó (levelwise) algoritmusoknak is hívják.

5.3.4. Az APRIORI módszer

Az eredeti Apriori algoritmust gyakori elemhalmazok kinyerésére használták, és mint az AIS algoritmus [Agrawal és tsa., 1993] továbbfejlesztett változatát adták közre. Rakesh Agrawal-tól és Ramakrishnan Srikant-tól [Agrawal és Srikant, 1994] függetlenül szinte ugyanezt az algoritmust javasolta Heikki Mannila, Hannu Toivonen és A. Inkeri Verkamo [Mannila és tsa., 1994]. Az öt szerző végül egyesítette a két írást [Agrawal és tsa., 1996]. Kis módosítással az algoritmust gyakori sorozatok kinyerésére is (APRIORIALL, GSP algoritmusok), sőt, alapelvét különféle típusú gyakori minta (epizód, fa stb.) keresésénél alkalmazhatjuk.

Az algoritmus rendkívül egyszerű, mégis gyors és kicsi a memóriaigénye. Talán emiatt a mai napig ez az algoritmus a legelterjedtebb és legismertebb gyakori mintakinyerő algoritmus.

Az Apriori szélességi bejárást valósít meg. Ez azt jelenti, hogy a legkisebb mintából kiindulva szintenként halad előre a nagyobb méretű gyakori minták meghatározásához. A következő szinten (iterációban) az eggyel nagyobb méretű mintákkal foglalkozik.

Az algoritmusban központi szerepet töltenek be az ún. *jelöltek*. Jelöltnek hívjuk egy adott iterációban azt a mintát, amelynek támogatottságát meghatározzuk, azaz, aminek figyelmünket szenteljük. *Hamis jelölt*eknek hívjuk azokat a jelölteket, amelyekről ki fog derülni, hogy ritka minták, *elhanyagolt minták* pedig azok a gyakori minták, amelyeket nem választunk jelöltnek – nem foglalkozunk velük :-). Nyilvánvaló, hogy csak azokról a mintákról tudjuk eldönteni, hogy gyakoriak-e, amelyeknek meghatározzuk a támogatottságát, tehát amelyek jelöltek valamikor. Ezért elvárjuk az algoritmustól, hogy minden gyakori mintát felvegyen jelöltnek. A teljesség feltétele, hogy ne legyen elhanyagolt minta, a hatékonyság pedig annál jobb, minél kevesebb a hamis jelölt.

A jelöltek definiálásánál a részminta antimonotinitási tulajdonságot használjuk fel, ami így szólt: „Gyakori minta minden részmintája gyakori.”. Az állítást indirekten nézve elmondhatjuk, hogy egy minta biztosan nem gyakori, ha van ritka részmintája! Ennek alapján ne legyen jelölt az a minta, amelynek van ritka részmintája. Az APRIORI algoritmus ezért építkezik lentről. Egy adott iterációban pontosan tudjuk, hogy a részminták gyakoriak vagy sem! Az algoritmus onnan kapta a nevét, hogy az ℓ-elemű jelölteket a bemeneti adat ℓ-edik átolvasásának megkezdése előtt (a priori) állítja elő.

Az algoritmus pszeudokódja a következő ábrán látható. Kezdeti értékek beállítása után belépünk egy ciklusba. A ciklus akkor ér véget, ha az ℓ-elemű jelöltek halmaza üres. A cikluson belül először a `támogatottság_meghatározás` eljárást hívjuk meg, amely a jelöltek támogatottságát határozza meg. Ha ismerjük a jelöltek támogatottságát, akkor ki tudjuk választani belőlük a gyakoriakat. A `jelölt_előállítás` függvény az ℓ-elemű gyakori mintákból $(\ell+1)$-

elemű jelölteket állít elő.

Algoritmus Az Apriori módszer

Require: B: bementei adat
min_supp: támogatottsági küszöb

```
ℓ ⇐ 0
J_ℓ ⇐ { az üres minta } {J_ℓ: Az ℓ-elemű jelöltek}
while |J_ℓ| ≠ 0 do
  támogatottság_meghatározás(B, J_ℓ)
  for all j ∈ J_ℓ do
    if supp(j) ≥ min_supp then
      GY_ℓ ⇐ GY_ℓ ∪ {j}
    end if
  end for
  J_{ℓ+1} ⇐ jelölt_előállítás(GY_ℓ)
  GY ⇐ GY ∪ GY_ℓ
  ℓ ⇐ ℓ + 1
end while
return GY: gyakori minták
```

Az Apriori elvet adaptáló algoritmusok mind a fenti lépéseket követik. Természetesen a különböző típusú mintáknál különböző módon kell elvégezni a támogatottság-meghatározás, gyakoriak kiválogatása, jelöltek előállítása lépéseket.

Az algoritmus hatékonyságának egyik alapfeltétele, hogy a jelöltek elférjenek a memóriában. Ellenkező esetben ugyanis rengeteg idő menne el az olyan I/O műveletekkel, amelynek során a jelölteket a háttér és a memória között ide-oda másolgatjuk. A fenti pszeudokód az eredeti Apriori egyszerűsített változatát írja le. Valójában ugyanis addig állítjuk elő az ℓ-elemű jelölteket, amíg azok elférnek a memóriában. Ha elfogy a memória, akkor ℓ növelése nélkül folytatjuk az algoritmust, majd a következő iterációban ott folytatjuk a jelöltek előállítását, ahol abbahagytuk.

Jelöltek előállítása

Az ℓ-elemű jelöltek előállításának egyszerű módja az, hogy vesszük az összes ℓ-elemű mintát, és azokat választjuk jelöltnek, amelyekre teljesül, hogy minden részmintájuk gyakori. Szükségtelen az összes részmintát ellenőrizni, ugyanis a támogatottság anti-monotonitásából következik az, hogy ha az összes $(\ell-1)$-elemű részminta gyakori, akkor az összes valódi részminta is gyakori.

Ez a módszer azonban nem túl hatékony, vagy úgy is megfogalmazhatnánk, hogy túl sok felesleges munkát végez, túl sok olyan mintát vizsgál meg, amelyek biztosan nem gyakoriak. Hívjuk *potenciális jelölteknek* azon mintákat, amelyeket előállítunk, majd ellenőrizzük, hogy részmintáik gyakoriak-e. Ha egy potenciális minta átesik a teszten, akkor jelölté válik.

Tudjuk, hogy ha egy minta jelölt lesz, akkor minden $(\ell-1)$-elemű részmintája gyakori, tehát célszerű az $(\ell-1)$-elemű gyakori mintákból kiindulni. Egy egyszerű megoldás lenne, ha sorra vennénk az $(\ell-1)$-elemű gyakori minták minimális valódi felső korlátait, mint potenciális jelölteket. Még jobb megoldás, ha a $(\ell-1)$-elemű gyakori mintapároknak vesszük a minimális valódi felső korlátait. Ekkor ugyanis csak olyan potenciális jelöltet állítunk elő, amelynek van két $(\ell-1)$-elemű gyakori részmintája. A minimális valódi felső korlátot egy *illesztési művelettel* fogjuk előállítani. A két gyakori mintát a potenciális jelölt generátorainak hívjuk. Az illesztési műveletet a $\otimes$-el fogunk jelölni. Akkor illesztünk két mintát, ha van $(\ell-2)$-elemű közös részmintájuk. Ezt a részmintát *magnak* (core) fogjuk hívni.

Ha az előállítás módja olyan, hogy nem állíthatjuk elő ugyanazt a potenciális jelöltet két különböző módon, akkor ezt a jelölt-előállítást *ismétlés nélkülinek* nevezzük. Nézzünk egy példát. Legyenek a mintatér elemei elemhalmazok. Akkor állítsuk elő két $(\ell-1)$-elemű gyakori elemhalmaznak a minimális valódi korlátját, ha metszetük $(\ell-2)$-elemű. A minimális valódi korlátok halmaza csak egy elemet fog tartalmazni, a két halmaz unióját. Ez a jelölt-előállítás nem ismétlés nélküli, ugyanis például az $(\{A,B\},\{A,C\})$ párnak ugyanaz a legkisebb felső korlátja, mint az $(\{A,B\},\{B,C\})$ párnak.

Az ismétlés nélküli jelölt-előállítást mindig a minta elemein értelmezett teljes rendezés fogja garantálni, ami a $\preceq$ rendezés egy lineáris kiterjesztése lesz. A teljes rendezésnek megfelelően végigmegyünk az $(\ell-1)$-elemű gyakori mintákon és megnézzük, hogy mely sorban utána következő $(\ell-1)$-elemű gyakori mintával illeszthető, illetve az illesztésként kapott potenciális jelölt minden $(\ell-1)$-elemű részmintája gyakori-e. Sok esetben a ismétlés nélküliségnek elégséges feltétele az lesz, hogy a két gyakori minta $(\ell-2)$-elemű prefixeik megegyezzenek. A minta típusának ismeretében a teljességet (minden minimális valódi felső korlátbeli elemet előállítunk) és az ismétlés nélküliséget könnyű lesz bizonyítani.

Algoritmus `Jelöltek előállítása`

Require: $GY_{\ell-1}$: $(\ell-1)$-elemű gyakori minták

```
for all gy ∈ GY_{ℓ-1} do
  for all gy' ∈ GY_{ℓ-1}, gy ≼ gy' do
    if gy és gy' illeszthető then
      Ĵ ⇐ minimális_valódi_felső_korlát(gy, gy') {Ĵ: potenciális jelöltek halmaza}
      for all ĵ ∈ Ĵ do
        if minden_részhalmaz_gyakori(ĵ, GY_{ℓ-1}) then
          J_ℓ ⇐ ĵ
        end if
      end for
    end if
  end for
end for
return J_ℓ: ℓ-elemű jelöltek
```

Zárt minták kinyerése, az APRIORI-CLOSE algoritmus

A zárt minták jelentőségét az 5.3.2 részben már tárgyaltuk. Itt most két feladat megoldásával foglalkozunk. Megnézzük, hogy az összes gyakori mintából hogyan tudjuk előállítani a zártakat, illetve bemutatjuk az APRIORI-CLOSE [Pasquier és tsa., 1998, Pasquier és tsa., 1999a, Pasquier és tsa., 1999b] algoritmust, amely már eleve csak a zárt mintákat határozza meg. Mindkét módszerhez az alábbi észrevételt használjuk fel:

5.3.1 Észrevétel *Ha az m minta nem zárt, akkor van olyan m-et tartalmazó eggyel nagyobb méretű minta, amelynek támogatottsága megegyezik m támogatottságával.*

Tegyük fel, hogy a legnagyobb méretű gyakori minta mérete k. A GY_k elemei zártak. Egy egyszerű algoritmus menete a következő:

Nézzük sorban $GY_{k-1}, GY_{k-2}, \ldots, GY_0$ elemeit. Ha $m \in GY_\ell$-hez találunk olyan $m' \in GY_{\ell+1}$ elemet, amelynek támogatottsága megegyezik m támogatottságával, akkor m nem zárt. Ha nincs ilyen tulajdonságú m', akkor m zárt.

Az APRIORI-CLOSE menete teljes mértékben megegyezik az Apriori algoritmus menetével. Az egyetlen különbség, hogy az ℓ-elemű gyakori minták meghatározása után törli az $(\ell-1)$-elemű nem zártakat. Miután eldöntötte, hogy az ℓ-elemű m minta gyakori, megvizsgálja az összes $(\ell-1)$-elemű részmintáját m-nek. Amennyiben van olyan részhalmaz, aminek támogatottsága egyenlő m támogatottságával, akkor ez a részminta nem zárt, ellenkező esetben zárt.

5.3.5. Sorozat típusú bemenet

A legáltalánosabb eset leírásánál nem tettünk semmi megkötést a bemenet típusára és a támogatottsági függvényre vonatkozóan. Az esetek többsége azonban egy speciális családba tartozik. Ennek a problémacsaládnak a jellemzője, hogy a bemenet egy véges sorozat, és a támogatottságot azon elemek száma adja, amelyek valamilyen módon *illeszkednek* a mintára[7]. Az illeszkedést egy illeszkedési predikátummal adhatjuk meg, melynek értelmezési tartománya a mintatér.

$$bemenet : \mathcal{S} = \langle s_1, s_2, \ldots, s_n \rangle$$

A támogatottság definíciója megköveteli, hogy ha egy minta illeszkedik egy sorozatelemre, akkor minden részmintája is illeszkedjen. A legtöbb esetben a sorozat elemei megegyeznek a mintatér elemeivel és az m minta akkor illeszkedik egy sorozatelemre, ha annak m a részmintája.

A szakirodalomban igen elterjedt a sorozatok helyett a halmazokkal leírt bemenet, ahol minden egyes elem egyedi azonosítóval van ellátva. A jegyzetben a sorozatos leírást fogjuk használni, akinek ez szokatlan, az tekintse azonosítóknak a sorozat elemeinek sorszámát.

Az m minta *gyakoriságát* (jelölésben: $freq_{\mathcal{S}}(m)$, ami a frequency szóra utal) az m támogatottsága és az $\mathcal{S}$ hosszának hányadosával definiáljuk. A gyakorisági küszöböt ($\frac{min\_supp}{|\mathcal{S}|}$) következetesen min_freq-el jelöljük. Az értelmesen megválasztott gyakorisági küszöb mindig 0 és 1 között van. Az esetek többségében támogatottsági küszöb helyett gyakorisági küszöböt adnak meg.

Sorozat típusú bemenet esetén merül fel azon elvárás az algoritmusokkal szemben, hogy ne legyen érzékeny a bemenet *homogenitására*. Intuitíve akkor homogén egy bemenet, ha nincsenek olyan részei, amelyben valamely minta gyakorisága nagyon eltér a teljes bemenet alapján számított gyakoriságától. Sok alkalmazásban ez a feltétel nem áll fenn, így azokat az algoritmusokat kedveljük, amelyek hatékonysága független a bemenet homogenitásától. Könnyű átgondolni, hogy az Apriori algoritmus rendelkezik ezzel a tulajdonsággal.

Apriori

Amennyiben a támogatottságot illeszkedési predikátum alapján definiáljuk, akkor az Apriori algoritmus a bemeneti elemeken egyesével végigmegy és növeli azon jelöltek számlálóját, amelyek illeszkednek az éppen aktuális bemeneti elemre. Azonos bemeneti elemeknél ez a művelet ugyanazt fogja csinálni ezért

[7] Ha csak a matematikai definíciókat tekintjük, akkor törekedhettünk volna a legegyszerűbb leírásra és használhattunk volna sorozatok helyett multihalmazokat. A valóságban azonban a bemenet tényleg sorozatok formájában adott, így nem tehetjük fel, hogy az azonos bemeneti elemek össze vannak vonva.

célszerű az azonos bemeneti elemeket összegyűjteni és csak egyszer meghívni az eljárást. A bemenet azonban túl nagy lehet, ezért ezt gyorsítási lépést csak akkor szokás elvégezni, amikor már rendelkezésünkre állnak az egyelemű gyakori minták. Ezek alapján további szűréseket lehet végezni. Például elemhalmaz/elemsorozat típusú bemeneti elemeknél töröljük a halmazból/sorozatból a ritka elemeket. Ez duplán hasznos, hiszen csökkentjük a memóriafogyasztást és mivel az azonos szűrt elemek száma nagyobb lehet, mint az azonos elemek száma a támogatottságok meghatározása még kevesebb időbe fog telni.

Vannak olyan minták, amelyeknél a illeszkedés eldöntése drága művelet. Például gráf típusú mintáknál az illeszkedés meghatározásához egy részgráf izomorfia feladatot kell eldönteni, ami bizonyítottan NP-teljes. Ilyen mintáknál hasznos, ha minden jelöltnél rendelkezésünkre állnak azon bemeneti elemek sorszámai, amelyekre illeszkednek a generátorok (nevezzük ezt a halmazt illeszkedési halmaznak). Az illeszkedési predikátum anti-monoton tulajdonságából következik, hogy a jelölt csak azon bemeneti elemekre illeszkedhet, amelyekre generátoraik is illeszkednek. A támogatottság meghatározása során a jelöltek illeszkedési halmazát is meg kell határoznunk hiszen a jelöltek lesznek a generátorok a következő iterációban. Természetesen a generátorok illeszkedési listáit törölhetjük miután meghatároztuk a jelöltek illeszkedési listáit.

DIC – A DIC (Dynamic Itemset Counting) algoritmus [**?**] az Apriori továbbfejlesztése. Gyakori elemhalmazok kinyerésére javasolták, de minden olyan gyakori mintákat kereső feladatban alkalmazható, amelyben a bemenet sorozat típusú, és a támogatottságot illeszkedési predikátum alapján definiáljuk. Az algoritmus nem tisztán szélességi bejárást valósít meg; a különböző elemszámú minták együtt vannak jelen a jelöltek között. Ha k a legnagyobb gyakori minta mérete, akkor várhatóan $(k+)$-nél kevesebbszer, de legrosszabb esetben $(k+1)$-szer kell végigolvasni a bemenetet.

A DIC algoritmusban – szemben az APRIORI-val – nem válik szét az egyes iterációkban a jelöltek előállítása, a támogatottságok meghatározása és a ritka minták törlése. Miközben vesszük a bemeneti elemeket és határozzuk meg a jelöltek támogatottságát, új jelölteket vehetünk fel és törölhetünk (azaz dinamikus elemszámlálást alkalmazunk, ahogyan erre az algoritmus neve is utal). Akkor veszünk fel egy mintát a jelöltek közé, ha minden valódi részmintájáról kiderült, hogy gyakori. Akárhol veszünk fel egy jelöltet, egy iterációval később ugyanott, ahol felvettük, törölnünk kell a jelöltek közül, hiszen a pontos támogatottság meghatározásához a teljes bemenetet át kell néznünk. Ha a törölt jelölt gyakori, akkor természetesen a mintát felvesszük a gyakori minták halmazába. Minden jelöl esetén tárolnunk kell, hogy hányadik bemeneti elemnél lett jelölt. A kiindulási állapotban minden egyelemű minta jelölt és akkor ér véget az algoritmus, amikor nincs egyetlen jelölt sem.

Elemhalmazok példáját nézve, ha az A és B elemek olyan sokszor fordul-

nak elő, hogy támogatottságuk, már a bemenet egyharmadának átolvasása után eléri *min_supp*-ot, akkor az $\{A, B\}$ két elemű halmaz már ekkor jelölt lesz, és előfordulásait el kell kezdeni összeszámolni. A bemenet végigolvasása után ismét az első bemeneti elemre lépünk és az egyelemű jelöltek törlése után folytatjuk a jelöltek támogatottságának meghatározását. Az A, B jelöltet a bemenet egyharmadánál töröljük a jelöltek közül. Ha nincs más jelölt, akkor az algoritmus véget ér. Látható, hogy ekkor a DIC algoritmus 1+1/3-szor olvassa végig a bemenetet, amit az Apriori kétszer tesz meg.

Az algoritmus hátránya, hogy minden bemeneti elemnél meg kell vizsgálni, hogy vannak-e törlendő jelöltek. Ez költséges művelet ezért célszerű „állomásokat" létrehozni. Például minden ezredik bemeneti elem lehet egy állomás. Csak az állomásoknál nézzük meg, hogy egy jelölt támogatottsága elérte-e *min_supp*-ot, így csak állomásnál veszünk fel, illetve törlünk jelölteket.

A DIC algoritmus, szemben az APRIORI-val, érzékeny az adatok homogenitására. Amennyiben egy minta a felszállóhelyétől nagyon távol koncentrálva gyakori, akkor az összes, őt részmintaként tartalmazó minta is csak sokára lesz jelölt. Ekkor a DIC hatékonysága rosszabb az Apriori algoritmusénál, hiszen ugyanannyiszor járja végig a bemenetet, mint az APRIORI, de eközben olyan munkát is végez, amit az Apriori nem (minden állomásnál ellenőrzi, hogy mely jelölteket kell törölni). Összességében elmondhatjuk, hogy a DIC csak abban az esetben lesz gyorsabb az APRIORI-nál, ha a bemenet olyan nagy, hogy a futási időben nagy szerepet játszik a bemenet beolvasása. A mai memóriakapacitások mellett ez ritkán áll fenn.

A következőkben olyan algoritmusokat ismertetünk, amelyek sorozat típusú bemenet és illeszkedés alapú támogatottság esetén tudják meghatározni a gyakori mintákat.

Zaki módszere

Zaki módszere [Zaki és tsa., 1997] szintén jelölteket használ a keresési tér bejárásához, de a bejárás típusa – szemben az APRIORI-val – mélységi. A $\mathcal{MK} = (\mathcal{M}, \preceq)$ mintakörnyezet esetén csak akkor használható, ha tudjuk definiálni a $\preceq$-nek egy lineáris kiterjesztését, ugyanis az algoritmus építőelemei a prefixek.

A prefix alapján definiálhatunk egy ekvivalencia relációt. Adott ℓ esetén két minta ekvivalens, ha ℓ-elemű prefixük megegyezik. A P prefixű minták halmazát $[P]$-vel jelöljük. A prefixek segítségével a minták halmazát diszjunkt részekre osztjuk, azaz a feladatot kisebb részfeladatokra vezetjük vissza.

Nézzük például az elemhalmazok esetét. Legyen $\mathcal{I} = \{A, B, C, D\}$ és $\mathcal{M} = (2^{\mathcal{I}}, \subseteq)$, akkor $I' \prec' I''$, ha $|I'| < |I''|$ vagy, ha $|I'| = |I''|$ és I' lexikografikusan megelőzi I''-t. Például $\{D\} \prec' \{A, C\}$ és $\{A, B, D\} \prec' \{B, C, D\}$.

Amennyiben $\ell = 1$, akkor például a $\{A, B\}, \{A, C\}, \{A, D\}$ egy ekvivalencia osztályba tartozik, aminek például a {B,C} nem eleme.

A prefix mellett Zaki módszerének központi fogalma az *illeszkedési lista*. Egy mintához tartozó illeszkedési lista tárolja a minta illeszkedéseit. Az illeszkedési lista két fontos tulajdonsággal bír:

1. Az illeszkedési listából könnyen megkapható a támogatottság.

2. Egy jelölt illeszkedési listája megkapható a generátorainak illeszkedési listáiból.

Például elemhalmaz típusú minták esetében (ha az illeszkedést a tartalmazási reláció alapján definiáljuk) egy elemhalmaz illeszkedési listája egy olyan lista lesz, amely a bemeneti sorozat azon elemeinek sorszámát tárolja, amelyeknek része az adott elemhalmaz. Például

$$\langle \{A, D\}, \{A, C\}, \{A, B, C, D\}, \{B\}, \{A, D\}, \{A, B, D\}, \{D\} \rangle$$

bemenet esetén az $\{A, C\}$ illeszkedési listája: $\langle \{1, 2\} \rangle$.

Zaki algoritmusának pszeudokódja az alábbi.

Algoritmus `Zaki módszere`

Require: B: bementei adat
min_supp: támogatottsági küszöb

$J \Leftarrow 1$ elemű minták halmaza$\{J$: jelöltek$\}$
$ILL(J) \Leftarrow$ `ILL_felépítés`(B, J) $\{ILL(J)$: jelöltek illeszkedési listája$\}$
for all $j \in J$ **do**
  if $|ILL(j)| \geq min\_supp$ **then**
    $GY_1 \Leftarrow GY_1 \cup j$
  end if
end for
`zaki_segéd`$(GY, ILL(GY), min\_supp)$ $\{GY = [0]^+\}$
return GY: gyakori minták

Először felépítjük az egyelemű minták illeszkedési listáit. Ezek alapján meghatározzuk a gyakori mintákat. A későbbiekben nem használjuk a bemenetet csak az illeszkedési listákat, ezekből ugyanis a támogatottságok egyértelműen meghatározhatók. Az algoritmus lényege a `zaki_segéd` rekurziós eljárás, amelynek pszeudokódja alább látható.

Algoritmus `zaki_segéd eljárás`

Require: $[P]^+$: P prefixű, P-nél eggyel nagyobb gyakori minták
$ILL[P]^+$: $[P]^+$-beli minták illeszkedési listája
min_supp: támogatottsági küszöb

for all $m \in [P]^+$ **do**
  for all $m' \in [P]^+, m \preceq m'$ **do**
    $J, ILL(J) \Leftarrow$ `minimális_valódi_felső_korlát`$(m, m', ILL(m, m'))$
    $\{J$: jelöltek,$ILL(J)$: jelöltek illeszkedési listája$\}$
    for all $j \in J$ **do**
      if $|ILL(j)| \geq min\_supp$ **then**
        $GY' \Leftarrow GY' \cup \{j\}$
      end if
    end for
  end for
  `zaki_segéd`$(GY', ILL(GY'), min\_supp)$
  $GY \Leftarrow GY \cup GY'$
end for
return GY: P prefixű összes gyakori minta

A Zaki féle jelölt előállításnak két feladata van. Természetesen az egyik a jelöltek előállítása, de emellett az illeszkedési listákat is előálltja. A jelölt-előállítás megegyezik az Apriori jelölt előállításának első lépésével (potenciális jelöltek előállítása). A második lépést nem is tudnánk elvégezni, ugyanis nem áll rendelkezésünkre az összes részminta, így nem is tudjuk ellenőrizni, hogy az összes részminta gyakori-e.

Nézzünk erre egy gyors példát. Amennyiben a mintákat elemhalmazok formájában keressük, akkor az Apriori és Zaki módszere is először meghatározza a gyakori elemeket. Legyenek ezek az A, C, D, G, M elemek. Az Apriori ezek után előállítana $\binom{5}{2}$ darab jelöltet, majd meghatározná támogatottságaikat. Zaki ehelyett csak az A prefixű kételemű halmazok támogatottságát vizsgálja. Ha ezek közül gyakori például az $\{A, C\}$, $\{A, G\}$, akkor a következőkben az {A,C,G}-t nézi, és mivel további jelöltet nem tud előállítani, ugrik a C prefixű elemhalmazok vizsgálatára, és így tovább.

Látnunk kell, hogy Zaki módszere csak több jelöltet állíthat elő, mint az APRIORI. A mélységi bejárás miatt ugyanis egy jelölt előállításánál nem áll rendelkezésünkre az összes részminta. Az előző példa esetében például az {A,C,G} támogatottságát hamarabb vizsgálja, mint a {C,G} halmazét, holott ez utóbbi akár ritka is lehet. Ebben a tekintetben tehát Zaki módszere rosszabb az APRIORI-nál, ugyanis több hamis jelöltet állít elő.

Zaki módszerének igazi ereje a jelöltek támogatottságának meghatározásában van. A minták illeszkedési listáinak előállítása egy rendkívül egyszerű és nagyon gyors művelet lesz. Emellett ahogy haladunk egyre mélyebbre a mélységi bejárás során, úgy csökken az illeszkedési listák hossza, és ezzel a

támogatottság meghatározásának ideje is.

A bemenet szűrésének ötletét az APRIORI algoritmusnál is elsüthetjük, de nem ilyen mértékben. Ha ismerjük a gyakori egyelemű mintákat, akkor törölhetjük azon sorozatelemeket, amelyek nem illeszkednek egyetlen gyakori egyelemű mintára sem. Sőt ezt a gondolatot általánosíthatjuk is: az ℓ-edik lépésben törölhetjük a bemeneti sorozat azon elemeit, amelyek nem illeszkednek egyetlen $(\ell-1)$-elemű mintára sem. Ez a fajta bemeneti tér szűkítés azonban nem lesz olyan hatékony, mint amilyen a Zaki módszerében. Ott ugyanis egyszerre csak 1 prefixet vizsgálunk, az APRIORI-nál azonban általában sok olyan minta van, aminek csak az üres minta a közös részmintája.

Összességében tehát az APRIORI kevesebb jelöltet generál, mint Zaki módszere, de a jelöltek támogatottságának meghatározása több időt vesz igénybe. Általánosságban nem lehet megmondani, hogy melyik a jobb módszer. Egyes adatbázisok esetén az APRIORI, másoknál a Zaki módszer. Sőt könnyen lehet olyan példát mutatni, amikor az egyik algoritmus nagyságrendileg több idő tölt a feladat megoldásával, mint a másik.

Zaki módszerénél könnyű kezelni a anti-monoton és a prefix anti-monoton kényszereket. A nem gyakori minták törlésekor töröljük azokat a mintákat is, amelyek nem elégítenek ki minden anti-monoton kényszert. A prefix anti-monoton kényszereket a jelöltek előállítása után kell figyelembe vennünk: törölhetjük azokat a generátorokat, amelyekre nem teljesül az anti-monoton kényszer. A `zaki_segéd` eljárásból következik, hogy ilyen m mintát legfeljebb olyan jelölt előállításánál fogunk felhasználni, aminek m a prefixe. Természetesen itt is bajban vagyunk, ha több prefix anti-monoton kényszer van adva, hiszen ezek $\prec$-nek különböző lineáris kiterjesztéseit használhatják.

Mintanövelő algoritmusok

A mintanövelő (pattern growth) algoritmus olyan mintakeresés esetén alkalmazható, amikor a bemenet minták sorozataként van megadva, és az illeszkedést a tartalmazás alapján definiáljuk, értelmezhető a prefix, és a minták *egyértelműen növelhetők*. Például a növelés művelet halmazok esetén az unió, sorozatok esetében a konkatenáció képzésének felel meg (és ebből látszik, hogy a növelés művelete nem feltétlenül kommutatív).

5.3.15. Definíció *Az $\mathcal{MK} = (\mathcal{M}, \preceq)$ mintakörnyezet mintái* egyértelműen növelhetők*, ha létezik egy olyan + „növelő” művelet, amellyel az $\mathcal{M}$ félcsoportot alkot.*

A növelés inverze a *csökkentés*, jelölése: -. Az $m - m'$ művelet eredménye az az m'' minta, amivel m'-t növelve m-et kapjuk.

A mintanövelő módszerek csak egyelemű jelölteket használnak, és emellett a bemeneten végeznek olyan műveleteket, amelyek eredményeként megkapjuk a gyakori mintákat. A két művelet a *szűrés* és a *vetítés*, amelyek az eredeti $\mathcal{S}$ bemenetből egy „kisebb" $\mathcal{S}'$ bemenetet állítanak elő. A szűrés a gyakori egyelemű mintákat használja és olyan $\mathcal{S}'$ bemenetet állít elő amelyben a gyakori minták megegyeznek az $\mathcal{S}$-beli gyakori mintákkal. Az $\mathcal{S}$ bemenet m mintára vetítése (jelölésben $\mathcal{S}|m$) pedig olyan $\mathcal{S}'$ bemenetet állít elő, amelyre igaz, hogy ha m-et az $\mathcal{S}'$-beli gyakori mintákkal növeljük, akkor megkapjuk az $\mathcal{S}$-beli, m-et tartalmazó gyakori mintákat. A m-et tartalmazó gyakori minták meghatározásához csak azokra a bemeneti elemekre van szükség, amelyekre illeszkedik m, ezért a vetítés első lépése mindig ezen elemek meghatározása lesz.

Ha például a bemenet elemei elemhalmazok és akkor illeszkedik egy elemhalmaz a bemenet egy elemére, ha annak része, akkor szűrés művelet az lesz, hogy a bemeneti elemekből töröljük a ritka elemeket. Nyilvánvaló, hogy ritka elem nem játszik szerepet a gyakori elemek meghatározásában. A bemenet X halmazra vetítését megkapjuk, ha töröljük azon bemeneti elemeket, amelyeknek nem része X, majd a kapott elemekből töröljük X-et. Legyen

$$\mathcal{S} = \langle \{A, C, F\}, \{B, G\}, \{A, C, D\}, \{A, C\}, \{B, C\}, \{C, D, E\}, \{A, B, C\} \rangle$$

amelynek szűrése 2-es támogatottsági küszöb esetén az

$$\tilde{\mathcal{S}} = \langle \{A, C\}, \{B\}, \{A, C, D\}, \{A, C\}, \{B, C\}, \{C, D\}, \{A, B, C\} \rangle$$

sorozat és $\tilde{\mathcal{S}}|\{A, C\} = \langle \{D\}, \{B\} \rangle$.

A mintanövelő módszer rendkívül egyszerű, tulajdonképpen a feladatot rekurzívan kisebb részfeladat megoldására vezeti vissza. A rekurziós eljárást a bemenet szűrésével és különböző mintákra vett vetítéseivel hívja meg, miközben a mintateret is csökkenti. Jelöljük $\mathcal{M} \setminus \bar{m}$-el azt a mintateret, amit úgy kapunk $\mathcal{M}$-ből, hogy töröljük azon mintákat, amelynek m részmintája ($\bar{m}$). Ha az m minta támogatottsága $\mathcal{S}$-ben $supp_{\mathcal{S}}(m)$ és az $m' \in \mathcal{M} \setminus \bar{m}$ támogatottsága $\mathcal{S}|m$-ben $supp_{\mathcal{S}|m}(m')$, akkor $m + m'$ támogatottsága is $supp_{\mathcal{S}|m}(m')$. A módszer pszeudokódja alább látható.

Algoritmus `Mintanövelő módszer`

Require: B: bemeneti adat
min_supp: támogatottsági küszöb
$\mathcal{M}$: mintatér

$J_1 \Leftarrow$1-elemű minták $\{J_1$: egyelemű jelöltek$\}$
`támogatottság_meghatározás`(B, J_1)
$GY_1 \Leftarrow$ `gyakoriak_kiválogatása`(J_1, min_supp)
$\tilde{B} \Leftarrow$ `szűrés`(B)

for all $gy \in GY^1$ **do**
  $GY' \Leftarrow$ `mintanövelő_módszer`$(\tilde{B}|gy, min\_supp, \mathcal{M} \setminus \tilde{gy})$
  for all $gy' \in GY'$ **do**
    $GY \Leftarrow GY \cup \{gy + gy'\}$
  end for
end for
return GY: gyakori minták

A módszer előnye abban rejlik, hogy szűrést, vetítést és az egyelemű jelöltek támogatottságát hatékonyan tudjuk megvalósítani. A hatékonyság növelése érdekében a vetített tranzakciók azonos elemeit csak egyszer tároljuk, általában egy fa-szerű struktúrában.

Az anti-monoton kényszerek kezelése a mintanövelő algoritmusok esetében is egyszerű. Ne folytassuk a rekurziót, ha a minta nem elégít ki minden anti-monoton kényszert.

Az egyes mintatípusok esetében úgy fogjuk megadni a növelés műveletet, hogy tetszőleges minta csökkentése a minta prefixét fogja adni. Ez azt eredményezi, hogy törölhetjük azt a mintát, amelyik nem elégíti ki a prefix anti-monoton kényszert, és leállhatunk a rekurzióval. Hasonlóan az Apriori és a Zaki módszeréhez itt sincs mód több prefix anti-monoton kényszer hatékony kezelésére. Az algoritmus menetét ugyanis egyértelműen megadja a növelés művelet, amit a prefix anti-monoton kényszerben felhasznált teljes rendezés alapján definiálunk.

Kétlépcsős technikák

A szélességi bejárást megvalósító algoritmusok az adatbázist legalább annyiszor olvassák végig, amekkora a legnagyobb gyakori minta mérete. Előfordulhatnak olyan alkalmazások, amelyeknél az adatbázis elérése drága művelet. Ilyenre lehet példa, amikor az adatbázis egy elosztott hálózatban található, vagy lassú elérésű háttértárolón.

A kétlépcsős algoritmusok [Savasere, és tsa., 1995, Toivonen, 1996] a teljes adatbázist legfeljebb kétszer olvassák végig. I/O tekintetében tehát legyőzik

például az APRIORI algoritmust, azonban olyan futási környezetben, ahol a futási időt nem szinte kizárólag az I/O műveletek határozzák meg (ha a bemenet elfér a memóriában akkor ez a helyzet áll fenn), az APRIORI algoritmus gyorsabban ad eredményt.

Naiv mintavételező algoritmus – Olvassuk be a teljes bemenet egy részét a memóriába (a rész nagyságára nézve lásd 235.oldal). Erre a kis részre futtassuk le az APRIORI algoritmust az eredeti min_freq gyakorisági küszöbbel. A kis részben megtalált gyakori minták lesznek a jelöltek a második fázisban, amelynek során a jelöltek támogatottságát a teljes adatbázisban meghatározzuk. Ezáltal ki tudjuk szűrni azokat a mintákat, amelyek ritkák, de a kis részben gyakoriak. Előfordulhat azonban a fordított helyzet, azaz a kis adatbázisban egy minta ritka, viszont globálisan gyakori, tehát nem kerül a jelöltek közé, és így nem is találhatjuk azt gyakorinak. Javíthatunk a helyzeten, ha csökkentjük a kis részben a gyakorisági küszöböt, amivel növeljük a jelöltek számát, de csökkentjük annak veszélyét, hogy egy gyakori mintát ritkának találunk.

Ennek az egyszerű algoritmusnak két hátránya van. Egyrészt nem ad arra garanciát, hogy minden gyakori mintát megtalálunk (azaz nem teljes), másrészt a gyakorisági korlát csökkentése miatt a hamis jelöltek száma túlzottan nagy lehet. A fenti két problémát küszöböli ki a partíciós, illetve a Toivonen-féle algoritmus.

Mivel a kétlépcsős algoritmusok egy kis rész kiválasztásán alapulnak, így nagyon érzékenyek az adatbázis homogenitására. Gondoljunk itt a szezonális elemekre, amelyek lokálisan gyakoriak, de globálisan ritkák. Például a kesztyűk eladása tél elején nagy, de mégis a kesztyű önmagában ritka elem. Amennyiben a kis rész kiválasztása a bemenet egy véletlen pontjáról történő szekvenciális olvasást jelentene, akkor az nagy eséllyel sok hamis és hiányzó jelöltet eredményezne.

Partíciós algoritmus – A partíciós algoritmus [Savasere, és tsa., 1995] kétszer olvassa végig a teljes adatbázist. Páronként diszjunkt részekre osztja a bemenetet ($\mathcal{S} = \langle \mathcal{S}_1, \mathcal{S}_2 \ldots, \mathcal{S}_r \rangle$), majd az egyes részekre meghívja az APRIORI algoritmust, ami megadja az egyes részekben gyakori mintákat (hívjuk őket lokálisan gyakori mintáknak). A második végigolvasásnál egy minta akkor lesz jelölt, ha valamelyik részben gyakori volt. Könnyen látható, hogy az algoritmus teljes, hiszen egy gyakori mintának legalább egy részben gyakorinak kell lennie, és ezt az APRIORI ki fogja szűrni (mivel az APRIORI is teljes).

Kérdés, hogy hány részre osszuk a teljes adatbázist. Nyilvánvaló, hogy minél nagyobb az egyes részhalmazok mérete, annál jobb képet ad a teljes adatbázisról, tehát annál kevesebb lesz a hamis jelölt. A részek nagy mérete azonban

azt eredményezi, hogy azok nem férnek el a memóriában, és így az APRIORI algoritmus sok időt tölt el partíciórészek ideiglenes háttérbe másolásával és visszaolvasásával. Habár globálisan csak kétszer olvassuk végig a teljes adatbázist, azonban az egyes partíciók I/O igényének összege legalább akkora, mintha a teljes adatbázisra futtatnánk le az APRIORI algoritmust. Végeredményben a második végigolvasás miatt a partíciós algoritmus I/O igénye nagyobb lesz, mint az APRORI algoritmusé.

Ha az egyes részek elférnek a memóriában, akkor nem lép fel a fenti probléma, hisz az APRIORI algoritmus nem fog I/O műveletet igényelni (feltéve, ha a jelöltek a számlálóikkal együtt is elférnek még a memóriában). Túl kis méret választása azonban azt eredményezheti, hogy a partíció nem ad hű képet a teljes adatbázisról, így a lokális gyakori minták mások (is!) lesznek, mint a globális gyakori minták, ami túl sok hamis jelöltet eredményezhet.

A helyes partícióméret tehát a rendelkezésünkre álló memóriától függ. Legyen minél nagyobb, de úgy, hogy a jelöltek számlálóikkal együtt is elférjenek a memóriában. Természetesen a jelöltek száma a gyakori minták méretétől függ, amiről a partícióméret meghatározásakor még nincs pontos képünk.

A partíciós algoritmus szintén érzékeny a bemenet homogenitására. Ezt az érzékenységet csökkenthetjük, ha módosítjuk egy kicsit az algoritmust. Ha egy m minta gyakori az $\mathcal{S}_i$ részben, akkor a rákövetkező $\mathcal{S}_{i+1}, \mathcal{S}_{i+2}, \ldots \mathcal{S}_{i+\ell}$ részekben is határozzuk meg a támogatottságát egészen addig, amíg $freq_{\cup_{j=i}^{i+\ell}\mathcal{S}_j}(m) \geq min\_freq$. Ha ezalatt eljutunk az utolsó részig, akkor vegyük fel m-et a második végigolvasás jelöltjei közé. Ellenkező esetben felejtsük el, hogy m gyakori volt ezen részekben. Ha egy mintát az összes részben vizsgáltunk, akkor ezt szintén szükségtelen felvenni jelöltnek a második végigolvasásnál, hiszen támogatottsága megegyezik az egyes résztámogatottságok összegével.

A partíciós algoritmus további előnye, hogy remekül párhuzamosítható. Saját memóriával rendelkező feldolgozó egységek végezhetik az egyes részek gyakori mintakeresését, és ezáltal mind az első, mind a második fázis töredék idő alatt elvégezhető.

Toivonen algoritmusa – Az naív mintavételező algoritmus nagy hátránya, hogy még csökkentett min_freq mellett sem lehetünk biztosak abban, hogy nem vesztettünk el gyakori mintát. Toivonen algoritmusa [Toivonen, 1996] az adatbázist egyszer olvassa végig, és ha jelenti, hogy minden mintát megtalál, akkor bizonyítható, hogy ez igaz. Az algoritmus nem más, mint a naív mintavételező algoritmus továbbfejlesztett változata. Az egyszerű algoritmusnál azonban több információt ad, ugyanis jelenti, ha biztos abban, hogy minden gyakori mintát előállított, és azt is jelenti, amikor lehetséges, hogy van hiányzó jelölt (olyan gyakori minta, ami nem jelölt, és így nem találhatjuk azt

gyakorinak). A lehetséges hiányzó jelöltekről információt is közöl.

Alapötlete az, hogy ne csak a kis részben található gyakori minták előfordulását számoljuk össze a teljes adatbázisban, hanem azok minimális valódi felső korlátait is. Mit jelent az, hogy az m minta tetszőleges $M \subseteq \mathcal{M}$ mintahalmaz minimális valódi felső korlátai közé tartozik (jelölésben $m \in MVFK(M)$)? Először is a valódi felső korlát formálisan: $m' \prec m$ minden $m' \in M$. A minimalitás pedig azt jelenti, hogy nem létezik olyan m'' minta, amely M-nek valódi felső korlátja és $m'' \prec m$. A gyakori minták minimális valódi felső korlátjai azok a ritka minták, amelyek minden részmintája gyakori.

Például elemhalmaz típusú minta esetén, ha $\mathcal{M} = 2^{\{A,B,C,D,E,F\}}$ és

$$M = \{\{A\}, \{B\}, \{C\}, \{F\}, \{A,B\}, \{A,C\}, \{A,F\}, \{C,F\}, \{A,C,F\}\}$$

, akkor

$$MVFK(M) = \{\{B,C\}, \{B,F\}, \{D\}, \{E\}\}.$$

Toivonen algoritmusában a teljes adatbázisból egy kis részt veszünk. Ebben meghatározzuk a gyakori minták halmazát és ennek minimális valódi felső korlátját. A teljes adatbázisban ezek támogatottságát vizsgáljuk, és gyűjtjük ki a globálisan gyakoriakat. A következő egyszerű tétel ad információt arról, hogy ez az algoritmus mikor teljes, azaz mikor lehetünk biztosak abban, hogy minden gyakori mintát meghatároztunk.

5.3.16. Tétel *Legyen $\mathcal{S}'$ az $\mathcal{S}$ bemeneti sorozat egy része. Jelöljük GY-vel az $\mathcal{S}$-ben, GY'-vel az $\mathcal{S}'$-ben gyakori mintákat és GY^*-al azokat az $\mathcal{S}$-ben gyakori mintákat, amelyek benne vannak $GY' \cup MVFK(GY')$-ben ($GY^* = GY \cap (GY' \cup MVFK(GY'))$). Amennyiben*

$$GY^* \cup MVFK(GY^*) \subseteq GY' \cup MVFK(GY')$$

teljesül, akkor $\mathcal{S}$-ben a gyakori minták halmaza pontosan a GY^*, tehát $GY^* \equiv GY$.

Bizonyítás – Indirekt tegyük fel, hogy létezik $m \in GY$, de $m \notin GY^*$, és a feltétel teljesül. A GY^* definíciója miatt ekkor $m \notin GY' \cup MVFK(GY')$. Vizsgáljuk azt a legkisebb méretű $m' \preceq m$-t, amire $m' \in GY$ és $m' \notin GY^*$ (ilyen m'-nek kell lennie, ha más nem, ez maga az m minta). Az m' minimalitásából következik, hogy minden valódi részmintája eleme $GY' \cup MVFK(GY')$-nek és gyakori. Ebből következik, hogy m' minden részmintája eleme GY^*-nak, amiből kapjuk, hogy $m' \in MVFK(GY^*)$. Ez ellentmondást jelent, hiszen a feltételnek teljesülnie kell, azonban van olyan elem (m'), amely eleme a bal oldalnak, de nem eleme a jobb oldalnak.

Tetszőleges GY' halmaz esetén az $MVFK(GY') \cup GY'$-t könnyű előállítani. Sőt, amennyiben a gyakori mintákat Apriori algoritmussal határozzuk meg, akkor $MVFK(GY')$ elemei pontosan a ritka jelöltek lesznek (hiszen a jelölt minden része gyakori).

Nézzünk egy példát Toivonen algoritmusára. Legyen a mintatér {A,B,C,D} hatványhalmaza. A kis részben az {A},{B},{C} elemhalmazok gyakoriak. Ekkor a minimális valódi felső korlát elemei az {A,B},{A,C},{B,C},{D} halmazok. Tehát ennek a 7 elemhalmaznak fogjuk a támogatottságát meghatározni a teljes adatbázisban. Ha például az {A},{B},{C} {A,B} halmazokat találjuk gyakorinak a teljes adatbázisban, akkor a tételbeli tartalmazási reláció fennáll, hiszen az {A},{B},{C},{A,B} halmaz minimális valódi felső korlátai közül mind szerepel a 7 jelölt között. Nem mondható ez, ha {D}-ről derül ki, hogy gyakori. Ekkor Toivonen algoritmusa jelenti, hogy előfordulhat, hogy nem biztos, hogy minden gyakori elemhalmazt megtalált. Az esetleg kimaradtak csak (!) az {A,D},{B,D},{C,D} halmazok lehetnek.

A zárt minták „törékenysége”

Tagadhatatlan, hogy a zárt mintákon alapuló memóriacsökkentés egy szép elméleti eredmény. Ne foglaljunk helyet a memóriában a gyakori, nem zárt mintáknak, hiszen a zárt, gyakori mintákból az összes gyakori minta meghatározható.

Ez a technika ritkán alkalmazható azon esetekben, amikor a bemenet sorozat formájában adott, a támogatottságot pedig egy illeszkedési predikátum alapján definiáljuk. És, mint azt már említettük, a legtöbbször ez áll fenn.

Ennek oka, hogy gyakori mintákat általában nagy, zajokkal terhelt adatbázisokban keresnek. Ilyen adatbázisban szinte az összes elemhalmaz zárt, így a módszerrel nem nyerünk semmit. Gondoljuk meg, hogy ha egy adatbázist úgy terhelünk zajjal, hogy véletlenszerűen beszúrunk egy-egy új elemet, akkor folyamatosan növekszik az esélye annak, hogy egy minta zárt lesz. A nemzártság tehát egy „sérülékeny” tulajdonság. Tetszőleges nem zárt m mintát zárttá tehetünk egyetlen olyan tranzakció hozzáadásával, amely illeszkedik m-re, de nem illeszkedik egyetlen olyan mintára sem, amelynek m valódi részmintája.

Dinamikus gyakori mintabányászat

Nagy adatbázisok esetén a gyakori minták kinyerése még a leggyorsabb algoritmusokat felhasználva is lassú művelet. Az adatbázisok többségében a tárolt adatok nem állandóak, hanem változnak: új elemeket veszünk fel, egyeseket módosítunk, vagy törlünk. Ha azt szeretnénk, hogy a kinyert gyakori minták konzisztensek legyenek az adatbázisban tárolt adatokkal, akkor bizonyos

időközönként a gyakori minták adatbázisát is frissíteni kell.

A konzisztenciát elérhetjük úgy, hogy lefuttatjuk valamelyik ismert (APRIORI, Zaki stb.) algoritmust minden módosítás után. Ennek az a hátránya, hogy lassú, hiszen semmilyen eddig kinyert tudást nem használ fel. Szükség van tehát olyan algoritmusok kifejlesztésére, ami felhasználja az adatbázis előző állapotára vonatkozó információkat és így gyorsabban ad eredményt, mint egy nulláról induló, hagyományos algoritmus [Cheung és tsa., 1996, Cheung és tsa., 1997, Omiecinski és Savasere, 1998, Sarda és Srinivas, 1998],
[Thomas és tsa., 1997, Ayan, 1999].

Itt most azt az esetet nézzük, amikor csak bővíthetjük a bemenetet, de a leírt módszerek könnyen általánosíthatók arra az esetre, amikor törölhetünk is a bemenetből. Adott tehát $\mathcal{S}$ bemeneti sorozat, amelyben ismerjük a gyakori mintákat ($GY^{\mathcal{S}}$) és azok támogatottságát. Ezen kívül adott az új bemeneti elemek sorozata $\mathcal{S}'$. A feladat a $\langle \mathcal{S}, \mathcal{S}' \rangle$-ben található gyakori minták ($GY^{\langle \mathcal{S}, \mathcal{S}' \rangle}$) és azok támogatottságának meghatározása.

FUP algoritmus – A FUP (Fast Update) [Cheung és tsa., 1996] a legegyszerűbb szabály- karbantartó algoritmus. Tulajdonképpen nem más, mint az APRIORI algoritmus módosítása.

Kétféle jelöltet különböztetünk meg: az első csoportba azok a minták tartoznak, melyek az eredeti adatbázisban gyakoriak voltak, a másodikba azok, amelyek nem. Nyilvánvaló, hogy az új adatbázisban mindkét csoport elemeinek támogatottságát meg kell határozni, a régi adatbázisban azonban elég a második csoport elemeit vizsgálni.

A FUP az alábbi trivialitásokat használja fel.

1. Ha egy minta $\mathcal{S}$-ban gyakori volt és $\mathcal{S}'$-ben is az, akkor az $\langle \mathcal{S}, \mathcal{S}' \rangle$-ben is biztos gyakori, előfordulása megegyezik $\mathcal{S}'$-beni és $\mathcal{S}$-beni előfordulások összegével.

2. Amennyiben egy elemhalmaz $\mathcal{S}$-ban ritka, akkor $\langle \mathcal{S}, \mathcal{S}' \rangle$-ben csak abban az esetben lehet gyakori, ha $\mathcal{S}'$-ben gyakori. Ezek szerint ne legyen jelölt olyan elemhalmaz, amely sem $\mathcal{S}$-ban, sem $\mathcal{S}'$-ben nem gyakori.

Ezekből következik, hogy csak olyan elemhalmazok lesznek jelöltek $\mathcal{S}$ végigolvasásánál, amelyek $GY^{\mathcal{S}}$-ban nem szerepeltek, de $\mathcal{S}'$-ben gyakoriak voltak.

Az algoritmus pszeudokódja alább látható.

Algoritmus `FUP algoritmus`

Require: S: régi bemeneti adat
S': új bemeneti adat
GY^S: régi gyakori minták
min_freq: gyakorisági küszöb

$\ell \Leftarrow 0$
$J_\ell^1 \Leftarrow GY_\ell^S$ {J_ℓ^1: 1-es csoportbeli jelöltek}
$J_\ell^2 \Leftarrow \{\text{üres minta}\} \setminus GY_\ell^S$ {J_ℓ^2: 2-es csoportbeli jelöltek}
while $|J_\ell^1| + |J_\ell^2| \neq 0$ **do**
  `támogatottság_meghatározás`$(S', J_\ell^1 \cup J_\ell^2)$
  $J_\ell^* \Leftarrow$ `gyakoriak_kiválogatása`(J_ℓ^2, min_freq)
  if $|J_\ell^*| \neq 0$ **then**
    `támogatottság_meghatározás`(S, J_ℓ^*)
  end if
  $GY_\ell \Leftarrow$ `gyakoriak_kiválogatása`$(J_\ell^1 \cup J_\ell^*, min\_freq)$
  $J_{\ell+1}^{**} \Leftarrow$ `jelölt_előállítás`(GY_ℓ)
  $\ell \Leftarrow \ell + 1$
  $J_\ell^1 \Leftarrow J_\ell^{**} \cap GY_\ell^S$
  $J_\ell^2 \Leftarrow J_\ell^{**} \setminus GY_\ell^S$
  `delete`(J_ℓ^{**})
end while
return $GY^{\langle S,S' \rangle}$: gyakori minták

A támogatottság meghatározás, gyakoriak kiválogatása és a jelölt-előállítás lépések teljes egészében megegyeznek a Apriori ezen lépéseivel.

A FUP algoritmust könnyű módosítani arra az esetre, amikor nem csak hozzáadunk új elemeket az eredeti bemeneti sorozathoz, hanem törlünk is néhányat a régi elemek közül (FUP2 algoritmus [Cheung és tsa., 1997]).

A FUP és FUP_2 algoritmusok nem mentesek az Apriori algoritmus legfontosabb hátrányától, attól, hogy a teljes adatbázist annyiszor kell átolvasni, amekkora a legnagyobb gyakori jelöltminta mérete. Ezen a problémán próbáltak segíteni a később publikált algoritmusok.

Esélyes jelölteken alapuló dinamikus algoritmus – Thomas és szerzőtársai Toivonen algoritmusában használt minimális valódi felső korlátokat használják annak érdekében, hogy csökkentsék a nagy adatbázist átolvasásának számát [Thomas és tsa., 1997]. Az adatbázis növekedése során először a minimális valódi felső korlátok válnak gyakorivá. Ha nem csak a gyakori minták előfordulását ismerjük a régi adatbázisban, hanem azok minimális valódi felső korlátait is, akkor lehet, hogy szükségtelen a régi adatbázist végigolvasni. Ha ugyanis az új tranzakciók felvételével egyetlen minimális valódi felső korlát sem válik gyakorivá, akkor biztos, hogy nem keletkezett új gyakori minta. Az 5.3.16.-as tétel ennél erősebb állítást fogalmaz meg: még ha bizonyos minimális valódi felső korlátok gyakorivá váltak, akkor is biztosak

lehetünk abban, hogy nem kell a régi adatbázist átvizsgálnunk, mert nem keletkezhetett új gyakori minta. Átültetve a tételt a jelenlegi környezetbe: ha $GY^{\mathcal{S}\cup\mathcal{S}'} \cup MVFK(GY^{\mathcal{S}\cup\mathcal{S}'}) \subseteq GY^{\mathcal{S}} \cup MVFK(GY^{\mathcal{S}})$, akkor biztosak lehetünk, hogy nem keletkezett új gyakori minta, és csak a támogatottságokat kell frissíteni.

5.4. Gyakori sorozatok, bool formulák és epizódok

A kutatások középpontjában a gyakori elemhalmazok állnak. Tovább léphetünk, és kereshetünk bonyolultabb típusú mintákat is. Erről szól ez a fejezet.

5.4.1. Gyakori sorozatok kinyerése

Napjainkban az elektronikus kereskedelem egyre nagyobb méretet ölt. A vevők megismerésével és jobb kiszolgálásával célunk a profitnövekedés mellett a vásárlói elégedettség fokozása. Az elektronikus kereskedelem abban különbözik a hagyományos kereskedelemtől, hogy az egyes tranzakciókhoz hozzárendelhetjük a vásárlókat. Eddig a tranzakciók (kosarak) óriási halmaza állt rendelkezésünkre, most ennél több: pontosan tudjuk, hogy ki, mikor, mit vásárol.

Az újabb adatok újabb információkinyeréshez adhatnak alapot. Nem csak általános vásárlási szabályokat állíthatunk elő, hanem ennél többet: személyre szabhatjuk a vásárlási szokásokat, vevők csoportjait alakíthatjuk ki, megkereshetjük a sok, illetve kevés profitot hozó vásárlási csoportokat, stb.

Ebben a fejezetben a vevők között gyakran előforduló vásárlói minták kinyerésével foglalkozunk. Két példa: sok vevő a „Csillagok háborúja" DVD megvétele után a „Birodalom visszavág" című filmet, majd később a „Jedi visszatér" című filmet is megveszi DVD-n, vagy a vevők 30%-a új okostelefon vásárlása és új mobilinternet-előfizetés megkötése után mp3-as zenéket is vásárol egy elektronikus zeneboltban (letölt egy webes áruház honlapjáról).[8]

Kereskedelmi cégek a kinyert gyakori mintákat, epizódokat újabb profitnövekedést hozó fogásokra használhatják. Például kiderülhet, hogy a videomagnót vásárlók nagy aránya a vásárlást követő 3-4 hónappal kamerát is vásárolnak. Ekkor ha valaki videomagnót vesz, küldjünk ki postán kamerákat reklámozó prospektusokat a vásárlást követően 2-3 hónappal. A szekvenciális mintakinyerés (és egyéb epizódkutató algoritmusok) nem csak az on-line áruházakra jellemző. Felhasználási területük egyre bővül, a további kutatásokat a gyakorlatban előforduló problémák is igénylik. Jellemző terület a direkt marketing,

[8] A példa illusztratív.

de további felhasználási területre lehet példa az alábbi: páciensek tüneteit és betegségeit tartalmazó adatbázisokból kinyert minták nagy segítségre lehetnek az egyes betegségek kutatásánál, nevezetesen, hogy az egyes betegségeket milyen tünetek, vagy más betegségek előzik meg gyakran.

Mielőtt rátérünk arra, hogy miként lehet kinyerni elemhalmazokat tartalmazó sorozatokból a gyakoriakat, egy egyszerűbb esettel foglalkozunk, ahol a sorozat elemei atomi események.

A Gyakori Sorozat Fogalma

A gyakori sorozatok kinyerésének feladata annak a feladatkörnek egy esete, amikor a támogatottságot a tartalmazási predikátum alapján definiáljuk. Feltételezzük, hogy az olvasó tisztában van az 5.3.5 részben definiált fogalmakkal.

Adott ($\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ elemek (vagy termékek) halmaza és v darab $\mathcal{I}$ felett értelmezett sorozat. Tehát a bemenet sorozatoknak egy sorozata:

$$\textit{bemenet} : \mathcal{S} = \langle S_1, S_2, \ldots, S_v \rangle,$$

ahol $S_k = \langle i_1^{(k)}, i_2^{(k)}, \ldots, i_{n^{(k)}}^{(k)} \rangle$, és $i_j^{(k)} \in \mathcal{I}$.

Definiáljuk a $\mathcal{M} = (\mathcal{M}, \preceq)$ mintakörnyezet tagjait sorozatok esetében. Az $\mathcal{M}$ elemei az $\mathcal{I}$ felett értelmezett sorozatok:

5.4.1. Definíció *$S = \langle i_1, \ldots, i_m \rangle$ sorozat tartalmazza $S' = \langle i'_1, \ldots, i'_n \rangle$ sorozatot (jelöléssel $S' \preceq S$), ha léteznek $j_1 < j_2 < \ldots < j_n$ egész számok úgy, hogy $i'_1 = i_{j_1}, i'_2 = i_{j_2}, \ldots, i'_n = i_{j_n}$.*

Amennyiben $S' \preceq S$, akkor S' az S *részsorozata*. Például a $\langle G, C, I, D, E, H \rangle$ sorozat tartalmazza a $\langle C, D, H \rangle$ sorozatot. Ebben a mintakörnyezetben $||$ függvény a sorozat hosszát adja meg. A fentiek alapján a fedés, TID lista, támogatottság, gyakoriság, gyakori sorozat definíciója egyértelmű.

Egy alap mintakinyerési feladatban adott S_i sorozatok sorozata, továbbá *min_supp* támogatottsági küszöb, elő kell állítani a gyakori sorozatokat.

APRIORI

A fent definiált feladat a gyakori mintakinyerés egy speciális esete, így alkalmazhatók rá az általános algoritmusok, például az APRIORI. Az általános leírást megadtuk az 5.3.4 részben, itt most csak azon speciális részleteket vizsgáljuk, amelyek sorozat típusú mintatér esetén érvényesek. Két lépést vizsgálunk közelebbről a jelöltek előállítását és a támogatottság meghatározását.

Jelöltek előállítása – Az APRIORI jelöltelőállítása két lépésből áll: potenciális jelöltek előállítása, majd a potenciális jelöltek részmintáinak vizsgálata.

Akkor lesz egy ℓ-elemű potenciális jelöltből jelölt, ha minden $\ell - 1$ elemű részsorozata gyakori. Általánosan annyit mondtunk el, hogy egy potenciális jelölt két $\ell - 1$ elemű gyakori mintáknak (ezeket hívtuk a jelölt generátorainak) a minimális valódi felső korlátja. Sorozat típusú minta esetén akkor lesz két $\ell - 1$ elemű gyakori mintáknak a minimális valódi felső korlátja ℓ elemű, ha van $\ell - 2$ elemű közös részsorozatuk.

A hatékonyság szempontjából fontos lenne, ha a jelöltek előállítása ismétlés nélküli lenne. Ehhez szükségünk van a sorozatokon értelmezett teljes rendezésre. Az $\mathcal{I}$ elemein tudunk egy tetszőleges teljes rendezést definiálni, ami szerinti lexikografikus rendezés megfelel a célnak. A rendezés alapján értelmezhetjük egy sorozat tetszőleges elemű prefixét. Két $\ell - 1$ elemű gyakori mintákból akkor képzek potenciális jelöltet, ha $\ell - 2$ elemű prefixük megegyeznek (hasonlóan a halmazok eseténél). A minimális valódi felső korlát a az utolsó elemmel bővített sorozatok lesznek.

A generátorok lehetnek azonos sorozatok is. Például az $\langle G, C, I \rangle$ sorozat önmagával a $\langle G, C, I, I \rangle$ jelöltet fogja előállítani. Látnunk kell, hogy ez a jelöltelőállítás ismétlés nélküli, ugyanis tetszőleges jelölteknek egyértelműen meg tudjuk mondani a generátorait.

Támogatottság meghatározása – A jelölt sorozatok támogatottságának meghatározás szinte megegyezik a jelölt halmazok támogatottságának meghatározásával. Erről részletesen szóltunk az 5.1.2 részben. Itt csak az apró különbségekre térünk ki.

A kételemű jelölteknél nem csak a kétdimenziós tömb egyik felét fogjuk használni, hanem a teljes tömböt. Ez abból következik, hogy számít a sorrend, tehát például az $\langle A, B \rangle$ sorozat különbözik az $\langle B, A \rangle$ sorozattól.

„Kínában, ahol sokan fogyasztják rendszeresen, lehetőség volt hosszas kísérletek folytatására, melyek során bebizonyosodott, hogy azok a férfiak és nők, akik hetente legalább egyszer isznak teát, kevesebb eséllyel betegednek meg végbél, hasnyálmirigyés vastagbéldaganatban, illetve a betegség esetleges kialakulása során lelassul a rákos sejtek burjánzása.”
Forrás: http://www.vital.hu/themes/alter/bio9.htm

Kettőnél nagyobb jelölteket célszerű szófában tárolni. A szófa felépítése, a jelöltek támogatottságának meghatározása 1 apró részlettől eltekintve teljesen megegyezik a halmazoknál leírtakkal. A szófa bejárásakor ügyelni kell arra, hogy a sorozatban lehetnek ismétlődő elemek, illetve az elemek nincsenek sorba rendezve. A rekurziós lépés nem két rendezett lista közös elemeinek meghatározását jelenti, hanem egy rendezett lista (az adott belső pontból kiinduló élek címkéi) azon elemeinek meghatározását, amelyek szerepelnek egy másik listában (az aktuális bemeneti sorozat vizsgálandó része).

Elemhalmazokat tartalmazó gyakori sorozatok

Az előző részben definiált feladat általánosítása, amikor a bemeneti sorozat és a mintahalmaz elemei nem elemek sorozata, hanem elemhalmazoké. Azaz megenegedünk $\langle AB, B, ABC, E\rangle$ típusú sorozatokat is. Vásárlásoknál például nem csak egy terméket vásárolnak az emberek, hanem termékek egy halmazát.

Formális leírás – A bemeneti sorozatok és a mintatér elemei a $2^{\mathcal{I}}$ felett értelmezett sorozatok, azaz a sorozat elemei az $\mathcal{I}$ részhalmazai. A bemeneti sorozat elemeit szokás *vásárlói sorozatoknak* is hívni, utalva arra, hogy először vásárlói sorzatok esetén került elő a feladat.

Hasonlóan az eddigiekhez a támogatottságot a tartalmazási reláció alapján definiáljuk.

5.4.2. Definíció *$S = \langle I_1, \ldots, I_m\rangle$ sorozat tartalmazza $S' = \langle I'_1, \ldots, I'_n\rangle$ sorozatot (jelöléssel $S' \preceq S$), ha léteznek $j_1 < j_2 < \ldots < j_n$ egész számok úgy, hogy $I'_1 \subseteq I_{j_1}, I'_2 \subseteq I_{j_2}, \ldots, I'_n \subseteq i_{j_n}$.*

Ezzel a tartalmazási relációval egy sorozat mérete a sorozat elemeinek méretösszege (tehát például a $\langle AB, B, ABC, E\rangle$ sorozat mérete 7). A támogatottság, gyakoriság, TID lista, gyakori sorozat fogalmai megegyeznek az eddigiekkel. Feladatunk kinyerni az elemhalmazokból felépülő gyakori sorozatokat [Agrawal és Srikant, 1995].

APRIORIALL – Ismét APRIORI! De minek törjük az agyunkat új módszereken, ha van már módszer, ami jól megoldja a feladatot. Csak a jelöltek előállítását kell tisztázni (pontosabban csak annak első lépését), és készen is vagyunk, mehetünk pihenni :-). Ennél még kényelmesebb megoldást javasoltak az APRIORIALL kitalálói[9]. Visszavezették ezt a feladatot az előző részben bemutatott APRIORI megoldásra.

Bevezethetjük a gyakori elemhalmaz fogalmát. Az I elemhalmaz támogatottsága megegyezik azon sorozatok számával, amelyek valamelyik eleme tartalmazza I-t. Az I gyakori, ha támogatottsága nagyobb *min_supp*-nál. Nyilvánvaló, hogy gyakori sorozat minden eleme gyakori elemhalmaz. Ezeket a gyakori elemeket tekinthetjük atomi elemeknek, és használhatjuk az előző részben bemutatott algorimust. A gyakori elemhalmazok meghatározásához pedig tetszőleges gyakori elemhalmazt kinyerő algoritmust használhatunk. Ügyelnünk kell azonban arra, hogy a támogatottság meghatározásánál egy sorozat csak eggyel növelheti egy jelölt méretét akkor is ha több elemének része a jelölt.

[9]Ez nem meglepő, hiszen sem az ismétlés nélküli jelöltelőállítás sem a támogatottság meghatározása nem triviális feladat. Érdemes elgondolkozni azon, hogy miért nem.

A feladat visszavezetése az előző feladat APRIORI megoldására nem jelenti azt, hogy ez a megoldás megegyezik az absztrakt APRIORI adaptálásával elemhalmazokat tartalmazó sorozatokra. Az APRIORIALL ugyanis az iterációk során eggyel hosszabb jelöltsorozatokat hoz létre, amelyek mérete nem feltétlenül eggyel nagyobb generátoraiknál. Az APRIORIALL nagyobb léptékben halad, így kevesebb iterációs lépést hajt végre, de ugyanakkor jóval több hamis jelöltet generálhat. Ez tehát egy kényelmes, de veszélyes megoldás.

Időkényszerek bevezetése – A gyakori sorozatok kinyerését – hasonlóan a gyakori minták kinyeréséhez – alkalmazói igények keltették életre. A feladat sikeres megoldása, a kapott eredmények újabb feladathoz vezettek [Zaki, 2000] [Srikant és Agrawal, 1996] .

1. **Időkényszerek bevezetése.** A felhasználók gyakran specifikálni akarják a sorozatban található szomszédos elemek között eltelt idő maximális és minimális megengedett értékét. Például nem tulajdonítunk túl nagy jelentőséget annak, ha valaki vesz egy tusfürdőt majd három év múlva egy ugyanolyan márkájú szappant.

2. **Kosarak definíciójának lazítása.** Sok alkalmazásnál nem számít ha a sorozat egy elemét 2 (vagy több) egymás utáni kosár tartalmazza, ha azok vásárlási ideje bizonyos időablakon belül van. Amennyiben egy vevő 5 perc múlva visszatér az áruházba, akkor valószínű, hogy ezt nem az előző vásárlásának hatására tette (még kicsomagolni sem volt ideje az árut), hanem inkább elfelejtett valamit. Logikus, hogy a két vásárlást összevonhatjuk, és lehet, hogy az összevont kosárhalmazban már megtalálható lesz a sorozat egy eleme, míg az eredeti kettőben külön-külön nem. A tranzakciók definíciójának ilyen lazításánál a sorozatok elemeit kosarak uniója tartalmazhatja, ahol az unióban szereplő kosarak vásárlási idejeinek egy előre megadott időablakon belül kell lenniük.

Gyakori sorozat fogalma időkényszerek esetén – Ismét vásárlási sorozatok sorozataként adott a bemenet, de most a vásárlási sorozatok elemei nem pusztán elemhalmazok, hanem olyan párok, amelyek első tagja egy elemhalmaz, második tagja pedig egy időbélyeg. Tehát, legyen ismét $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ elemek (vagy termékek) halmaza. Egy vásárlói sorozat most $\mathcal{T} = \langle \hat{t}_1, \hat{t}_2, \ldots, \hat{t}_n \rangle$ tranzakciók sorozata, ahol $\hat{t}_j = (t_j, TIME_j)$, $t_j \subseteq \mathcal{I}$, $TIME_j \in \mathbb{R}$. A $\hat{t} = (t, TIME)$ tranzakció tartalmazza $I \subseteq \mathcal{I}$ elemhalmazt (jelölésben $I \subseteq \hat{t}$), ha $I \subseteq t$. A $\hat{t}$ tranzakció idejére a továbbiakban $\hat{t}.TIME$-al hivatkozunk, tranzakciójára $\hat{t}.t$-vel.

A mintakörnyezet definíciója megegyezik a hagyományos, sorozatokat tartalmazó mintakörnyezettel. Mivel ebben az esetben a bemenet és a mintatér elemeinek típusa különbözik (párokból álló sorozat, illetve elemhalmazokból álló sorozat) ezért definiálnunk kell a támogatottságot.

5.4.3. Definíció *A* $\mathcal{T} = \langle \hat{t}_1, \hat{t}_2, \ldots, \hat{t}_n \rangle$ *vásárlói sorozat tartalmazza az* $M = \langle I_1, \ldots, I_m \rangle$ *mintasorozatot, ha léteznek* $1 \leq l_1 \leq u_1 < l_2 \leq u_2 < \ldots < l_m \leq u_m \leq n$ *egész számok úgy, hogy*

1. $I_j \subseteq \cup_{k=l_j}^{u_j} \hat{t}_k.t, 1 \leq j \leq m,$
2. $\hat{t}_{u_i}.TIME - \hat{t}_{l_i}.TIME \leq$ idő_ablak, $1 \leq i \leq m,$
3. $\hat{t}_{l_i}.TIME - \hat{t}_{u_{i-1}}.TIME >$ min_eltelt_idő, $2 \leq i \leq m$
4. $\hat{t}_{u_i}.TIME - \hat{t}_{l_{i-1}}.TIME \leq$ max_eltelt_idő, $2 \leq i \leq m$

A fentiekből látszik, hogy az 5.4.1. definícióval ellentétben tetszőleges elemhalmazt tranzakciók elemhalmazainak uniója tartalmazhat, ahol a tranzakcióknak *idő_ablak*on belül kell lenniük (2. feltétel).

Ez alapján az M mintasorozat *támogatottsága* legyen az M-et tartalmazó vásárlói sorozatok száma. Egy mintasorozat gyakori, ha támogatottsága nem kisebb egy előre megadott támogatottsági küszöbnél (min_supp).

Definiáltunk egy gyakori mintákat kinyerő problémát, amit nyilvánvalóan meg tudunk oldani egy APRIORI algoritmussal. A jelöltek előállításának módja egyezzen meg az APRIORIALL jelöltelőállításának módjával (lévén a mintakörnyezet ugyanaz), a támogatottságok meghatározásánál pedig vegyük figyelembe az időkényszereket, annak érdekében, hogy a helyes támogatottságokat kapjuk. Ha lefuttatnánk így az algoritmust, és vizsgálnánk az eredményt, akkor megdöbbenve vennénk észre, hogy az APRIORI algoritmus nem állította elő az összes gyakori sorozatot. Mi az oka ennek? Bizonyítottuk, hogy az APRIORI teljes, de akkor hol bújt el a hiba? A következő részben eláruljuk a megoldást.

GSP algoritmus – A GSP (Generalized Sequential Patterns) algoritmus alkalmas olyan sorozatok kinyerésre, amelynél időkényszereket alkalmazhatunk és lazíthatjuk a tranzakciók definícióját. A most következő leírás látszólag teljesen eltér a GSP-t publikáló írástól. Ennek oka az, hogy ragaszkodunk az egységes leíráshoz, amit a 5.3.1 részben adtunk. Ennek a leírásnak nagy előnye az, hogy ha a problémát meg tudjuk fogalmazni ebben a keretben, akkor a megoldás is azonnal adódik.

Térjünk vissza arra a kérdésre, hogy hol a hiba. Tekintsük a következő mintát: $M = \langle A, B, C \rangle$, és nézzük a következő vásárlói sorozatot:

$$mathcalT = \langle (A, 1.0), (B, 2.0), (C, 3.0) \rangle.$$

Ha *max_ eltelt_ idő=1.5*, akkor $\mathcal{T}$ tartalmazza M-et, de nem tartalmazza annak $M' = \langle A, C \rangle$ részmintáját, ugyanis az A és C elem időbélyege között nagyobb a különbség *max_ eltelt_ idő*-nél. Ezek szerint az M támogatottsága nagyobb, mint M' részmintájának támogatottsága. Azaz a fent definiált támogatottsági függvény nem teljesít a támogatottsági függvénnyel szembeni elvárásunkat! Hát ez a hiba, ezért nem fog helyes eredményt adni az APRIORI.

Ahelyett, hogy új problémát definiálnánk és új algoritmus keresnénk, próbálkozzunk azzal, hogy átírjuk a feladatot úgy, hogy az új feladat megoldásai megegyezzenek az eredeti feladat megoldásaival, és az új feladat beilleszkedjen egységes keretünkbe. A bemenet, a keresett minta típusa és a támogatottsági függvény adott, így csak a $\mathcal{MK} = (\mathcal{M}, \prec)$ mintakörnyezet második tagját változtathatjuk meg.

5.4.4. Definíció *Az $M = \langle I_1, \ldots, I_n \rangle$ sorozatnak M' részsorozata (vagy az M tartalmazza M'-t, $M' \prec M$), amennyiben az alábbi 3 feltétel közül teljesül valamelyik:*

1. *M'-t megkaphatjuk M-ből I_1 vagy I_n törlésével.*
2. *M'-t megkaphatjuk M-ből egy legalább 2 elemű I_i valamely elemének törlésével.*
3. *M' részsorozata M''-nek, ahol M'' részsorozata M-nek.*

Ebben a mintakörnyezetben a $||$ függvény ismét a sorozat elemei méretének összegét adja meg. Nézzünk példákat részsorozatokra. Legyen $M = \langle AB, CD, E, F \rangle$. Ekkor a $\langle B, CD, E \rangle$, $\langle AB, C, E, F \rangle$ és a $\langle C, E \rangle$ mind részsorozatai M-nek, de a $\langle AB, CD, F \rangle$ és $\langle A, E, F \rangle$ sorozatok nem azok.

Észrevétel – A fenti tartalmazási relációra nézve a támogatottsági függvény rendelkezik a monotonitás tulajdonságával.

Ha visszatérünk ahhoz a példához, amelyen bemutattuk, hogy az eredeti támogatottsági függvény nem igazi támogatottsági függvény, akkor láthatjuk, hogy nem baj, ha $\langle A, B, C \rangle$ támogatottsága nagyobb, mint az $\langle A, C \rangle$ támogatottsága, ugyanis $\langle A, C \rangle$ nem része az $\langle A, B, C \rangle$ sorozatnak.

Most már alkalmazhatjuk az APRIORI algoritmust. Ezzel kapcsolatban egyetlen kérdést kell tisztáznunk, mégpedig az, hogyan és mikor állítsunk elő két $\ell - 1$ elemű gyakori sorozatból ℓ elemű jelöltet.

Két k-méretű sorozatból (S_1, S_2) potenciális jelöltet generálunk akkor, ha törölnénk S_1 első elemének legkisebb sorszámú elemét ugyanazt a sorozatot kapnánk, mintha S_2-ből az utolsó elem legnagyobb sorszámú elemét törölnénk. A jelölt sorozat az S_2 utolsó elemének legnagyobb sorszámú elemével bővített S_1 sorozat lesz. Az új elem külön elemként fog megjelenni a jelöltben, amennyiben S_2-ben is külön elem volt, ellenkező esetben S_1 utolsó eleméhez csatoljuk. A fentiek alól kivétel az 1-elemes sorozatok illesztése, ahol az új elemet mind a kétféleképpen fel kell venni, tehát mint új elem, és mint bővítés is. Ezek szerint $\langle(i)\rangle$ és $\langle(j)\rangle$ illesztésénél $\langle(i,j)\rangle$, és $\langle(j),(i)\rangle$ is bekerül a jelöltek közé (egyértelmű, hogy mindkét jelöltnek mindkét 1-elemes sorozat részsorozata).

3 méretű gyakoriak	4 méretű jelöltek	
	potenciális jel.	jelölt
$\langle(A,B),(C)\rangle$	$\langle(A,B),(C,D)\rangle$	$\langle(A,B),(C,D)\rangle$
$\langle(A,B),(D)\rangle$	$\langle(A,B),(C),(E)\rangle$	
$\langle(A),(C,D)\rangle$		
$\langle(A,C),(E)\rangle$		
$\langle(B),(C,D)\rangle$		
$\langle(B),(C),(E)\rangle$		

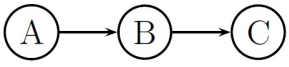
5.7. táblázat. Példa: GSP jelöltgenerálás

A fenti táblázat egy példát mutat a jelöltek előállítására. Az $\langle(A,B),(C)\rangle$ sorozatot a $\langle(B),(C,D)\rangle$ és a $\langle(B),(C),(E)\rangle$ sorozathoz is illeszthetjük. A többi sorozatot egyetlen másik sorozathoz sem tudjuk illeszteni. Például az $\langle(A,B),(D)\rangle$ illesztéséhez $\langle(B),(Dx)\rangle$ vagy $\langle(B),(D),(x)\rangle$ alakú sorozatnak kéne szerepelnie a gyakoriak között, de ilyen nem létezik. A törlési fázisban az $\langle(A,B),(C),(E)\rangle$ sorozatot töröljük, mert az $\langle(A),(C),(E)\rangle$ részsorozata nem gyakori.

A jelöltek támogatottságának meghatározását nem részletezzük.

Sorozat típusú minta általánosítása

Tetszőleges elemsorozatot ábrázolhatunk egy gráffal. Például a $\langle A,B,C\rangle$ sorozat megfelelője az alábbi gráf:



Az általunk definiált sorozatot, mindig egy nagyon egyszerű gráffal ábrázolnánk, ami egy irányított, körmentes, címkézett út. Mi sem természetesebb,

hogy a sorozat általánosítása egy olyan valami, amit teljesen általános irányított, körmentes, címkézett gráffal ábrázolunk. Például lehet egy általános mintához tartozó gráf az 5.13 ábrán látható.

Érezzük, hogy ezt a mintát tartalmazzák például a $\langle A, D, C, B, C, B\rangle$ vagy az $\langle E, D, A, B, B, CC, B\rangle$ sorozatok, de nem tartalmazzák a $\langle A, D, C, C, B, B\rangle$ illetve a $\langle A, D, B, C, B, C\rangle$ sorozatok.

Ugyanezt az általános leírást kapnánk, ha egy sorozatra nem mint út tekintünk, hanem mint olyan halmazon értelmezett teljes rendezés, amelynek elemei azonosító, elem párok. A teljes rendezés általánosítása ugyanis a részben rendezés, amit körmentes, irányított gráffal szokás ábrázolni.

Nézzük formálisan. Legyen $\mathcal{I}$, illetve TID elemek és azonosítók halmaza. A mintatér elemei ekkor (tid, i) párokon értelmezett részben rendezés, ahol $tid \in TID, i \in \mathcal{I}$. A *tid címkéjén* az i elemet értjük.

5.4.5. Definíció *Az* $m = (\{(tid_1, i_1), \ldots, (tid_m, i_m)\}, \leq)$ *minta* tartalmazza *az* $m' = (\{(tid'_1, i'_1), \ldots, (tid'_n, i'_n)\}, \leq')$ *mintát (jelöléssel* $m' \preceq m$*), ha létezik* $f : \{tid'_1, \ldots tid'_m\} \to \{tid_1, \ldots tid_n\}$ *injektív függvény úgy, hogy* tid'_j *címkéje megegyezik* $f(tid'_j)$ *címkéjével (*$1 \leq j \leq m$*), és* $(tid'_k, i'_k) \leq' (tid'_l, i'_l)$ *esetén* $(f(tid'_k), i'_k) \leq (f(tid'_l), i'_l)$ *is teljesül minden (*$1 \leq k, l \leq m$*) indexre.*

Az általános minta keresésénél a bemenet $\mathcal{I}$ felett értelmezett elemsorozatok sorozataként adott. Egy bemeneti sorozat tulajdonképpen felfogható általános mintának, ahol a rendezés teljes rendezés. Egy minta *támogatottsága* megegyezik azon sorozatok számával, amelyek tartalmazzák a mintát.

5.4.2. Gyakori bool formulák

Legyenek a bemenet n-esek halmaza. A felhasználó megad predikátumokat, amelyek a bemenet elemein vannak értelmezve, és akár többváltozósak is lehetnek. A mintatér elemei ezen predikátumokon értelmezett bool formula. A formulában megengedjük az *és*, *vagy* illetve *negáció* operátorokat [Mannila és Toivonen, 1996], de hatékonysági okok miatt célszerű csak a diszjunktív normál formulákra szorítkozni.

„*Kutatási eredmények igazolják, hogy a csoportban működőknek teljesebb szülésélményben van részük, körükben alacsonyabb a koraszülések száma, és a babák súlya is nagyobb az egyéni felkészülésben részesülőknél.*”
Forrás: Baba Patika X. évfolyam 10. szám, 56. oldal 2007. október

Nézzünk példákat. Tegyük fel, hogy egy telekommunikációs hálózatban egy eseménynek 4 attribútuma van: típus, modul, szint, időbélyeg. Az első megadja egy riasztás típusát, a második a modult, ami a riasztást küldte, a

harmadik a riasztás erősségét, a negyedik pedg riasztás időpontját. Ebben a környezetben mintára lehet példa az alábbi:

$$\text{p(x,y)}=x.típus=2356 \wedge y.típus=7401 \wedge x.time \leq y.time \wedge x.modul=y.modul$$

ami azt jelenti, hogy egy 2356 és egy 7401 típusú riasztás érkezett ebben a sorrendben ugyanabból a modulból. Bevezethetjük például a *szomszédja* – modul attribútumra vonatkozó – kétváltozós predikátumot, ha úgy gondoljuk hogy fontos lehet ennek vizsgálata. Ekkor a

$$\text{p'(x,y)}=x.típus=2356 \wedge y.típus=7401 \wedge szomszédja(x.modul=y.modul)$$

azt fejezi ki hogy a 2356 és 7401 típusú riasztások szomszédos modulból érkeztek.

A $p(x_1, x_2, \ldots x_m)$ m változós minta *illeszkedik* az $\langle S_1, S_2, \ldots, S_v \rangle$ sorozatra, ha léteznek $i_1, i_2, \ldots i_m$ egészek úgy, hogy $p(S_{i_1}, S_{i_2}, \ldots, S_{i_m})$ igaz értéket ad.

5.4.3. Gyakori epizódok

Az eddigi részekben sok elemhalmaz, sorozat volt adva, és kerestük a gyakori mintákat. Ezek a minták általánosan érvényes információt adtak: az adott vásárlói minta sok vásárlóra jellemző. Ha a sok sorozatból kiválasztunk egyet és azt elemezzük, akkor az adott sorozatra jellemző információt nyerünk ki. Megtudhatjuk például, mi jellemző az adott ügyfélre, amit felhasználhatunk akkor, amikor személyre szabott ajánlatot szeretnénk tenni (például azért mert az ügyfél elégedetlen szolgáltatásainkkal, és vissza akarjuk szerezni bizalmát).

Epizódkutatásról beszélünk, ha egyetlen sorozat van adva, és ebben keressük a gyakran előforduló mintákat[Mannila és tsa., 1995, **?**]. Az epizódkutatásnak egyik fontos területe a telekommunikációs rendszerek vizsgálata. Az olyan epizódok feltárása, amelyben riasztás is előfordul, alkalmas lehet a riasztás okának felderítésére, vagy előrejelzésére.

Nem vezetünk be új típusú mintát, tehát most is elemhalmazokat, sorozatokat keresünk, de a formalizmus könnyen általánosítható elemhalmazokat tartalmazó sorozatokra, vagy általános mintára is. A támogatottsági függvény lesz új, ami abból fakad, hogy egyetlen bemeneti sorozat van adva.

A támogatottság definíciója

Legyen $\mathcal{I}$ elemek (items) halmaza. A bemenet az $\mathcal{I}$ felett értelmezett sorozat.

$$bemenet : \mathcal{S} = \langle i_1, i_2, \ldots, i_n \rangle,$$

ahol $i_k \in \mathcal{I}$ minden k-re,

5.4.6. Definíció *Az $\mathcal{S} = \langle i_1, i_2, \ldots, i_n \rangle$ sorozatnak a $\langle i_j, i_{j+1}, \ldots, i_{j+w-1} \rangle$ sorozat egy w elem széles* összefüggő részsorozata*, ha $1 \leq j \leq n + 1 - w$.*

Ha $w < n$, akkor valódi összefüggő részsorozatról beszélünk.

Legyen adva $\mathcal{MK}$ mintakörnyezet, és értelmezzük valahogy a τ anti-monoton *illeszkedési predikátumot.* $\tau_{\mathcal{S}}(m)$ igaz értéket ad, ha az m minta illeszkedik az $\mathcal{S}$ sorozatra.

5.4.7. Definíció *A m minta minimálisan illeszkedik az $\mathcal{S}$ sorozatra, ha $\mathcal{S}$-nek nincsen olyan valódi összefüggő részsorozata, amelyre illeszkedik m.*

Ha például a mintatér elemei $\mathcal{I}$ részhalmazai, akkor a $\mathcal{S} = \langle i_1, i_2, \ldots, i_n \rangle$ sorozatra illeszkedik az I halmaz, amennyiben minden $i \in I$-hez létezik $1 \leq j \leq n$, amelyre $i = i_j$. Elemsorozat típusú minta esetén S akkor illeszkedik az $\mathcal{S}$ sorozatra, ha S részsorozata $\mathcal{S}$-nek, ahol a részsorozat definíciója megegyezik az 5.4.1. részben megadottal.

Két különböző támogatottsági definíció terjedt el.

5.4.8. Definíció *Legyen $\mathcal{S}$ bemeneti sorozat, $\mathcal{MK} = (\mathcal{M}, \preceq)$ mintakörnyezet és τ anti-monoton illeszkedési predikátum. Az $m \in \mathcal{M}$ minta támogatottsága megegyezik*

1. *S azon összefüggő részsorozatainak számával, amelyekre m minimálisan illeszkedik.*

2. *S azon w széles részsorozatainak számával, amelyekre m illeszkedik. Itt w előre megadott konstans.*

Ha a támogatottság így van definiálva, akkor a mintatér elemeit *epizódoknak* nevezzük. Egy epizód *gyakori*, ha támogatottsága nem kisebb egy előre megadott korlátnál, amit általában *min_supp*-al jelölünk.

Epizódkutatásnál adott $\mathcal{S}$ bemeneti sorozat $\mathcal{MK} = (\mathcal{M}, \preceq)$ mintakörnyezet (esetleg w) és τ illeszkedési predikátum, célunk megtalálni a gyakori epizódokat.

APRIORI

Az illeszkedési predikátum anti-monoton tulajdonságából következik a támogatottság anti-monotonítása, amiből jön, hogy gyakori epizód minden részepizódja gyakori. Mi sem természetesebb, hogy a gyakori epizódok kinyeréséhez az APRIORI algoritmust használjuk. Az jelöltek-előállítása és a gyakori epizódok kiválogatása ugyanaz, minta a támogatottságot a régi módszerrel definiálnánk (lásd 5.1.2 5.4.1 rész). Egyedül a támogatottság meghatározásán kell változtatnunk. A következőkben feltesszük, hogy a támogatottságot a második definíció szerint értjük (w széles ablakok száma).

A támogatottság meghatározásának egy butuska módszere lenne, ha az eseménysorozaton egyszerűen végigmasírozva minden összefüggő részsorozatnál meghatároznánk, hogy tartalmazza-e az egyes jelölt epizódokat. Hatékonyabb algoritmushoz juthatunk, ha felhasználjuk azt, hogy szomszédos sorozatok között pontosan két elem eltérés van. Vizsgájuk meg az első sorozatot, majd nézzük az eggyel utána következőt, és így tovább addig, amíg el nem érjük az utolsót. Mintha egy ablakot tolnánk végig a sorozaton.

Vezetjük be a következő változókat. Minden i elemhez tartozik:

- *i.számláló*, ami megadja, hogy a jelenlegi összefüggő részsorozatba hányszor fordul elő az i elem.
- *i.epizódjai* lista, amelyben az i elemet tartalmazó epizódok találhatók.

„*Nemzetközi tanulmányok alapján elmondhatjuk, hogy a magzati fejlődési rendellenességek (az agykoponya hiánya, nyitott hátgerinc), továbbá a szív és a vese rendellenességei megelőzhetők, ha a terhes kismama a fogamzást megelőzően legalább négy hétig, majd a terhesség első három hónapjában folsav tartalmú készítményt szed.*" Forrás: Baba Patika X. évfolyam, 10. szám, 48. oldal, 2007. október

Epizódjelöltekhez pedig a következőkre lesz szükségünk:

- *j.kezdeti_index*: annak a legkorábbi elemnek az indexe, amely után minden részsorozatban előfordult az epizód egészen a jelenlegi részsorozatig.
- *j.számláló*, ami megadja, hogy hány *kezdeti_index* előtti összefüggő részsorozatban fordult elő j jelölt. A bemenet feldolgozása után e változó fogja tartalmazni a jelölt támogatottságát.
- *j.hiányzás* egész szám adja meg, hogy j elemei közül hány nem található a jelenlegi összefüggő részsorozatban. Nyilvánvaló, hogy ha φ előfordul a jelenlegi részsorozatban, akkor *j.hiányzás=0*.

Elemhalmazok támogatottságának meghatározása – Amikor lépünk a következő részsorozatra, akkor egy új elem kerül bele az ablakba, amit jelöljünk $i_{\text{új}}$-al, ugyanakkor egy elem eltűnik a sorozatból, ezt pedig jelöljük $i_{\text{régi}}$-vel.

Egy elem kilépésének következtében epizódok is kiléphetnek. $i_{régi}$*.számláló* segítségével megállapíthatjuk, hogy maradt-e még ilyen elem az ablakban, mert ha igen, akkor az eddig tartalmazott epizódokat az új ablak is tartalmazza. Ha nem maradt, akkor *i.epizódjai* és epizódok *hiányzás* számlálója alapján megkaphatjuk azon epizódokat, amelyek kiléptek a sorozatból. Ezek előfordulásának értékét kell növelni. Ebben segítségünkre van a *kezdeti_index* érték,

ami megadja, hogy mióta van jelen az epizód a sorozatokban. Az algoritmus pszeudokódja az alábbi ábrán látható.

Könnyű kitalálni ezek alapján, hogy mit kell tenni egy új elem belépésénél. Ha az új elem még nem szerepelt az ablakban, akkor végig kell nézni az új elemet tartalmazó epizódokat. Azon epizód kezdeti indexét kell a jelenlegi indexre beállítani, amelyekből csak ez az egyetlen elem hiányzott (5.15 ábra).

Elemsorozatok támogatottságának meghatározása – Az elemsorozatok felismerése determinisztikus véges automatákkal történik, amelyek az egyes elemsorozatokat fogadják el. Az epizód alapján az automata előállítása egyszerű, az 5.16. ábra erre mutat példát.

A teljes elemsorozatot egyesével olvassuk végig az első elemtől kezdve. Ha valamely epizód első eleme megegyezik az éppen olvasott elemmel, akkor új automatát hozunk létre. Ha ez az elem elhagyja az ablakot, akkor töröljük az automatát. Amikor egy automata elfogadó állapotba lép (jelezve, hogy az epizód megtalálható az ablakban), és nincs ehhez az epizódhoz tartozó másik – szintén elfogadó állapotban lévő – automata, akkor *kezdeti_index* felveszi az aktuális elem indexét. Amennyiben egy elfogadó állapotban lévő automatát törlünk, és nincs más, ugyanahhoz az epizódhoz tartozó elfogadó állapotú automata, akkor a *kezdeti_index* alapján növeljük az epizód *számlálóját*, hiszen tudjuk, hogy az epizód a kezdeti idő utáni összes részsorozatban megtalálható volt egészen az aktuális részsorozat előtti részsorozatig.

Vegyük észre, hogy felesleges adott epizódhoz tartozó, ugyanabban az állapotban lévő automatákat többszörösen tárolni: elég azt ismernem, amelyik utoljára lépett be ebbe az állapotba, hiszen ez fog utoljára távozni. Emiatt j jelölthöz maximum j darab automatára van szükség.

Egy új elem vizsgálatakor nem kell az összes automatánál megnéznünk, hogy új állapotba léphetnek-e, mert az elem *epizódjai* listájában megtalálható az őt tartalmazó összes epizód.

Az előzőekben ismertetett epizódkutatási algoritmus olyan adatbányászati problémára adott megoldást, ami az ipari életben merült fel, és hagyományos eszközök nem tudták kezelni. Az algoritmus telekommunikációs hálózatok riasztásáról eddig nem ismert, az adatokban rejlő információt adott a rendszert üzemeltető szakembereknek. Erről bővebben az alábbi cikkekben olvashatunk: [Klemettinen, 1999][Lee és Stolfo, 1998] [Lee és tsa., 1999][Lee és Stolfo, 2000][Hatonen, 1996].

5.5. Gyakori fák és feszített részgráfok

Amikor gyakori elemhalmazokat kerestünk, akkor azt néztük, hogy mely elemek fordulnak elő együtt gyakran. Sorozatok keresésénél ennél továbbléptünk, és

azt is néztük, hogy milyen sorrendben fordulnak elő az elemek, azaz melyek elemek előznek meg más elemeket. Ez már egy bonyolultabb kapcsolat. Még általánosabb kapcsolatok leírására szolgálnak a gráfok: a felhasználási terület entitásainak felelnek meg a gráf csúcsai vagy a csúcsainak címkéi, amelyeket él köt össze, amennyiben van közöttük kapcsolat. A kapcsolat típusát, sőt az entitások jellemzőit is kezelni tudjuk, amennyiben a gráf csúcsai és élei címkézettek.

Ezt a fejezetet először a gráf egy speciális esetével, a gyökeres fák vizsgálatával kezdjük, majd rátérünk a gyakori általános gráfok keresésére. Ellentétben az elemhalmazokkal vagy a sorozatokkal, a támogatottságot megadó illeszkedési predikátumot a gráfoknál többféleképpen definiálhatjuk: részgráf, feszített részgráf, topologikus részgráf. Ez tovább bővíti a megoldandó feladatok körét.

5.5.1. Az izomorfia problémája

Ha gráfokra gondolunk, akkor szemünk előtt vonalakkal – irányított gráfok esetében nyilakkal – összekötött pontok jelennek meg. Címkézett gráfoknál a pontokon és/vagy az éleken címkék, általában számok szerepelnek. Különböző pontoknak lehetnek azonos címkéi. Egy ilyen pontokat és vonalakat tartalmazó rajz a gráf egy lehetséges ábrázolása. Matematikailag egy gráf egy páros, amelynek első eleme egy alaphalmaz, a második eleme ezen alaphalmazon értelmezett bináris reláció.

Különböző gráfoknak lehet azonos a rajzuk. Például a $G_1 = (\{a, b\}, \{a, b\})$ és a $G_1 = (\{a, b\}, \{b, a\})$ gráfok rajza ugyanaz lesz: az egyik pontból egy nyíl indul a másik pontba. Ugyanúgy azonos ábrát készítenénk, ha az egyetlen élnek címkéje lenne, vagy a két pontnak ugyanaz lenne a címkéje. Az alkalmazások többségében a gráf rajza, topológiája továbbá a címkék az érdekesek és nem az, hogy a pontokat hogyan azonosítjuk annak érdekében, hogy a bináris relációt fel tudjuk írni. Ezen alkalmazásokban nem akarjuk megkülönböztetni az *izomorf* gráfokat (pontos definíciót lásd alapfogalmak gráfelmélet részében). Ez a helyzet áll fenn, például amikor kémiai vegyületeket vizsgálunk. Itt a gráf címkéi jellemzik az atomot (esetleg még további információt, pl. töltést) az élek a kötést, az élek címkéi pedig a kötés típusát (egyszeres kötés, kétszeres kötés, aromás kötés). Amikor gyakori gráfokat keresünk, akkor mindenképpen el kell döntenünk, hogy az izomorf gráfokat megkülönböztetjük, vagy nem. Mielőtt rátérünk a gyakori gráfok keresésére, járjuk egy kicsit körül az izomorfia kérdését.

Két gráf izomorfiájának eldöntésére nem ismerünk polinom idejű algoritmust, sőt azt sem tudjuk, hogy a feladat NP-teljes-e. Hasonló feladat a *részgráf izomorfia* kérdése, ahol azt kell eldönteni, hogy egy adott gráf izomorf-e egy másik gráf valamely részgráfjával. Ez a feladat NP-teljes. Ha ugyanis az egyik

gráf egy k-csúcsú teljes gráf, akkor a feladat az, hogy keressünk egy gráfban k-csúcsú klikket, ami bizonyítottan NP-teljes. Szerencsére kisebb méretű gráfok esetében az izomorfia eldöntése egyszerűbb algoritmusokkal is megoldható elfogadható időn belül. A két legismertebb részgráf izomorfiát eldöntő algoritmus Ullmanntól a backtracking [Ullmann, 1976] és B.D.McKaytól a Nauty [McKay, 1981].

A gráf izomorfiát eldöntő módszerek a csúcsok *invariánsait* használják. Az invariáns tulajdonképpen egy tulajdonság. Például invariáns a csúcs címkéje, fokszáma, illetve irányított gráfok esetében a befok és a kifok is két invariáns. Amennyiben a G_1, G_2 gráfok a ϕ bijekció alapján izomorfak, akkor az u csúcs minden invariánsa megegyezik a $\phi(u)$ csúcs megfelelő invariánsaival a G_1 minden u csúcsára. Ez tehát egy szükséges feltétel: az u csúcshoz csak azt a csúcsot rendelheti a bijekció, amelynek invariánsai páronként azonosak az u invariánsaival.

Az izomorfia eldöntésének naív módszere az lenne, ha az összes bijekciót megvizsgálnánk egyesével. Egy bijekció a csúcsoknak egy permutációja, így n csúcsú gráfok esetében $n!$ bijekció létezik. Csökkenthetjük ezt a számot az invariánsok segítségével. Osszuk részekre a csúcsokat. Egy csoportba azon csúcsok kerüljenek, amelyeknek páronként minden invariánsuk azonos. Nyilvánvaló, hogy az olyan bijekciókat kell megvizsgálni, amelyek csak ugyanazon invariánsok által leírt csoportba tartoznak. Ha az invariánsokkal a V csúcsokat szétosztottuk a $V_1, \ldots, V_k$ csoportokba, akkor a szóba jövő bijekciók száma $\prod_{i=1}^{k} \left(|V_i|!\right)$-re csökken. Minél több csoportot hoznak létre az invariánsok annál többet nyerünk ezzel az egyszerű trükkel. Az invariánsok nem csökkentik asszimptotikusan a számítás komplextását. Ha például a gráf reguláris és a csúcsoknak nincsenek címkéjük, akkor minden csúcs azonos csoportba kerül, azaz nem nyerünk a trükkel semmit.

„*A legújabb kutatások szerint bizonyos vitaminok képesek a hibás gének okozta fejlődési rendellenességek kivédésére.*"
Forrás: Baba Patika X. évfolyam 10. szám, 44. oldal, 2007. október

Eddigi ismereteink alapján elmondhatjuk, hogy minél bonyolultabb gyakori mintát keresünk, annál nehezebb a feladat és annál erőforrás-igényesebbek a megoldó algoritmusok. A címke nélküli gráfok egy általánosítása a címkézett gráfok, így azt várjuk, hogy címkézett gráfokhoz még több számítást kell majd végezni. Az előbb bemutatott módszer szerencsére az ellenezőjét állítja, hiszen a címke egy invariáns, ami újabb csoportokat hozhat létre. Sőt minél több a címke, annál több a csoport és annál gyorsabban döntjük el, hogy két gráf izomorf-e.

A gráf izomorfiából született probléma a gráfok *kanonikus kódolásának* problémája.

5.5.1. Definíció *A gráfok* kanonikus kódolása *(vagy* kanonikus címkézése*) egy olyan kódolás, amely az izomorf gráfokhoz és csak azokhoz azonos kódsorozatot rendel.*

Nyilvánvaló, hogy egy kanonikus kódolás előállítása ugyanolyan nehéz feladat, mint két gráf izomorfiájának eldöntése, hiszen két gráf izomorf, ha kanonikus kódjaik megegyenek. Például egy egyszerű kanonikus kód az, amit úgy kapunk, hogy egy adott gráf összes lehetséges szomszédossági mátrixát tekintjük (a csúcsok különböző sorrendezéséhez különböző szomszédossági mátrixok tartoznak), egy-egy szomszédossági mátrix sorait egymás után írva minden egyes szomszédossági mátrixot egy-egy kóddá alakítunk, majd ezen kódok közül kiválasztjuk a lexikografikus rendezés szerinti legkisebbet.

Nézzük példaként az 5.17 ábrán látható csúcs- és élcímkézett gráfot (a csúcsokban szereplő számok a csúcsok azonosítói). Legyen $cimke(1) = e$, $cimke(2) = e$, $cimke(3) = e$, $cimke(4) = f$. Ennek kanonikus kódja $\langle e00A0e0B00fAABAe\rangle$ lesz, ha a címkéken az abc szerinti rendezést vesszük és a rendezésben a 0 minden betűt megelőz.

Kanonikus kódok ennél kifinomultabb eljárásokkal is előállíthatók, például mélységi és szélességi keresés alapján [Borgelt, 2007].

5.5.2. A gyakori gráf fogalma

Annak alapján, hogy az izomorf gráfokat megkülönböztetjük, vagy sem, a gyakori gráfok kinyerésének feladatát két csoportra osztjuk. Legyen $\mathcal{V} = \{v_1, v_2, \ldots, v_m\}$ csúcsok halmaza. A mintakörnyezet ekkor az $MK = (\{G_1 = (V_1, E_1), G_2 = (V_2, E_2), \ldots\}, \preceq)$ pár, ahol $V_i \subseteq \mathcal{V}$, minden gráf összefüggő és $G_i \preceq G_j$, amennyiben G_i a G_j-nek részgráfja. A bemenet szintén olyan gráfok sorozata, amelyek csúcshalmaza $\mathcal{V}$-nek részhalmazai. A gráfok csúcsainak és/vagy éleinek lehetnek címkéi. A továbbiakban az élek és csúcsok címkéjét a c_E és c_V függvények adják meg. Az általánosság megsértése nélkül feltehetjük, hogy a címkék pozitív egész számok.

A támogatottságot illeszkedési predikátum alapján definiáljuk. Attól függően, hogy a csúcsok értéke fontos, vagy csak a címkéjük, az illeszkedést kétféleképpen definiálhatjuk: G' gráf illeszkedik a G bemeneti gráfra, ha

- G' részgráfja/feszített részgráfja/topologikus részgráfja G-nek,
- létezik G-nek olyan részgráfja/feszített részgráfja/topologikus részgráfja, amely izomorf G'-vel.

A fenti lehetőségek közül az alkalmazási terület ismerete alapján választhatunk.

A topologikus részgráf fogalma nem tartozik az alapfogalmak közé, így ennek jelentését meg kell adnunk.

5.5.2. Definíció *A $G' = (V', E')$ gráf a $G = (V, E)$ gráf topologikus részgráfja, ha $V' \subseteq V$ és $(u, v) \in E'$ akkor és csak akkor, ha u-ból vezet út v-be a G gráfban.*

Gráfok esetében használt fogalom a *súlyozott támogatottság*, melynek kiszámításához illeszkedési predikátum helyett illeszkedési függvényt használunk. Az illeszkedési függvény megadja a bemeneti gráf különböző részgráfjainak/feszített részgráfjainak/topologikus részgráfjainak számát, amely azonosak/izomorfak a mintagráffal. A G gráf súlyozott támogatottsága a bemeneti elemeken vett illeszkedési függvény összege.

Mielőtt rátérnénk az általános eset tárgyalására nézzük meg, hogyan lehet kinyerni a gyakori címkézett fákat.

5.5.3. Gyakori gyökeres fák

Ebben a részben feltesszük, hogy a mintatér és a bemeneti sorozat elemei csúcscímkézett gyökeres fák. Egy fa mérete a csúcsainak számát adja meg. Csak a címkék fontosak, ezért az illeszkedési predikátumnak a második fajtáját használjuk: akkor illeszkedik egy mintafa egy bementi fára, ha annak létezik olyan topologikus részgráfja, amellyel a mintafa izomorf.

A gyakori fák kinyerése hasznos a bioinformatikában, a webelemzésnél, a félig strukturált adatok vizsgálatánál stb. Az egyik legszemléletesebb felhasználási terület a webes szokások elemzése. Gyakori elemhalmaz-kinyerő algoritmussal csak azt tudnánk megállapítani, hogy melyek a gyakran látogatott oldalak. Ha gyakori szekvenciákat keresünk, akkor megtudhatjuk, hogy az emberek milyen sorrendben látogatnak el az oldalakra leggyakrabban. Sokkal élethűbb és hasznosabb információt kapunk, ha a weboldalakból felépített gyakori fákat (vagy erdőket) keresünk. Egy internetező viselkedését egy fa jobban reprezentálja, mint egy sorozat.

Rendezett gyökeres fáknál további feltétel, hogy az egy csúcsból kiinduló élek a gyerek csúcs címkéje szerint rendezve legyenek. Ez tulajdonképpen egy átmenet afelé, hogy az izomorf gráfokat ne különböztessük meg, vagy másként szólva a mintatérben ne legyenek izomorf gráfokat. Ha a címkék rendezése abc szerint történik, akkor például a következő 3 fa közül csak az első tartozik a mintatér elemei közé.

A rendezettség nem biztosítja azt, hogy a mintatérben ne legyenek izomorf fák. Például a következő ábrán látható két rendezett fa izomorf egymással, és mindketten a mintatérnek elemei.

Mivel az illeszkedés során izomorf részfákat keresünk, ezért feltehetjük, hogy a fa csúcsai természetes számok, és az i csúcs azt jelenti, hogy a csúcsot az i-edik lépésben látogatjuk meg a gráf preorder, mélységi bejárása során. Legyen a gyökér csúcs a 0. Az F fa i csúcsjának címkéjét $c_F(i)$-vel a szülőjét pedig

$szulo_F(i)$-vel jelöljük. Elhagyjuk az F alsó indexet azokban az esetekben, ahol ez nem okozhat félreértést.

Az 5.20 ábrán egy példát láthatunk illeszkedésre (topologikus részfára). A fák csúcsaiba írt számok a csúcsok címkéit jelölik. Az F' és F'' is illeszkedik a F fára.

Amennyiben egy gráf ritka (kevés élet tartalmaz), akkor azt szomszédossági listával (lásd alapfogalmak 2.4 rész) célszerű leírni. Fák esetében a *címkeláncok* még kevesebb helyet igényelnek a memóriából. A címkeláncot úgy kapjuk meg, hogy bejárjuk a fát preorder, mélységi bejárás szerint, és amikor új csúcsba lépünk akkor hozzáírjuk az eddigi címkelánchoz az új csúcs címkéjét. Amikor visszalépünk, akkor egy speciális címkét (*) írunk. Például az előző ábrán F címkelánca: $\mathcal{F} = 0, 1, 3, 1, *, 2, *, *, 2, *, *, 2, *$ és $\mathcal{F}' = 1, 1, *, 2, *$. Címkesorozatnak hívjuk és $l(F)$-vel jelöljük azt a sorozatot, amit a F gráf címkeláncából kapunk meg, ha elhagyjuk a $*$ szimbólumot. Nyilvánvaló, hogy a címkesorozat – a címkelánccal ellentétben – nem őrzi meg a fa topológiáját.

Hasonlóan a gyakori elemhalmazok kereséséhez most is megkülönböztetünk horizontális és vertikális adatábrázolási módot. Horizontális ábrázolásnál a bemenet gráfok leírásának (például címkelánc) sorozata. Vertikális tárolásnál minden címkéhez tartozik egy párokból álló sorozat. Az i címkéhez tartozó sorozatban a (j, k) pár azt jelenti, hogy a j-edik bemeneti gráf preorder bejárás szerinti k-adik csúcs címkéje i.

TreeMinerH

A TreeMinerH [Zaki, 2001] az APRIORI sémára épül (annak ellenére, hogy Zaki publikálta). Nézzük meg, hogyan állítjuk elő a jelölteket és hogyan határozzuk meg a támogatottságukat.

Jelöltek előállítása – Egy ℓ-elemű jelöltet két $(\ell-1)$-elemű gyakori fa (F' és F'') illesztésével (jelölésben: $\otimes$) kapjuk meg. Hasonlóan az eddigiekhez a két $(\ell-1)$-elemű fa csak a legnagyobb elemükben különböznek, amely esetünkben azt jelenti, hogy ha elhagynánk a legnagyobb csúcsot (és a hozzá tartozó élt), akkor ugyanazt a fát kapnánk. Az általánosság megsértése nélkül feltehetjük, hogy $szulo_{F'}(\ell-1) \leq szulo_{F''}(\ell-1)$, ahol $szulo_{F'}(\ell-1)$ az $(\ell-1)$-dik csúcs szülőjét jelöli, az összhasonlítás a szülő csúcsokban tárolt értékekre vonatkozik. Az 5.21. ábrán például $\ell = 4$ elemű jelöltek előállítását látjuk, $szulo_{F'}(\ell - 1) = szulo_{F'}(3) = szulo_{F'}(\text{"}4\text{"}) = \text{"}1\text{"}$ illetve $szulo_{F''}(\ell-1) = szulo_{F''}(3) = szulo_{F''}(\text{"}3\text{"}) = \text{"}2\text{"}$. A potenciális jelölt a F' gráf egy csúccsal való bővítése lesz, ahol az új csúcs címkéje a $c_{F''}(\ell-1)$ lesz.

Kételemű (egy élt tartalmazó) jelöltek előállításánál nincs sok választás: az új élt egyetlen helyre illeszthetjük. Ha $\ell > 2$, akkor két esetet kell megkü-

lönböztetnünk. Az első esetben $szulo_{F'}(\ell-1) = szulo_{F''}(\ell-1)$. Ekkor két jelöltet állítunk elő. Az elsőben az új élt a $szulo(\ell-1)$ csúcshoz, a másodikban a $szulo(\ell-1)+1$ csúcshoz kapcsoljuk. Ha $szulo_{F'}(\ell-1) < szulo_{F''}(\ell-1)$, akkor az új élt a $szulo_{F''}(\ell-1)$-hez csatoljuk. Jelölt-előállításra mutat példát a következő ábra:

Szokásos módon a jelöltek előállításának második lépésében minden $\ell-1$ elemű részfát ellenőrizni kell, hogy gyakori-e.

Támogatottság meghatározása

Az egy- és kételemű fák támogatottságát vektorral, illetve tömbbel célszerű meghatározni. A vektor i-edik eleme tárolja a támogatottságát az i-edik címkének. A tömb i-edik sorának j-edik eleme tárolja a támogatottságát annak a kételemű fának, amelyben a gyökér címkéje az i-edik gyakori címke, a másik csúcs címkéje a j-edik gyakori címke.

A kettőnél nagyobb elemszámú fák támogatottságának meghatározásánál szófa jellegű adatstruktúrát javasoltak. A szófát a fák címkesorozatai alapján építjük fel, de a levelekben a címkeláncot tároljuk. Egy levélben több jelöltfa is lehet, hiszen különböző fáknak lehet azonos a címkeláncuk. Amikor egy bemeneti fára illeszkedő jelölteket kell meghatároznunk, akkor a bemenet címkesorozata alapján eljutunk azokhoz a jelöltekhez, amelyek illeszkedhetnek a bemeneti fára. Egy jelölt címkesorozatának illeszkedése szükséges feltétele annak, hogy maga a jelölt is illeszkedjen a bemeneti fára. Ha eljutunk egy levélbe, akkor az ott található címkesorozatok mindegyikét megvizsgáljuk egyesével, hogy topologikusan illeszkedik-e a bemenet címkeláncra. Ennek részleteit nem ismertetjük.

TreeMinerV

A TreeMiner algoritmus [Zaki, 2002] Zaki módszerét használja, melyet az 5.3.5 részben mutattunk be. A vertikális adatbázisból kiindulva előállítja a egyelemű fák illeszkedési listáit és a továbbiakban már csak ezen listákkal dolgozik.

5.5.4. A gyakori feszített részgráfok

Ebben a részben bemutatjuk a legismerteb gyakori feszített részgráfokat kinyő algoritmust. A $\mathcal{MK} = (\mathcal{G}, \preceq)$-ben mintatér elemei címkézett egymással nem izomorf gráfok és $G' \preceq G$, ha G' a G-nek feszített részgráfja. A gráf méretét a csúcsainak száma adja meg. A bemenet címkézett gráfok sorozata. A G gráf támogatottságán azon bemeneti elemek a számát értjük, amelyeknek létezik G-vel izomorf feszített részgráfjuk (feszített részgráf fogalmát lásd a 2.4 részben).

Az AcGM algoritmus

Az AcGM algoritmus [**?**] – ami az AGM javított változata [**?**] – a gyakori feszített részgráfokat nyeri ki. Az algoritmus az APRIORI sémát követi. Ahhoz, hogy az összes összefüggő feszített részgráfot megtalálja előállítja a *félig összefüggő* feszített részgráfokat is. Egy gráf félig összefüggő, ha összefüggő, vagy két összefüggő komponensből áll, ahol az egyik komponens egyetlen csúcsot tartalmaz.

Az egész algoritmus során a gráfok szomszédsági mátrixszaival dolgozunk. A szomszédossági mátrix eredeti definíciója alapján nem tárolja a címkéket, ezért ebben a részben a $G = (V, E, c_V, c_E)$ gráf f bijekciójához tartozó $A_{G,f}$ szomszédossági mátrixának elemei (a_{ij} a mátrix i-edik sorának j-edik elemét jelöli):

$$a_{i,j} = \begin{cases} c(e_{ij}) & \text{, ha } i \neq j \text{ és } (f^{-1}(i), f^{-1}(j)) \in E, \\ c(f^{-1}(i)) & \text{, ha } i = j, \\ 0 & \text{, különben} \end{cases}$$

Az $A_{G,f}$ elemeiből és a csúcsok címkéiből egy kódot rendelhetünk a G gráfhoz:

$$CODE(A_{G,f}) = a_{1,1}, a_{2,2}, \ldots, a_{k,k}, c_V(f^{-1}(k)a_{1,2}, a_{1,3}, a_{2,3}, a_{1,4}, \ldots, a_{k-2,k}, a_{k-1,k},$$

azaz először felsoroljuk a csúcsok címkéit, majd a szomszédossági mátrix felső háromszög-mátrixának elemeit.

Különböző bijekciók különböző szomszédossági mátrixot, és így különböző kódokat eredményeznek. Amennyiben a címkéken tudunk egy rendezést definiálni, akkor a kódokat is tudjuk rendezni. Legyen a G gráf kanonikus kódolása az a kód, amelyik a legnagyobb ezen rendezés szerint. A kanonikus kódhoz tartozó szomszédossági mátrixot *kanonikus szomszédossági mátrixnak* hívjuk.

Az eddigiekhez hasonlóan most is azt kell tisztáznunk, hogy miként állítjuk elő a jelölteket és hogyan határozzuk meg a támogatottságukat.

Jelöltek előállítása

Az $X = A_{G',f}$ és $Y = A_{G'',g}$ $\ell \times \ell$ méretű szomszédossági mátrixokat, ahol G' összefüggő, G'' pedig félig összefüggő gráf, akkor illesztjük, ha teljesül három feltétel:

- Ha az X és Y-ból töröljük az utolsó sort és oszlopot, akkor azonos (T) mátrixot kapunk, és a csúcsok címkéi is rendre megegyeznek:

$$X_\ell = \begin{pmatrix} T & x_1 \\ x_2^T & x_{l,l} \end{pmatrix}, Y_\ell = \begin{pmatrix} T & y_1 \\ y_2^T & y_{l,l} \end{pmatrix},$$

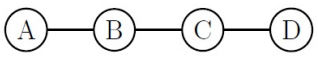
- T egy kanonikus szomszédossági mátrix,
- ha $x_{l,l} = y_{l,l}$, akkor legyen $code(X) < code(Y)$, ellenkező esetben $x_{l,l} < y_{l,l}$ vagy G' ne legyen összefüggő.

A potenciális jelölt szomszédossági mátrixa a következő lesz:

$$Z_{\ell+1} = \begin{pmatrix} T & x_1 & y_1 \\ x_2^T & x_{l,l} & z_{\ell,\ell+1} \\ y_2^T & z_{\ell+1,\ell} & y_{l,l} \end{pmatrix},$$

ahol $z_{\ell,\ell+1}$ és $z_{\ell+1,\ell}$ 0-át és az összes lehetséges élcímke értékét felvehetik. Irányítatlan gráfok esetében a két értéknek meg kell egyeznie. Az ilyen módon létrehozott szomszédossági mátrixot a szerzők *normál formájú* szomszédossági mátrixnak nevezik.

Az első feltétel szerint nem csak azt várjuk el, hogy a két ℓ-elemű illesztendő mintának legyen $(\ell - 1)$-elemű közös részmintája, hanem még azt is, hogy ez a részminta mindkét generátor prefixe is legyen. Tulajdonképpen ez biztosítja azt, hogy az illesztésként kapott jelölt mérete $\ell + 1$ legyen. Ha a második és harmadik feltételnek nem kellene teljesülnie, akkor sokszor ugyanazt a potenciális jelöltet hoznánk létre. Az algoritmus nem lenne teljes, amennyiben csak összefüggő gráfok lehetnének a generátorok. Az



gráfot például a fenti jelölt előállítással nem lehetne kinyerni.

Nézzünk egy példát. Az 5.22. ábrán két gyakori 3 csúcsú gráfot láthatunk, amelyből a jelölt előállítás során a jobb oldalon látható gráfot hozzuk létre. Az első gráf szomszédossági mátrixa $\left(\begin{smallmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{smallmatrix}\right)$, a másodiké $\left(\begin{smallmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{smallmatrix}\right)$, az illesztés során kapott szomszédossági mátrix pedig $\left(\begin{smallmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & z \\ 0 & 1 & z & 0 \end{smallmatrix}\right)$.

A jelölt-előállítás második fázisában minden ℓ elemű feszített részgráfról el kell dönteni, hogy gyakori-e. Amennyiben az összes részgráf gyakori, akkor a potenciális jelölt valódi jelölt lesz, ami azt jelenti, hogy meg kell határozni a támogatottságát.

Sajnos ez a második lépés nem annyira egyszerű, mint elemhalmazok, sorozatok, gyökeres fák esetében. A feszített részgráf egy szomszédossági mátrixát megkaphatjuk, ha töröljük a mátrix adott indexű sorát és oszlopát. Eközben figyelnünk kell arra, hogy egy gráfnak több szomszédossági mátrixa is létezhet.

Támogatottságok meghatározása – Mivel egy gráfnak több szomszédossági mátrixa is létezhet, a jelöltek előállítása után minden mátrixhoz hozzá

kell rendelni az általa reprezentált gráf kanonikus kódját. A továbbiakban már csak ezekkel dolgozunk, tehát csak ezekhez rendelünk – kezdetben 0 értékű – számlálókat.

A bemeneti gráfokat egyesével vesszük és minden jelöltet megvizsgálunk, hogy izomorf-e a bemeneti gráf valamely feszített részgráfjával. Feltételezzük, hogy a bemeneti mátrix kanonikus szomszédossági mátrixa rendelkezésünkre áll.

Ez a részfeladat tulajdonképpen a részgráf izomorfia feladata, amiről tudjuk, hogy NP-teljes. A feladatot azonban gyorsan megoldhatjuk, ha tudjuk, hogy a jelölt ℓ-elemű feszített részgráfja a bemeneti gráf melyik feszített részgráfjával volt izomorf. Nem kell mást tennünk, mint megvizsgálni, hogy az új csúcs és a hozzá tartozó él illeszkedik-e a bemeneti gráf részgráfjára.

5.5.5. A gyakori részgráfok keresése

Ebben a részben feltesszük, hogy a mintatér elemei összefüggő gráfok és $G' \preceq G$, ha G' a G gráfnak részgráfja. Eben a mintakörnyezetben egy gráf méretét az éleinek száma adja meg. A bemenet címkézett gráfok sorozata. A G gráf támogatottságán azon bemeneti elemeknek a számát értjük, amelyeknek létezik G-vel izomorf részgráfja. A következőkben áttekintjük az egyik legismertebb algoritmust, az FSG-t.

Az FSG algoritmus

Az FSG algoritmus [Kuramochi és Karypis, 2001] az APRIORI sémára épül. A gráfok tárolásához szomszédossági listát használ. Amikor egy gráfnak elő kell állítani a kanonikus kódját, akkor a szomszédssági listát szomszédossági mátrixá alakítja. Amennyiben a gráfok ritkák, a szomszédossági listák kevesebb helyet igényelnek, mint a mátrixok.

Megszokhattuk már, hogy a fő lépés a jelöltek előállítása.

Jelöltek előállítása – Két ℓ-elemű $G_1 = (V_1, E_1), G_2$ gráfot akkor illesztünk, ha van $(\ell-1)$-elemű közös részgráfjuk (ezt hívtuk magnak), és az G_1 kannonikus kódja nem nagyobb G_2 kannonikus kódjánál. Ez azt jelenti, hogy minden gráfot önmagával is illesztünk. Két gráf illesztésénél – akárcsak két elemsorozatok esetében – több gráf jön létre. Jelöljük a G_2-nek a magba nem tartozó élét $e = (u, v)$-vel. Az előállított gráfok a G_1 bővítése lesz egy olyan $e' = (u', v')$ éllel, amelyre $u' \in V_1$, $e' \notin E_1$, $c_E(e) = c_E(e')$, $c_V(u) = c_V(u')$ és $c_V(v) = c_V(v')$. Tehát egy megfelelően címkézett élt helyezünk be a G_1 gráfba. Ezt többféleképpen tehetjük, így több potenciális jelöltet hozunk létre.

Lehet, hogy az új él új csúcsot is fog eredményezni, de az is lehet, hogy csak két meglévő pont között húzunk be egy új élt. Ezt szemlélteti a 5.23 ábra.

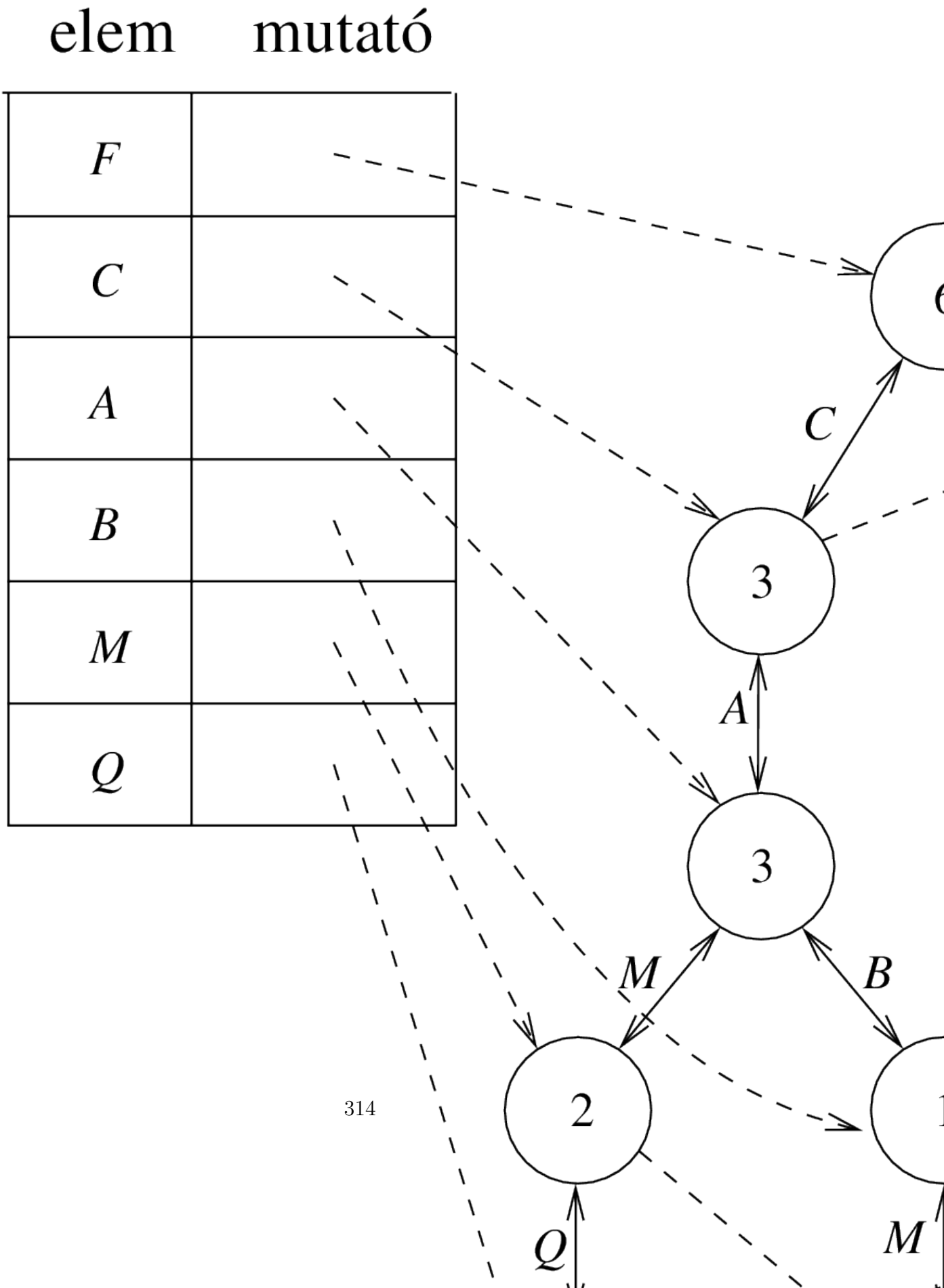
Az előállított potenciális jelöltek számát növeli az a tény is, hogy az új élt sokszor több csúcshoz is illeszthetjük. Erre mutat példát a következő ábra.

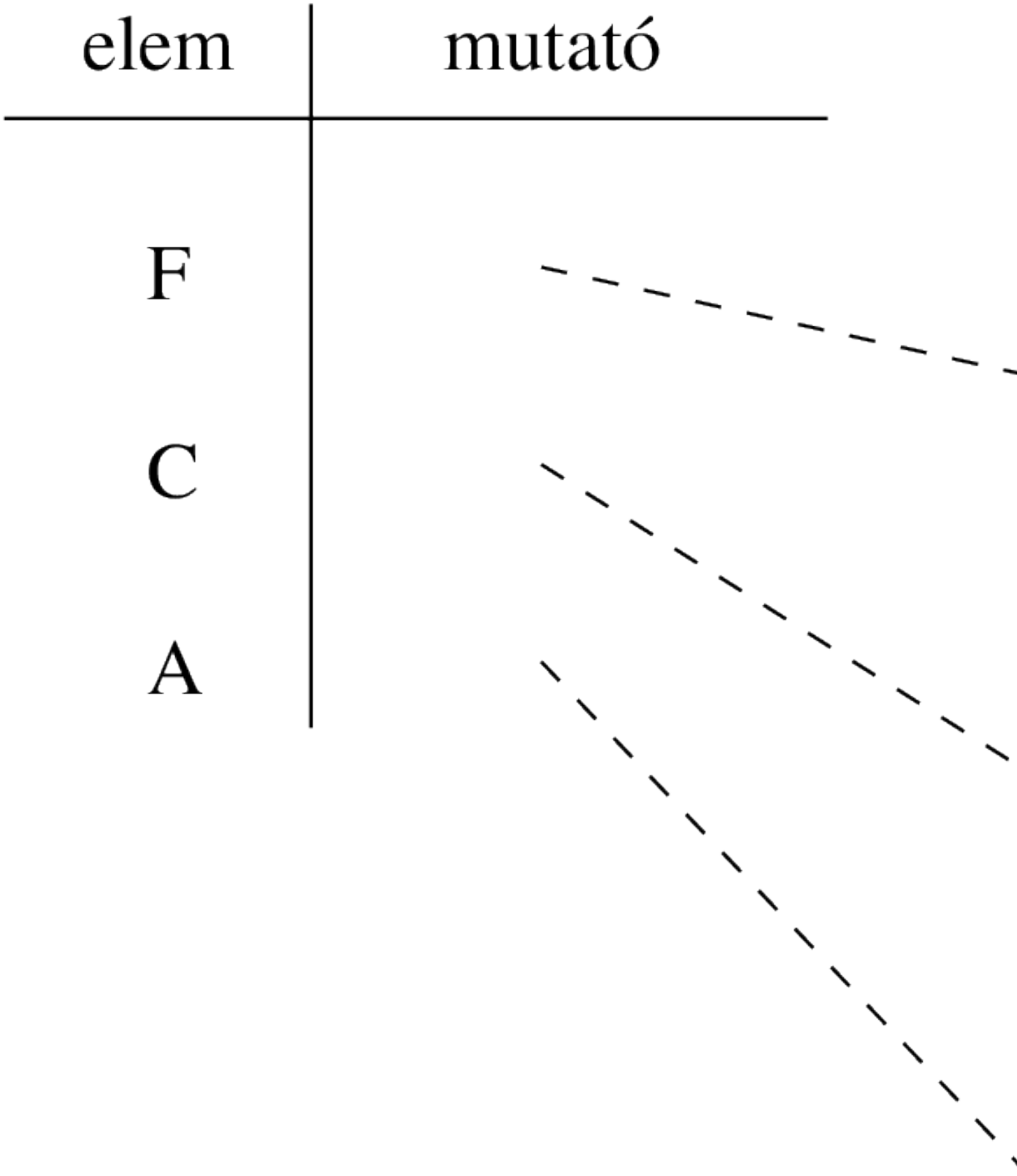
A harmadik ok, amiért két gráf több potenciális jelöltet állíthat elő az, hogy két gráfnak több közös részgráfja (magja) is lehet. Egy ilyen eset látható az 5.25 ábrán.

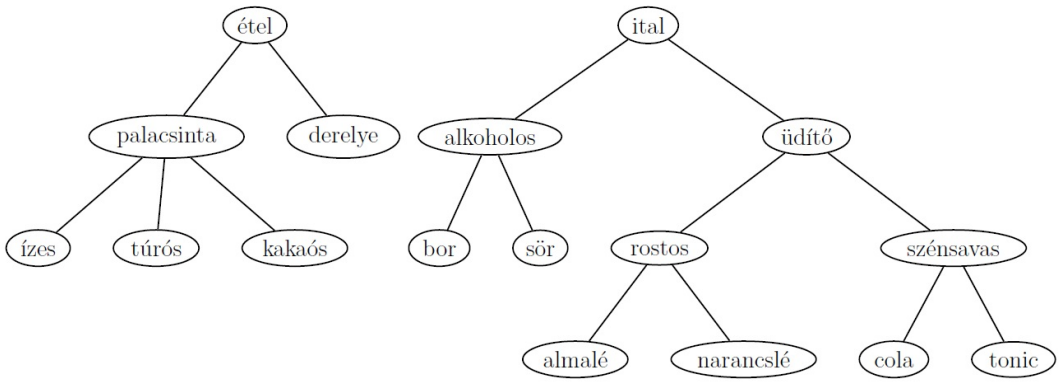
Miután előállítottuk a potenciális jelölteket, minden potenciális jelölt ($\ell - 1$)-elemű részgráfját ellenőrizzük, hogy gyakori-e. Azok a potenciális jelöltek leszenek jelöltek, amelyek minden valódi részhalmaza gyakori és még nem vettük fel a jelöltek közé. Ez utóbbi feltétel már sejteti, hogy a fenti jelölt-előállítás nem ismétlés nélküli. Az algoritmus a gráfok kanonikus kódolását használja annak eldöntésére, hogy egy potenciális jelölt adott részgráfja gyakori-e, illetve a jelölt szerepel-e már a jelöltek között.

A jelöl-előállításának tehát három fő lépése van: mag azonosítás (ha létezik egyáltalán), él-illesztés és a részgráfok ellenőrzése. Az első lépést gyorsíthatjuk, ha minden gyakori gráfnak egy listában tároljuk az ($\ell-1$)-elemű részgráfjainak kanonikus kódjait. Ekkor a közös mag meghatározása tulajdonképpen két lista metszetének meghatározását jelenti.

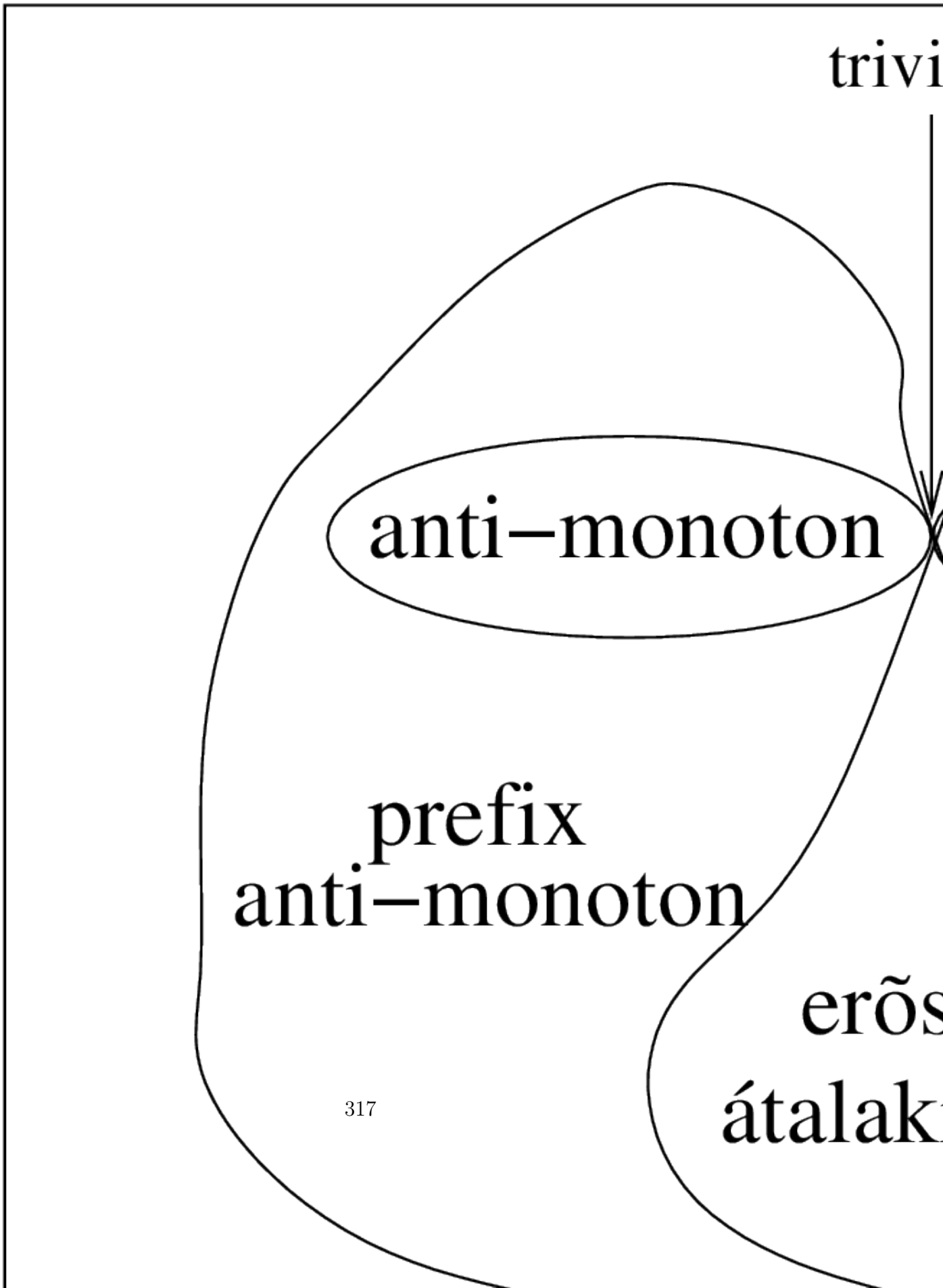
Támogatottság meghatározása – A bemeneti gráfokat egyesével vizsgálva meg kell határozni, hogy melyek azok a jelöltek, amelyek izomorfak a bemeneti gráf valamely részgráfjában. A részgráf izomorfia eldöntése NP-teljes, de ezen feladat eldöntésére használt lépések számát csökkenthejük, ha minden részgráfnak rendelkezésünkre áll a TID-hamaza, azon bemeneti gráfok sorszámai, amelyek tartalmazzák a részgráfot. Egy jelölt vizsgálatánál csak azon bemeneti elemeket kell megvizsgálnunk (ha ezek száma nagyobb min_supp-nál), amely sorszáma minden részgráf TID-halmazában szerepel.

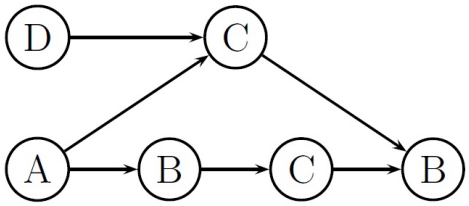

elem
mutató
F
C
A
B
M
Q
C
3
A
3
M
B
2
Q
M


elem
mutató
F
C
A



5.11. ábra. Példa: képzeletbeli büfé termék-taxonomiája


trivi
anti–monoton
prefix
anti–monoton
erõs
átalak



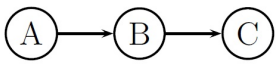
5.13. ábra. Példa: sorozat általánosítása

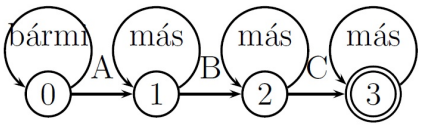
```
i_régi.számláló ← i_régi.számláló-1;
if( i_régi.számláló = 0)
   forall j in i_régi.epizódjai
   {
      j.hiányzás ← j.hiányzás+1;
      if( j.hiányzás = 1) then
         j.számláló ← j.számláló + j.kezdeti_index-jelenlegi_index;
   }
```

5.14. ábra. Régi elem kilépése

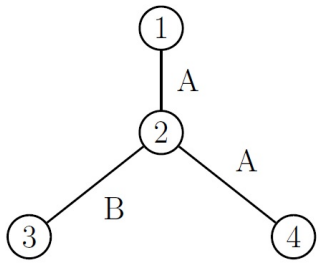
```
i_új.számláló ← i_új.számláló+1;
if( e_új.számláló = 1 )
   forall j in i_új.epizódjai
   {
      j.hiányzás ← j.hiányzás-1;
      if j.hiányzás=0 then
         j.kezdeti_index ← jelenlegi_index;
   }
```

5.15. ábra. Új elem belépése

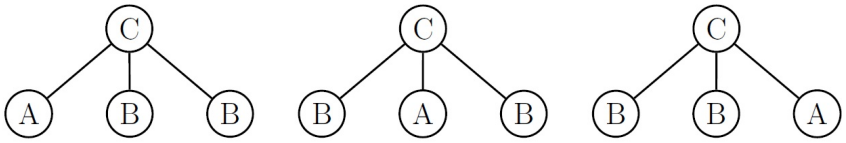




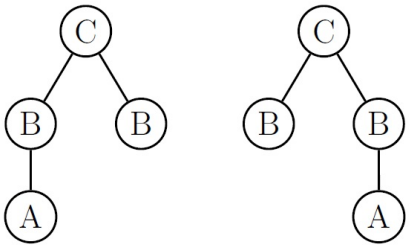
5.16. ábra. Példa: automata előállítása epizód alapján



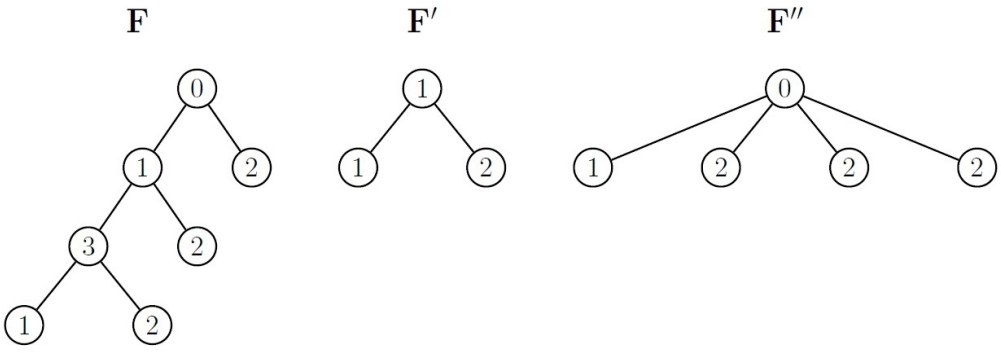
5.17. ábra. Példa kanonikus kódolásra



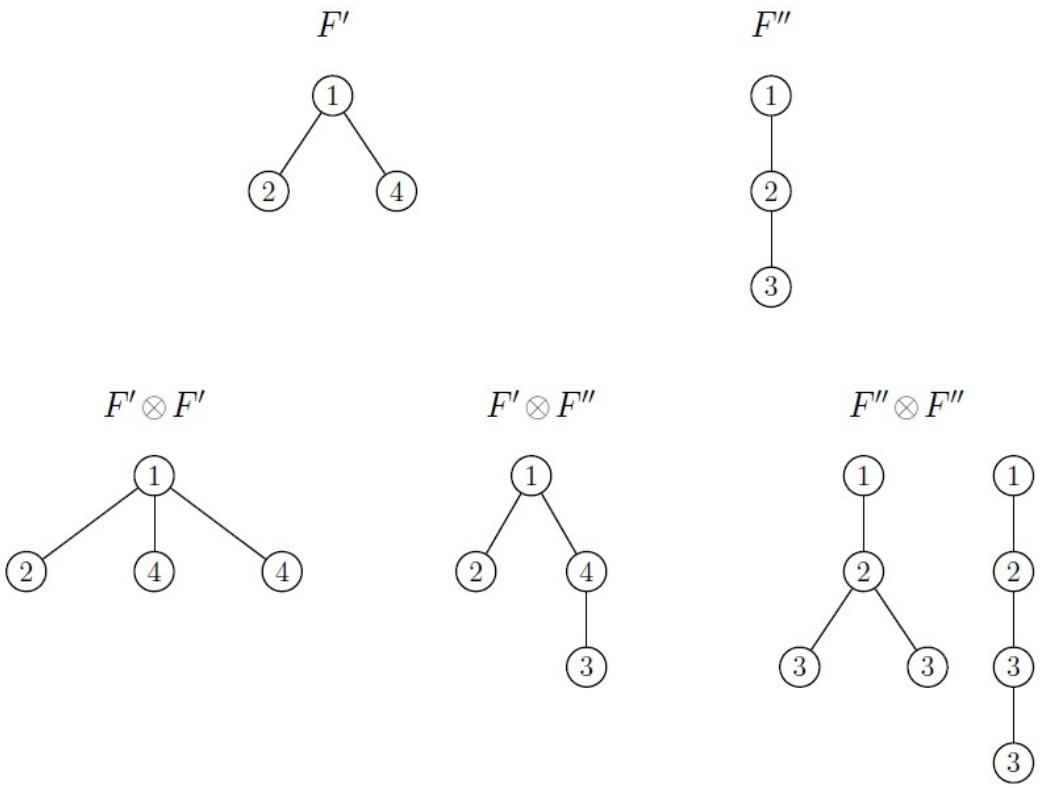
5.18. ábra. Példa: rendezés nélküli, címkézett, gyökeres fák



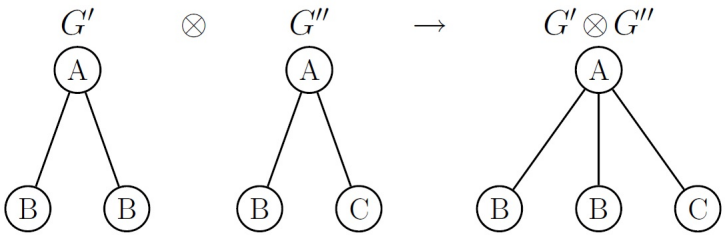
5.19. ábra. Példa: izomorf rendezett, gyökeres fák



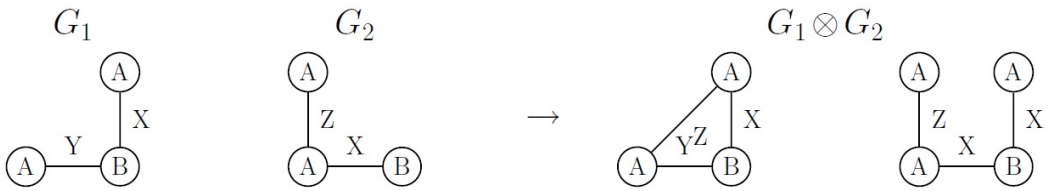
5.20. ábra. Példa: gyökeres részfák tartalmazására



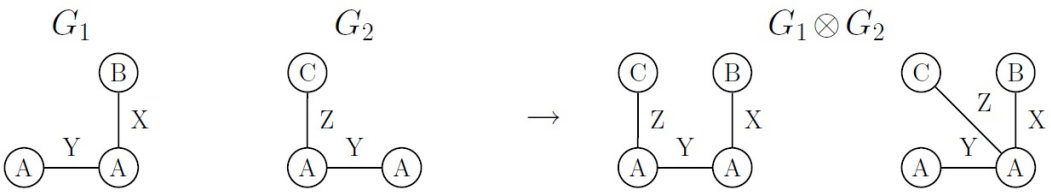
5.21. ábra. Példa jelöltek előállítására



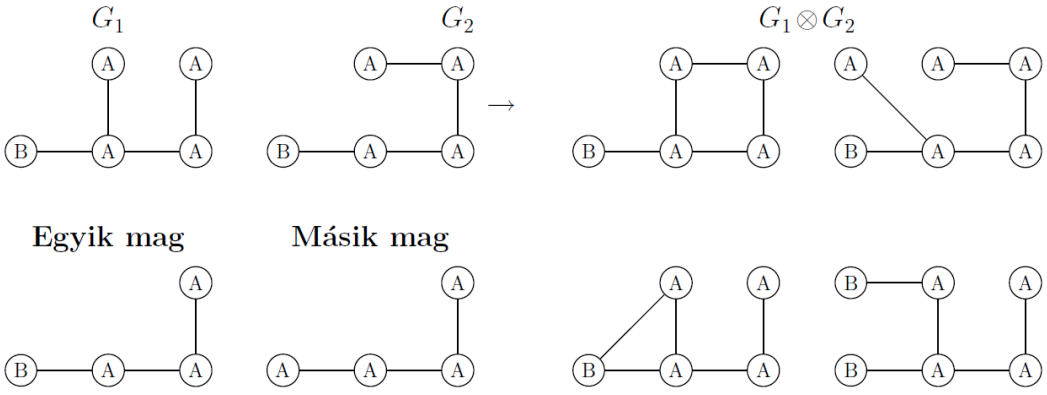
5.22. ábra. Példa jelöltek előállítására



5.23. ábra. Példa: gráf illesztése



5.24. ábra. Példa: gráf illesztése - az új élt több csúcshoz is illeszthetjük



5.25. ábra. Példa: gráf illesztése - több közös mag

6. fejezet

Klaszterezés

Klaszterezésen elemek csoportosítását értjük. Úgy szeretnénk a csoportosítást elvégezni, hogy a hasonló elemek ugyanazon, míg az egymástól eltérő elemek külön csoportba kerüljenek.

A klaszterezés az adatbányászat egyik legrégebbi és leggyakrabban alkalmazott eszköze. Csoportosítanak ügyfeleket, weboldalakat, géneket, betegségeket stb. Az egyik legdinamikusabban fejlődő terület a személyre szabott szolgáltatásoké, ahol az ügyfeleket, ill. vásárlókat klaszterezik (csoportosítják), és az egyes csoportokat eltérően kezelik. A klaszterezésre azért van szükség, mert az ügyfelek számossága miatt a kézi kategorizálás túl nagy költséget jelentene.

Gyakran nem az a fontos, hogy az egyes elemeket melyik csoportba soroljuk, hanem az, hogy mi jellemző a különböző csoportokra. Például egy banki stratégia kialakításánál nem érdekel bennünket, hogy Kis Pista melyik csoportba tartozik, hanem csak az, hogy milyen ügyfélcsoportokat célszerű kialakítani és ezekre a csoportokra mi jellemző. A klaszterezés segítségével egy veszteséges tömörítést végeztünk. A teljes ügyfeleket tartalmazó adatbázist egy kisebb, átláthatóbb, emészthetőbb ügyfélcsoport adatbázissá alakítottuk. A klaszterezést tehát egy nagy adatbázis struktúrájának feltárására is használhatjuk, arra, hogy egy jó áttekintő képet kapjuk arról, hogy milyen objektumok találhatók az adatbázisban.

Sajnos a „jó” csoportok kialakítása nem egyértelmű feladat, hiszen az emberek gyakran más-más szempontokat vesznek figyelembe a csoportosításnál. Ugyanazt azt adathalmazt, alkalmazástól és szokásoktól függően, eltérően klaszterezneák az emberek. Például az 52 darab francia kártyát sokan 4 csoportra osztanák (szín szerint), sokan 13-ra (figura szerint). A Black Jack játékosok 10 csoportot hoznának létre (ott a 10-es, bubi, dáma, király között nincs különbség), míg a Pikk Dáma játékot kedvelők hármat (pikk dáma, a körök és a többi lap). Klaszterezéskor tehát az adathalmaz mellett meg kell adnunk, hogy miként definiáljuk az elemek hasonlóságát, továbbá, hogy mi alapján csopor-

tosítsunk (összefüggő alakzatokat keressünk, vagy a négyzetes hibát minimalizáljuk stb.). Egy-egy konkrét alkalmazásban azonban legtöbbször sikerül az adott alkalmazás szempontjából megfelelő, gyakorlatban jól használható módon definiálni a jóságot.

A jóság általános és egzakt definíciójának hiánya mellett problémát jelent az óriási keresési tér. Ha n pontot akarunk k csoportba sorolni, akkor a lehetséges csoportosítások számát a Stirling számok adják meg:

$$\mathcal{S}_n^{(k)} = \frac{1}{k!}\sum_{i=0}^{k}(-1)^{k-i}\binom{k}{i}i^n.$$

Még egy egészen pici adathalmaz mellett is megdöbbentően sokféleképpen csoportosíthatunk. Például 25 elemet 5 csoportba $\mathcal{S}_{25}^{(5)} = 2,436,684,974,110,751$ különböző módon partíciónálhatunk. Ráadásul, ha a csoportok számát sem tudjuk, akkor a keresési tér még nagyobb ($\sum_{k=1}^{25}\mathcal{S}_{25}^{(k)} > 4\cdot 10^{18}$).

Az osztályozáshoz hasonlóan a klaszterezés során is csoportokba soroljuk az elemeket. Az osztályozással ellentétben, a klaszterezés során a csoportok nincsenek előre megadva, a klaszterezési feladat része a csoportok megtalálása. Ebből adódóan, az osztályozással ellentétben, a klaszterezésnél *nincs* tanítóhalmaz, amelynek objektumaról (példányairól) előre tudnánk, hogy azok mely csoportokba tartoznak. Ezért hívják a klaszterezést a *felügyelet nélküli tanulás*i eljárások (unsupervised learning) közé sorolják.

A fejezet további részében először egy elsőre meghökkentőnek látszó kutatási eredményről számolunk be és ezt értékeljük, majd a hasonlóság meghatározásával foglalkozunk, végül a legismertebb klaszterező algoritmusokat mutatjuk be.

6.1. Legfontosabb lépések a klaszterezés elméleti alapjainak megértéséhez

A klaszterezés az egyik legnehezebben átlátható adatbányászati terület. Napról napra újabb és újabb cikkek jelennek meg különböző „csodaalgoritmusokról", amelyek szupergyorsan és helyesen csoportosítják valamely adatbázis objektumait. Míg az algoritmusokat igazoló teszteredményekből nincs hiány, az elméleti elemzésekről viszonylag ritkák, a klaszterezési feladat elméleti alapjainak teljeskörű megértése, a klaszterezés szilárd elméletének kidolgozása irányába csak viszont kevés előrelépés történt. Ebben a részben a klaszterezés elméleti alapjaival kapcsolatos legfontosabb eredmények közül mutatunk be néhányat.

Amint már utaltunk rá, az egyik alapvető kérdés, hogy mikor nevezzük az objektumok egy csoportosítását jónak. Ezzel egy külön alfejezetben fogunk

foglalkozni (6.4. fejezet). Az, hogy egy konkrét esetben egy klaszterező eljárás "jó" eredményt adott (akármit is értsünk az alatt, hogy "jó"), nyilván nem jelenti azt, hogy az eljárás egy "jó" klaszterező algoritmusnak tekinthető: lehet, hogy csak véletlenül volt szerencsénk az adott algoritmussal. A következőkben azzal foglalkozunk, hogyan tudjuk formalizálni azt, hogy mikor jó egy *klaszterező eljárás*, milyen követelményeket támaszthatunk egy klaszterező algoritmussal szemben.

6.1.1. Kleinberg lehetetlenség-elmélete

Ebben a részben Jon Kleinberg „An Impossibility Theorem for Clustering (A Klaszterezés egy lehetetlenség-elmélete)" című munkáját [Kleinberg, 2002] tekintjük át, amely az adatbányászatra az utóbbi évtizedben legnagyobb hatást kiváltó művek egyike. A cím már sejteti az elszomorító eredményt, miszerint *nem létezik jó, távolság alapú*[1] *klaszterező eljárás*! Ezt a meglepő állítást úgy bizonyítja, hogy három tulajdonságot mond ki, amellyel egy klaszterező eljárásnak rendelkeznie kell, majd belátja, hogy nem létezhet klaszterező eljárás, amelyre ez igaz. A tulajdonságok az alábbiak:

Skála-invariancia: Ha minden elempár távolsága helyett annak az α-szorosát vesszük alapul (ahol $\alpha > 0$), akkor a klaszterező eljárás eredménye ne változzon.

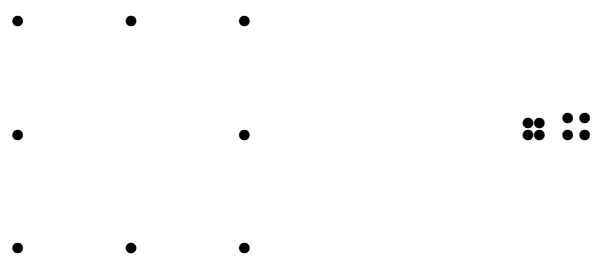
Gazdagság (richness): Tetszőleges előre megadott csoportosításhoz tudjunk megadni távolságokat úgy, hogy a klaszterező eljárás az adott módon csoportosítson.

Konzisztencia: Tegyük fel, hogy a klaszterező eljárás valahogy csoportosítja az elemeket. Ha ezután tetszőleges, azonos csoportban lévő elempárok között a távolságot csökkentjük, illetve külön csoportban lévő elempárok távolságát növeljük, akkor az újonnan kapott távolságok alapján működő eljárás az eredetivel megegyező csoportosítást adja.

A fenti tulajdonságok teljesen természetesek, azt gondolnánk, hogy minden algoritmus ilyen. Ezért nem túl bíztató a következő tétel:

6.1.1. Tétel *Amennyiben az elemek száma nagyobb 1-nél, akkor nem létezik olyan klaszterező eljárás, ami rendelkezik a Skála-invariancia, a Gazdagság és a Konzisztencia tulajdonságokkal.*

[1] A különbözőség megállapításához használt távolságfüggvénynek szemi-metrikának kell lennie, tehát a háromszög egyenlőtlenségnek nem kell teljesülnie



6.1. ábra. Példa arra, hogy a konzisztencia, mint elvárás nem mindig egyezik az emberi intuícióval.

Kleinberg azt is bebizonyítja, hogy bármely két tulajdonsághoz létezik klaszterező eljárás, amely rendelkezik a választott tulajdonságokkal. Például a single-linkage eljárás (lásd 6.7.1. rész) skála-invariáns és konzisztens. Ezen kívül az is igaz, hogy a partíciónáló algoritmusok (pl.: k-means, k-medoid), ahol a cél a középpontoktól vett távolság függvényének (például négyzetes hiba összege) minimalizálása, nem konzisztensek.

Vitatkozhatunk azon, hogy a konzisztencia jogos elvárás-e egy klaszterező algoritmussal szemben. Tekintsük a 6.1. ábrát. Bal oldalon láthatjuk az eredetileg megadott pontokat, jobb oldalon pedig az átmozgatás során kapottakat. Legtöbben a bal oldali pontokat egy csoportba vennék (nagy négyzetet reprezentáló pontok), a jobb oldalon láthatókat viszont két külön csoportba sorolnák (két kis négyzethez tartozó pontok). A klaszteren belüli távolságokat tehát csökkentettük, a klaszterezés mégis megváltozott, azaz klaszterezési eljárásunk nem rendelkezik a konzisztencia tulajdonsággal.

Kleinberg erre az észrevételre is tud elszomorítóan reagálni. A konzisztencia fogalmát lazíthatjuk. Amennyiben a klasztereken belüli távolságokat csökkentjük, a klaszterek közötti távolságokat növeljük, és ezáltal bizonyos klaszterek kisebb klaszterekké bomlanak, akkor a klaszterező eljárás *finomítás–konzisztens*. Belátható, hogy nem létezik olyan klaszterező eljárás, ami skála-invariáns, gazdag és finomítás–konzisztens.

Ha viszont a gazdagságból is engedünk egy kicsit, nevezetesen, hogy a klaszterező algorimus sose tudjon minden pontot külön klaszterbe sorolni – de tetszőleges más módon tudjon particionálni –, akkor létezik klaszterező eljárás, amely kielégíti a három tulajdonságot.

Érvelhetünk úgy, hogy a finomítás-konzisztencia közelebb áll az emberi intuícióhoz, mint a konzisztencia eredeti definíciója. Hasonlóképpen: ha elemek (objektumok) *csoportosítását* keressük, vajon tényleg lényeges, hogy az algoritmus képes legyen megtalálni azon triviális csoportosítást, amikor minden elem (objektum) külön csoportba kerül? Egy ilyen csoportosítás semmivel sem

.....

.....

6.2. ábra. Példa: az emberi intuíciónak megfelelő klaszterezés nem mindig a távolság-alapúl klaszterezés.

mond többet, mint az eredeti adatbázis, ezért érvelhetünk úgy, hogy a gazdagság módosított definíciója legalább annyira "jó", mint az eredeti. Akár azt is felvethetjük, vajon megfelelőek-e a kleinbergi elvárások, nem lenne-e célszerűbb valamilyen más módon formalizálni azt, hogy milyen kritériumoknak megfelelő klaszterező algoritmust tekintünk jónak.

Mielőtt továbblépnénk, gondolkodjunk el azon, hogy jogos-e a hasonlóságot és különbözőséget pusztán egy távolság alapján definiálni. A klaszterezés eredeti célja az, hogy a hasonló elemek egy csoportba, míg a különböző elemek eltérő csoportba kerüljenek. Ebből következik, hogy egy tetszőleges elem különbsége (távolsága) a saját csoportbeli elemeitől kisebb lesz, mint a különbsége más csoportban található elemektől. Biztos, hogy jó ez? Biztos, hogy az ember is így csoportosít, tehát ez a természetes klaszterezés? Sajnos nem lehet a kérdésre egyértelmű választ adni. Van amikor az ember így csoportosít, van, amikor máshogy. Tekintsük a 6.2. ábrán elhelyezkedő pontokat. Valószínűleg kivétel nélkül minden ember két csoportot hozna létre, az alsó szakaszhoz tartozó pontokét és a felső szakaszhoz tartozó pontokét. Mégis, ha megnézzük, akkor az alsó szakasz bal oldali pontja sokkal közelebb van a felső szakasz bal oldali pontjaihoz, mint azokhoz a pontokhoz, amelyek az alsó szakasz jobb oldalán helyezkednek el. Mégis ragaszkodunk ahhoz, hogy a bal- és jobboldali pontok egy csoportba kerüljenek. Úgy érezzük, egymáshoz tartoznak, mert mindannyian az alsó szakasz elemei.

Következésképpen a klaszterezés célja – az eredetivel szemben – gyakran az, hogy úgy csoportosítsunk, hogy egy csoportba kerüljenek az elemek akkor, ha ugyanahhoz az absztrakt objektumhoz (fogalomhoz) tartoznak, és különbözőbe, ha más absztrakt objektum részei. A klaszterezés nehézsége pont abban rejlik, hogy automatikusan kell felfedezni az absztrakt objektumokat, fogalmakat az elemek alapján, ami ráadásul nem egyértelmű feladat.

Ha a klaszterezés során az absztrakt objektumokat összefüggő alakzatok formájában keressük (pl. vonal, gömb, amőba, pálcikaember stb.) akkor van esély jól megoldani a feladatot. A lehetetlenség-elmélet tehát nem zárja ki az összefüggő alakzatokat felfedező eljárás létezését.

Kleinberg lehetetlenségelmélete mellett, a klaszterezéshez kapcsolódó negatív eredmények között említjük meg, hogy a klaszterezési feladat – több, különbözőféle formalizálás mellett is – NP-nehéz [Krivánek és Morávek, 1986, Aloise és tsa., 2009, Buza, 2011b].

6.1.2. Stabilitás és 'Klaszterezhetőség'

Kleinberg lehetetlenségelméletét a kutatók egy része "kincset érő gyöngyszemként" és a "helyes irányvonal megvilágításaként", mint "megalapozott elméleti eredményt" ünnepelték. Akik valamilyen okból egyébként sem szerették a klaszterezést, igazolva látták benne, hogy a klaszterezés tudományos szempontból néve értelmetlen, hiszen még olyan klaszterező algoritmus sem létezik, amely az egyszerű kleinbergi feltételének megfelelne. Vegyük azonban észre, hogy az ilyen megállapítások már a szubjektív értékelés részét képezik, nem pedig objektív, bebizonyított igazságot: Kleinberg tétele nem többet (és nem kevesebbet!) állít, csak annyit, hogy nincs olyan klaszterező algoritmus, amely az általa definiált három kritériumot egyidejűleg teljesíti. Nem tudjuk azonban, hogy tényleg *jogosak-e* a kleinbergi kritériumok. Ha gyanakodnánk, hogy nem jogosak, akkor ezt erősítené az is, hogy láttuk, hogy Kleinberg eredeti kritériumait kicsit változtatva már olyan kritériumokat kapunk, amelyek egyidejűleg teljesíthetők, amelyek értelmében létezik jó klaszterező algoritmus. Nyilván nincs értelme arról vitatkozni, hogy vajon a konzisztencia, vagy a finomítás-konzisztencia a jobb kritérium, de megpróbálhatjuk más módon definiálni azt, hogy mikor tekintünk egy klaszterező algoritmust jónak.

Megkísérelhetjük a jó klaszterező algoritmus fogalmát az algoritmus stabilitásán keresztül definiálni [Ben-David, 2006]. Ennek alapgondolata a következő: természetes elvárásunk egy jó klaszterező algoritmussal szemben, hogy a bemeneti adatok minimális változtatása mellett nagyjából ugyanazt a csoportosítást kapjuk. Hamar kiderül azonban, hogy még egy jó klaszterező algoritmus sem feltétlenül teljesíti ezt az elvárást: a klaszterezési feladatnak ugyanis több közel-optimális megoldása lehet, azaz több, lényegesen különböző csoportosítás is lehet lényegében egyformán jó megoldása egy adott klaszterezési feladatnak. Hogy a több, különböző közel-optimális megoldás közül egy adott bemenetre melyik a legjobb, illetve melyiket találja meg egy jó klaszterező eljárás, nagyban függhet a bemeneti adatoktól, a bemeneti adatok minimális változtatása lényegesen befolyásolhatja az egyébként kíváló klaszterező algoritmus által adott csoportosítást. A csoportosítás előbbi értelemben vett stabilitása tehát sokkal inkább a megoldás unikalitását méri, mintsem a klaszterező algoritmus jóságát.

Margareta Ackerman és Shai Ben-David bevezették egy adatbázis 'klaszterezhetőségének' fogalmát [Ackerman és Ben-David, 2009]. Intuitíve: akkor

klaszterezhető jól egy adatbázis, ha a klaszterek jól elkülönülnek egymástól, kevés az átfedés. Formálisan nyilván a klaszterezhetőség fogalmát is többféle módon definiálhatjuk, amint az említett szerzőpáros is több lehetséges definíciót tekintett. Mint említettük, a klaszterezési feladat NP-nehéz. Ackerman és Ben-David viszont arra a meglepő eredményre jutottak, hogy a feladat NP-nehézsége ellenére, amennyiben egy adatbázis jól klaszterezhető, akkor létezik olyan algoritmus, amely gyorsan megtalál egy közel-optimális klaszterezést.

6.2. Hasonlóság mértéke, adatábrázolás

Adott n elem (vagy más néven objektum, egyed, megfigyelés stb.). Amint a 3.2. fejezetben láttuk, tetszőleges két elem (x és y) között értelmezzük a *távolságukat* vagy más szóval: *különbözőségüket*, $d(x, y)$-t. A $d(x, y)$-tól elvárjuk metrika legyen (a metrika definícióját lásd a 3.2. fejezetben).

A klaszterezés legáltalánosabb esetében minden egyes elempár távolsága előre meg van adva. Az adatokat ekkor egy ún. távolság mátrixszal reprezentáljuk:

$$\begin{bmatrix} 0 & d(1,2) & d(1,3) & \cdots & d(1,n) \\ & 0 & d(2,3) & \cdots & d(2,n) \\ & & 0 & \cdots & d(3,n) \\ & & & \ddots & \vdots \\ & & & & 0 \end{bmatrix},$$

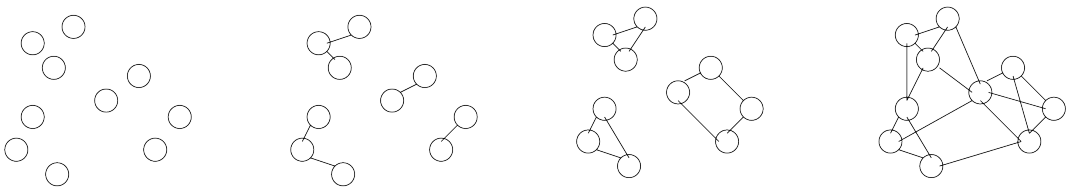
ahol $d(i, j)$ adja meg az i-edik és a j-edik elem különbözőségét.

A gyakorlatban az n elem (vagy objektum) attribútumokkal van leírva, és a különbözőséget az attribútumok alapján definiálhatjuk valamilyen *távolságfüggvénnyel*. Ha megadjuk a távolságfüggvényt, akkor elvben felírhatjuk a fenti mátrixot. Sok esetben azonban az elemek száma olyan nagy, hogy a mátrix rengeteg helyet foglalna. Modellünkben ezért rendelkezésünkre állnak az attribútumokkal megadott elemek halmaza és a távolságfüggvény. Az n értéke nagy lehet, így nem tehetjük fel, hogy az adatok elférnek a memóriában.

Sokszor fogjuk a klaszterezést gráfparticionálási feladatként tekinteni. Az elemekre tekinthetünk úgy, mint egy $G = (V, E)$ súlyozott, irányítatlan, teljes gráf pontjaira, ahol az éleken található súlyok a távolságot (vagy a hasonlóságot) adják meg. Az $(u, v) \in E$ él súlyát $w(u, v)$-vel jelöljük.

Vannak algoritmusok, amelyek nem az eredeti gráfon dolgoznak, hanem az úgynevezett *k-legközelebbi szomszéd gráf*on, amit G_k-val jelölünk. G_k-ban is a pontoknak az elemek, az éleken található súlyok pedig a hasonlóságoknak felelnek meg, de itt csak azokat az éleket tároljuk, amelyek egy pont és annak k k legközelebbi szomszédait kötik össze. A 6.3. ábrán ilyen gráfokat láthatunk.

A k legközelebbi szomszéd reláció, amint arról már volt szó, asszimetrikus, ezért a k-legközelebbi szomszéd gráfok valójában irányított gráfok. Az élek irányításától azonban most eltekintünk.



6.3. ábra. Példa k-legközelebbi szomszéd gráfokra k=0,1,2,3 esetén

Ha az adathalmazt a k-legközelebbi szomszéd gráffal ábrázoljuk, akkor ugyan vesztünk némi információt, de a lényeg megmarad, és jóval kevesebb helyre van szükségünk. Az egymástól nagyon távoli elemek nem lesznek összekötve G_k-ban. További előny, hogy amennyiben egy klaszter sűrűségét a benne található élek összsúlyával mérjük, akkor a sűrű klasztereknél ez az érték nagy lesz, ritkáknál pedig kicsi.

6.3. A klaszterek jellemzői

A C klaszter elemeinek számát $|C|$-vel jelöljük. A klaszter „nagyságát” próbálja megragadni a klaszter *átmérője* ($D(C)$). A két legelterjedtebb definíció az elemek közötti átlagos, illetve a maximális távolság:

$$D_{avg}(C) = \frac{\sum_{p\in C}\sum_{q\in C} d(p,q)}{|C|^2},$$

$$D_{max}(C) = \max_{p,q\in C} d(p,q).$$

Ízlés kérdése, hogy a klaszter átmérőjének számításakor figyelembe vesszük-e a pontok önmaguktól vett távolságát (ami 0). Nyugodtan használhatjuk az átmérő $D'_{avg}(C) = \frac{\sum_{p,q\in C, p\neq q} d(p,q)}{\binom{|C|}{2}}$ definícióját is. A *klaszterek közötti távolság*ot ($d(C_i, C_j)$) is többféleképpen értelmezhetjük.

Minimális távolság: $d_{min}(C_i, C_j) = \min_{p\in C_i, q\in C_j} d(p,q)$.

Maximális távolság: $d_{max}(C_i, C_j) = \max_{p\in C_i, q\in C_j} d(p,q)$.

Átlagos távolság: $d_{avg}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{p \in C_i} \sum_{q \in C_j} d(p, q)$, ami a külön klaszterben lévő pontpárok átlagos távolságát adja meg.

Egyesített klaszter átmérője: $d_D(C_i, C_j) = D(C_i \cup C_j)$

A vektortérben megadott elemeknél gyakran használt fogalmak a klaszter *középpont*ja ($\vec{m}_C$) és a *sugara* (R_C).

$$\vec{m}_C = \frac{1}{|C|} \sum_{p \in C} \vec{p},$$

$$R_C = \frac{\sum_{p \in C} |\vec{p} - \vec{m}_C|}{|C|}.$$

A klaszterek közötti távolság mérésére pedig gyakran alkalmazzák a középpontok közötti távolság értékét:

$$d_{mean}(C_i, C_j) = |\vec{m}_i - \vec{m}_j|.$$

Az átlagok kiszámításánál – például átmérő, sugár esetében – számtani közepet használtunk. Bizonyos cikkekben négyzetes közepet alkalmaznak helyette. Tulajdonképpen tetszőleges közép használható, egyik sem rendelkezik elméleti előnnyel a többivel szemben. Gondoljuk meg azonban, hogy a hatvány alapú közepeknél jóval nagyobb számokkal dolgozunk, így ezek számítása esetleg nagyobb átmeneti tárat kíván.

A négyzetes középnek előnye a számtani középpel szemben, hogy könnyű kiszámítani, amennyiben vektortérben dolgozunk. Ezt a BIRCH algoritmusnál (6.7.3. rész) is kihasználják, ahol nem tárolják a klaszterekben található elemeket, hanem csak 3 adatot: $|C|$, $\vec{LS}_C = \sum_{p \in C} \vec{p}$, $SS_C = \sum_{p \in C} \vec{p}\vec{p}^T$. Könnyű belátni, hogy a fenti három adatból két klaszter (C_i, C_j) közötti átlagos távolság (és hasonlóan az egyesített klaszter átmérője) közvetlenül adódik: $d_{avg}(C_i, C_j) = \frac{SS_{C_i} + SS_{C_j} - 2\vec{LS}_{C_i}\vec{LS}_{C_i}^T}{|C_i||C_j|}$.

6.4. A klaszterezés „jósága”

Mint már említettük, a klaszterezés jósága alkalmazásspecifikus, nem lehet minden szempontot kielégítő, objektív mértéket adni. Ennek ellenére néhány függvény minimalizálása igen elterjedt a klaszterező algoritmusok között.

A továbbiakban n darab elemet kell k rögzített számú csoportba sorolni úgy, hogy a csoportok diszjunktak legyenek, és minden csoportba kerüljön legalább egy elem.

6.4.1. Klasszikus mértékek

Az alábbi problémákat különböztetjük meg a minimalizálandó célfüggvény alapján:

Minimális átmérő probléma: Célunk itt a legnagyobb klaszterátmérő minimalizálása. Átmérőnek ez esetben D_{max}-ot szokás használni.

k-medoid probléma: Válasszuk ki az n elem közül k ún. reprezentáns elemet, amelyek a minimális hibaösszeget adják. Egy elem hibája a hozzá legközelebbi reprezentáns elem távolsága. A feladat NP-nehéz, még akkor is, ha olyan síkba rajzolható gráfokra szorítkozunk, amelyeknek a maximális fokszáma 3 (ha a gráf fa, akkor már lehet polinomrendű algoritmust adni, $p = 2$ esetében a feladat lineáris időben megoldható)[Kariv és Hakimi, 1979]. A feladat NP-nehéz marad, ha a gráf Euklideszi térbe képezhető, sőt, konstans szorzó erejéig közelítő megoldást adni, még ilyenkor is, nehéz feladat [Megiddo és Supowitz, 1984].

k-center probléma: Ez a feladat a k-medián módosítása, csak itt a hibaösszeg helyett a legnagyobb hibát kell minimalizálni.

k-klaszter probléma: Célunk itt a klaszteren belüli távolságösszegek azaz

$$\sum_{i=1}^{k} \sum_{p,q \in C_i} d(p,q) = \sum_{i=1}^{k} |C|^2 D_{avg}(C_i)$$

minimalizálása. A feladat (és konstans szorzó erejéig annak közelítése) NP-nehéz $k \geq 2$ ($k \geq 3$) esetén [Sahni és Gonzales, 1976].

Legkisebb (négyzetes) hibaösszeg: Csoportosítsuk úgy a pontokat, hogy a középpontoktól való távolság összege

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} (|\vec{p} - \vec{m}_{C_i}|)$$

minimális legyen. Nyilvánvaló, hogy ez a megközelítés csak olyan esetekben használható, amikor értelmezni tudjuk a klaszterek középpontját ($\vec{m}_{C_i}$-t). Sok esetben a középpontoktól való távolságösszeg helyett a távolság négyzeteinek összegét kívánjuk minimalizálni:

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} (|\vec{p} - \vec{m}_{C_i}|^2)$$

Legkisebb (négyzetes) hibaösszeg probléma eléggé hasonlít a k-klaszter problémához.

6.4.1 Észrevétel $\sum_{i=1}^{k} \sum_{p,q \in C_i} d(p,q) = \sum_{i=1}^{k} \sum_{p \in C_i} ||p - \vec{m}_{C_i}||^2$, *ahol* $\vec{m}_C = \frac{1}{|C|} \sum_{q \in C} \vec{q}$.

Bizonyítás:

$$\sum_{i=1}^{k} \sum_{p \in C_i} ||p - \vec{m}_{C_i}||^2 = \sum_{i=1}^{k} \sum_{p \in C_i} ||p - \frac{1}{|C|} \sum_{q \in C_i} \vec{q}||^2 = \sum_{i=1}^{k} \sum_{p \in C_i} \sum_{q \in C_i} \frac{1}{|C|} ||p - q||^2$$
$$= \sum_{i=1}^{k} \frac{1}{|C_i|} \sum_{(p,q) \in C_i} ||p - q||^2 = \sum_{i=1}^{k} |C_i| D_{avg}(C_i)$$

Azok az algoritmusok, amelyek a fenti célfüggvényeket minimalizálják, az elemeket kis kompakt felhőkbe csoportosítják. Ez valamennyire elfogadhatónak tűnik, azonban ezeknek a megközelítéseknek számos súlyos hátránya van.

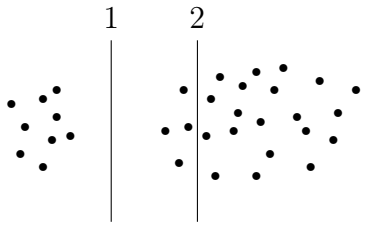
1. Legfontosabb, hogy csak elliptikus klasztereket generál, tehát tetszőleges amőba alakú, de kompakt klasztert felvág kisebb kör alakú klaszterekre.

2. Rosszul csoportosít, ha a klaszterek között nagyok a méretkülönbségek. Ennek oka az, hogy a nagy klaszterben lévő pontok távol esnek a középponttól, ami nagy hibát eredményez. Tehát hiába kompakt egy nagy klaszter, a hibát minimalizáló algoritmusok kis részekre fogják felosztani.

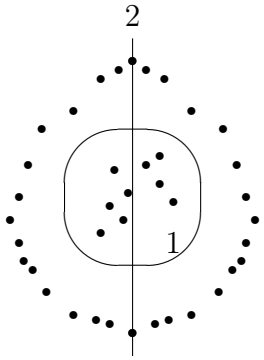
3. A négyzetes hibaösszeget minimalizáló eljárások további hibája, hogy érzékeny a távol eső (outlier) pontokra, hiszen egy távoli pont a klaszter középpontját nagyon „elhúzhatja".

Elrettentő példaként nézzük a 6.4. és 6.5. ábrákon látható pontokat. A 6.4. ábrá pontjain, ha a maximális átmérőt minimalizáljuk, akkor a 2. egyenes alapján osztjuk ketté a pontokat. Ennek ellenére minden „rendszerető" ember a két csoportot inkább az 1-es egyenes mentén szeparálná. A gyenge klaszterezés oka, hogy a klaszterekben belüli maximális eltérést annak árán minimalizáljuk, hogy sok különböző pont egy klaszterbe kerül. (Megjegyzés: ugyanezt a rossz eredményt kapnánk, ha a közepektől való távolságot, esetleg a távolságösszeget akarnánk minimalizálni.)

A 6.5. ábrán látható pontokat a 2-medián problémát megoldó algoritmusok a 2-es egyenes szerint csoportosítanák.



6.4. ábra. Hibás klaszterezés: eltérő méretű klaszterek esetén



6.5. ábra. Hibás klaszterezés: egymást tartalmazó klaszterek esetén

6.4.2. Konduktancia alapú mérték

Az adatbányászat terjedésével hamar kiderült, hogy az előbb bemutatott klasszikus mértékeknek nem minden gyakorlati esetben sikerül jellemezniük, hogy mennyire "jó" egy adott csoportosítás. Új mértékek, új megközelítések születtek. Ezek közül az egyik legígéretesebb a konduktancia alapú mérték [Kannan és tsa., 2000].

Tekintsünk az adathalmazunkra, mint egy $G = (V, E)$ gráfra, de most az éleken található súly a hasonlósággal legyen arányos, ne pedig a távolsággal (különbözőséggel). Jelöljük egy $T \subseteq E$ élhalmazban található élek súlyainak összegét $w(T)$-vel ($w(T) = \sum_{e \in T} w(e)$), $C \subseteq V$ klaszterben található elemek számát $|C|$-vel, $E(S) = E(V \backslash S)$-el (kiolvasva: edge(S)-sel illetve edge(V kivéve S)-sel) pedig az $(S, V \backslash S)$ vágást keresztező élek halmazát: $E(S) = \{(p, q) | p \in S, q \in V \backslash S\}$.

Vizsgáljuk azt az egyszerű esetet, amikor $k = 2$, tehát a gráf pontjait két részre akarjuk osztani. Klaszterezésnél az egyik célunk, hogy az elemeket úgy csoportosítsuk, hogy a különböző elemek külön klaszterbe kerüljenek. Ez alapján mondhatnánk, hogy egy minimális összsúlyú vágás jól osztaná ketté a pontokat. Sajnos ez a módszer legtöbb esetben kiegyensúlyozatlan (nagyon eltérő méretű) csoportokat hozna létre. Gondoljuk meg, hogy ha az egyik klaszterben csak 1 elem található, akkor $n - 1$ súlyt kell összegezni, míg egyenletes

kettéosztásnál ugyanez az érték $(\frac{n}{2})^2$.

A vágás összsúlya helyett tehát célszerű olyan mértéket bevezetni, amely figyelembe veszi valahogy a gráf kiegyensúlyozottságát is, és kisebb jelentőséget tulajdonít az olyan vágásnak, amely kis elemszámú részhez tartozik. Egy gráf *kiterjedése* (expansion) a vágás súlya és a vágást alkotó két ponthalmaz közül a kisebbik elemszáma közti arányt méri. Formálisan az $(S, V\backslash S)$ vágás kiterjedése:

$$\varphi(S) = \frac{w(E(S))}{\min(|S|, n - |S|)}.$$

Látható, hogy a számláló a kis vágásértéket, míg a nevező a kiegyensúlyozottságot preferálja. Egy gráf kiterjedése pedig a vágások minimális kiterjedése, egy klaszter kiterjedését pedig a hozzá tartozó részgráf kiterjedésével definiálhatjuk. A klaszterezés jóságát, ez alapján, a klaszterek minimális kiterjedésével adhatjuk meg.

Sajnos a kiterjedés képletében a nevező nem veszi figyelembe az élek súlyait. Azt szeretnénk, hogy azok a pontok, amelyek nagyon különbözőek az összes többi ponttól, kisebb összsúllyal szerepeljenek a „jóság" definíciójában, mint azok a pontok, amelyeknek jóval több ponthoz hasonlítanak. A kiterjedés általánosítása a *konduktancia* (conductance).

6.4.1. Definíció *Legyen $G = (V, E)$ gráf egy vágása $(S, V\backslash S)$. A vágás konduktanciáját a következőképpen definiáljuk:*

$$\phi(S) = \frac{w(E(S))}{\min(a(S), a(V\backslash S))},$$

ahol $a(S) = \sum_{p\in S, q\in V} w(p, q)$.

A gráf konduktanciája pedig a vágások minimális konduktanciája: $\phi(G) = \min_{S\subseteq V} \phi(S)$.

A konduktancia könnyen általánosítható k klaszter esetére. Egy $C \subseteq V$ klaszter konduktanciája megegyezik a vágásai legkisebb konduktanciájával, ahol az $(S, C\backslash S)$ vágás konduktanciája: $\phi(S) = \frac{\sum_{p\in S, q\in C\backslash S} w(p,q)}{\min(a(S), a(C\backslash S))}$. Egy klaszterezés konduktanciája a klaszterek minimális konduktanciájával egyezik meg. A klaszterezés célja tehát az, hogy keressük meg azt a klaszterezést, ami a legnagyobb konduktanciát adja. A 6.4 és a 6.5 ábrákon látható pontokat a konduktancia alapú klaszterező eljárások helyesen csoportosítják.

Sajnos a konduktancia alapú mérték még nem tökéletes. Ha például egy jó minőségű klaszter mellett van néhány pont, amelyek mindentől távol esnek, akkor a klaszterezés minősége igen gyenge lesz (hiszen a minőség a leggyengébb klaszter minősége). A probléma egy lehetséges kiküszöbölése, ha a klaszterezés

minősítésére két paramétert használunk. A konduktancia mellett bevezethetjük azt a mértéket, amely megadja, hogy az összes él súlyának hányad részét nem fedik le a klaszterek.

6.4.2. Definíció *A* $\{C_1, C_2, \ldots, C_k\}$ *a* (V, E) *gráf egy* (α, ϵ)*-partíciója, ha:*

1. *minden* C_i *klaszter konduktanciája legalább* α*,*
2. *a klaszterek közötti élek súlya legfeljebb* ϵ *hányada az összes él súlyának.*

A klaszterezés célja ekkor az lehet, hogy adott α mellett találjunk olyan (α, ϵ)-partíciót, amely minimalizálja ϵ-t, vagy fordítva (adott ϵ mellé találjunk olyan (α, ϵ)-partíciót, amely maximalizálja α-t). A feladat NP-nehéz.

6.4.3. Referencia-klaszterekhez való viszonyítás

Eddig végig azt feltételeztük, hogy a helyes csoportosítás teljesen ismeretlen. Néhány alkalmazásban előfordulhat, hogy az adatok egy (kis) részének (az adott alkalmazás szemszögéből nézett) helyes csoportosítása ismert, és "csak" az a feladat, hogy az adatbázis többi objektumát (jellemzően az objektumok nagyobb részét) helyesen csoportosítsuk. Ha eközben nem adódnak új csoportok (és ezt előre tudjuk), akkor osztályozási feladattal van dolgunk. Ha azonban a még csoportosítatlan objektumok csoportosítása várhatóan új csoportokhoz fog vezetni, klaszterező eljárásokat használhatunk. Ekkor az adatoknak azt a részhalmazát, amelyen már ismert a helyes csoportosítás, használhatjuk a klaszterezés minőségének becslésére: megnézhetjük, hogy az algoritmus által adott csoportok mennyiben illeszkednek a helyes csoportokra, az osztályozó algoritmusok kiértékelésénél megismert precision-t, recall-t, F-measure-t számíthajtuk a referenciaklaszterekhez viszonyítva [Radovanović, 2011]. Ha egy ilyen protokollt követünk, azzal az implicit feltételezéssel élünk, hogy ha a klaszterező algoritmus jól csoportosítja az adatbázis azon objektumait, amelyekről már előre tudjuk, hogy mely csoportokba tartoznak, akkor várhatóan a többit is jól fogja csoportosítani.

Példa – Egy klaszterező algoritmus által adott csoportosĂtást referencia-csoportokhoz viszonyítjuk. Az alábbi táblázat mutatja az egyes klaszterekbe tartozó példányok számát referecia-csoportonként (nyilván csak az adatbázis azon részét tekintve, amelyre ismerjük a helyes csoportosítást). Az egyszerűség kedvéért a referencia-csoportokat osztályoknak nevezzük. A példán keresztül azt mutatjuk be, hogyan számíthatjuk ki a klaszterező algoritmus eredményének aggregált F-measure-jét.

	1. klaszter	2. klaszter	3. klaszter	4. klaszter
1. osztály	40	20	15	15
2. osztály	5	20	5	10
3. osztály	5	10	5	25

Úgy tekintjĂźk, hogy a j-dik klaszter válasz egy olyan lekérdezésre, amelyre a helyes válasz az i-dik osztálybeli példányok halmaza. Kiszámoljuk az összes osztály-klaszter párra a precision-t és recall-t. Jelöljük precision(i, j)-val illetve recall(i, j)-val a j-dik klaszter i-dik osztályra vonatkozó precision-jét illetve recall-ját:

precision(1,1) = 40 / 50 precision(1,3) = 15 / 25
recall(1,1) = 40 / 90 recall(1,3) = 15 / 90
precision(1,2) = 20 / 50 precision(1,4) = 15 / 50
recall(1,2) = 20 / 90 recall(1,4) = 15 / 90
precision(2,1) = 5 / 50 precision(2,3) = 5 / 25
recall(2,1) = 5 / 40 recall(2,3) = 5 / 40
precision(2,2) = 20 / 50 precision(2,4) = 10 / 50
recall(2,2) = 20 / 40 recall(2,4) = 10 / 40
precision(3,1) = 5 / 50 precision(3,3) = 5 / 25
recall(3,1) = 5 / 45 recall(3,3) = 5 / 45
precision(3,2) = 10 / 50 precision(3,4) = 25 / 50
recall(3,2) = 10 / 45 recall(3,4) = 25 / 45

Ezt követően F-measure-t számolunk az osztály-klaszter-párokra az

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

összefüggés alapján:

f(1,1) = 2*0.8*0.44 / (0.8+0.44) = 0.57
f(1,2) = 2*0.4*0.22 / (0.4+0.22) = 0.28
...

Így kapjuk, hogy:

f(1,1) = <u>0.57</u> f(2,1) = 0.11 f(3,1) = 0.11
f(1,2) = 0.28 f(2,2) = <u>0.44</u> f(3,2) = 0.21
f(1,3) = 0.26 f(2,3) = 0.15 f(3,3) = 0.14
f(1,4) = 0.21 f(2,4) = 0.22 f(3,4) = <u>0.53</u>

A klaszterezés aggregált F-measurjének számításához minden osztályra megnézzük, hogy melyik klaszter adja a maximális F-measure-t (ezt jelöltük alá-húzással), és ezeket súlyozottan átlagoljuk az osztályok mérete alapján:

$$F_{aggr} = (90/175) \times 0.57 + (40/175) \times 0.44 + (45/175) \times 0.53 = 0.53.$$

6.4.4. Klaszterező algoritmusok feladat-alapú kiértékelése

Egy klaszterezési feladat gyakran egy nagyobb feladat részfeladatként fordul elő. Ilyen lehet például az ügyfelek klaszterezése online kereskedelemben, azért, hogy jobb személyre szabott ajánlatokat tehessünk. A korábban már említett ajánlórendszerek hátterében futó ajánló algoritmusok között számos akad, amelyik inputként – az egyes ügyfelek által vásárolt termékek mellett – az ügyfelek valamilyen klaszterezését is fogadják. Minél jobban használható egy adott csoportosítás a személyre szabott ajánlatok előállításához, annál jobb, ezen feladat kontextuában, az adott csoportosítás. Az eddigiek látott kiértékelési lehetőségek mellett egy alternatív lehetőség tehát csoportosítások, és ezáltal egy klaszterező algoritmusok, egy adott feladat kontextusában történő kiértékelése: ha adott egy F feladat, melynek részfeladata egy klaszterezési feladat, azt a klaszterező algoritmust tekintjük a legjobbnak, amelynek segítségével a legjobban tudjuk megoldani az F feladatot. Ezt az eljárást feladat-alapú kiértékelésnek (task-based evaluation) nevezik.

6.5. Klaszterező algoritmusok típusai

A szakirodalomban jónéhány klaszterező algoritmus található. Az egyes alkalmazások jellegétől függ, hogy melyik algoritmust célszerű választani. A klaszterező algoritmusokat 5 kategóriába sorolhatjuk.

Partíciós módszer: A partíciós módszerek a pontokat k diszjunkt csoportra osztják úgy, hogy minden csoportba legalább egy elem kerüljön. A csoportok a klasztereknek felelnek meg. Egy kezdeti particionálás után egy újraparticionálási folyamat kezdődik, mely során egyes pontokat más csoportba helyezünk át. A folyamat akkor ér véget, ha már nem „mozognak" az elemek.

Hierarchikus módszer: A hierarchikus módszerek a klaszterekből egy hierarchikus adatszerkezetet (általában fát, amit a szakirodalomban *dendogram*nak neveznek) építenek fel.

Spektrál módszerek: Spektrál módszerek közé soroljuk az olyan algoritmusokat, amelyek a csoportok meghatározásához az adathalmazt reprezentáló mátrix sajátértékeit, illetve sajátvektorait használja fel.

Sűrűség-alapú módszerek: A legtöbb klaszterező algoritmus csak elliptikus alakú klasztereket tud kialakítani. A sűrűség-alapú módszerek ennek a hibának a kiküszöbölésére születtek meg. Az alapvető ötlet az, hogy egy klasztert addig növesztenek, amíg a sűrűség a „szomszédságban" meghalad egy bizonyos korlátot. Pontosabban: egy klaszteren belüli elemekre mindig igaz, hogy adott sugarú körön belül mindig megtalálható bizonyos számú elem. A sűrűség-alapú módszereket a klaszterezés mellett kivételek, kívülálló elemek felderítésére (outlier analysis) is alkalmazzák.

Grid-alapú módszerek: A grid-alapú módszerek az elemeket rácspontokba képezik le, és a későbbiekben már csak ezekkel a rácspontokkal dolgoznak. Ezeknek az algoritmusoknak a gyorsaság a fő előnyük.

Klaszterező algoritmusokkal Dunát lehetne rekeszteni. Szinte bármilyen „butuska" klaszterező algoritmushoz tudunk generálni olyan adathalmazt, amit az fog a legjobban csoportosítani. Sajnos ezt a tényt a cikkek szerzői is gyakran kihasználják. A végeredményen kívül akadnak még szempontok, amelyeket meg lehet vizsgálni az egyes klaszterező algoritmusoknál. A legfőbb elvárásaink az alábbiak lehetnek:

Skálázhatóság: Sok algoritmus csak akkor hatékony, ha az elemek elférnek a memóriában. Gyakorlati alkalmazásokban időnként olyan nagy adatbázisokat kell feldolgozni, hogy ez a feltétel nem tartható.

Adattípus: Vannak algoritmusok, amelyek csak intervallum típusú attribútumokkal megadott elemeken működnek. Nyilvánvaló, hogy ez a feltétel szűkíti az alkalmazások körét.

Tetszőleges alakú, méretű és sűrűségű klaszterek: A legtöbb klaszterező algoritmus csak elliptikus klasztereket képes felfedezni. A gyakorlati életben azonban ritkán elliptikusak a klaszterek. Jogos elvárás, hogy az algoritmus akár amőba alakú, sőt egymásba ágyazódó klasztereket is meg tudjon határozni. Emellett jól tudjon csoportosítani eltérő méretű és sűrűségű elemhalmazokat.

Előzetes ismeretek: Elvárjuk, hogy az algoritmusok automatikusan meghatározzák a szükséges klaszterek számát. Sajnos vannak algoritmusok, amelyeknek előre meg kell adni ezt a paramétert.

Zajos adatok, távol eső elemek kezelése: A legtöbb adatbázis tartalmaz valamekkora zajt, kivételes, a többségtől távol eső elemeket. Rossz tulajdonsága egy algoritmusnak, ha ezeknek az elemeknek nagy hatása van a klaszterek kialakítására.

Adatok sorrendjére való érzékenység: Miért fogadnánk el az algoritmus eredményét, ha az teljesen megváltozik, mihelyt más sorrendben vesszük az elemeket? Az eredményként kapott klaszterek nem függhetnek az adatok feldolgozásának sorrendjétől.

Dimenzió: Bizonyos algoritmusok csak alacsony dimenzió esetén hatékonyak. Vannak azonban olyan alkalmazások, ahol az elemek nagyon sok paraméterrel vannak leírva, azaz az elemeket egy magas dimenziójú tér elemeiként kell felfognunk.

Értelmezhetőség: A felhasználók azt várják el a klaszterező algoritmusoktól, hogy olyan klasztereket találjanak, amelyek jól meghatározott jegyekkel bírnak, és viszonylag könnyű magyarázatot adni arra, hogy milyen tulajdonságú elemek tartoznak az egyes klaszterbe.

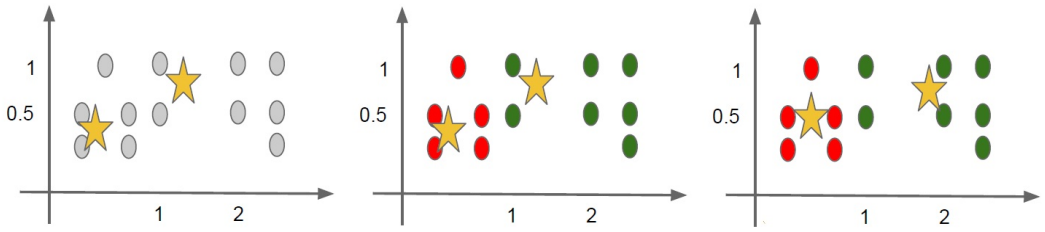
6.6. Particionáló eljárások

A particionáló algoritmusoknál a csoportok száma előre adott (k). Azért nevezzük ezeket az eljárásokat particionáló eljárásoknak, mert a legelső lépésben particionáljuk az elemeket, és a továbbiakban csak áthelyezgetünk bizonyos elemeket az egyik részből a másikba. Akkor kerül egy elem egy másik részbe, ha ezáltal javul a klaszterezés minősége. A klaszterezés minőségére az egyes partíciós algoritmusok eltérő célfüggvényt használnak. Egy lépés során a célfüggvény javítására általában több lehetőség is kínálkozik. Általában az algoritmusok a legjobbat választják ezek közül, tehát a „legmeredekebb lejtő" irányába lépnek a célfüggvény völgyekkel teli görbéjén.

6.6.1. Forgy k-közép algoritmusa

A k-közép algoritmus (k-means algorithm) [Forgy, 1965] az egyik legrégebbi (1965-ből származik) és legegyszerűbb klaszterező algoritmus vektortérben megadott elemek csoportosítására. A klaszterezés minőségének jellemzésére a négyzetes hibafüggvényt használja.

Az algoritmus menete a következő: kezdetben választunk k darab véletlen elemet. Ezek reprezentálják eleinte a k klasztert. Ezután besorolunk minden pontot ahhoz a klaszterhez, amely reprezentáns eleméhez az a leginkább hasonló. A besorolás után új reprezentáns pontot választunk, éspedig a klaszter középpontját. A *besorolást*, és az ezt követő *új középpont választást* addig ismételjük, amíg történik változás. Az algoritmus egy iterációjára láthatunk példát a 6.6. ábrán.



6.6. ábra. A k-means algoritmus egy iterációja $k = 2$ klaszter esetén. A példában feltesszük, hogy az adatbázis objektumai egy kétdimenziós tér pontjainak felelnek meg, azaz két szám típusú attribútummal rendelkeznek. A baloldali ábra mutatja az iteráció elején érvényes klaszerközéppontokat, a középső ábrán az egyes elemek klaszterközéppontokhoz való hozzárendelése látható, a jobboldali ábrán az új (a k-means következő iterációjának kezdetén érvényes) klaszterközéppontokat ábrázoltuk.

Jancey 1966-ban Forgy-tól teljesen függetlenül ugyanezt az algoritmust javasolta egy apró módosítással [Jancey, 1966]. Az új reprezentáns pont ne az új középpont legyen, hanem a régi és az új középpontot összekötő szakasz valamely pontja, például a szakasz középső pontja. Ez egy visszafogottabb, kisebb lépésekben haladó algoritmus. A kisebb lépések előnye, hogy esetleg nem lesznek túllövések, túl nagy oszcillációk. Elméletileg azonban egyikről sem lehet elmondani, hogy jobb lenne a másiknál, bizonyos helyzetekben az egyik, máskor a másik ad jobb eredményt.

Az algoritmus szép eredményeket hoz, amennyiben a klaszterek egymástól jól elszigetelődő kompakt „felhők”. Előnye, hogy egyszerű és jól skálázható, futási ideje $O(nkt)$, ahol t az iterációk számát jelöli. A k-közép algoritmusnak azonban számos hátránya van.

1. Lehet, hogy az algoritmus lokális optimumban áll meg, tehát az iterációk során nem változik semmi, mégis létezik olyan csoportosítás, ahol a négyzetes hiba kisebb.

2. Csak olyan elemek csoportosítására használható, amelyek vektortérben vannak megadva, hiszen értelmezni kell az elemek középpontját. Ezek szerint a k-közép nem használható olyan alkalmazásokban, ahol az elemek attribútumai között például kategória típusú is szerepel.

3. Rendelkezik a négyzetes hibát minimalizáló algoritmusok minden hibájával (lásd a 6.4.1-es részt).

A lokális optimumba kerülés esélyének csökkentése érdekében érdemes az algoritmust többször futtatni különböző kezdeti pontokkal. Azt a csoportosítást

fogadjuk el, amelynek legkisebb a négyzetes hibája. Vegyük észre, hogy ez a megoldás erős heurisztika! Minél nagyobb n, az adatbázisbeli objektumok (elemek) száma, elvben annál több lokális optimum lehet, annál nagyobb az esélye, hogy lokális optimumban kötünk ki. Ismereteink szerint nincs olyan képlet, ami megmondja, hogy adott elemszám esetén hányszor kell futtatni az algoritmust, hogy biztosan (vagy adott valószínűséggel) megtaláljuk a globális optimumot.

6.6.2. A k-közép néhány további változata

A k-közép algoritmusnak számos változata létezik. Ha például az új klaszterközéppontokata klaszterbe tartozó elemek (attribútumonkénti) mediánjaként számítjuk, ahelyett, hogy a klaszterbe tartozó elemek attribútumonkénti átlagát vennénk, akkor a négyzetes hibaösszeg helyett az átlagos abszolút hibát (mean absolute error) optimalizálja az algoritmus [Tan és tsa., 2005]. Az előző mondatban a hiba alatt a klaszterközépponttól való távolság értendő.

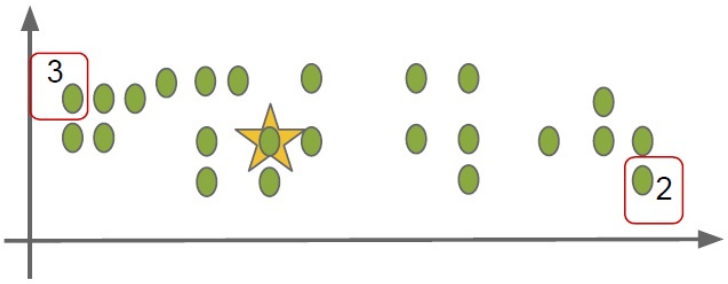
Többféleképp választhatjuk a kezdeti klaszterközéppontokat is: ahelyett, hogy a kezdeti klaszterközéppontokat véletlenszerűen választjuk az adatbázisbeli objektumok (elemek) közül, választhatjuk a klaszterközéppontokat tudatosan úgy, hogy távol legyenek egymástól. Az első két klaszterközéppontnak választhatjuk például a két legtávolabbi elemet. A további klaszterközéppontokat egyesével választjuk úgy, hogy klaszterközéppontnak mindig azt az elemet választjuk, amelyre a már kiválasztott klaszterközéppontoktól mért távolságok összege (vagy négyzetösszege) maximális.

Egy másik lehetőség, hogy az első klaszerközéppontot véletlenszerűen választjuk, majd – az előbbihez hasonlóan – a további klaszterközéppontokat úgy választjuk, hogy mindig azt az elemet választjuk klaszterköéppontnak, amelyre a már kiválasztott klaszterközéppontoktól mért távolságok összege (vagy négyzetösszege) maximális. Egy ilyen választást mutat a 6.7. ábra.

A *FarthestFirst* algoritmus a k-közép egy olyan változata, amely kezdeti klaszterközéppontnak egymástól távol lévő elemeket választ, és mindössze egyetlen iterációt hajt végre: a klaszterközéppontok kezdeti megválasztása után minden elemet a legközelebbi klaszterközéppontnak megfelelő csoporthoz rendel hozzá [Hochbaum és Shmoys, 1985].

6.6.3. A k-medoid algoritmusok

Ezek az algoritmusok a k-közép két hibáját próbálják kiküszöbölni. Egyrészt az eredmény kevésbé érzékeny a kívülálló pontokra, másrészt csak a hasonlósági értékeket használja. Tehát nem feltétel, hogy az elemek vektortérben legyenek megadva.



6.7. ábra. A k-means vagy ahhoz hasonló algoritmusok esetében a kezdeti klaszterközéppontokat választhatjuk úgy, hogy azok lehetőleg távol legyenek egymástól. Ha első klaszterközéppontnak véletlenszerűen a csillaggal jelölt elemet választjuk, akkor a második és harmadik klaszterközéppontoknak a "2"-vel illetve "3"-mal jelölt elemeket választhatjuk.

A k-medoid algoritmusokban egy klasztert nem a középpont reprezentál, hanem a leginkább középen elhelyezkedő elem, a medoid. A reprezentáns tehát, a k-means-zel ellentétben, most egy olyan objektum lesz, amely ténylegesen előfordul az adatbázisban. Továbbra is egy négyzetes hiba jellegű függvényt próbálunk minimalizálni, de a négyzetes hiba itt a medoidoktól való távolságok összegét jelenti (k-medoid probléma, lásd 6.4.1 rész).

A PAM algoritmus

A PAM (Partitioning Around Medoids) algoritmus [Kaufman és Rousseeuw, 1990] menete lényegében megegyezik a k-közép menetével. Kezdetben választunk k véletlen elemet, ezek lesznek először a medoidok. Ezután elkezdődik az iteratív folyamat, mely során az elemhozzárendelés medoidokhoz és új medoid választása lépések közt alternálunk. Az elemhozzárendelés során egy elemet a legközelebbi medoidhoz rendelünk.

Abban az esetben választunk új medoidot egy klaszterben, ha ezzel csökken a négyzetes hiba. Határozzuk meg az összes (x, x_m) nemmedoid-medoid párra, hogy mennyivel változna a négyzetes hiba, ha x_m-nek átvenné a szerepét x. Nyilvánvaló, hogy nincsenek hatással a négyzetes hiba változására azok az elemek, amelyekhez van x és x_m-nél közelebbi medoid. A négyzetes hiba változásánál nem csak x_m klaszterébe tartozó elemeket kell megvizsgálnunk, hiszen lehet, hogy a medoidváltás hatására egyes elemek új klaszterbe kerülnek. Ha vannak olyan párok, amelyeknél a négyzetes hiba csökken, akkor cseréljen szerepet annak a párosnak a két eleme, amelyhez tartozó négyzetes hiba változás abszolút értékben a legnagyobb.

Sajnos az algoritmus lassú, hiszen egy iteratív lépés időigénye $O(k(n-k)^2)$. Összehasonlítva a k-közép algoritmussal elmondhatjuk, hogy lassabb, viszont kevésbé érzékeny a távol eső pontokra.

A CLARA és CLARANS algoritmusok

A PAM algoritmus nem alkalmas nagy adathalmazok klaszterezésére, mert lassú. A CLARA és CLARANS algoritmusok a PAM algoritmus kiterjesztései. Nem vizsgálnak meg minden medoid, nem medoid párt, hanem csak néhányat. Így az iterációs lépésben elvégzendő ellenőrzések száma kisebb, ami által az algoritmusok nagyobb adathalmazokon is használhatók.

A CLARA algoritmus [Kaufman és Rousseeuw, 1990] az eredeti adatbázis helyett egy véletlenszerűen választott mintán dolgozik. A medoidok csak ebből a véletlen mintából kerülhetnek ki, de a négyzetes hibát a teljes adatbázisra nézve számítja. Az iterációs lépés időigénye így $O(k(n'-k)(n-k))$, ahol n' a minta nagysága.

Ha a legkisebb négyzetes hibát eredményező k elem nincs a mintában, akkor a CLARA nem fogja megtalálni az optimális megoldást. Célszerű ezért az algoritmust több véletlenszerűen kiválasztott mintán lefuttatni, és a legkisebb négyzetes hibát szolgáltatót elfogadni.

A CLARANS algoritmus [Ng és Han, 1994] az eredeti adathalmazon dolgozik. Nem az összes lehetséges csere által eredményezett négyzetes hiba változását számítja ki, hanem csak néhány, véletlenszerűen választott párét. Ha egy pár esetében a hiba csökken, akkor a pár elemei szerepet cserélnek (a nemmedoidból medoid lesz és a medoidból nemmedoid), ellenkező esetben új párt sorsolunk. A felhasználó egy paraméterrel szabályozhatja a sikertelen választások maximális számát, amely elérésével az algoritmus véget ér.

A CLARANS sem biztos, hogy megtalálja a legkisebb négyzetes hibát adó k medoidot mielőtt a sikertelen próbálkozások száma elérné a küszöböt. Ezért érdemes többször futtanunk az algoritmust mindig más-más kezdeti medoidokkal.

Vegyünk észre egy fontos tulajdonságot: eredményezhetik ugyanazt a klaszterezést különböző medoidok. Valószínű, hogy az optimálishoz közeli megoldás ugyanazt a csoportosítást adja, mint a legkisebb négyzetes hibát szolgáltató medoidok. Ezért javasolt a fenti heurisztikák alkalmazása nagy adathalmazok esetén.

Ester és szerzőtársai a CLARANS nagy adathalmazokon való alkalmazhatóságával foglalkoztak [Ester és tsa., 1995]. R*-fák használatával feloldják azt a kényszert, miszerint a pontok férjenek el a memóriában. A PAM és rokonainak hibája, hogy a k-medián problémát próbálja megoldani, így csak egyszerű, eliptikus klasztereket képesek felfedezni.

k-Hubs

A k-Hubs algoritmus a k-Medoids algoritmus egy olyan változata, amely tekintettel van a 3.3.7. fejezetben leírt csomósodás jelenségére: a klaszterközéppontok választásakor figyelembe veszi, hogy egy-egy elem hányszor fordul elő más elemek legközelebbi szomszédai között [Tomasev és tsa., 2011].

6.7. Hierarchikus eljárások

A hierarchikus eljárások onnan kapták a nevüket, hogy az elemeket, klasztereket, alklasztereket hierarchikus adatszerkezetbe rendezik el. Két fajta hierarchikus eljárást különböztetünk meg: a *lentről építkező*t, más néven *egyesítő*t és a *fentről építkező*t, avagy az *osztó*t. A lentről építkező eljárásoknál kezdetben minden elem külön klaszter, majd a nagyon közeli klasztereket egyesíti, amennyiben azok teljesítenek bizonyos feltételt. A fentről építkezők fordítva működnek: kezdetben egyetlen klaszter létezik, amit kisebb alklaszterekre osztunk, majd ezeket finomítjuk tovább.

A hierarchikus algoritmusok kényes pontja az egyesítendő vagy osztandó klaszterek kiválasztása. Miután egy egyesítés (osztás) megtörténik, az összes további műveletet az új klaszteren végezzük. Ezek a műveletek tehát nem megfordítható folyamatok, így rossz választás rossz minőségű klaszterezéshez vezet.

6.7.1. Single-, Complete-, Average Linkage Eljárások

A legegyszerűbb egyesítő, hierarchikus eljárás az alábbi.

1. Kezdetben minden pont külön klaszterhez tartozik.
2. Keressük meg, majd egyesítsük a legközelebbi klasztereket.
3. Ha a klaszterek száma lecsökkent k-ra, akkor álljunk meg, ellenkező esetben ugorjunk a 2. lépésre.

Ez az egyszerű eljárás a távolság mátrixszal dolgozik, feltételezi, hogy az elfér a memóriában. A távolság mátrix egy felső háromszög-mátrix, amelynek i-edik sorának j-edik eleme tárolja az i és j elemek távolságát. Célszerű kiegészíteni minden sort két extra információval: a legközelebbi klaszter sorszámával és annak távolságával.

Az eljárás tár- és időigénye (az összehasonlítások száma) $O(n^2)$. A mai tárkapacitások mellett (néhányszor 10 Gbyte memóriával rendelkező gép hétköznapinak számít) ez azt jelenti, hogy az elemek száma nagyságrendileg nem lehet több néhányszor 100 ezernél.

Amennyiben két klaszter távolságát a legközelebbi elemeik távolságával definiáljuk, akkor az eljárást *single linkage eljárás*nak nevezzük. A single linkage rendkívül alkalmas jól elkülönülő, tetszőleges alakú klaszterek felfedezésére. Mivel a minimális távolságon alapszik, ezért ha két klaszterek túl közel kerül egymáshoz, a single linkage hajlamos összekötni őket.

A single link további hibája, hogy érzékeny az outlierekre. Általában az outlierek távol esnek a többi ponttól, így ezeket a pontokat nem fogja egyesíteni. Például két jól elszeparálódó, sok pontot tartalmazó klasztert és egy nagyon távoli pontot úgy oszt két részre, hogy az egyik részben lesz a távoleső pont, a másikban pedig az összes többi. Ezen hibát megpróbálhatjuk olyan módon kiküszöbölni, hogy egy más jellegű leállási feltételt adunk meg: addig egyesítjük a klasztereket, amíg a két legközelebbi klaszter távolsága nem nagyobb egy adott küszöbszámnál. A küszöbszám megválasztása nem-triviális feladat, a küszöbszám megfelelő értéke nagyban függ az alkalmazási területtől, illetve a konkrét klaszterezendő adatbázistól. Ha tudjuk, hogy olyan adathalmazt kell klaszterezünk, ahol a (tetszőleges alakú) csoportok jól elkülönülnek egymástól, továbbá nincsenek outlierek, akkor a single eljárás – az eredeti leállási feltétellel is – jó munkát végez.

Ha az eljárást gráfelméleti szemszögből nézzük (teljes gráfban a pontoknak az elemek, az éleken lévő súlyoknak pedig a távolságok felelnek meg), akkor a single-linkage eljárás egy minimális feszítőfát fog találni, amennyiben a klaszterek számának egyet adunk meg értékül. Ha k darab csoportba akarjuk sorolni a pontokat, akkor ezt a minimális feszítőfa $k-1$ legnagyobb élének elhagyásával nyerhetjük. Azon elemek lesznek egy klaszterben, amelyek egy komponensbe kerültek. Rájöhetünk arra is, hogy a single-linkage eljárás nem más, mint Kruskal algoritmusa más köntösben.

Ha két klaszter távolságának megállapításához a legközelebbi elemeik távolsága helyett a legtávolabbi elemeik távolságát használjuk, akkor *complete linkage* eljárásról beszélünk. Ha pedig a C_i és C_j klaszterek távolságát az $(x, y), x \in C_i, y \in C_j$ elempárok távolságainak átlagaként definiáljuk, akkor *average linkage* eljárásról beszélünk.

Példa – Példaként tekintsünk egy 5 objektum (elem) klaszterezéséből álló klaszterezési feladatot. Az egyes elemek távolságai az alábbi mátrix-szal adottak:

	1	2	3	4	5
1	0	0.5	1.5	2	6
2	0.5	0	1	4	7
3	1.5	1	0	5.5	6
4	2	4	5.5	0	2
5	6	7	6	2	0

Kezdetben minden elem egy önálló. A hierarchikus klaszterező mindhárom tárgyalt változata (Single Linkage, Complete Linkage, Average Linkage) az egyesítési lépések során a mindig a két leginkább hasonló klasztert egyesíti. Ahogy írtuk, a külöbség a három eljárás köött az, ahogyan *két klaszter* távolságát számítják. Amíg minden elem egy-egy önálló klaszter, a klaszterek távolságai megegyeznek az elemek távolságaival. Kezdetben tehát a két legkisebb távolságú klaszter az $\{1\}$ és $\{2\}$ egyelemű klaszerek, hiszen az 1-es és 2-es elemek távolsága a legkisebb. Ezeket egyesítjük az első egyesítési lépésben. Így az első egyesítési lépés után az alábbi klaszterek adódnak: $\{1,2\}, \{3\}, \{4\}, \{5\}$.

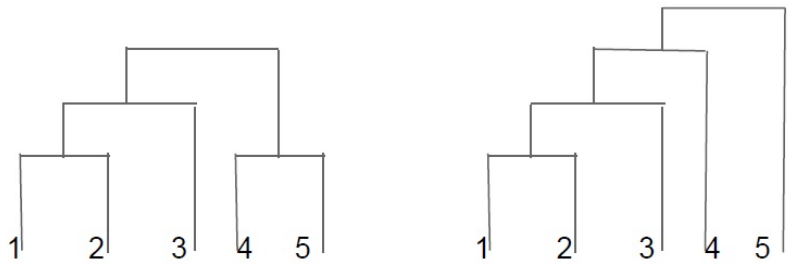
A *Single Linkage* a klaszterek távolságainak számításakor a klaszterek legközelebbi elemeit veszi figyelembe, így például az $\{1,2\}$ és $\{3\}$ klaszterek távolsága: $min(d(1,3), d(2,3)) = min(1.5, 1) = 1$. Hasonlóképpen az $\{1,2\}$ és $\{4\}$ klaszterek távolsága: $min(d(1,4), d(2,4)) = min(2,4) = 2$ illetve az $\{1,2\}$ és $\{5\}$ klaszterek távolsága: $min(d(1,5), d(2,5)) = min(6,7) = 6$.

A hierarchikus klaszterező *Complete Linkage* változata esetén az $\{1,2\}$ és $\{3\}$ klaszterek távolsága: $max(d(1,3), d(2,3)) = max(1.5, 1) = 1.5$, az $\{1,2\}$ és $\{4\}$ klaszterek távolsága: $max(d(1,4), d(2,4)) = max(2,4) = 4$ illetve az $\{1,2\}$ és $\{5\}$ klaszterek távolsága: $max(d(1,5), d(2,5)) = max(6,7) = 7$.

Average Linkage esetén az $\{1,2\}$ és $\{3\}$ klaszterek távolsága: $(1.5+1)/2 = 1.25$, az $\{1,2\}$ és $\{4\}$ klaszterek távolsága: $(2+4)/2 = 3$ illetve az $\{1,2\}$ és $\{5\}$ klaszterek távolsága: $(6+7)/2 = 6.5$.

Az egyelemű klaszterek távolsága – mindhárom változatban – továbbra is megegyezik a klasztereket alkotó elemek távolságával. Mivel bármely két egyelemű klaszter távolsága legalább 2, a két egymáshoz legközelebbi klaszter, mindhárom változat esetében az $\{1,2\}$ és $\{3\}$ lesz, ezek egyesítése után az alábbi klaszterezést kapjuk: $\{1,2,3\}, \{4\}, \{5\}$.

Az egyesítési lépéseket folytathatjuk tovább. Az egyesítések lefolyását szemléltei a 6.8. ábrán látható dendogram. A Single Linkage esetén a következő lépésben választhatunk, hogy az $\{1,2,3\}$ és $\{4\}$ klasztereket egyesítjük-e vagy a $\{4\}$ és $\{5\}$ klasztereket: mindkét klaszterpár távolsága ugyanannyi. Ezért a Single Linkage lefutása a jobb vagy baloldali dendogramnak felel meg. A másik két változat, Average Linkage és Complete Linkage esetében nincs ilyen



6.8. ábra. A példabeli elemek hierarchikus klaszterezőkkel történő klaszterezésének menetét ábrázoló dendogrammok.

választási lehetőségünk. Ennek a két változatnak a baloldali dendogram felel meg.

6.7.2. Ward módszere

Ward módszere a particionáló eljárásokhoz hasonlóan a legkisebb négyzetes hibát próbálja minimalizálni (tehát csak vektortérben megadott pontoknál alkalmazható). Az egyszerű hierarchikus eljárástól csak az egyesítendő klaszterek kiválasztásának módjában különbözik. Azt a két klasztert egyesíti, amelyek a legkisebb négyzetes hibanövekedést okozzák (nyilvánvalóan kezdetben – amikor minden pont külön klaszter – a négyzetes hibaösszeg 0).

A módszer rendelkezik a négyzetes hibát minimalizáló eljárások minden hibájával. Emellett nem skálázható, hiszen a távolságmátrixszal dolgozik, és végül nem garantált, hogy megtalálja a globális minimumot.

Vegyük észre, hogy attól függően, hogyan definiáljuk két klaszter távolságát, a hierarchikus egyesítő algoritmust különféle klaszterezési célfüggvények optimumának keresésére használhatjuk. Egy alkalmazás-specifikus klaszterezésre, az adott alkalmazás speciális célfüggvényének optimalizására láthatunk példát a SOHAC (Storage-Optimizing Hierarchical Agglomerative Clustering) algoritmusban [Nagy és Buza, 2012].

6.7.3. A BIRCH algoritmus

Ezt az algoritmust nagy adathalmazok klaszterezésére találták ki. Csak vektortérben adott elemeket tud klaszterezni. Egy klasztert a $CF = (N, \vec{LS}, SS)$ hármas (Cluster Feature) jellemez, ahol N a klaszterben található elemek száma, $\vec{LS} = \sum_{i=1}^{N} \vec{x}_i$ és $SS = \sum_{i=1}^{N} |\vec{x}_i|$. Egy klaszter CF-je a klaszter statisztikai

jellemzőit tárolja: a nulladik, első és második momentumokat. Az algoritmus során a klaszterek CF-értékeit tároljuk, a benne lévő elemeket nem. Egy klaszter CF-jéből ki tudjuk számolni a klaszter átmérőjét vagy akár két klaszter átlagos távolságát.

A CF-ekből az algoritmus egy ún. CF-fát épít fel. A CF-fa egy gyökeres, (lefelé) irányított fa. A fa leveleiben CF-értékek vannak, egy belső pont pedig a pontból induló alfához tartozó klaszterek egyesítéséből kapott CF-értéket tárolja. A fának két paramétere van. Az első határozza meg a belső pontból induló ágak maximális számát (ágszám korlát). A második paraméter adja meg, hogy mekkora lehet maximálisan a levélhez tartozó klaszterek átmérője. Ennek a paraméternek nagy hatása van a fa méretére: minél kisebb a maximális átmérő, annál több kis klasztert kell létrehozni, tehát annál nagyobb lesz a fa.

A BIRCH algoritmus két fázisra oszlik. Az elsőben az elemek egyszeri végigolvasása során felépítjük a CF-fát. Ez már eleve egyfajta klaszterezést eredményez. A második fázisban minden elemet besorolunk a hozzá legközelebbi klaszterbe, majd esetleg ebből kiindulva lefuttatunk egy particionáló algoritmust (például a k-közepet).

Az első fázis során kapott CF-fa – az ágszám korlát miatt – nem valószínű, hogy meg fog felelni a természetes klaszterezésnek. Lehet, hogy bizonyos pontok, amelyeknek egy klaszterben kellene lenniük, szét lesznek választva, és a fordítottja is előfordulhat. Sőt, az is megtörténhet, hogy ugyanazok az elemek a fa építésének különböző fázisaiban különböző levelekbe fognak kerülni!

A cikk szerzői javaslatot adnak az outlierek kiszűrésére: ha a CF-fában valamely levélben található pontok száma „jóval kevesebb", mint a levelekben található pontok átlagos száma, akkor töröljük a levelet, mert valószínűleg outliereket tartalmaz. A felhasználónak kell megadni az elemszámra vonatkozó küszöbszámot, ami alatt töröljük a leveleket. Vegyük észre, hogy ez a megoldás erős heurisztika. Számtalan példát lehetne mutatni, amikor fontos pontokat is töröl, miközben valódi outliereket a fában hagy.

A BIRCH algoritmus tehát tényleg meg tud bírkozni igazán nagy méretű adathalmazokkal, azonban rendelkezik szinte az összes rossz klaszterezési tulajdonsággal. Mivel a második szakaszban egyfajta k-közép algoritmust futtatunk, ezért a BIRCH-re is igazak a k-középről mondottak. De ezen kívül érzékeny az adatok sorrendjére, és nem skála-invariáns, hiszen a CF-fában lévő klaszterek maximális átmérőjét a felhasználónak kell megadnia.

6.7.4. A CURE algoritmus

A CURE (Clustering Using REpresentatives) algoritmus [Guha és tsa., 1998] átmenet a BIRCH és a single linkage eljárás között a reprezentáns elemek számát illetően: a BIRCH-ben a középpont a reprezentáns pont, a single linkage-

ben a klaszter összes elemét számon tartjuk. Egy klasztert c darab (ahol c előre megadott konstans) elem jellemez. Ezek az elemek próbálják megragadni a klaszter alakját. Ennek következménye, hogy a CURE nem ragaszkodik az eliptikus klaszterekhez.

Hierarchikus eljárás lévén, kezdetben minden elem külön klaszter, majd elkezdődik a klaszterek egyesítése. Az egyesítésnek három lépése van.

1. A reprezentáns pontok alapján kiválasztja a két legközelebbi klasztert. Két klaszter távolságát reprezentáns pontjaik távolságának minimuma adja.

2. Egyesíti a klasztereket, majd a $2c$ reprezentáns pont közül kiválaszt c darabot, amelyek várhatóan jól fogják reprezentálni az egyesített klasztert. Ennek módja a következő. Legyen az első reprezentáns pont a középponttól legtávolabbi elem. A következő $c-1$ lépésben mindig az előzőleg kiválasztott ponthoz képest a legtávolabbit válasszuk reprezentáns elemnek.

3. A reprezentáns pontokat „összehúzza”, azaz az általuk kijelölt középpont felé mozdulnak úgy, hogy az új távolság a középponttól az α-szorosa legyen az eredeti távolságnak. Ennek a lépésnek a célja az outlierek hatásának csökkentése.

Az egyesítés akkor ér véget, amikor a klaszterek száma eléri k-t. Az eljárás végeztével rendelkezésünkre áll c reprezentáns ponttal jellemzett k darab klaszter. Ezután a teljes adatbázis egyszeri végigolvasásával az elemeket besoroljuk a hozzá legközelebbi klaszterbe a legközelebbi reprezentáns pont alapján.

A CURE algoritmust felkészítették, hogy kezelni tudjon nagy adathalmazokat is. Véletlen mintát vesz, azt felosztja részekre, majd az egyes részeket klaszterezi (ez a rész tehát párhuzamosítható). A kapott klaszterekből végül kialakítja a végső klasztereket. A részletek és az algoritmus során felhasznált adatszerkezetek (k-d fa és kupac) iránt érdeklődöknek ajánljuk Guha cikkét [Guha és tsa., 1998].

A CURE algoritmus számos hibával rendelkezik:

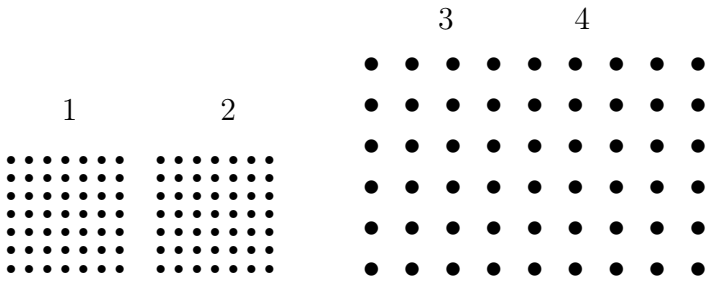
1. Az elemeknek vektortérben adottnak kell lenniük.

2. Minden klasztert rögzített számú reprezentáns pont jellemez. De miért jellemezzen ugyanannyi pont egy kis kör alakú klasztert és egy nagy amőba alakút? Természetesebb lenne, ha a reprezentáns pontok száma függene a klaszter méretétől és alakjától.

3. A reprezentáns pontok kiválasztása sem túl kifinomult. A módszer nem a klaszter alakját fogja meghatározni, hanem inkább egy konvex burkot. Gondoljuk meg, ha egy kör alakú klaszterben van egy bemélyedés, akkor a bemélyedés környékén található pontokat nem fogja az eljárás reprezentásnak választani, hiszen ők közel vannak a többi ponthoz. Amőba alakú klaszternél tehát a reprezentáns pontok alapján nem lehet megállapítani, hogy hol vannak a bemélyedések, így azt sem dönthetjük el, hogy egy nagy ellipszissel van dolgunk, vagy egy érdekes alakú amőbával.

4. Klaszter egyesítése után a reprezentáns pontokat összehúzzuk a középpont felé. Nagy klaszter esetében sok egyesítés, így sok összehúzás van. Az összehúzás által távolabb kerülnek a reprezentáns pontjai más reprezentáns pontoktól, így egyre ritkábban lesz kiválasztva egyesítésre. Mintha az algoritmus „büntetné" a nagy klasztereket.

5. Rosszul kezeli az eltérő sűrűségű pontokat. Ezt leginkább az alábbi ábra illusztrálja. A CURE az 1-es és 2-es klasztereket fogja egyesíteni (azok



6.9. ábra. Hibás klaszterezés: eltérő sűrűségű klaszterek esetén

reprezentáns pontjai vannak egymáshoz legközelebb) a 3-as és 4-es klaszterek egyesítése helyett.

6.7.5. A Chameleon algoritmus

A Chameleon két nagy fázisra oszlik. Kiindulásként előállítja a k-legközelebbi gráfot, majd ezt részekre osztja. A második fázisban bizonyos részeket egyesít, előállítva ezzel a végleges csoportokat.

A Chameleon az első olyan hierarchikus algoritmus, amely a klaszterek egyesítésénél nem csak a klaszterek távolságát ($d(C_i, C_j)$) veszi figyelembe, hanem az egyes klasztereken belüli távolságokat is, pontosabban a *relatív belső kapcsolódás*ukat ($RI(C_i, C_j)$) is (relative inter-connectivity). Abban az esetben egyesít két klasztert, amennyiben $d(C_i, C_j)$ és $RI(C_i, C_j)$ is nagy értéket vesz

fel. Ennek az ötletnek köszönhető, hogy a Chameleon – szemben az eddigi algoritmusokkal – jól tud klaszterezni eltérő sűrűségű adathalmazokat is.

6.8. Sűrűség-alapú módszerek

A sűrűség alapú módszerek (density based methods) szerint egy klaszteren belül jóval nagyobb az elemek sűrűsége, mint a klaszterek között. Ez alapján lehet elválasztani a klasztereket és azonosítani az outliereket.

6.8.1. A DBSCAN algoritmus

A DBSCAN a legelső sűrűség-alapú eljárás [Ester és tsa., 1996]. A sűrűség meghatározásához két paramétert használ, egy sugár jellegű mértéket (eps) és egy elemszám küszöböt ($minpts$). A p elem *szomszédai* ($N_{eps}(p)$) azok a elemek, amelyek p-től legfeljebb eps távolságra vannak. A q elem a p-ből *sűrűség alapon közvetlen elérhető*, ha $q \in N_{eps}(p)$ és $|N_{eps}(p)| \geq minpts$. Naivan azt gondolhatnánk, hogy egy klaszterben található elemek sűrűség alapon közvetlen elérhetők egymásból. Ez nem így van, hiszen a klaszter határán lévő elemek eps távolságán belül nincs mindig $minpts$ darab elem.

A q elem *sűrűség alapon elérhető* p-ből, ha léteznek $p = p_1, p_2, \ldots, p_n = q$ elemek úgy, hogy p_{i+1} sűrűség alapon közvetlen elérhető p_i-ből. A p és q elemek *sűrűség alapon összekötöttek*, ha létezik olyan o elem, amelyből p és q sűrűség alapon elérhető. A klaszter definíciója ezek alapján:

6.8.1. Definíció *Az elemek egy C részhalmaza klaszter, amennyiben*

1. *Ha $p \in C$ és q sűrűség-alapon elérhető p-ből, akkor $q \in C$ (maximalitás).*
2. *Ha $p, q \in C$, akkor p és q sűrűség alapon összekötöttek.*

Egy elemet *zajnak* (noise) hívunk, ha nem tartozik egyetlen klaszterbe sem.

Legyen a C klaszter egy p eleme olyan, hogy $|N_{eps}(p)| \geq minpts$. Ekkor könnyű belátni, hogy C megegyezik azoknak a elemeknek a halmazával, amelyek p-ből sűrűség alapján elérhetők. E tulajdonságot használja az algoritmus. Válasszunk egy tetszőleges elemet (p) és határozzuk meg a sűrűség alapján elérhető elemeket. Amennyiben $|N_{eps}(p)| \geq minpts$ feltétel teljesül, akkor meghatároztunk egy klasztert. A feltétel nemteljesülés nem jelenti azt, hogy p zaj, lehet, hogy egy klaszter határán helyezkedik el. $|N_{eps}(p)| < minpts$ esetén egyszerűen válasszunk egy új elemet. Ha már nem tudunk új elemet választani, akkor az algoritmus véget ér. Azokat az elemeket tekintjük zajnak (outliernek), amelyeket nem soroltunk semelyik klaszterbe.

A DBSCAN algoritmus előnye, hogy tetszőleges alakú klasztert képes felfedezni, és ehhez csak az elemek távolságát használja. Hátránya, hogy rendkívűl érzékeny a két paraméterre (*eps*, *minpts*). Sőt amennyiben a klaszterekben található elemek sűrűsége eltérő, akkor nem biztos, hogy lehet olyan paramétereket adni amivel a DBSCAN jó eredményt ad.

7. fejezet

Idősorok elemzése

Mindeddig azt tételeztük fel, hogy az elemzendő adatok egy táblázat formájában adottak, minden objektum ugyanazon attribútumokkal rendelkezik. Annak ellenenére, hogy különböző típusú attribútumokkal foglalkoztunk, ez a táblázatos modell számos gyakorlati alkalmazásra így sem illeszkedik megfelelően. A más jellegű struktúrával rendelkező adatok egyik legegyszerűbb esetével, az idősorokkal foglalkozunk ebben a fejezetben.

Idősor alatt megfigyelések egy sorozatát érjük. Ha csak egyetlen számot figyelünk meg, *egyváltozós* (univariate) idősorokról beszélünk. Ha például a levegő hőmérsékletét mérjük egymást követő időpontokban, a mért értékek egy egyváltozós idősort alkotnak. Többváltozós (multivariate) idősorról akkor beszélünk, ha egymást követő időpontokban több attribútum értékét is megfigyeljük. A kézírás például felfogható egy többváltozós idősorként: miközben egy érintésérzékeny képernyőre betűket írunk, eltárolhatjuk a toll végpontjának vízszintes és függőleges koordinátáit egymást követő időpillanatokban, például századmásopercenként egyszer. A pozíciót leíró koordináták mellett akár a nyomás erősségét is tárolhatjuk. Ezek alapján felismerhetjük, hogy milyen betűt/szót írt a felhasználó az érintésérzékeny képernyőre.

A szenzortechnika és adattárolás egyre olcsóbbá válásával és széleskörű terjedésével az idősorokkal kapcsolatos adatbányászati feladatok jelentősége folyamatosan növekszik. Az idősorok osztályozása számos gyakorlatban felmerülő felismerési feladat közös elméleti hátterét képezi, a már említett kézírásfelismerés mellett ilyen az aláírások automatikus (számítógépi) ellenőrzése, a számítógépes beszédfelismerés vagy a jelbeszédi jelek felismerése. Idősorokat osztályozó algoritmusok hozzájárulhatnak továbbá a szívbetegségek felismeréséhez, ha ezen algoritmusokkal a szív elektromos aktivitását leíró EKG (elektrokardiográf) jeleket elemezzük, lásd pl. [Bortolan és Willems, 1993, Olszewski, 2001, Melgani és Bazi, 2008, Syed és Chia, 2011, Buza, 2011c]. Számos érdekes fel-

ismerési feladat köthető az agy elektromos aktivitását leíró (EEG) jelekhez is.[1] A Berlini Műszaki Egyetem kutatói például az autók féktávolságának csökkentésével próbálkozott az autót vezető sofőr agyhullámainak (EEG-jének) automatikus elemzése révén.[2] Mielőtt egy sofőr vészfékezésbe kezd, észleli a veszélyhezetet és erre reagálva helyezi át jobb lábát a gázról a fékre. Ha sikerül az agyhullámok (EEG) alapján automatikusan felismerni, hogy a sofőr szándékában áll fékezni, az autó már azelőtt belekezdhet a fékezésbe, hogy a sofőr elkezdené nyomni a féket. Így tehát lerövidíthető a féktávolság. A viszonylag olcsó EEG szenzoroknak köszönhetően folyamatosan terjednek a EEG-re épülő, "gondolatainkkal irányítható" számítógépes játékok is.

Idősorok adatbányászata számos további alkalmazásban fordul elő. Ilyen például az alakfelismerés képekben [Jalba és tsa., 2005]. Ha sikerül felismernünk egy alakzatot határoló görbét, ez a görbe átalakítható egy idősorrá, és így idősorokat osztályozó algoritmusok használhatók képeken lévő alakzatok feismerésére [Yankov és tsa. 2007, Wang és tsa., 2008].

Az idősorok számos sajátos tulajdonsággal rendelkeznek, amelyek miatt érdemes külön témaként foglalkozni az idősorokkal kapcsolatos adatbányászati feladatokkal, első sorban az idősorok osztályozásával és klaszterezésével. Az egyik ilyen tulajdonság, hogy az idősorokban az egymást követő megfigyelések – általában – erősen *korrelálnak egymással*, nem függetlenek egymástól: például az, hogy hány fokos a levegő hőmérséklete délelőtt 10 óra 15 perckor, nyilván nem független attól, hogy hány fokos volt a hőmérséklete 10 óra 5 perckor illetve 10 óra 10 perckor. Míg néhány alkalmazásban az objektumokhoz tartozó idősorok hossza kötött és ebből adódóan minden példányra azonos, legtöbbször *különböző hosszúságú idősorokkal* kell dolgoznunk, különböző hosszúságú idősorokat kell összehasonlítanunk. Nem várhatjuk el például, hogy a felhasználók minden betűt, minden alkalommal, századmásodperc pontossággal azonos idő alatt írjanak le, egy-egy szívverés pontosan ugyanannyi ideig tartson, stb. Az idősorokban található mintázatok valamilyen valós esemény "lenyomatai", ugyanazon eseményhez tartozó mintázatok kisebb-nagyobb mértékben különbözhetnek. A mintázatok lehetséges megnyúlását, eltoldódását figyelembe kell tehát venni az idősorok összehasonlításakor.

Nemcsak a korábban már megismert adatbányászati feladatok (osztályozás, klaszterezés) megoldása igényel új technikákat idősorok esetében, hanem a korábbi feladatok új változatai is megjelentek. Idősorok osztályozásakor felmerülő igény például, hogy a lehető leghamarabb találjuk meg azt az osztályt (csoportot), amibe az idősor tartozik, még *azelőtt*, hogy az idősorhoz tartozó összes

[1] Lásd pl. http://archive.ics.uci.edu/ml/datasets/EEG+Database

[2] http://www.origo.hu/tudomany/20110728-aghullamok-segithetik-a-fekezest-a-jovo-autoiban.html

számértéket megfigyeltük volna. *Early classificationről* (korai osztályozásról) beszélünk, ha ilyen feladattal állunk szemben [Xing és tsa., 2011]. Ennek mintájára definiálhatjuk az *early clustering*-et is, amely során a célunk az, hogy az idősorok alkotta klasztereket mielőbb megtaláljuk.

A következő fejezetben áttekintjük az idősorokkal kapcsolatos adatbányászati eljárások alapjait: először az idősorok ábrázolására leginkább elterjedt technikákkal ismerkedünk, majd két idősor összehasonlításának kérdésével foglalkozunk részletesebben. Végezetül az idősorok osztályozására és klaszterezésére használt leggyakoribb technikákat tekintjük át.

7.1. Idősorok ábrázolása

A legkézenfekvőbb, hogy egy l hosszúságú x idősort egy l elemű számsorozatnak tekintsünk:

$$x = (x[0], ..., x[l-1]). \tag{7.1}$$

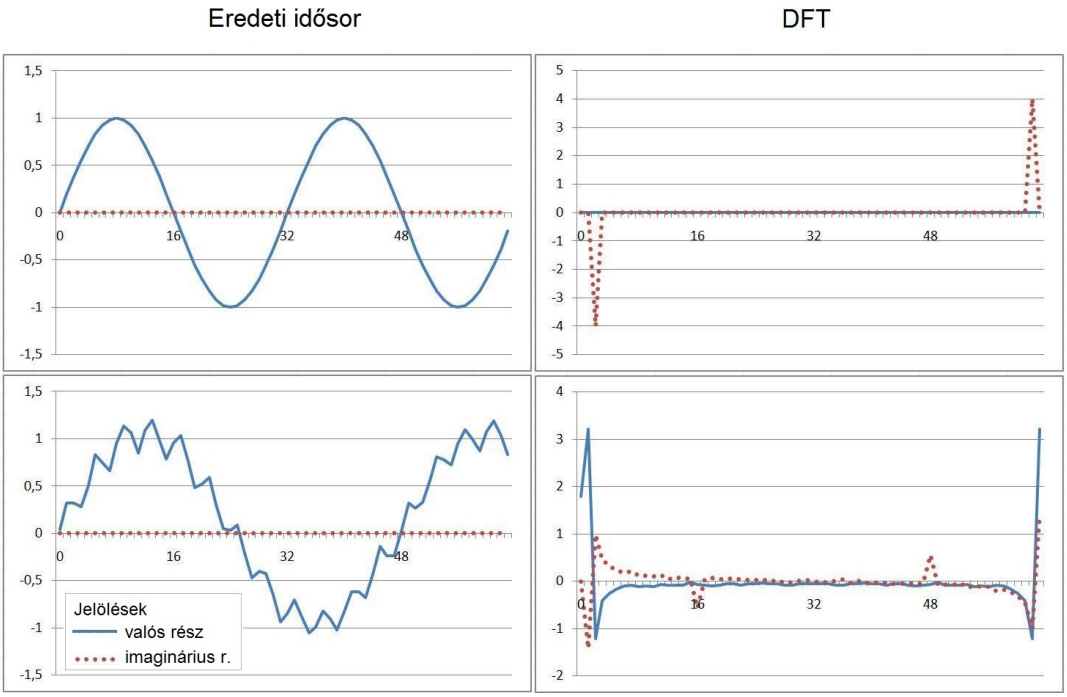
Ez az elemi reprezentáció azonban számos alkalmazásban nem előnyös. A beszédfelismerésben például a hanghullám frekvenciája sokkal kifejezőbb az elemi hullámnál: ha az angol *yes* és *no* szavakat szeretnénk megkülönböztetni, a magas frekvencián ejtett *s*-nek köszönhetően lényegében csak annyi a dolgunk, hogy megnézzük, hogy a jel tartalmazza-e a megfelelő magasfrekvenciás komponenst. Más esetekben célszerű az idősorok egymást követő értékeit aggregálni (pl. átlagolni), és gyakran az idősor lokális és globális tulajdonságai egyaránt számítanak.

A széleskörű alkalmazások változatos igényeinek megfelelően különféle ábrázolási módok terjedtek el a Fourier- és Wavelet transzformációktól kezdve [Chan és Fu, 1999, Mörchen, 2003, Wu és tsa., 2000] a szakaszonkénti approximációkon [Yi és Faloutsos, 2000, Shieh és Keogh, 2008, Keogh és tsa., 2001, Keogh és Pazzani, 1998, Geurts, 2001, Lin és tsa., 2003, Olszewski, 2001] át a singuláris érték-felbontáson (SVD) alapuló eljárásokig [Korn és tsa., 1997].

A következőkben a Fourier-transzformációt, Wavelet-transzformációt és a szimbólikus aggregált approximációt (SAX) tekintjük át. Az idősor-ábrázolási ejárások áttekintését és taxonomikus kategorizációját lásd [Lin és tsa., 2003]-ben.

7.1.1. Diszkrét Fourier-transzformáció (DFT)

A Fourier-transzformáció során egy idősort, képletesen szólva, szinuszos hullámok összegére bontunk fel. Ezáltal a Fourier-transzformáció azt ragadja meg, hogy milyen frekvencia-komponensek vannak jelen az idősorban.



7.1. ábra. Valós értékű idősorok (bal oldalon) és azok diszkrét Fourier-transzformáltja (jobb oldalon).

Az idősorokat, mint időben diszkrét jeleket tekintjük (lásd 7.1. képlet), ezért a Diszkrét Fourier-transzformációt (DFT) használjuk. A DFT az l hosszúságú $x = (x[0], ..., x[l-1])$ idősorhoz l darab komplex együtthatót, $c_0^F, \ldots, c_{l-1}^F$-t, rendel hozzá. Ezen együtthatókat az alábbiak szerint definiáljuk:

$$c_j^F(x) = \frac{1}{\sqrt{l}} \sum_{i=0}^{l-1} x[i] e^{-\frac{2\pi \mathcal{I} j i}{l}} \quad , \quad 0 \leq j \leq l-1 \tag{7.2}$$

ahol $\mathcal{I} = \sqrt{-1}$ (immaginárius egység), $e \approx 2.718$, $\pi \approx 3.142$ és $x[i]$ az x idősor i-edik elemét jelöli. Az x idősor *Diszkrét Fourier-Transzformáltja* az előbbiekben definiált c_j^F Fourier-együtthatók sorozata:

$$DFT(x) = (c_0^F(x), ..., c_{l-1}^F(x)) \tag{7.3}$$

.

Példaként két valós értékű idősort és azok diszkrét Fourier-transzformáltját mutatja a 7.1. ábra.

Bár elvileg számíthatnánk a DFT-t a fenti definíció (7.2. képlet) alapján, a gyakorlatban ez nem célszerű, mert létezik egy sokkal kevésbé számításigényes módja a DFT kiszámításának, amelyet az angol irodalomban Fast Fourier Transformation-nek (FFT) neveznek. Az FFT alapgondolata az, hogy ha egy idősor páros hosszúságú és ismertjük a páros illetve páratlan sorszámú értékekbő álló idősorok, azaz $x[0], x[2], x[4], ...x[l-1]$ illetve $x[1], x[3], x[5], ..., x[l]$ DFT-jét akkor ezek segítségével nagyon gyorsan kiszámolhatjuk az eredeti idősor DFT-jét. Ha az idősor hossza kettő valamely hatványa, akkor az előbbi ötletet rekurzíven alkalmazhatjuk. Az FFT pszeudokódja:

Algoritmus Fast Fourier Transformation (FFT)

Require: $x = (x[0], x[1], ..., x[l-2], x[l-1])$
Ensure: DFT(x)

if l páros **then**
  $a = FFT(\ (x[0], x[2], ..., x[l-2])\ )$
  $b = FFT(\ (x[1], x[3], ..., x[l-1])\ )$
  for $j = 0; j < l;\ j++$ **do**
    $a_0 = a[j \bmod (l/2)]$.realPart
    $a_1 = a[j \bmod (l/2)]$.imaginaryPart
    $b_0 = b[j \bmod (l/2)]$.realPart
    $b_1 = b[j \bmod (l/2)]$.imaginaryPart
    c_j^F.realPart= $\Big(a_0 + b_0 \cos(2\pi j/l) + b_1 \sin(2\pi j/l)\Big)/\sqrt{2}$
    c_j^F.imaginaryPart= $\Big(a_1 + b_1 \cos(2\pi j/l) - b_0 \sin(2\pi j/l)\Big)/\sqrt{2}$
  end for
  return $(c_1^F, ..., c_{l-1}^F)$
else
  return DFT a definíció szerint
end if

7.1.2. Diszkrét Wavelet Transzformáció

Amint láttuk, a Fourier-transzformáció szinuszos jelek összegére bont fel egy idősort és ezáltal az idősor *globális periodikus* jellemzőit ragadja meg. Ezzel szemben a Wavelet transzformáció célja, hogy az idősor lokális és globális jellemzőit egyaránt megragadja [Hastie és tsa., 2001].

A diszkrét wavelet transzformáció (DWT) számos változata közül csak a legegyszerűbbet, a Haar-waveleteket mutajuk be. A Haar-wavelet transzformáció (HWT) során az idősor egymást követő értékeit páronként aggregáljuk, ezáltal lényegében az idősor egy alacsonyabb felbontású változatát hozzuk létre. Ahhoz, hogy az eredeti idősor helyreállítható legyen, részletező tényezőket

(detail coefficients) is eltárolunk. A Haar-Wavelet transzformáció pszeudokódja:[3][4]

Algoritmus HWT

Require: $x = (x[0], x[1], ..., x[l-2], x[l-1])$ (l páros)
Ensure: HWT(x)

```
for j = 0...l/2 − 1 do
    a[j] = 1/√2 (x[2j] + x[2j + 1])
    c[j] = 1/√2 (x[2j] − x[2j + 1])
end for
if l > 2 then
    return concat(HWT(a),c)        (HWT(a) és c szekvenciák összefűzése)
else
    return ( a[0], c[0] )
end if
```

7.1.3. Szimbólikus Aggregált Approximáció (SAX)

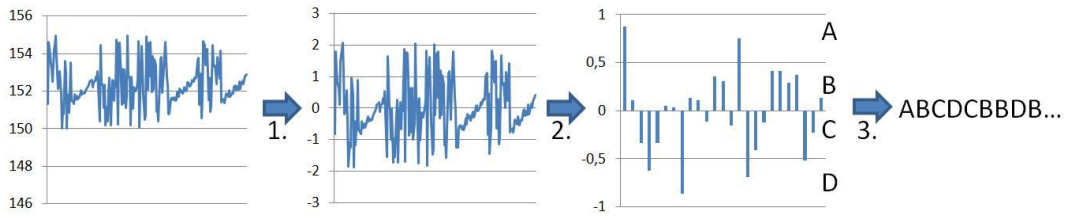
Egy egyszerű, de elterjedt eljárás a szimbólikus aggregált approximáció (SAX), amely során három transzformációs lépést végzünk az idősoron:

1. Normalizáció: kiszámoljuk az x idősor elemeinek átlagát (a_x) és szórását (s_x), majd az idősor egyes elemeit az alábbi képlet szerint transzformáljuk:
$$x[i]^j = \frac{x[i]^{rgi} - a_x}{s_x}.$$
2. Aggregáció: a normalizált idősor egymást követő elemeit átlagoljuk.
3. Diszkretizáció: az átlagolással kapott értékeket diszkretizáljuk, a diszkrét értékeknek szimbólumokat feleltethetünk meg. A diszkretizáció során a tartományokat úgy választjuk meg, hogy a normáleloszlás minden egyes tartomány felett vett integrálja azonos legyen. Azt feltételezve tehát, hogy az idősor elemei normális eloszlásúak, a transzformlált szekvenciában minden szimbólum valószínűsége azonos lesz.

A SAX-ot szemlélteti a 7.2. ábra, további részletek [Lin és tsa., 2003]-ben olvashatók.

[3] http://www.ismll.uni-hildesheim.de/lehre/ip-08w/script/imageanalysis-2up-05-wavelets.pdf

[4] A HWT itt bemutatott változata akkor működik, ha az idősor hossza kettő valamely hatványa.



7.2. ábra. Szimbólikus aggregált approximáxió (SAX)

7.2. Idősorok távolsága

Amint láttuk, mind osztályozás, mind klaszterezés esetében egyik kulcsfogalom az objektumok közti távolság. Említettük, hogy két idősor összehasonlításakor nem feltétlenül célszerű az egyik idősor i-dik elemét a másik idősor ugyanazon elemével összehasonlítani, hiszen ugyanazon jelenség a valóságban kisebb-nagyobb időbeli eltérésekkel játszódhat le, és ezért az idősorok összehasonlításakor is figyelmet kell szentelnünk a mintázatok lehetséges eltolódásának, megnyúlásának, összenyomódásának. Ezt szemlélteti a 7.3. ábra.

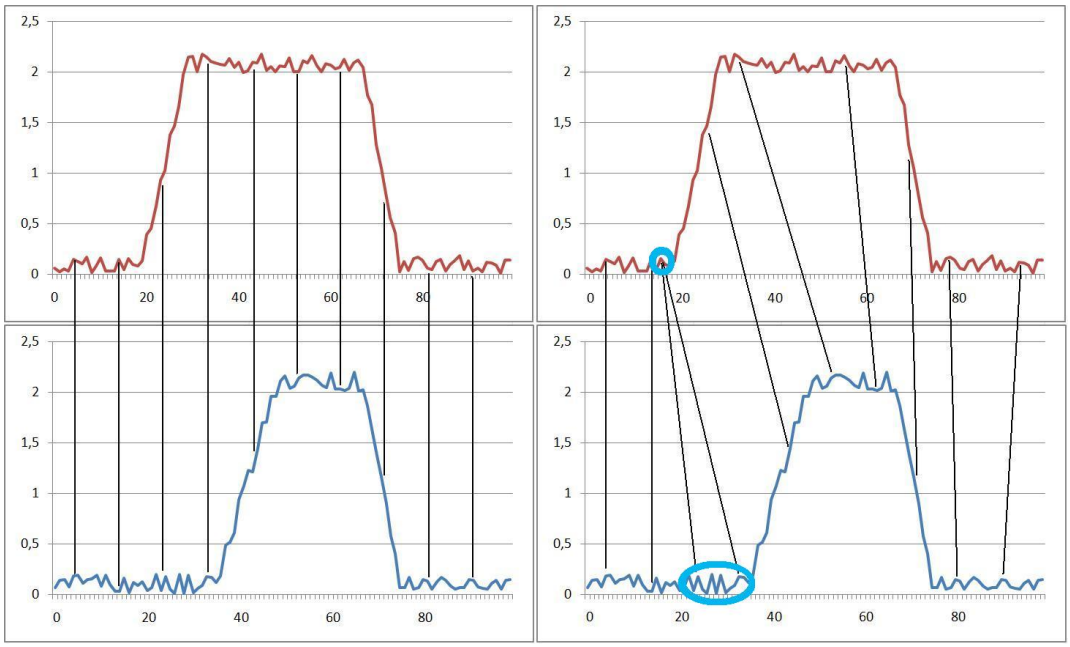
A dinamikus idővetemítés (dynamic time warping, DTW) egy olyan távolságmérték, amely két idősor távolságának számításakor figyelembe veszi a mintázatok lehetséges eltolódását, megnyúlását, összenyomódását. A DTW-t eredetileg a beszédfelismerési feladatokhoz fejlesztették [Sakoe és Chiba, 1978], de a közelmúltban népszerűvé vált az idősorokkal kapcsolatos adatbányászatban, lásd pl. [Keogh és Pazzani, 2000, Ding és tsa., 2008, Radovanović és tsa., 2010b].

A következőkben röviden leírjuk, hogyan számítható két idősor, x_1 és x_2, DTW-távolsága. A DTW egy szerkesztési távolság, azt adja meg, hogy mekkora "költséggel" lehet az egyik idősort a másikba átalakítani. A számítást tekinthetjük úgy, hogy egy mátrix celláit töltjük ki. Úgy képzelhetjük, hogy az egyik idősort a mátrix első sora fölé írtuk, a másodikat pedig az első oszlop mellé. A mátrix minden egyes cellájába írt szám a a két idősor egy-egy prefixumának a DTW-távolsága. Ezt mutatja a 7.4. ábra bal oldala. Az ábra jobb oldala azt mutatja, hogy milyen sorrendben töltjük ki a mátrix celláit.

A DTW-számításához meg kell adnunk egy *belső távolságfüggvényt*, amelyet $c_{tr}^{DTW}(x_1[i], x_2[j])$-vel jelölünk. A belső távolságfüggvénnyel az idősorok egyes *elemeinek* távolságát mérjük. Legegyszerűbb esetben:

$$c_{tr}^{DTW}(x_1[i], x_2[j]) = |x_1[i] - x_2[j]|.$$

Meg kell adnunk továbbá a nyújtás illetve összenyomás "költségét", c_{el}^{DTW}-t, amely a legegyszerűbb esetben nulla.

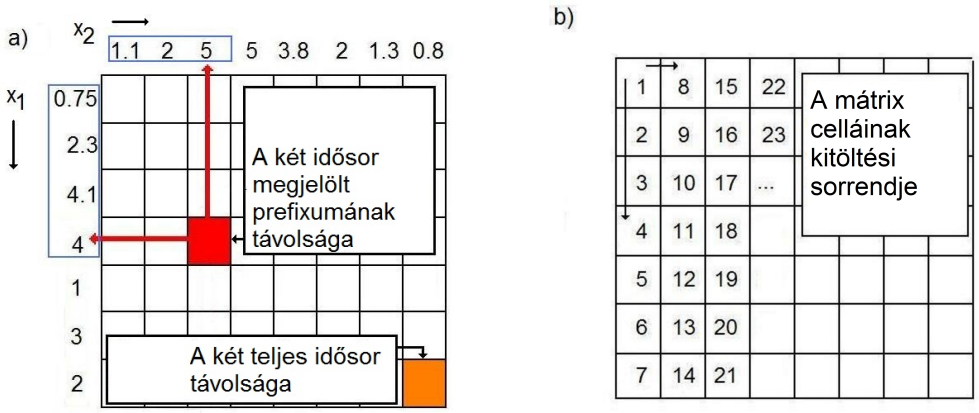


7.3. ábra. Idősorok összehasonlításakor figyelmet kell szentelnünk a mintázatok lehetséges eltolódásának, megnyúlásának, összenyomódásának: ahelyett, hogy a felső és alsó ábrákon látható idősorok összehasonlításakor az egyik idősor i-dik elemét mindig a másik idősor i-dik elemével hasonlítanánk össze (bal oldali ábrák), célszerűbb a mintázatok szerint összeillő elemek összehasonlítása, melynek során előfordulhat, hogy az egyik idősor egy rövid szakaszát hasonlítjuk a másik idősor egy hosszabb szakaszához (jobb oldali ábrák).

A mátrix i-edik sorának j-dik elemét $d_0^{DTW}(i,j)$-vel jelöljük, a mátrix bal felső sarka $d_0^{DTW}(0,0) = c_{tr}^{DTW}(x_1[0], x_2[0])$. A mátrix további celláit pedig az alábbi összefüggéssel határozzuk meg:

$$d_0^{DTW}(i,j) = c_{tr}^{DTW}(x_1[i], x_2[j]) + \min \left\{ \begin{array}{l} d_0^{DTW}(i,j-1) + c_{el}^{DTW} \\ d_0^{DTW}(i-1,j) + c_{el}^{DTW} \\ d_0^{DTW}(i-1,j-1) \end{array} \right\} \qquad (7.4)$$

Ebben a képletben a minimum második és harmadik tagja az egyik illetve másik idősorbeli nyújtásnak illetve összenyomásnak felel meg. A számítások során a minumumban csak azon tagokat vesszük figyelembe, amelyek definiáltak: amikor például az i=0-dik sor elemeit számítjuk, a minimum tagjai közül csak $d_0^{DTW}(i,j-1)+c_{el}^{DTW}$ definiált, a minimumban szereplő másik két tag az $i-1 = -1$-dik sorra vonatkozna, ezért ezt a két tagot ebben az esetben figyelmen kívül hagyjuk. A DTW-mátrix számítására mutat példát a 7.5. ábra.



7.4. ábra. The DTW-matrix. Két idősor, x_1 és x_2 DTW-távolságának számítását tekinthetjük annak, hogy egy mátrix celláit töltjük ki. a) A két idősor: $x_1 = (0.75, 2.3, 4.1, 4, 1, 3, 2)$ és $x_2 = (1.1, 2, 5, 5, 3.8, 2, 1.3, 0.8)$, melyeket úgy képzelhetünk, hogy a mátrix mellé (fentről lefelé) illetve fölé írtunk. A mátrix minden egyes cellájába írt szám a a két idősor egy-egy prefixumának a DTW-távolsága. b) A cellák kitöltésének sorrendje.

Megfelelően megválasztott belső távolságfüggvény mellett többváltozós idősorokat is összehasonlíthatunk DTW-vel. Példaként tegyük fel, hogy kétváltozós idősorokkal dolgozunk. Kézírásfelismerés esetén ez a két változó lehet a toll végpontjának vízszintes és függőleges koordinátája. Mindkét változó értékét időben egymást követő pillanatokban mérjük, ha az egyik változót a-val, a másikat b-vel jelöljük, a kapott x idősort jelölhetjük így:

$$x = \Big((x[0].a, x[0].b), \quad (x[1].a, x[1].b), \quad \dots \quad , \quad (x[l-1].a, x[l-1].b)\Big) \tag{7.5}$$

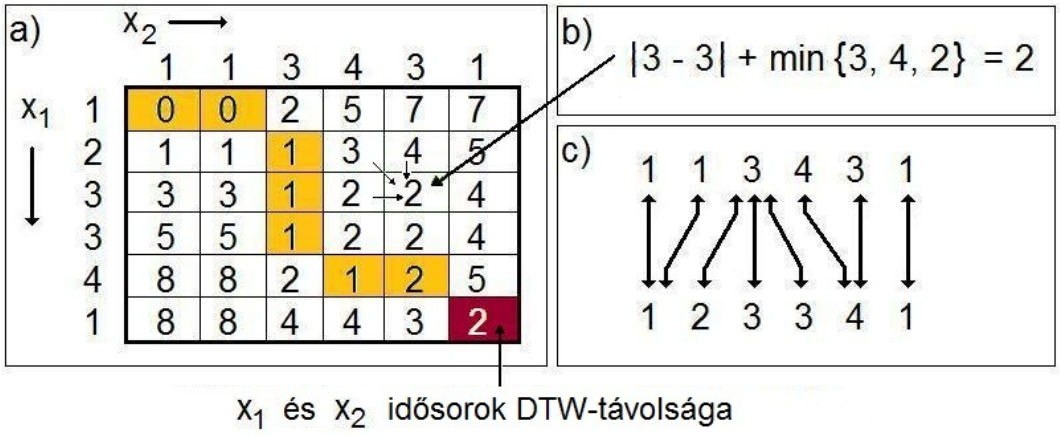
Belső távolságként választhatjuk az euklideszi távolságot:

$$c_{tr}^{DTW}(x_1[i], x_2[j]) = \sqrt{(x_1[i].a - x_2[j].a)^2 + (x_1[i].b - x_2[j].b)^2}. \tag{7.6}$$

Idősorok adatbányászata során használt további gyakran használt távolságmérték az EDR (Edit distance on Real Sequences), ERP (Edit Distance with Real Penalty), LCSS (Longest Common Subsequences) és a DISSIM, lásd [Buza, 2011a]-t és az ott hivatkozott irodalmat.

7.3. Idősorok osztályozása és klaszterezése

Különféle alkalmazási területekről származó, valós idősorokat tartalmazó adatbázisokon végzett vizsgálatok azt mutatták, hogy a legközelebbi szomszéd osz-



7.5. ábra. Példa két idősor DTW-távolságának kiszámítása. a) A DTW-mátrix. b) Egy cella értékének számítása. c) A DTW által (implicite) kiszámolt hozzárendelés a két idősor különböző értékei között.

tályozó a DTW-hasonlóságfüggvény mellett általában véve versenyképes számos bonyolult osztályozó algoritmussal [Ding és tsa., 2008]. Ennek köszönhetően a DTW-alapú legközelebbi szomszéd osztályozó (és annak különböző változatai) nagy népszerűségre tettek szert az utóbbi időben.

A DTW egyik hiányossága azonban, hogy számításigényes: az előbbi eljárás szerint két l hosszúságú idősor összehasonlításának futásideje $\mathcal{O}(l^2)$ nagyságrendű. Az előbb látott példa alapján sejthetjük, hogy két idősor DTW-távolságának kiszámításakor a mátrix főátlójának közelében található elemek kapják a legfontosabb szerepet, azt várjuk, hogy a gyakorlatban előforduló esetek nagyrészében nem ront sokat az osztályozás pontosságán, ha csak a főátló közelében lévő cellákat számoljuk. Adódik tehát az ötlet, hogy csak a főátló környezetében lévő elemeket számoljuk. Empírikus eredmények azt mutatják, hogy a DTW-számítás ilyen módosítása nem csak nem rontja, hanem egyenesen *javítja* a legközelebbi szomszéd osztályozás pontosságát [Ratanamahatana és Keogh, 2004, Ratanamahatana és Keogh, 2005], amellett, hogy természetesen jelentősen gyorsít.

A DTW-alapú legközelebbi szomszéd osztályozó további gyorsításai alapulnak azon, hogy egy x idősor legközelebbi szomszédjainak meghatározásához gyakran nem szükséges a DTW-távolság pontos ismerete, elég, ha tudjuk, hogy x és az épp vizsgált idősor távolsága nagyobb, mint az eddig talált legközelebbi szomszédé [Kim és tsa., 2001, Yi és tsa., 1998]. Az idősorok egymást követő elemeinek aggregálásával csökkenthető az idősorok hossza és ezáltal szintén csökken a DTW távolság számításához szükséges futásidő. A példánykivá-

lasztás (instance selection) alapötlete az, hogy az osztályozandó x idősort a legközelebbi szomszéd osztályozó csak a tanítóhalmaz néhány, reprezentatív elemével hasonlítja össze, ahelyett, hogy x-t a tanítóhalmaz minden elemével összehasonlítaná. A reprezentatív idősorok kiválasztása általános esetben NP-nehéz (ekvivalens a halmaz-fedési problémával), a gyakorlatban azonban jó heurisztikák adhatók a csomósodás jelenségének leírásakor bemutatott $GN(x)$ és $BN(x)$ mértékek alapján, lásd [Buza, 2011d].

Idősorok klaszterezése a korábban megismert klaszterező algoritmusok DTW-t használó változatával történhet, például a hierarchikus klaszterező algoritmusok, vagy a k-Medoids könnyen adaptálhatók olyan módon, hogy távolságfüggvényként a DWT-t használják.

Idősorok semi-supervised protokoll szerinti osztályozása során klaszterező algoritmusok segíthetnek az osztályozás pontosabbá tételében [Marussy és Buza, 2013].

Az említett technikákon kívül idősorok osztályozására gyakran használnak neurális hálókat [Petridis és Kehagias, 1990] és Hidden Markov Modelleket (HMM) [Bishop, 2006]. Az idősorokkal kapcsolatban a korábban hivatkozott irodalmon felül [Lindsey, 2007]-t ajánljuk.

8. fejezet

Anomáliák feltárása

Anomáliák (más néven outlier-ek vagy különc pontok) alatt szokatlan, meglepő objektumokat értünk, olyanokat, amelyek a sokaság egészétől lényegesen eltérnek. Az anomáliák gyakran valamilyen meghibásodás, csalás, vagy más szokatlan esemény következményei. A "normális" viselkedés leírásakor célszerű lehet az anomáliákat figyelmen kívül hagyni, míg más esetekben érdemes lehet alaposabban megvizsgálni, hogy honnan származik a szokatlan objektum: *anomáliák viszonylag ritkán fordulnak elő, de amikor előfordulnak, igencsak dramatikus hatást válthatnak ki, mégpedig a szó negatív értelmében.*[1] Valós anomáliák többek között hitelkátyacsalásokhoz, biztosítási csalásokhoz, internetes rendszerek feltöréséséhez kapcsolódhatnak.

Akár alaposabban akarjuk megvizsgálni az anomáliákat, akár figyelmen kívül akarjuk hagyni azokat, szükségünk van olyan eljárásokra, amelyek képes *megtalálni* az anomáliákat.

Típusuk szerint megkülönböztethetjük a pont-anomáliákat, környezeti (koxtextuális) anomáliákat és az együttes (kollektív) anomáliákat. *Pont-anomália* alatt olyan objektumot értünk, amely önmagában is nagyban eltér a sokaság egészétől, az adatbázisbeli többi objektumtól. *Környezeti anomáliák* közé tartoznak például olyan mérési eredmények, amelyek önmagukban véve nem szokatlanok, de az adott szituációban igen: például a -5 fokos nem szokatlan januárban, de szokatlan augusztusban, ugyanazon bankkártya tranzakció az egyik ügyfélcsoportban szokatlan, a másikban hétköznapi lehet (például üzleteemberek vs. pedagógusok). Az tehát, hogy a mérési eredmény anomália-e vagy sem nagyban függ attól, hogy milyen kontextusban figyeltük meg a jelenséget. *Együttes anomáliák* alatt olyan mérési eredményeket értünk, amelyek önmagukban nem tekinthetők anomáliának, együttesen viszont igen. Ha például egy hallgató nulla pontos zárthelyit ír, az nem anomália, ha mindenki

[1] http://www.siam.org/meetings/sdm08/TS2.ppt

nulla pontos zárthelyit ír, az viszont valamilyen rendellenesség jele lehet.

Az anomália-kereső eljárások közül kiemeljük a *távolság-alapúakat*, az *osztályozásra, regresszióra illetve klaszterezésre épülő* anomália-feltáró eljárásokat, valamint a *statisztikai megközelítésen alapuló* anomália-keresést.

8.1. Távolság-alapú anomália-kereső eljárások

Számos anomália-kereső ejárás minden objektumhoz egy-egy anomália-pontszámot (anomaly score) rendel. Távolság-alapú anomália-kereső eljárások esetében egy x objektum anomália-pontszámát a többi objektumtól való távolsága alapján számoljuk. Egy ilyen anomália-pontszám például a k-adik legközelebbi szomszédtól mért távolság. Az x objektum anomália-pontszámát az alapján is számíthatjuk, hogy hány objektum található az x objektum d sugarú környezetében. Jelölje $n(x, d)$ az x-től legfeljebb d távolságra található objektumok számát, ekkor egy anomália-pontszám az $s(x) = 1/n(x, d)$.

Az egyes objektumokat sorbaállíthatjuk anomália-pontszámaik szerint, vagy anomáliának tekinthetjük azon objektumokat, amelyek egy küszöbszámnál nagyobb anomália-pontszámmal rendelkeznek.

8.2. Osztályozásra és regresszióra épülő anomália-kereső eljárások

Az anomália-keresési feladatot tekinthetjük egy kétosztályos osztályozási feladatnak: az egyik osztályba tartoznak az anomáliák, a másikba pedig a normális objektumok. Ha vannak tanítóadataink, azaz olyan objektumok, amelyekről előre tudjuk, hogy anomáliák-e vagy sem, a korábban megismert osztályozó eljárásokat használhatjuk anomália-keresésre. Az osztályozó algoritmus megválasztásakor és kiértékelésekor figyelembe kell vennünk, hogy az anomáliák ritkák, azaz egy kiegyensúlyozatlan osztályozási feladattal (imbalanced classification) állunk szemben.

Ha az anomáliákat 1-es, a normális objektumokat pedig 0-ás cimkével jelöljük, regressziós modelleket is taníthatunk és segítségükkel új objektumok anomália-pontszámát becsülhetjük.

8.3. Klaszterezés-alapú anomália-keresés

A klaszterezési feladatot az anomália-keresés duálisának is tekinthetjük: anomáliáknak tekinthetjük azokat az objektumokat, amelyeket nem sikerült "ér-

telmesen" klaszterekbe sorolni, azaz egyelemű klasztert alkotnak vagy távol esnek a kapott klaszerközéppontoktól.

8.4. Statisztikai megközelítésen alapuló anomália-keresés

A statisztikai megközelítésen alapuló anomáliakeresés során a normális viselkedést valamilyen valószínűségi eloszlással modellezzük, és azokat az objektumokat tekintjük anomáliának, amelyek nagyon valószínűtlenek a normális viselkedést leíró eloszlás tükrében.

Az anomália-kereséssel kapcsolatos további részletek tekintetében ajánljuk [Chandola és tsa., 2009]-t és a benne hivatkozott cikkeket.

9. fejezet

Adatbányászat a gyakorlatban: Weka

Ebben a fejezetben egy adatbányászati szoftert mutatunk be. A Weka egy adatbányászati szoftver, melyet az Új-Zélandon található Waikatoi Egyetemen fejlesztenek. A folyamatosan bővülő szoftvercsomagban megtaláljuk a legfontosabb adatbányászati algoritmusok implementációit. A szoftvercsomag oktatási és kutatási célra ingyenesen letölthető a

`http://www.cs.waikato.ac.nz/ml/weka`

oldalról.

A Wekát Java programozási nyelven készítették, forráskódja is nyilvános. Többféle grafikus felhasználói felülettel is rendelkezik (Explorer, Experimenter, Knowledge Flow), de egy Weka-parancssorból (Simple CLI) is meghívhatjuk az egyes algoritmusokat, valamint használhatjuk a Wekát függvénykönyvtárként saját Java nyelvű programjainkban. A következőkben a Wekához kapcsolódó legfontosabb tudnivalókat foglaljuk össze. A további részletek iránt érdeklődő olvasónak ajánljuk a fent említett weblapról kiindulva elérhető dokumentációt, [Witten és tsa., 2011]-t, valamint a magyar nyelvű szakirodalomból Abonyi János könyvét [Abonyi, 2006] illetve B. Kis Piroska és Buza Antal jegyzetét [B. Kis és Buza, 2013].

9.1. A Weka indítása

A Weka által használható memória (heap) maximális méretét a `RunWeka.ini` fájlban adhatjuk meg a `maxheap` tulajdonságnál. Első indítás előtt célszerű a `maxheap` kellőképpen nagyra állítani azért, hogy nagyobb méretű adatbázisokat is be tudjunk tölteni Weka-val.

A Weka indítása után megjelenő ablakból választhatjuk ki, hogy melyik a három grafikus felületet vagy a parancssoros interfész közül melyiket szeretnénk használni. Az egyszerűség kedvéért a következőkben végig az Explorer felület használatát tételezzük fel.

9.2. Adatok betöltése, az ARFF formátum, attribútumtípusok Weka-ban

A Weka többféle forrásban adott bemeneti adatokat képes kezelni. Ilyen források: SQL nyelven lekérdezhető adatbázis, internet (URL cím) vagy fájlok. Fájl megnyitásához klikkeljünk a `Prepocess` fül bal felső sarkában található `Open file...` gombra. A Weka többféle formátumot is támogat, leginkább az ARFF, a Weka saját formátumának használata ajánlott. A weka saját fájlformátumát `Arff`-nak nevezik. Az Arff formátum (Attribute-Relation File Format) egy olyan ASCII szöveges fájl formátum, mely azonos attribútummal rendelkező objektumok (rekordok) tárolására alkalmas. Két részből áll: *fejléc* (header) és *adat* (data).

A fejlécet egy "@RELATION" kulcsszóval kezdődő sor vezeti be. A kulcsszó után adjuk meg az adattábla (reláció) elnevezését. A fejléc tartalmazza az attribútumokat és azok típusát. A wekában használt adattípusok a következők: kategória (`nominal`), szám (`numeric`), karakterlánc (`string`) és dátum (`date`). Kategória típusú attribútumoknál fel kell sorolnunk az attribútum lehetséges értékeit. A sorrend fontos lehet bizonyos előfeldolgozási szűrőknél. A dátum típusú attribútumoknál megadhatjuk a dátum formátumát is.

Az adatrészt a "@DATA" kulcsszó vezeti be és minden sorában egy-egy rekord szerepel, amelynek attribútumértékei vesszővel vannak elválasztva. A hiányzó értékeket a ? jelzi.

A % jellel kezdődő sorok a megjegyzéseket jelölik.

Az alábbiakban egy ARFF fájlra mutatunk példát.[1] A példabeli reláció elnevezése *iris*, 5 attribútummal rendelkezik, melyek közül az első 4 numerikus, az utolsó kategória típusú, ennek lehetséges értékeit egyenként fel kell sorolni a fejlécben.

```
% 1.  Title:  Iris Plants Database
%
% 2.  Sources:
% (a) Creator:  R.A. Fisher
```

[1] Forrás: http://www.cs.waikato.ac.nz/ml/weka/arff.html

```
% (b) Donor:  Michael Marshall (MARSHALL% (c) Date:  July, 1988
%
@RELATION iris

@ATTRIBUTE sepallength NUMERIC
@ATTRIBUTE sepalwidth NUMERIC
@ATTRIBUTE petallength NUMERIC
@ATTRIBUTE petalwidth NUMERIC
@ATTRIBUTE class {Iris-setosa,Iris-versicolor,Iris-virginica}

@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
5.4,3.9,1.7,0.4,Iris-setosa
4.6,3.4,1.4,0.3,Iris-setosa
5.0,3.4,1.5,0.2,Iris-setosa
4.4,2.9,1.4,0.2,Iris-setosa
4.9,3.1,1.5,0.1,Iris-setosa
```

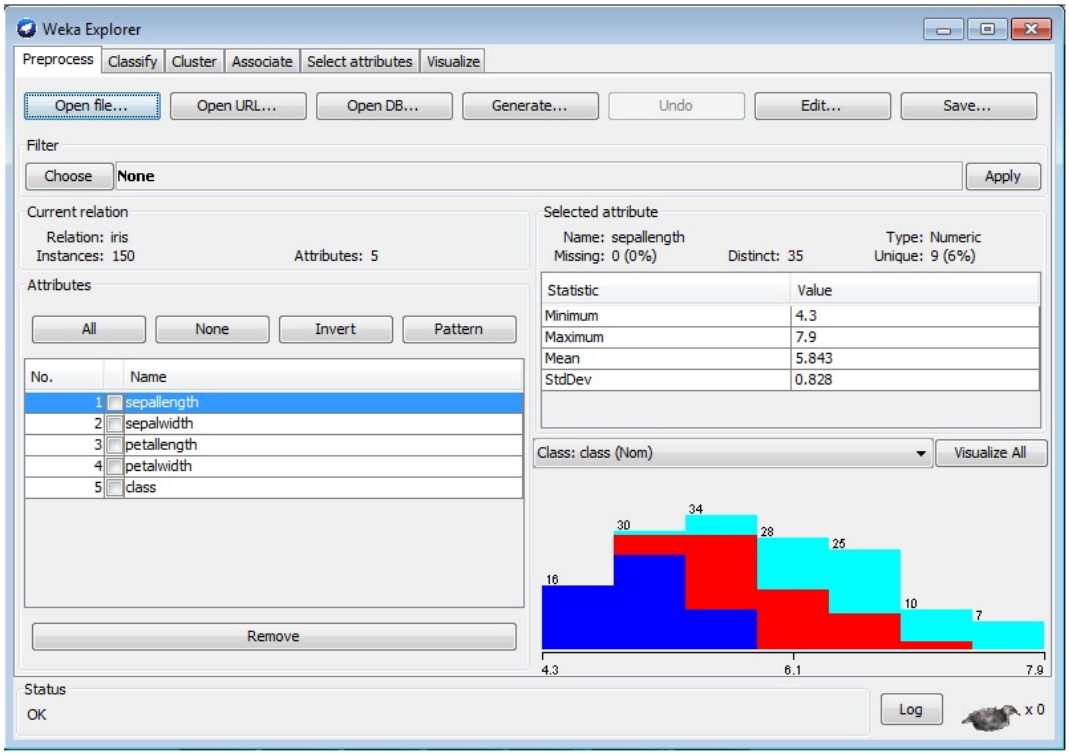
Ha az adatbázisban sok nulla érték szerepel akkor a `sparse arff` formátumot célszerű használni. Ennél a formátumnál a `data` részben attribútum sorszámból és attribútum értékből álló párok vesszővel elválasztott sorozata áll. A nulla értékeket nem rögzítjük.

A fenti fájl betöltését követően a 9.1. ábrán látható ablak jelenik meg.

9.3. Előfeldolgozás Weka-ban

Az előfeldolgozási módszereket az Explorer `preprocess` fülén (lásd 9.1. ábrát) keresztül érhetjük el. Az `Edit...` gombra klikkelve könnyen olvasható formában megjelenik az adat, amelyet közvetlen módosíthatunk is. Ha valamelyi oszlop fejlécére kattintunk, akkor az adatokat az oszlop szerint rendezve láthatjuk.

Minden előfeldolgozási eljárást két csoportba soroljuk. A `supervised` (felügyelt) módszereknél meg kell adni egy osztályattribútumot, az `unsupervised` (felügyelet nélküli) módszereknél minden attribútum egyforma. Ezen csoportokon belül megkülönböztetünk `attribute` és `instance` eljárásokat attól füg-



9.1. ábra. Az Explorer felület Preprocess füle

gően, hogy attribútumokra vagy objektumokra vonatkoznak.

Az előfeldolgozási lépéseket szűrők (filter) segítségével hajthatjuk végre. A használni kívánt szűrőt a `Filter` panel alatti `Choose` gombra klikkelve választhatjuk ki. A szűrő paramétereinek megadása szolgáló ablak a szűrő `Choose` gomb *mellett* megjelenő nevére való klikkeléssel érhető el. A beállított transzformáció elvégzéséhez, azaz a szűrő alkalmazásához az `Apply` gombra kell klikkelnünk.

A következőkben áttekintjük a Wekában megtalálható legfontosabb szűrőket.

9.3.1. Adatok konvertálása

A `weka.filters.unsupervised.attribute.MergeTwoValues` szűrő összevonja egy kategória típusú attribútum két értékét. Ha az eredeti attribútum k különböző értéket vehet fel, akkor a szűrő alkalmazása után már csak $(k-1)$-et.

A `weka.filters.unsupervised.attribute.ChangeDateFormat` szűrő egy dátum formátumú attribútum formátumát átalakít egy másik formátumba. Így

egy részletes dátumformátumból (például év, hónap, nap, óra, perc, másodperc) részinformációt (például óra, perc) nyerhetünk ki.

A `weka.filters.unsupervised.attribute.NominalToBinary` minden kategória típusú attribútumot átvált bináris attribútummá. Minden olyan A attribútumot, amely k különböző értéket vehet fel ($k > 2$), helyettesítünk k darab bináris attribútummal. Ha egy elem A attribútumának értéke az i-edik attribútum érték volt, akkor csak i-edik új attribútum értéke lesz egy, a többié pedig nulla.

A `weka.filters.unsupervised.attribute.NumericToNominal` szűrő a szám típusú attribútumokból kategória típusúakat állít elő. Ezt egyszerűen úgy végzi, hogy minden egyes számot kategória típusú attribútum egy értékeként kezel, és hozzáadja az attribútum értékhalmazához.

9.3.2. Hiányzó értékek kezelése

A `weka.filters.unsupervised.attribute.ReplaceMissingValues` szűrő a hiányzó értékek helyettesítésére szolgál. Kategória típusú attribútumoknál a leggyakrabban előforduló értékkel (módusz), szám típusúaknál pedig az átlaggal helyettesít.

9.3.3. Új attribútumok létrehozása

A `weka.filters.unsupervised.attribute.Add` szűrő egy új attribútumot hoz létre. Minden elem ezen attribútuma üres (hiányzó) lesz. Kategória típusú attribútum létrehozásához meg kell adnunk a lehetséges értékeket.

A `weka.filters.unsupervised.attribute.AddExpression` szűrővel új attribútumot származtathatunk meglévő attribútumokból. Az meglévő attribútumokra, mint `a1`, `a2`, ... hivatkozhatunk. A felhasználható operátorok a következők: összeadás, kivonás, szorzás, osztás, hatványozás, logaritmus, exponenciális, szinus, coszinus, tangens, egészrész képzés és a kerekítés.

A `weka.filters.unsupervised.attribute.AddID` szűrő egy azonosító attribútumot ad az adathalmazhoz. Minden elem (sor) azonosítója egyedi lesz.

A `weka.filters.unsupervised.attribute.Copy` egy meghatározott attribútumhalmazt duplikál. Ezt a szűrőt általában olyan más szűrőkkel együtt szokás használni, amelyek felülírják az adatokat. Ebben az esetben lehetővé válik az eredeti attribútum megőrzése az új mellett.

A `weka.filters.unsupervised.attribute.FirstOrder` szűrő szám típusú attribútumok k elemű sorozatából készít egy $(k-1)$-eleműt, az egymást követő tagok különbségének képzésével. Például az 1,2,1 sorozatból 1,-1 sorozatot készít.

A `weka.filters.unsupervised.attribute.MathExpression` végrehajt egy megadott függvényt a kiválasztott típusú attribútumokon. A függvényt az `expression` paraméterrel adjuk meg ('A' betűvel lehet az attribútumra hivatkozni). A $MIN, MAX, MEAN, SD$ változók az attribútum minimumát, maximumát, átlagát és szórását jelölik. A támogatott műveletek listája az alábbi: +, -, *, /, `pow`, `log`, `abs`, `cos`, `exp`, `sqrt`, `tan`, `sin`, `ceil`, `floor`, `rint`, `(`, `)`,`A`, `COUNT`, `SUM`, `SUMSQUARED`, `ifelse`.

A `weka.filters.unsupervised.attribute.NumericTransform` szűrő a szám típusú attribútumokon hajt végre egy eljárást. A `className` paraméterrel adható meg az osztály, amely a felhasználni kívánt eljárást tartalmazza. A `methodName` opció segítségével adjuk meg a metódus nevét.

9.3.4. Attribútumok törlése

A `weka.filters.unsupervised.attribute.Remove` törli az általunk megadott attribútumokat. Használjuk a `weka.filters.unsupervised.attribute.RemoveType` szűrőt, ha az összes, adott típusú attribútumot törölni kívánjuk.

`weka.filters.unsupervised.attribute.RemoveUseless` szűrő a haszontalan attribútumokat törli, azokat, melyek vagy egyáltalán nem vagy nagyon sokat változnak.

9.3.5. Zajszűrés, hibás bejegyzések eltávolítása

A `weka.filters.unsupervised.attribute.InterquartileRange` szűrő a különc pontokat és az extrém értékeket deríti fel. Jelöljük $Q1$, $Q3$-mal a 25%-hoz és 75%-hoz tartozó kvantiliseket, legyen $IQR = Q3 - Q1$, továbbá OF és EVF a felhasználó által megadott két érték (Outlier Factor és Extreme Value Factor). Extrémnek nevezünk egy értéket, ha az nagyobb, mint $Q3 + EVF * IQR$, vagy kisebb, mint $Q1 - EVF * IQR$. Különc pontok közé soroljuk azokat az értékeket, amelyen nem extrémek és nem esnek a $[Q1 - OF * IQR, Q3 + OF * IQR]$ intervallumba sem. Ha az `outputOffsetMultiplier` paramétert igazra állítjuk, akkor a szűrő új attribútumot hoz létre, amelynek értéke a $(A - median)/IQR$ lesz (A az attribútum érték jelöli).

A `weka.filters.unsupervised.instance.RemoveWithValues` szűrővel azokat az elemeket törölhetjük az adathalmazból, amelyek adott attribútuma adott értéket vesz fel.

A `weka.filters.unsupervised.attribute.NumericCleaner` szűrő a `maxThreshold` paraméternél nagyobb értékeket `maxDefault` értékkel, a `minThreshold` paraméternél kisebbeket `minDefault` értékkel és a `closeTo` paraméterhez közeli (`closeToTolerance`) értékeket `closeToDefault` értékkel helyettesíti.

A `weka.filters.unsupervised.instance.RemoveMisclassified` lefuttat egy osztályozó módszert, majd törli a rosszul osztályzott elemeket.

9.3.6. Adatok torzítása

A `weka.filters.unsupervised.attribute.AddNoise` osztály az elemek adott részének megváltoztatja adott attribútumának értékét.

A `weka.filters.unsupervised.attribute.Obfuscate` szűrő megváltoztatja az attribútumok nevét és átnevezi az attribútumértékeket.

9.3.7. Diszkretizálás

A 3.3.4. fejezetben bemutatott egyenlő szélességű vagy egyenő gyakoriságú intervallumokat kialakító diszkretizációs eljárásokat a `weka.filters.unsupervised.attribute.Discretize` szűrőn keresztül érhetjük el. A `useEqualFrequency` paraméterrel adhatjuk meg, hogy a két lehetőség közül melyiket választjuk.

A PKI módszert a `weka.filters.unsupervised.attribute.PKIDiscretize` osztály implementálja.

9.3.8. Normalizálás

A 3.3.5. fejezetben látott két normalizáló eljárást a `weka.filters.unsupervised.attribute.Normalize` és a `weka.filters.unsupervised.attribute.Standardize` szűrők implementálják. Itt kell megemlítenünk a `weka.filters.unsupervised.attribute.Center` osztályt is, amely csak annyit tesz, hogy minen értékből kivonja az átlagot ($a'_j = a_j - \bar{A}$).

9.3.9. Mintavételezés

A `weka.filters.unsupervised.instance.RemovePercentage` szűrő az elemek egy adott százalékát törli.

A `weka.filters.unsupervised.instance.RemoveFolds` megkeveri az adatbázist, majd egyenlő méretű részekre osztja és megtartja az egyik részt. A szűrőt kereszt-validációhoz szokták használni (lásd a 4.10.2. fejezetet). Amennyiben azt szeretnék, hogy az osztályok eloszlása megegyezzen minden részben (lásd a 4.10.1.. fejezetet), akkor használjuk a `weka. filters. supervised. instance. StratifiedRemoveFolds` szűrőt .

A `weka.filters.unsupervised.instance.Resample` szűrő egy véletlenszerű részhalmazát képzi az adathalmaznak. A `sampleSizePercent` opcióval százalékosan fejezhetjük ki az részhalmaz méretét az eredeti adathalmazhoz képest. Megadjuk, hogy a mintavételezéshez visszatevéses vagy visszatevés

nélküli módszert használjon. Az eredeti adathalmaznak el kell férnie a memóriában. A szűrő felügyelt változata (`weka.filters.supervised.instance.Resample`) abban különbözik a felügyelet nélküli változattól, hogy befolyásolhatjuk a mintában az osztály eloszlását. Ha a `biasToUniformClass` értéke 0, akkor nem változik az osztály eloszlása, ha viszont 1, akkor minden osztály ugyanannyiszor fog előfordulni.

A `weka.filters.unsupervised.instance.ReservoirSample` Vitter "R" algoritmusát felhasználva mintavételez.

A `weka.filters.supervised.instance.SpreadSubsample` mintavételező szűrő olyan részhalmazt fog előállítani, amelyben a leggyakoribb és a legritkább osztályok előfordulásának hányadosa kisebb, mint egy előre megadott konstans (`distributionSpread`).

9.3.10. Dimenziószámcsökkentés

A SVD-t a `weka.attributeSelection.LatentSemanticAnalysis` osztályon keresztül érhetjük el. Amennyiben a `rank` értéke egynél nagyobb, akkor a `rank` a k-t adja meg (figyelembe vett szinguláris értékek számát). Ellenkező esetben a közelítés relatív pontosságát definiálhatjuk (lásd a 3.2 képletet). Ha a `normalize` paraméternek igaz értéket adunk, akkor a weka az SVD elvégzése előtt az attribútumokat normalizálni fogja. Célszerű a normalizálást elvégezni, ugyanis az SVD során a hibát a Frobeniusz normával számítjuk, amely attribútumértékek különbségének négyzetével számol, így a nagy értékekkel rendelkező attribútumok nagy jelentőséget kapnak.

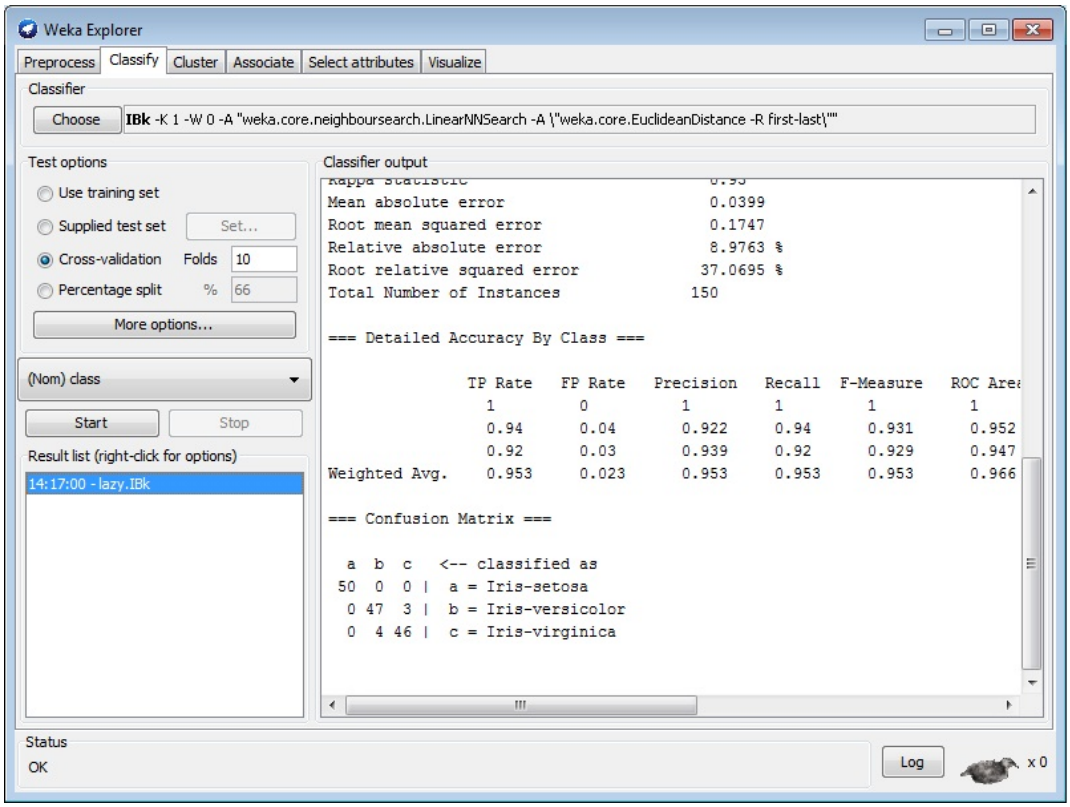
A főkomponens analízist a `weka.filters.unsupervised.attribute.PrincipalComponents` szűrő hajtja végre.

9.4. Osztályozó eljárások Weka-ban

A wekában az osztályozó módszereket a `Classify` fülön keresztül érjük el, lásd a 9.2. ábrát.

A használni kívánt osztályozó eljárást a `Choose` gombbal választhatjuk ki. A kiválasztott eljárás nevét tartalmazó szövegmezőre klikkelve megjelenő ablakban adhatjuk meg az eljárás paramétereit. A kiválasztott osztályozó módszer alkalmazását a `Start` gombra klikkelve indíthatjuk el.

A wekában az osztályozás kiértékelésének módját a `Test options` panelen adhatjuk meg. `Use training set` esetén a hibát és egyéb paramétereket a tanítóhalmazon mérjük. `Supplied test set` estén külön teszthalmazt adhatunk meg, `Cross-validation` választásakor kereszt-validációt használunk. A



9.2. ábra. Az Explorer felület Classify füle

`Folds` paraméterrel adhatjuk meg, hogy hány részre ossza a weka a tanítóhalmazt. Ha a hagyományos tanítóhalmaz, teszthalmaz kettéosztást kívánjuk használni, akkor válasszuk a `Percentage split` opciót. Ilyenkor megadhatjuk tanítóhalmazba kerülő elemek százalékos arányát. A kiértékelési protokollokat részletesebben lásd a 4.10.2. fejezetben.

Osztályozás végrehajtása után a weka alapértelmezés szerint kirajzolja a keveresési mátrixot a kimeneti panelen (`Classifier output`). Ha erre nem vagyunk kíváncsiak, akkor a `Test options` panelen klikkeljünk a `More options...` feliratú gombra. Ez felhozza a `Classifier evaluation options` panelt, itt tőrőljük az `output confusion matrix` kijelölését. Itt állíthatjuk többek kőzőtt azt is, hogy megjelenjen-e az osztályozó által előállított modell (döntési szabályok, fák, feltételes valószínűségi táblák – Bayes osztályozók esetében, stb.).

A kimeneti panelen (`Classifier output`) mindig megjelenik a jól/rosszul osztályozott és a nem osztályozható elemek száma (`Correctly/Incorrectly Classified Instances`, `UnClassified Instances`) és ezen értékek összes ta-

nítóponthoz viszonyított aránya, a kappa statisztika, az abszolut hibák átlaga (`Mean absolute error` – lásd 176 oldal), a négyzetes hibaátlag (`Root mean squared error`), a relatív abszolut hibaátlag (`Relative absolute error`), a relatív négyzetes hibaátlag (`Root relative squared error`).

Ha a `Classifier evaluation options` ablakban (melyet a `Test options` panel `More options...` feliratú gombján keresztül érhetünk el) bejelőljük az `Output per-class stats` opciót, akkor minden osztályhoz megkapjuk a TP és FP arányt, a precisiont, a felidézést (recall-t), az F-mértéket ($\alpha = 0.5$ mellett) és a ROC görbe alatti területet.

A következőkben a Wekában implementált osztályozó eljárások közül tekintünk át néhányat.

9.4.1. Legközelebbi szomszéd osztályozó

A legközelebbi szomszéd módszerét (tehát amikor csak egy szomszédot veszünk figyelembe) a `weka.classifiers.lazy.IB1` osztály implementálja. Két pont távolságának meghatározásánál az euklideszi normát használja. Ha több legközelebbi pontja van egy osztályozandó pontnak, akkor az elsőként megtalált alapján fog osztályozni.

A k-legközelebbi szomszéd futtatásához $k > 1$ esetén használjuk a `weka.classifiers.lazy.IBk` osztályt. A legközelebbi szomszédok számát, k-t, nem kell feltétlenül megadnunk: amennyiben a `crossValidate` értéke igaz, a Weka a tanítóhalmazon végzett leave-one-out módszerrel (lásd a 172 oldal) megpróbálja a megfelelő k értéket meghatározni.

Használhatjuk a súlyozott legközelebbi szomszéd módszert is (lásd a 112). Ekkor választanunk kell a `distanceWeighting` paraméterrel, hogy a súly a távolság reciproka, vagy 1−távolság legyen.

A `nearestNeighbourSearchAlgorithm` kiválasztóval megadhatjuk, hogy a legközelebbi szomszédok meghatározásához milyen módszert/adatstruktúrát használjon a weka. Az alapértelmezett az egyszerű lineáris keresés, de választhatunk KD-fa, Ball tree és Cover tree alapú megoldások közül.

9.4.2. Regressziós eljárások

A Winnow, illetve a kiegyensúlyozott Winnow módszert a wekában a `weka.classifiers.functions.Winnow` osztály implementálja. A `balanced` paraméter igazra állításával adhatjuk meg, ha kiegyensúlyozott Winnow módszert szeretnénk alkalmazni. A súlyok kezdeti értékét a `defaultWeight` paraméterrel, az iterációk számát a `numIterations` paraméterrel szabályozhatjuk. A Θ paraméter a wekában a `treshold` paraméternek felel meg.

A `weka.classifiers.functions.SimpleLinearRegression` osztály egyetlen attribútum szerinti lineáris regressziót hajt végre. Azt az attribútumot választja, amely a legkisebb négyzetes hibát adja. Csak szám típusú attribútumokkal tud dolgozni és hiányzó értékeket nem enged meg.

A `weka.classifiers.functions.LinearRegression` osztály szintén lineáris regressziót hajt végre, de ez már több attribútumot is figyelembe tud venni. Lehetőség van a regresszióba felhasználandó attribútumok automatikus kiválasztására is az `attributeSelectionMethod` paraméterrel.

A négyzetes hibák átlaga helyett a hibák mediánját minimalizálja a `weka.classifiers.functions.LeastMedSq` osztály.

Logisztikus regressziót a `weka.classifiers.functions.Logistic` és a `weka.classifiers.functions.SimpleLogistic` függvények implementálják.

9.4.3. Neurális hálózatok

A backpropagation tanító módszert használó neurális hálózatot a `weka.classifiers.functions.MultilayerPerceptron` osztály implementálja. A hálózatot felépíthetjük kézzel vagy automatikusan. A neuronokban használt nemlinearitás a szigmoid függvény. Az osztálynak számos paramétere van:

1. A `GUI` paraméterrel bekapcsolhatunk egy grafikus interfészt, melyen keresztül láthatjuk, illetve módosíthatjuk a neurális hálózatot, lásd a 9.3. ábrát. A hálózat módosításához bal egérgombbal történő klikkeléssel jelölhetünk ki egy neuront. Ezt követően, ha a bal egérgombbal üres helyre klikkelünk, új neuront vehetünk fel a hálózatba, amely össze lesz kötve a kijelölt neuronnal. Ha ki van jelölve egy neuron, és egy meglévő neuronra klikkelünk, új kapcsolatot hozhatunk létre. A jobboldali egérgombbal üres területre klikkelve megszüntethetjük a kijelölést. Egy meglévő neuront úgy törölhetünk, hogy a jobboldali egérgombbal klikkelünk rá. A neuronok és kapcsolatok szerkesztése után a `Start`-ra klikkelve indíthatjuk a backpropagation algoritmust, amely az élsúlyokat határozza meg. Ha ezt követően az `Accept`-re klikkelünk, a Weka a teszthalmaz osztályozására fogja használni a hálózatot és értékelni fogja az osztályozást.

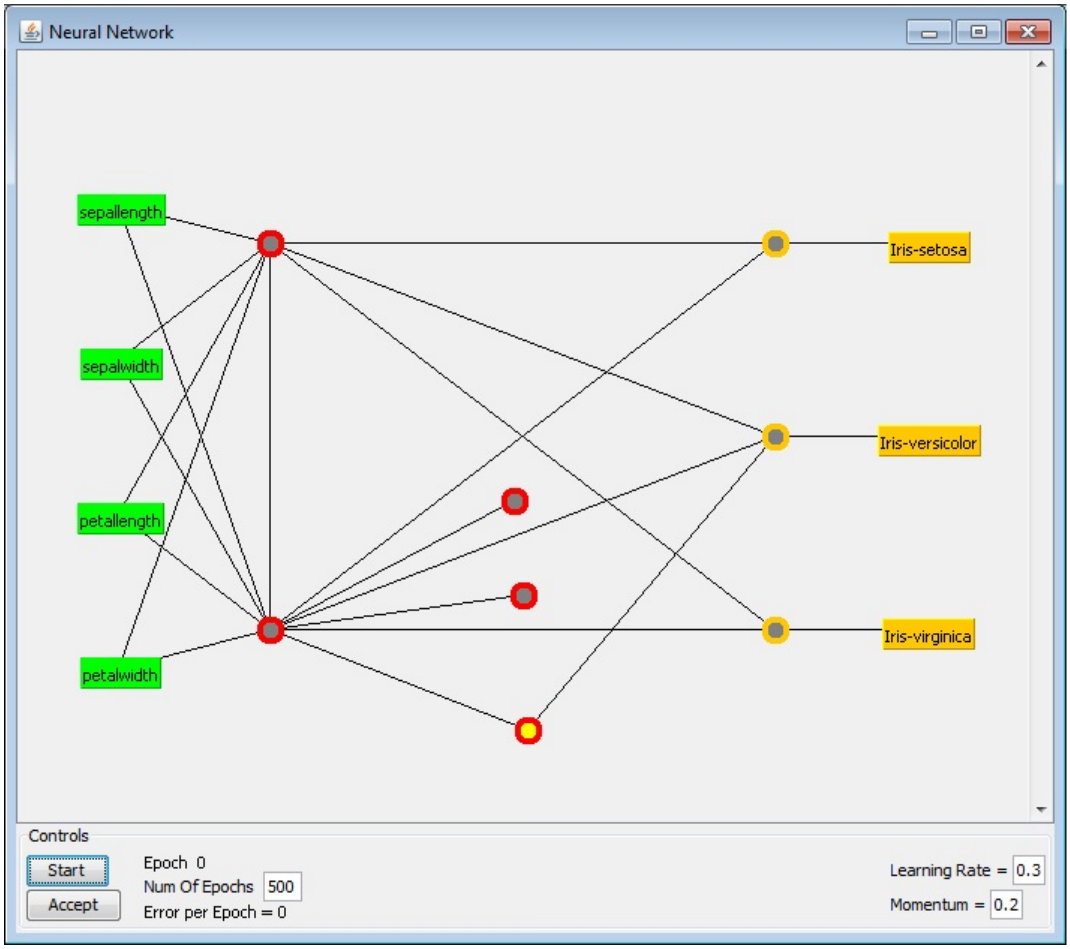
2. Az `autoBuild` paraméter engedélyezésével a hálózat automatikusan bővül további rejtett rétegekkel.

3. A `hiddenLayers` paraméter adja meg a neurális hálózat rejtett rétegeinek a számát. Az attribútumok előfeldolgozására ad lehetőséget a `nominalToBinaryFilter` paraméter. A kategória típusú attribútumokat bináris típusúvá alakítja (lásd 9.3.1 részt).

4. Az attribútumok normalizálását a `normalizeAttributes` paraméterrel tudjuk engedélyezni.

5. A `normalizeNumericClass` paraméter az osztályattribútumot normalizálhatjuk, amennyiben az szám típusú.

6. A `validationSetSize` paraméter a teszthalmaz százalékos méretét adja meg. A tesztelés leállását szabályozza a `validationThreshold`. Ez az érték adja meg, hogy egymás után hányszor romolhat a tesztelési hiba, mielőtt leállna a tanítás.



9.3. ábra. Neurális hálózatok a Weka grafikus felületén keresztül szerkeszthetőek.

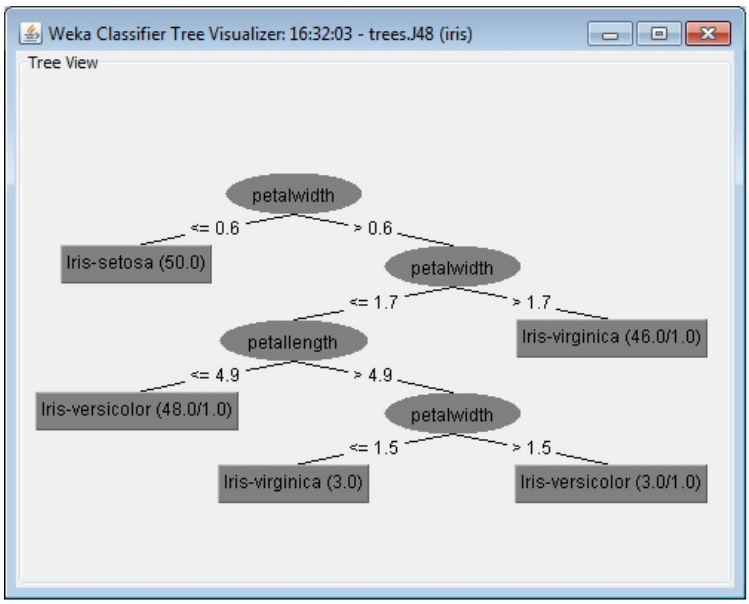
9.4.4. Szabály-alapú osztályozók

A wekában a 0R és 1R módszereket a `weka.classifiers.rules.ZeroR` és a `weka.classifiers.rules.OneR` osztályok implementálják. Az 1R módszer egyetlen paramétere a diszkretizálás során használt elemszám küszöb.

A wekában a Prism módszert a `weka.classifiers.rules.Prism` osztály implementálja.

9.4.5. Döntési fák és regressziós fák

A döntési fákkal kapcsolatos módszereket a `weka.classifiers.trees` csomagban találjuk. A `Classifier output` ablakban a döntési fát szövegesen megjelenítve láthatjuk, amennyiben nem kapcsoljuk ki a `Classifier evaluation options` panelen az `Output model` kapcsolót. A döntési fa grafikus megjelenítéséhez jobb gombbal klikkeljünk a `Result list` panel a megfelelő elemére és válasszuk a `Visualize tree` lehetőséget. A döntési fa grafikus megjelenítésére mutat példát a 9.4. ábra.



9.4. ábra. Döntési fa grafikus megjelenítése.

A döntési fa interaktív előállítását teszi lehetővé a `weka.classifiers.trees.UserClassifier` osztály. A módszer elindítása után egy ablak jelenik meg amelynek két füle van. A `Tree Visualizer` fülön az aktuális fát láthatjuk, a `Data Visualizer` pedig a kijelölt fa csomópontjának tanítópontjai jeleníti

meg. Itt állíthatjuk elő a vágási függvényt, amelyhez vizuális segítséget kapunk. Az osztály eloszlását láthatjuk két tetszőlegesen kiválasztható attribútum értékeinek függvényében. Ez alapján kijelölhetünk egy téglalapot, poligont vagy összekötött szakaszokat, amely kettéválasztja a pontokat. Akkor jó a kettéválasztás, ha az osztályattribútum szerint homogén csoportok jönnek létre.

A Wekában az Id3 algoritmust a `weka.classifiers.treea.Id3` osztály implementálja. A C4.5 egy továbbfejlesztett változatának java implementációja a `weka.classifiers.trees.J48` osztály.

9.4.6. Bayes-osztályozók

Néhány Bayes-alapú módszer található a `weka.classifiers.bayes` csomagban.

A naív Bayes osztályozót, amely a szám típusú attribútumoknál normális eloszlást feltételez a `weka.classifiers.bayes.NaiveBayesSimple` osztály implementálja.

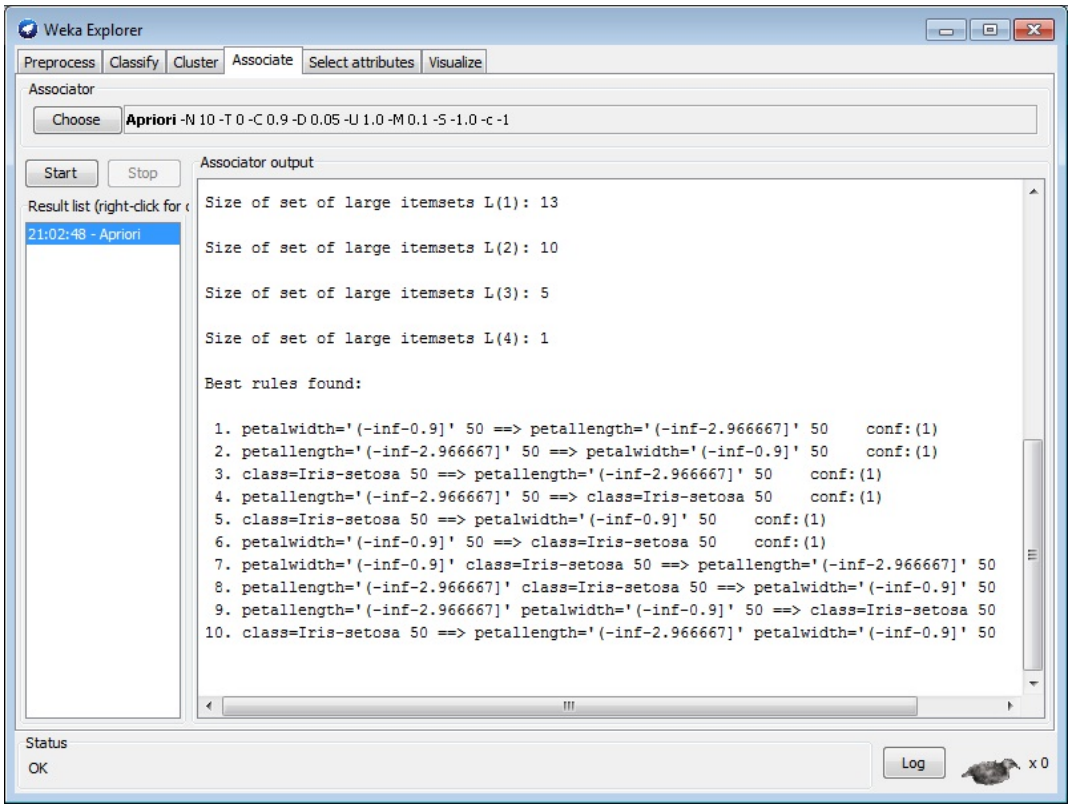
A `weka.classifiers.bayes.NaiveBayes` a normalitásra tett feltételt enyhíti. Alapesetben ez az osztályozó ún. kernel becslőt használ a keresett valószínűségek meghatározásához. Ha pedig a `useSupervisedDiscretization` paramétert igazra állítjuk, akkor a szám típusú attribútumokat kategória típusúvá alakítja egy felügyelt diszkretizáló módszerrel (`weka.filters.supervised.attribute.Discretize` szűrővel).

9.5. Asszociációs szabályok bányászata

Az asszociációs szabályokkal kapcsolatos osztályokat az `Explorer Associate` fülén keresztül érhetjük el.

Asszociációs szabályokat a `weka.associations.Apriori` osztály segítségével nyerhetünk ki. Használatához a numerikus attribútumokat nominálissá kell alakítanunk a `Preprocess` fülről elérhető szűrők segítségével.

A `weka.associations.Apriori` osztály nem a klasszikus asszociációs szabály kinyerésének feladatát oldja meg – amely szerint adott min_supp, min_conf, min_lift mellett határozzuk meg az *összes* érvényes asszociációs szabályt – hanem csak `numRules` darab szabályt ad meg, ahol `numRules` a felhasználó által megadott paraméter. Ehhez a min_supp értéket egy kiindulási értékről (`upperBoundMinSupport` paraméter) mindig `delta` értékkel csökkenti és ellenőrzi, hogy van-e legalább `numRules` darab érvényes szabály. Ha van, akkor kiírja a legjobb `numRules` szabályt, ha nincs, akkor tovább csökkenti min_supp-ot. A minimális támogatottsági küszöböt nem csökkenti `lowerBoundMinSupport` alá.



9.5. ábra. Az Explorer felület Associate füle

A `metricType` paraméterrel adhatja meg a felhasználó, hogy mi alapján rangsorolja az asszociációs szabályokat a weka. Az empirikus kovarianciát a `Leverage` jelöli.

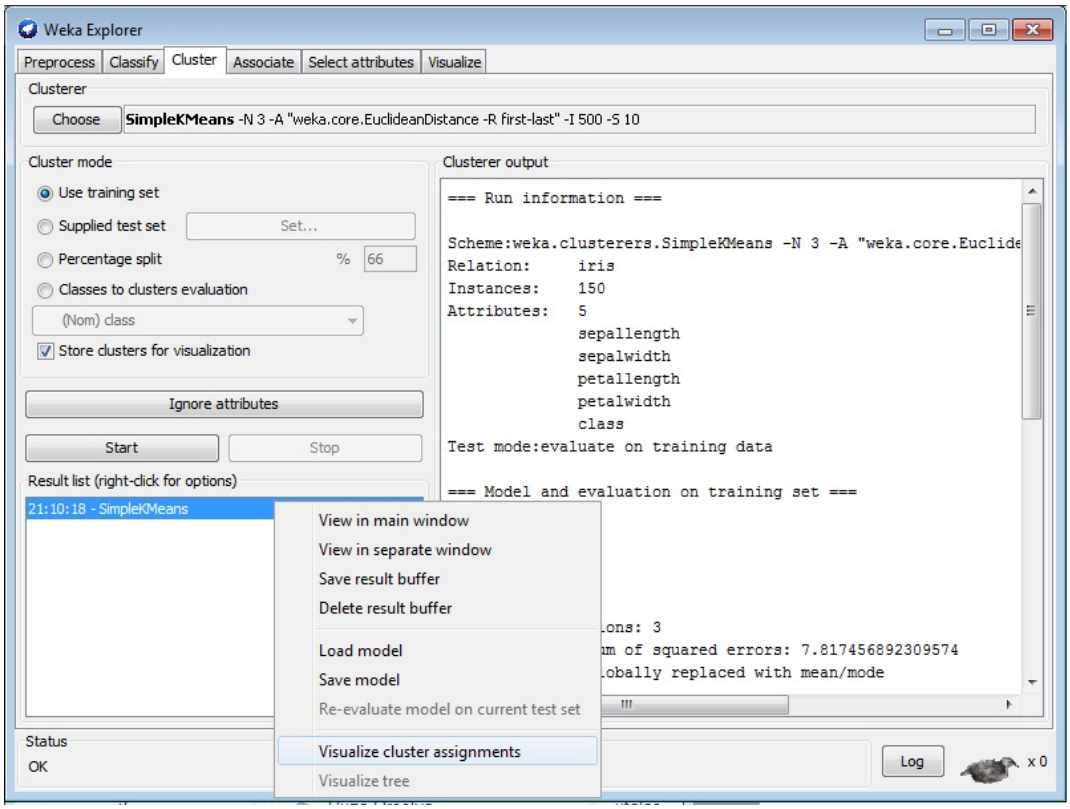
Lehetőségünk van egy osztályattribútumot kijelölni a `car` paraméter igazra állításával és a `classIndex` megadásával. Ekkor csak olyan szabályokat fog a weka előállítani, amelyek következményrészében csak az osztályattribútum szerepel.

Gyakori elemhalmazokat úgy nyerhetünk ki, ha az `outputItemSets` paramétert igazra állítjuk.

9.6. Klaszterező eljárások Weka-ban

A klaszterező módszereket az `Experimenter` alkalmazás `Cluster` fülén keresztül érhetjük el (9.6. ábra).

A k-közép algoritmust a `weka.clusterers.SimpleKMeans`, a DBScan al-



9.6. ábra. Az Explorer felület Cluster füle

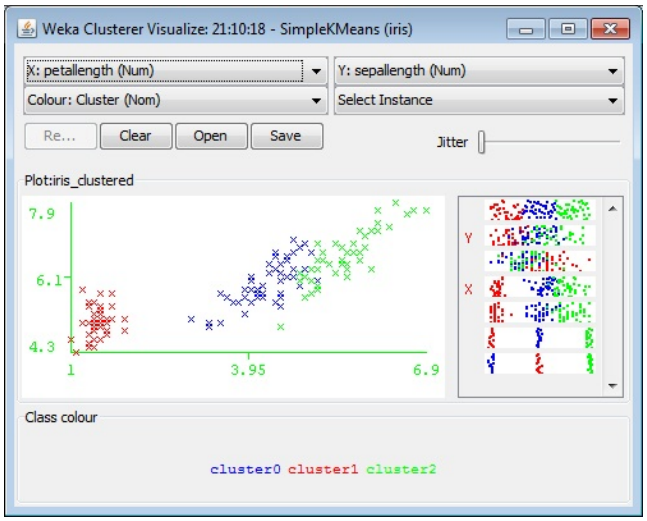
goritmust a `weka.clusterers.DBScan` osztály implementálja. A klaszterezés eredményének grafikus megtekintéséhez (9.7. ábra) jobb egérgombbal klikkeljünk a `ResultList` panel megfelelő elemére és válasszuk a `Visualize Cluster Assignments`-t.

9.7. Weka használata függvénykönytárként

Az alábbi példa illusztrálja a Weka függvénykönyvtárként történő használatát saját Java kódban. Ahhoz, hogy a Weka osztályait el tudjuk érni, a classpathnak tartalmaznia kell a `weka.jar`-t. A Java nyelv részletes ismertetése túlmutat a jelen jegyzet keretein, az érdeklődő olvasónak ajánljuk [Lakatos és Nyékiné, 2009]-t valamint ezt a weblapot:

`http://weka.wikispaces.com/Use+Weka+in+your+Java+code`

```
import java.io.BufferedReader;
```



9.7. ábra. Klaszterezés eredményének grafikus megjelenítése

```
import java.io.FileReader;
import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.trees.J48;
import weka.clusterers.ClusterEvaluation;
import weka.clusterers.Clusterer;
import weka.clusterers.SimpleKMeans;
import weka.core.Instances;
import weka.filters.Filter;
...
// Ezek mintájára importálandó az összes osztály,
// amit használni fogunk

public class HelloWekaWorld {

public static void main(String[] args) throws Exception {

Instances data = new Instances(new BufferedReader(
new FileReader(" (...)
breast-cancer.arff")));
data.setClassIndex(data.numAttributes()-1);

Instances traindata = data.trainCV(10, 0);
```

```
Instances testdata = data.testCV(10, 0);

Classifier c = new J48();
((J48) c).setReducedErrorPruning(true);
((J48) c).setMinNumObj(5);
c.buildClassifier(traindata);

for (int i=0;i<testdata.numInstances();i++) {
double predicted_class = c.classifyInstance(testdata.instance(i));
System.out.println(i+" "+predicted_class);
}

Evaluation eval = new Evaluation(traindata);
eval.evaluateModel(c, testdata);
System.out.println(eval.toSummaryString("Eredmény:", false));

for (int i=0;i<data.numClasses();i++) {
System.out.println("AUC (osztály="+i+")"+
eval.areaUnderROC(i));
}

System.out.println("Accuracy:"+ (eval.pctCorrect()/100) );


// Klaszterezés

// Az osztályattribútum eltávolítása a klaszterezend{o} adatból
// a data.classindex() metódus 0-val kezd{o}d{o}en számolja az attribútumokat,
// a filter.setAttributeIndices viszont String-et vár és 1-gyel kezd{o}d{o}en
számolja // az attribútumokat weka.filters.unsupervised.attribute.Remove
filter =
new weka.filters.unsupervised.attribute.Remove();
filter.setAttributeIndices("" + (data.classIndex() + 1));
filter.setInputFormat(data);
Instances dataClusterer = Filter.useFilter(data, filter);

Clusterer clus = new SimpleKMeans();
// Ha nem Euklideszi távolságot akarunk használni...  //( (SimpleKMeans)
clus).setDistanceFunction(new weka.core.ChebyshevDistance());
( (SimpleKMeans) clus).setNumClusters(2);
```

```
/* További klaszterező például:
Clusterer clus = new HierarchicalClusterer();
((HierarchicalClusterer)clus).setOptions(new String[]
{ "-N", "3", "-L", "SINGLE", "-A", "weka.core.ChebyshevDistance" });
*/
// Link options:  SINGLE COMPLETE AVERAGE
// Distance functions:  ChebyshevDistance, EuclideanDistance, ManhattanDistance

clus.buildClusterer(dataClusterer);

ClusterEvaluation eval1 = new ClusterEvaluation();
eval1.setClusterer(clus);
eval1.evaluateClusterer(data);
System.out.println(eval1.clusterResultsToString());

// Igy kapjuk számként, a clustering_error elnevezesu float típusú
// változóban a helytelenul klaszterezett példányok számát
String cluster_results = eval1.clusterResultsToString();
float clustering_error = getIncorrectlyClusteredInstances(cluster_results);
System.out.println(clustering_error);


// Asszociációs szabályok

Apriori a = new Apriori();
a.setOptions(new String[] { "-M", "0.3", "-C", "0.7", "-N", "10000"});
a.buildAssociations(data);
System.out.println(a.toString());

FastVector[] rules = a.getAllTheRules();
int[] items = ((AprioriItemSet) rules[0].elementAt(0)).items();
for (int i=0;i<items.length;i++) System.out.println( items[i] );
} /* main() */

// Ez a metódus "kihámozza" a klaszterézes eredményeként kapott stringből
// a helytelenül klaszterezett példányok számát, és visszaadja számként
public static float getIncorrectlyClusteredInstances(String clustering_result)
{
String pattern = "Incorrectly clustered instances :";
String s = clustering_result.substring(
clustering_result.indexOf(pattern)+pattern.length());
```

```
s = s.trim();
return new Float(s.split("")[0]);
} /* getIncorrectly...() */
} /* class HelloWekaWorld */
```

Tárgymutató

χ^2 próba, 54
átlagos négyzetes hiba, 177
átlagos négyzetes hibagyök, 177
út (gráfban), 58
min_supp, 205
Apriori, 209
Eclat algoritmus, 226
FP-growth algoritmus, 230

A minta nagysága, 235
abszolút hibaátlag, 177
adatbázis
    horizontális, 206
    vertikális, 206
adjacency matrix, 58
aggregáció, 103
algoritmus
    helyesen működő, 272
    teljes, 272
anomália, 364
anti-monoton, 266
anytime decision tree, 147
APRIORI
    módszer, 272
APRIORI-CLOSE, 276
asszociációs szabály, 242
    érdekessége, 245
    érvényes, 243
    bizonyossága, 242
    egzakt, 243
    hierarchikus, 258, 259
    támogatottsága, 242
AUC, 175

average linkage, 345

backward selection, 115
bagging, 167
bemeneti sorozat, 204
boosting, 166

centrális momentum, 48
collaborative filtering, 183
complete linkage, 345
csökkentő módszer, 115
csomósodás, 99
curse of dimensionality, 88

döntési fa, 30
dimenzióátok, 88
dinamikus idővetemítés, 359
DTW, 359
duplikátumkeresés gépi tanulással, 103
duplikátumok, 101
Duquenne–Guigues-bázis, 244
dynamic time warping, 359

early classification, 355
early clustering, 355
egyváltozós, 353
ekvivalencia-reláció, 43
előmetszés, 155
elemhalmaz, 292
    fedés, 205
    gyakori, 205
    gyakorisága, 205
    pszeudo-zárt, 244
    zárt, 244

elemi modell, 167
elméleti regressziós görbe, 110
eloszlás
    χ^2, 51
    binomiális, 50
    hipergeometrikus, 51
    normális, 49
    Poisson, 50
eloszlásfüggvény, 45
ensemble modellek, 163
entrópia, 52
erdő, 58
Euklideszi-norma, 70

független valószínűségi változók, 49
függetlenségvizsgálat, 55
főkomponens analizis, 90
fa, 58
FarthestFirst, 341
feature extraction, 188
felügyelet nélküli tanulás, 323
feladat-alapú kiértékelés, 337
felidézés, 174
feltételes valószínűség, 47
feltételesen független valószínűségi változók, 49
ferdeség, 67
feszítő fa, 58
forwald selection, 115
FP-fa
    vetített, 232
FUP algoritmus, 289

Galois-kapcsolat, 237
Galois-lezárás operátor, 237
gráf, 57
gráfok izomorfiája, 58
GSP, 296
gyökeres fa, 58
gyakorisági küszöb, 277
gyakorisági küszöböt, 205

halmaz, 42
    lokálisan véges, 265
    rangszámozott, 266
halmazcsalád, 223
hibajavító kód, 187
hierarchikus asszociációs szabály
    érdekessége, 260
Hoeffding-korlát, 51
hub, 99
hybrid collaborative filtering, 184

invariáns távolság, 69
irányított gráf, 57
Ismételt mintavételezés, 172
ismétlődő részfa probléma, 145
item-based collaborative filtering, 184
izomorf gráfok, 58

Jaccard-koefficiens, 69
jelölt, 208, 273
    hamis, 273
jelölt-előállítás
    ismétlés nélküli, 209
jellemző kinyerés, 188

k-legközelebbi szomszéd gráf, 328
kör (gráfban), 58
különc pont, 364
kényszer
    erősen átalakítható, 270
kanonikus reprezentáció, 223
kappa statisztika, 173
kereszt-validáció, 172
kernel, 163
kernel trükk, 163
keveredési mátrix, 173
kimerítő keresés, 190
Klaszterezés, 322
kontingencia-táblázat, 55
korai osztályozás, 355
korrelációs együttható, 177
koszinusz-mérték, 73

Laplace estimation, 158
lapultság, 67
leave-one-out, 172
lexikografikus rendezés, 43
lineáris kiterjesztés, 267
lineárisan szeparálható osztályok, 120
logisztikus függvény, 130
logit függvénynek, 130

mátrix faktorizáció, 183
módusz, 48
Manhattan-norma, 70
margin, 162
maximal margin hyperplane, 162
MDS, 94
metamodell, 167
metszés (döntési fa), 154
min_freq, 205, 277
Minkowski-norma, 70
minta, 266
    üres, 266
    elhanyagolt, 273
    gyakori, 266
    gyakorisága, 277
    jelölt, 273
    mérete, 266
    nem bővíthető, 269
    ritka, 266
    támogatottsága, 266
    zárt, 269
mintahalmaz, 266
mintatér, 266
modellek kombinációja, 163
mohó bővítő eljárás, 115
monotonizáció, 104
multiclass osztályozás, 182
multidimensional scaling, 94
multilabel osztályozás, 182
multinomial logistic regression, 133
multiresponse logistic regression, 133
multivariate time-series, 353

normális eloszlás, 49

one-versus-all, 132
outlier, 364

paraméterkeresés, 189
partíciós algoritmus, 285
PATRICIA fa, 61
precision, 174
predikátum
    anti-monoton, 270
    monoton, 270
    prefix anti-monoton, 270
    prefix monoton, 270
    triviális, 270
prefix, 267

részben rendezés, 43
részben strukturált adatok, 188
részfa felhúzás, 155
részfa helyettesítés, 155
részgráf, 58
részminta, 266
    valódi, 266
rétegzett particionálás, 171
Receiver Operator Curve, 175
regressziós fa, 156
relatív abszolút hiba, 177
relatív négyzetes hiba, 177
relatív négyzetes hibagyök, 177
replicated subtree problem, 145
reprezentatív minta, 88
resubstitution error, 168
ROC, 175
Rocchio-eljárás, 125

sűrűségfüggvény, 45
semi-structured data, 188
semi-supervised learning, 180
semi-supervised osztályozás, 180
single linkage eljárás, 345
sorozat, 44

sorted neighborhood, 102
strukturálatlan adatok, 188
strukturált adatok, 188
SVM, 162
szófa, 59, 213
    láncolt listás implementáció, 60
    nyesett, 60
    táblázatos implementáció, 60
szórás, 47
szűkebb értelemben vett adatbányászat, 20
szerkesztési távolság, 73
szigmoid, 130
szinguláris felbontás, 90
szomszédossági lista, 58
szomszédossági mátrix, 58
szupport vektor gépek, 162

többcímkés osztályozás, 182
többosztályos osztályozás, 182
többségi szavazás, 164
többválaszú logisztikus regresszió, 133
többváltozós idősor, 353
támogatottsági függvény, 266
támogatottsági küszöb, 205, 266
task-based evaluation, 337
taxonómia, 259
Taylor-Silverman elvárások, 149
teljes rendezés, 43
TID-halmaz, 227
tranzakció, 204
triviális osztályozó, 141

univariate time-series, 353
user-based collaborative filtering, 183
utómetszés, 155

vágás (gráfban), 58
várható érték, 46
valószínűségi változó, 45
valószínűségi változó szórása, 47
valószínűségi változó várható értéke, 46
variáns távolság, 69
visszahelyettesítéses hiba, 168

Ward módszer, 347
weka
    `Associate` fül, 380
    `Classify` fül, 374
    Arff formátum, 368
    sparse arff formátum, 369
weka.associations
    Apriori, 380
    Leverage, 381
weka.attributeSelection
    LatentSemanticAnalysis, 374
weka.classifiers
    bayes.NaiveBayes, 380
    bayes.NaiveBayesSimple, 380
    Classifier evaluation options, 375, 376
    Classifier output, 375
    functions.LeastMedSq, 377
    functions.LinearRegression, 377
    functions.Logistic, 377
    functions.MultilayerPerceptron, 377
    functions.SimpleLinearRegression, 376
    functions.Winnow, 376
    lazy.IB1, 376
    lazy.IBk, 376
    Result list panel, 379
    rules.OneR, 379
    rules.Prism, 379
    rules.ZeroR, 379
    Test options panel, 374, 375
    trees csomag, 379
    trees.UserClassifier, 379
weka.clusterers
    DBScan, 381
    SimpleKMeans, 381
weka.filters.supervised
    instance.Resample, 374

instance.SpreadSubsample, 374
instance.StratifiedRemoveFolds, 373
weka.filters.unsupervised
attribute.Add, 371
attribute.AddExpression, 371
attribute.AddID, 371
attribute.AddNoise, 373
attribute.Center, 373
attribute.ChangeDateFormat, 370
attribute.Copy, 371
attribute.Discretize, 373
attribute.FirstOrder, 371
attribute.InterquartileRange, 372
attribute.MathExpression, 371
attribute.MergeTwoValues, 370
attribute.NominalToBinary, 371
attribute.Normalize, 373
attribute.NumericCleaner, 372
attribute.NumericToNominal, 371
attribute.NumericTransform, 372
attribute.Obfuscate, 373
attribute.PKIDiscretize, 373
attribute.PrincipalComponents, 374
attribute.Remove, 372
attribute.RemoveType, 372
attribute.RemoveUseless, 372
attribute.ReplaceMissingValues, 371
attribute.Standardize, 373
instance.RemoveFolds, 373
instance.RemoveMisclassified, 372
instance.RemovePercentage, 373
instance.RemoveWithValues, 372
instance.Resample, 373
instance.ReservoirSample, 374

zárt elemhalmaz, 238

Irodalomjegyzék

[Abonyi, 2006] Abonyi János és tsa. (2006): *Adatbányászat, a hatékonyság eszköze*, ComputerBooks Kiadói Kft.

[Ackerman és Ben-David, 2009] Margareta Ackerman, Shai Ben-David (2009): *Clusterability: A Theoretical Study* Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS), Clearwater Beach, Florida, USA.

[Aloise és tsa., 2009] Daniel Aloise, Amit Deshpande, Pierre Hansen, Preyas Popat (2009): *NP-hardness of Euclidean sum-of-squares clustering*, Machine Learning, Volume 75, Number 2, pp. 245–248

[Agrawal és Srikant, 1994] Rakesh Agrawal and Ramakrishnan Srikant (1994): *Fast Algorithms for Mining Association Rules*, Proceedings of the 20th International Conference Very Large Data Bases (VLDB), Morgan Kaufmann, pp. 487–499

[Agrawal és Srikant, 1995] Rakesh Agrawal, Ramakrishnan Srikant (1995): *Mining Sequential Patterns*, Proceedings of the 11th International Conference on Data Engineering (ICDE), IEEE Computer Society, pp. 3–14

[Agrawal és tsa., 1993] Rakesh Agrawal, Tomasz Imielinski, Arun N. Swami (1993): *Mining Association Rules between Sets of Items in Large Databases*, Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, D.C., pp. 207–216

[Agrawal és tsa., 1996] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, A. Inkeri Verkamo (1996): *Fast Discovery of Association Rules*, Advances in Knowledge Discovery and Data Mining, pp. 307–328

[Agrawal és tsa., 2001] Ramesh C. Agarwal, Charu C. Aggarwal, V. V. V. Prasad (2001): *A Tree Projection Algorithm for Generation of Frequent Item*

Sets, Journal of Parallel and Distributed Computing, Volume 61, Number 3, pp. 350–371.

[Alon and Spencer, 2000] Noga Alon, Joel H. Spencer (2000): *The Probabilistic Method*, second edition, John Wiley & Sons, Inc.

[Amir és tsa., 1997] Amihood Amir, Ronen Feldman, Reuven Kashi (1997): *A New and Versatile Method for Association Generation*, Principles of Data Mining and Knowledge Discovery, pp. 221–231.

[Aurenhammer, 1991] Franz Aurenhammer (1991): *Voronoi diagrams — a survey of a fundamental geometric data structure*, ACM Computing Surveys, Volume 23, Number 3, pp. 345–405, ACM, New York, NY, USA

[Ayan, 1999] Necip Fazil Ayan, Abdullah Uz Tansel, M. Erol Arkun (1999): *An Efficient Algorithm to Update Large Itemsets with Early Pruning*, Knowledge Discovery and Data Mining, pp. 287–291

[Bastide és tsa., 2000] , Yves Bastide, Rafik Taouil, Nicolas Pasquier, Gerd Stumme, Lotfi Lakhal (2000): *Mining frequent patterns with counting inference*, SIGKDD Explor. Newsl., Volume 2, Number 2, pp. 66–75, ACM Press, New York, NY, USA

[Ben-David, 2006] Shai Ben-David, Ulrike von Luxburg, Dávid Pál (2006): *A Sober Look at Clustering Stability* Learning Theory, Lecture Notes in Computer Science, 2006, Volume 4005/2006, Springer, pp. 5-19

[Bentley, 1975] Jon Louis Bentley (1975): *Multidimensional binary search trees used for associative searching*, Commun. ACM, Volume 18, Number 9, pp. 509–517, ACM, New York, NY, USA

[Beygelzimer és tsa., 2006] Alina Beygelzimer, Sham Kakade, John Langford (2006): *Cover trees for nearest neighbor*, ICML '06: Proceedings of the 23rd international conference on Machine learning, pp. 97–104, ACM, New York, NY, USA

[Bishop, 2006] Christopher M. Bishop (2006): *Pattern Recognition and Machine Learning*, Springer

[Blanzieri és Bryl, 2008] Enrico Blanzieri, Anton Bryl (2008): *A survey of learning-based techniques of email spam filtering*, Artificial Intelligence Review Volume 29, Number 1, pp. 63-92, Springer

[Blum és Rivest, 1988] A. Blum, R. L. Rivest (1988): *Training a 3-node neural network is NPComplete*, Extended abstract, Proceedings of the Workshop on Computational Learning Theory, pp. 9–18, San Francisco, CA. Morgan Kaufmann.

[Bodon, 2003a] Bodon Ferenc (2003): *Hash-fĂĄk ĂŠs szĂłfĂĄk az adatbĂĄnyĂĄszatban*, Alkalmazott Matematikai Lapok, pp. 1–24, Volume 21

[Bodon, 2003b] Ferenc Bodon (2003): *A fast APRIORI implementation*, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03), CEUR Workshop Proceedings, Volume 90, Melbourne, Florida, USA

[Bonchi és tsa., 2003] Francesco Bonchi, Fosca Giannotti, Alessio Mazzanti, Dino Pedreschi (2003): *ExAnte: Anticipated Data Reduction in Constrained Pattern Mining*, Knowledge Discovery in Databases: PKDD 2003, 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, Springer, Lecture Notes in Computer Science, Volume 2838, pp. 59–70

[Borg és Groenen, 2005] Ingwer Borg, Patrick J.F. Groenen (2005): *Modern Multidimensional Scaling*, Springer Series in Statistics, Springer.

[Borgelt, 2003] Christian Borgelt (2003): *Efficient Implementations of Apriori and Eclat*, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03), CEUR Workshop Proceedings, Volume 90, Melbourne, Florida, USA

[Borgelt, 2007] Christian Borgelt (2007): *Canonical Forms for Frequent Graph Mining*, Advances in Data Analysis, Studies in Classification, Data Analysis, and Knowledge Organization, Springer, pp. 337–349

[Borgelt és Kruse, 2002] Christian Borgelt, Rudolf Kruse (2002): *Induction of Association Rules: Apriori Implementation*, Proceedings of the 15th Conference on Computational Statistics (Compstat 2002, Berlin, Germany), Physika Verlag.

[Bortolan és Willems, 1993] G. Bortolan, J. Willems (1993): *Diagnostic ECG classification based on neural networks*, Journal of Electrocardiology, Vol. 26, p. 75

[Breiman és tsa., 1984] Leo Breiman, Jerome Friedman, Charles J. Stone, R. A. Olshen (1984): *Classification and Regression Trees*, Chapman & Hall/CRC

[Briandais, 1959] R. de la Briandais (1959): *File searching using variable-length keys*, Western Joint Computer Conference, pp. 295–298, San Francisco, United States

[Burdick és tsa., 2001] Douglas Burdick and Manuel Calimlim and Johannes Gehrke (2001): *MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases*, Proceedings of the 17th International Conference on Data Engineering, pp. 443–452, IEEE Computer Society, Heidelberg, Germany

[Buza, 2011a] Krisztian Buza (2011): *Fusion Methods for Time Series Classification*, Peter Lang Verlag.

[Buza, 2011b] K. Buza, A. Buza, P.B. Kis (2011): *A Distributed Genetic Algorithm for Graph-Based Clustering* Man-Machine Interactions 2, Advances in Intelligent and Soft Computing, Volume 103/2011, Springer, pp. 323–331

[Buza, 2011c] K. Buza, A. Nanopoulos, L. Schmidt-Thieme, J. Koller (2011): *Fast Classification of Electrocardiograph Signals via Instance Selection*, First IEEE Conference on Healthcare Informatics, Imaging, and Systems Biology (HISB)

[Buza, 2011d] K. Buza, A. Nanopoulos, L. Schmidt-Thieme (2011): *INSIGHT: Efficient and Effective Instance Selection for Time-Series Classification*, 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Lecture Notes in Computer Science/Lecture Notes in Artificial Intelligence (LNCS/LNAI), Volume 6635, pages 149–160, Springer

[Buza és Galambos, 2013] K. Buza, I. Galambos (2013): *An Application of Link Prediction in Bipartite Graphs: Personalized Blog Feedback Prediction*, Japanese-Hungarian Symposium on Discrete Mathematics and Its Applications

[B. Kis és Buza, 2013] B. Kis Piroska, Buza Antal (2013): *Bevezetés az adatbányászat egyes fejezeteibe*

[Cendrowska, 1987] Jadzia Cendrowska (1987): *PRISM: An Algorithm for Inducing Modular Rules*, International Journal of Man-Machine Studies, Volume 27, Number 4, pp. 349–370

[Cismondi és tsa., 2011] F. Cismondi, A.S. Fialho, S.M. Vieira, J.M.C. Sousa, S.R. Reti, M.D. Howell, S.N. Finkelstein (2011): *Computational Intelligence Methods for Processing Misaligned, Unevenly Sampled Time Series Containing Missing Data*, Symposium Series on Computational Intelligence, Symposium on Computational Intelligence and Data Mining (CIDM), IEEE

[Chan és Fu, 1999] K.P. Chan és W.C. Fu (1999): *Efficient Time Series Matching by Wavelets*, Proceedings of the 15th International Conference on Data Engineering (ICDE), pp. 126–133, IEEE Computer Society

[Chandola és tsa., 2009] Varun Chandola, Arindam Banerjee, Vipin Kumar (2009): *Anomaly Detectiom: A Survey*, ACM Computing Surveys

[Chapman és tsa., 1999] Pete Chapman, Julian Clinton, Randy Kerber, Thomas Khabaza, Thomas Reinartz, Colin Shearer, Rüdiger Wirth (1999): *Cross Industry Standard Process for Data Mining (CRISP-DM) – Step by Step Data Mining Guide*, technical report, ftp://ftp.software.ibm.com/software/analytics/spss/support/Modeler/Documentation/14/UserManual/CRISP-DM.pdf

[Cheung és tsa., 1996] David Wai-Lok Cheung, Jiawei Han, Vincent Ng, C. Y. Wong (1996): *Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique*, ICDE, pp. 106–114

[Cheung és tsa., 1997] David Wai-Lok Cheung, Sau Dan Lee, Ben Kao (1997): *A General Incremental Technique for Maintaining Discovered Association Rules*, Database Systems for Advanced Applications, pp. 185–194

[Christen, 2008] Peter Christen (2008): *Automatic record linkage using seeded nearest neighbour and support vector machine classification*, Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '08), pp. 151-159

[Cormack, 2007] Gordon V. Cormack (2007): *Email Spam Filtering: A Systematic Review*, Foundations and Trends in Information Retrieval, Volume 1, Number 4, pp. 335-455

[Cover, 1991] Thomas M. Cover and Joy A. Thomas (1991): *Elements of Information Theory*, Wiley Series in Telecommunications

[Datar és tsa., 2004] M. Datar, N. Immorlica, P. Indyk, V. S. Mirrokni (2004): *Locality-sensitive hashing scheme based on p-stable distributions*, SCG

â04: Proceedings of the Twentieth Annual Symposium on Computational Geometry, New York, NY, USA, pp. 253–262

[De Mántaras, 1991] R. López De Mántaras (1991): *A Distance-Based Attribute Selection Measure for Decision Tree Induction*, Mach. Learn., Volume 6, Number 1, pp. 81–92, Kluwer Academic Publishers, Hingham, MA, USA

[Devroye és tsa., 1996] L. Devroye, L. Györfi, G. Lugosi (1996): *A Probabilistic Theory of Pattern Recognition*, Springer-Verlag, New York.

[Dietterich és tsa., 1996] T. G. Dietterich, M. Kearns, Y. Mansour (1996): *Applying the Weak Learning Framework to Understand and Improve C4.5*, Proceedings of the 13th International Conference on Machine Learning (ICML'96), Morgan Kaufmann, pp. 96–104, San Francisco, CA, USA

[Dietterich, 2000] Thomas G. Dietterich (2000): *Ensemble Methods in Machine Learning*, Multiple Classifier Systems, Lecture Notes in Computer Science, Volume 1857/2000, Springer, pp. 1-15

[Ding és tsa., 2008] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, E. Keogh (2008): *Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures*, Proceedings of the VLDB Endowment, Volume 1, Number 2, 1542–1552

[Drumond, 2012] Lucas Drumond, Steffen Rendle, Lars Schmidt-Thieme (2012): *Predicting RDF Triples in Incomplete Knowledge Bases with Tensor Factorization*, Proceedings of the 27th ACM International Symposium on Applied Computing, Riva del Garda, Italy.

[Dunham, 2002] Margaret H. Dunham (2002): *Data Mining: Introductory and Advanced Topics*, Prentice Hall PTR, Upper Saddle River, NJ, USA

[Dunteman, 1989] George H. Dunteman (1989): *Principal Components Analysis*, Newbury Park, California.

[Edelstein, 1999] Herb Edelstein (1999): *Mining Large Databases – A Case Study*, Two Crows Corporation

[Ester és tsa., 1995] M. Ester, H.-P. Kriegel, X. Xu (1995): *A Database Interface for Clustering in Large Spatial Databases*, Proceedings of the Knowledge Discovery and Data Mining Conference, Montreal, Canada pp. 94–99

[Ester és tsa., 1996] Martin Ester, Hans-Peter Kriegel, Jorg Sander, Xiaowei Xu (1996): *A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise*, Second International Conference on Knowledge Discovery and Data Mining, AAAI Press, pp. 226–231

[Farhangfar és tsa., 2008] Alireza Farhangfar, Lukasz Kurgan, Jennifer Dy (2008): *Impact of imputation of missing values on classification error for discrete data* Pattern Recognition, Volume 41, pp. 3692–3705

[Fayyad, 1996] Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth (1996): *From Data Mining to Knowledge Discovery: An Overview*, Advances in Knowledge Discovery and Data Mining, AAAI Press/The MIT Press, pp. 1-34

[Fazekas, 2000] Fazekas István (2000): *Bevezetés a matematikai statisztikába*, Debreceni Egyetem Kossuth Egyetemi Kiadója

[Feller, 1978] William Feller (1978): *Bevezetés a Valószínűségszámításba és Alkalmazásaiba*, Műszaki Könyvkiadó

[Fleiner, 2011] Fleiner Tamás (2011): *Bevezetés a számításelméletbe 2.*, http://www.cs.bme.hu/~fleiner/jegyzet/BSz2.1.5.pdf

[Fleiner, 2012] Fleiner Tamás (2012): *Bevezetés a számításelméletbe 1.*, http://www.cs.bme.hu/~fleiner/jegyzet/BSz1.1.11.pdf

[Forgy, 1965] E. W. Forgy (1965): *Cluster Analysis of Multivariate Data: Efficiency Versus Interpretability of Classifications*, Biometric Soc. Meetings, Riverside, California, Volume 21, p. 768

[Fortin és Liu, 1996] Scott Fortin, Ling Liu (1996): *An Object-Oriented Approach to Multi-Level Association Rule Mining*, CIKM, pp. 65–72

[Fu, 1996] Y. Fu (1996): *Discovery of multiple-level rules from large databases*, Ph. D. thesis, Simon Fraser University

[Futó, 1999] Futó Iván (1999): *Mesterséges Intelligencia*, Aula Kiadó, Budapest

[Fredkin, 1960] Edward Fredkin (1960): *Trie memory*, Communications of the ACM, Volume 3, Number 9, pp. 490–499, ACM Press

[Freund és Schapire, 1994] Y. Freund, R. Schapire (1994): *A decision-theoretic generalization of on-line learning and an application to boosting*, Proceedings EuroCOLTâ94: European Conference on Computational Learning Theory. LNCS, Springer

[Garcia-Molina és tsa., 2008] H. Garcia-Molina, J. Ullman, J. Widom (2008): *Database Systems: the Complete Book*, 2nd Edition, Prentice Hall

[Gelfand és tsa., 1991] S.B. Gelfand, C.S. Ravishankar, E.J. Delp (1991): *An Iterative Growing and Pruning Algorithm for Classification Tree Design*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Volume 13, Number 2, pp. 163–174, IEEE Computer Society, Los Alamitos, CA, USA

[Geurts, 2001] P. Geurts (2001): *Pattern Extraction for Time Series Classification*, Principles of Data Mining and Knowledge Discovery, pp. 115–127, Springer.

[Goethals, 2002] Bart Goethals (2002): *Survey on Frequent Pattern Mining*, Manuskript

[Goethals és Zaki, 2003] Bart Goethals, Mohammed J. Zaki (2003): *Advances in Frequent Itemset Mining Implementations: Introduction to FIMI03*, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03), CEUR Workshop Proceedings, Volume 90, Melbourne, Florida, USA

[Grahne és Zhu, 2003] Gosta Grahne, Jianfei Zhu (2003): *Efficiently Using Prefix-trees in Mining Frequent Itemsets*, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03), CEUR Workshop Proceedings, Volume 90, Melbourne, Florida, USA

[Guha és tsa., 1998] Sudipto Guha, Rajeev Rastogi, Kyuseok Shim (1998): *CURE: an efficient clustering algorithm for large databases*, ACM SIGMOD International Conference on Management of Data, pp. 73–84

[Hagerup és Rüb, 1990] Torben Hagerup, C. Rüb (1990): *A guided tour of Chernoff bounds*, Inf. Process. Lett., Volume 33, Number 6, pp. 305–308, Elsevier North-Holland, Inc.

[Han és Fu, 1995] J. Han and Y. Fu (1995): *Discovery of multiple-level association rules from large databases*, Proceedings of the 21st International Conference on Very Large Databases (VLDB), Zurich, Switzerland

[Han és Kamber, 2006] Jiawei Han, Micheline Kamber (2006): *Data mining: concepts and techniques*, second edition, Morgan Kaufmann Publisher

[Han és tsa., 2000] Jiawei Han and Jian Pei and Yiwen Yin (2000): *Mining frequent patterns without candidate generation*, ACM SIGMOD International Conference on Management of Data, ACM Press, pp. 1–12

[Hastie és tsa., 2001] Trevor Hastie, Robert Tibshirani, Jerome Friedman (2001): *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, Springer-Verlag

[Hatonen, 1996] K. Hatonen, Mika Klemettinen, Heikki Mannila, P. Ronkainen, Hannu Toivonen (1996): *Knowledge discovery from telecommunication network alarm databases*, Proceedings of the twelfth International Conference on Data Engineering, New Orleans, Louisiana, IEEE Computer Society Press, MD, USA pp. 115–122

[Hennessy és Patterson, 2011] John L. Hennessy, David A. Patterson (2011) : Computer Architecture: A Quantitative Approach, Fifth Edition, The Morgan Kaufmann Series in Computer Architecture and Design

[Hochbaum és Shmoys, 1985] D.S. Hochbaum, D.B. Shmoys (1985): *A best possible heuristic for the k-center problem*, Mathematics of Operational Research, Volume 10, Number 2, pp. 180–184

[Holte, 1993] Robert C. Holte (1993): *Very Simple Classification Rules Perform Well on Most Commonly Used Datasets*, Machine Learning, Volume 11, Number 1, pp. 63–90, Kluwer Academic Publishers, Hingham, MA, USA

[Hornik és tsa., 1990] K. Hornik, M. Stinchcombe, H. White (1990): *Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks*, Neural Networks, Volume 3, pp. 551â560

[Horváth, 2006] Tomáš Horváth, Peter Vojtás (2006): *Ordinal Classification with Monotonicity Constraints*, Advances in Data Mining. Applications in Medicine, Web Mining, Marketing, Image and Signal Mining, Lecture Notes in Computer Science, Volume 4065/2006, Springer, pp. 217–225

[Horváth és tsa., 2011] T. Horváth, A. Eckhardt, K. Buza, P. Vojtás, L. Schmidt-Thieme (2011): *Value-transformation for Monotone Prediction by Approximating Fuzzy Membership Functions*, 12th IEEE International Symposium on Computational Intelligence and Informatics

[Houtsma és Swami, 1993] Maurice Houtsma, Arun Swami (1993): *Set-Oriented Mining of Association Rules*, Research Report RJ 9567, IBM Almaden Research Center, San Jose, California

[Hull, 1994] D. A. Hull (1994): *Improving text retrieval for the routing problem using latent semantic indexing*, Proc. of SIGIR-94, 17th ACM Int. Conf. on Research and Development in Information Retrieval, pp. 282–289

[Hyafil és Rivest, 1976] L. Hyafil, R. L. Rivest (1976): *Constructing optimal binary decision trees is NP-Complete*, Information Processing Letters, Volume 5, Number 1, pp. 15–17

[Jalba és tsa., 2005] Andrei C. Jalba, Michael H.F. Wilkinson, Jos B.T.M. Roerdink, Micha M. Bayer, Stephen Juggins (2005): *Automatic diatom identi*

cation using con- tour analysis by morphological curvature scale spaces. Machine Vision and Applications, Volume 16, pp. 217–228

[Jancey, 1966] R. C. Jancey (1965): *Multidimensional Group Analysis*, Austral. J. Botany, Volume 14, pp. 127–130

[Jäschke és tsa., 2008] Robert Jäschke, Leandro Marinho, Andreas Hotho, Lars Schmidt-Thieme, Gerd Stumme (2008): *Tag recommendations in social bookmarking systems*, AI Communications, Volume 21, Number 4, pp. 231–247

[Joenssen és Bankhofer, 2012] Dieter William Joenssen, Udo Bankhofer (2012): *Hot Deck Methods for Imputing Missing Data - The Effects of Limiting Donor Usage*, Machine Learning and Data Mining in Pattern Recognition, Lecture Notes in Computer Science, Volume 7376/2012, Springer, pp. 63–75

[Jolliffe, 2005] Ian Jolliffe (2005): *Principal Component Analysis*, Encyclopedia of Statistics in Behavioral Science.

[Johnson és Wichern, 2002] Richard A. Johnson, Dean W. Wichern (2002): *Applied Multivariate Statistical Analysis*, Fifth Edition, Prentice-Hall, Upper Saddle River, NJ

[Kannan és tsa., 2000] Ravi Kannan, Santosh Vempala, Adrian Vetta (2000): *On Clusterings: Good, Bad and Spectral*, Proceedings of the 41th Annual Symposium on Fundations of Computer Science

[Karimi és tsa., 2012] R. Karimi, C. Freudenthaler, A. Nanopoulos, L. Schmidt-Thieme (2012): *Exploiting the Characteristics of Matrix Factorization for Active Learning in Recommender Systems*, Doctoral Symposium of the 6th Annual ACM Conference on Recommender Systems, Dublin, Irelan, pp. 317-320.

[Kariv és Hakimi, 1979] O. Kariv and S.L.Hakimi (1979): *An algorithmic approach to network location problems, Part II: p-medians*, SIAM J. Appl. Math., Volume 37, pp.539–560

[Kaufman és Rousseeuw, 1990] L. Kaufman and P.J. Rousseeuw (1990): *Finding Groups in Data: an Introduction to Cluster Analysis*, John Wiley & Sons

[Kearns és Mansour, 1996] Michael Kearns, Yishay Mansour (1996): *On the boosting ability of top-down decision tree learning algorithms*, STOC '96: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, pp. 459–468, ACM Press, New York, NY, USA

[Keogh és tsa., 2001] E. Keogh, K. Chakrabarti, M. Pazzani, S. Mehrotra (2001): *Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases*, ACM SIGMOD Record, Volume 30, Number 2, pp. 151–162

[Keogh és Pazzani, 1998] E. Keogh, M. Pazzani (1998): *An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback*, Proceedings of the 4th International Conference of Knowledge Discovery and Data Mining, pages 239–241. AAAI Press.

[Keogh és Pazzani, 2000] E. Keogh, M. Pazzani (2000): *Scaling up Dynamic Time Warping for Datamining Applications*, 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 285–289, ACM

[Kibriya és Frank, 2007] Ashraf M. Kibriya, Eibe Frank (2007): *An Empirical Comparison of Exact Nearest Neighbour Algorithms*, Proc 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, pp. 140–151, Springer, http://www.cs.waikato.ac.nz/ eibe/pubs/KibriyaAndFrankPKDD07.pdf

[Kim és tsa., 2001] S.W. Kim, S. Park, W.W. Chu (2001): *An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases*, International Conference on Computer Communications and Networks (ICCCN), page 0607, IEEE Computer Society

[Kleinberg, 2002] Jon Kleinberg (2002): *An Impossibility Theorem for Clustering*, Advances in Neural Information Processing Systems (NIPS), Volume 15

[Klemettinen, 1999] Mika Klemettinen (1999): *A Knowledge Discovery Methodology for Telecommunication Network Alarm Databases*, PhD thesis, University of Helsinki

[Koren, 2009] Yehuda Koren, Robert Bell, Chris Volinsky (2009): *Matrix Factorization Techniques for Recommender Systems*, IEEE Computer, Volume 42, Issue 8, pp. 30–37

[Korn és tsa., 1997] F. Korn, H.V. Jagadish, C. Faloutsos (1997): *Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences*, Proceedings of the International Conference on Management of Data (SIGMOD), pp. 289–300, ACM

[Korolova és tsa., 2009] Aleksandra Korolova, Krishnaram Kenthapadi, Nina Mishra, Alexandros Ntoulas (2009): *Releasing Search Queries and Clicks Privately*, 18th International World Wide Web Conference (WWW '2009), http://www2009.org/proceedings/pdf/p171.pdf

[Kriványek és Morávek, 1986] Mirko Kriványek und Jaroslav Morávek (1986): *NP-hard problems in hierarchical-tree clustering*, Acta Informatica, Volume 23, Number 3, pp. 311–323

[Kuramochi és Karypis, 2001] Michihiro Kuramochi and George Karypis (2001): *Frequent Subgraph Discovery*, Proceedings of the 2001 IEEE International Conference on Data Mining, pp. 313–320, IEEE Computer Society

[Kurucz és tsa., 2007] M. Kurucz, A. A. Benczúr, T. Kiss, I. Nagy, A. Szabó, B. Torma (2007): *Who Rated What: a combination of SVD, correlation and frequent sequence mining*, Proc. KDD Cup and Workshop, Volume 23

[Lakatos és Nyékiné, 2009] Lakatos Attila (Szerk.), Nyékiné Gaizler Judit (Szerk.) (2009): Java 2 Útikalauz programozóknak 5.0 I-II.

[Langley és Simon, 1995] Pat Langley, Herbert A. Simon (1995): *Applications of machine learning and rule induction*, Communications of the ACM, Volume 38, Issue 11, pp. 54–64

[Lee és Stolfo, 1998] Wenke Lee, Salvatore Stolfo (1998): *Data mining approaches for intrusion detection*, Proceedings of the 7th USENIX Security Symposium. San Antonio, TX

[Lee és Stolfo, 2000] Wenke Lee, Salvatore J. Stolfo (2000): *A Framework for Constructing Features and Models for Intrusion Detection Systems*, ACM Transactions on Information and System Security, Volume 3, Number 4

[Lee és tsa., 1999] Wenke Lee, Salvatore J. Stolfo, Kui W. Mok (1999): *A Data Mining Framework for Building Intrusion Detection Models*, IEEE Symposium on Security and Privacy, pp. 120–132

[Levandowsky és Winter, 1970] Michael Levandowsky, David Winter (1970): *Distance between Sets* Nature, Volume 234, pp. 34–35

[Liao és tsa., 2008] Shu-Hsien Liao, Chyuan-Meei Chen, Chung-Hsin Wu (2008): *Mining customer knowledge for product line and brand extension in retailing* Expert Systems with Applications, Volume 34, Issue 3, pp. 1763â1776

[Lin és tsa., 2003] J. Lin, E. Keogh, S. Lonardi, B. Chiu (2003): *A Symbolic Representation of Time Series, with Implications for Streaming Algorithms*, Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, ACM

[Lindsey, 2007] J.K. Lindsey (2007): Applying Generalized Linear Models, http://www.commanster.eu/ms/glm.pdf

[Mannila és Toivonen, 1996] Heikki Mannila and Hannu Toivonen (1996): *Discovering Generalized Episodes Using Minimal Occurrences*, Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96), pp. 146–151, AAAI Press

[Mannila és tsa., 1994] Heikki Mannila, Hannu Toivonen, A. Inkeri Verkamo (1994): *Efficient algorithms for discovering association rules*, AAAI Workshop on Knowledge Discovery in Databases (KDD-94), AAAI Press, Seattle, Washington, pp. 181–192

[Mannila és tsa., 1995] Heikki Mannila, Hannu Toivonen, A. Inkeri Verkamo (1995): *Discovering frequent episodes in sequences* Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95), pp. 210–215, AAAI Press, Montreal, Canada

[Mannila és tsa., 1997] rticlemannila97discovery, Heikki Mannila, Hannu Toivonen, A. Inkeri Verkamo (1997):, *Discovery of Frequent Episodes in Event Sequences*, Data Mining and Knowledge Discovery, Volume 1, Number 3, pp. 259–289

[Marussy és Buza, 2013] K. Marussy, K. Buza (2013): *SUCCESS: A New Approach for Semi-Supervised Classification of Time-Series*, 12th International Conference on Artificial Intelligence and Soft Computing, Lecture Notes Computer Science, Springer

[McKay, 1981] Brendan D. McKay (1981): *Practical graph isomorphism*, Congressus Numerantium, Volume 30, pp. 45–87

[Megiddo és Supowitz, 1984] N. Megiddo and K.Supowitz (1984): *On the complexity of some common geometric location problems*, SIAM J. Comput., pp. 182–196

[Melgani és Bazi, 2008] F. Melgani, Y. Bazi (2008): *Classification of electrocardiogram signals with support vector machines and particle swarm optimization*, IEEE Transactions on Information Technology in Biomedicine, Vol. 12, No. 5, pp. 667â677

[Mena, 2000] Jesus Mena (2000): *Data Mining und E-Commerce*, Symposion Publishing, Düsseldorf, http://www.symposion.de/datamining

[Meyer és tsa., 2003] Ulrich Meyer, Peter Sanders, Jop F. Sibeyn (2003): *Algorithms for Memory Hierarchies, Advanced Lectures, Dagstuhl Research Seminar, March 10-14, 2002*, Springer, Lecture Notes in Computer Science, Volume 2625.

[Mörchen, 2003] F. Mörchen (2003): *Time series feature extraction for data mining using dwt and dft*, Technical report.

[Mueller, 1995] Andreas Mueller (1995): *Fast Sequential and Parallel Algorithms for Association Rule Mining: A Comparison*, Technical Report, Departure of Computer Science, University of Maryland, CS-TR-3515, College Park, MD

[Nagy és Buza, 2012] Gabor I. Nagy, Krisztian Buza (2012): *SOHAC: Efficient Storage of Tick Data That Supports Search and Analysis*, Industrial Conference on Data Mining, Lecture Notes in Computer Science / Lecture Notes in Artificial Intelligence, Volume 7377, Springer, pp. 38–51

[Neumann, 1945] John von Neumann (1945): *First Draft of a Report on the EDVAC*, Contract No. W–670–ORD–4926 Between the United States Army Ordnance Department and the University of Pennsylvania, http://qss.stanford.edu/˜godfrey/vonNeumann/vnedvac.pdf

[Nerbonne és tsa., 1999] John Nerbonne, Wilbert Heeringa, Peter Kleiweg (1999): *Edit Distance and Dialect Proximity*, Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison, CSLI, Stanford, pp. v-xv.

[Ng és Han, 1994] Raymond T. Ng, Jiawei Han (1994): *Efficient and Effective Clustering Methods for Spatial Data Mining*, Proceedings of the 20th International Conference Very Large Data Bases (VLDB), Morgan Kaufmann, pp. 144–155

[Olszewski, 2001] R. Olszewski (2001): *Generalized feature extraction for structural pattern recognition in time-series data*, Ph.D. dissertation, School of Computer Science, Carnegie Mellon University.

[Omiecinski és Savasere, 1998] Edward Omiecinski, Ashoka Savasere (1998): *Efficient Mining of Association Rules in Large Dynamic Databases* British National Conference on Databases, pp. 49-63

[Omohundro, 1989] Stephen M. Omohundro (1989): *Five Balltree Construction Algorithms*, ICSI Technical Report tr-89-063, International Computer Science Institute

[Ozden és tsa., 1998] Banu Ozden, Sridhar Ramaswamy, Abraham Silberschatz (1998): *Cyclic Association Rules*, ICDE, pp. 412-421

[Park és tsa., 1995] Jong Soo Park and Ming-Syan Chen and Philip S. Yu (1995): *An Effective Hash Based Algorithm for Mining Association Rules*, Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, pp. 175–186

[Pasquier és tsa., 1998] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal (1998): *Pruning closed itemset lattices for association rules*, Proceedings of the BDA French Conference on Advanced Databases

[Pasquier és tsa., 1999a] Nicolas Pasquier, Yves Bastide, Rafik Taouil, Lotfi Lakhal (1999): *Discovering Frequent Closed Itemsets for Association Rules*, ICDT, pp. 398–416

[Pasquier és tsa., 1999b] N. Pasquier, Y. Bastide, R. Taouil, L. Lakhal (1999): *Efficient mining of association rules using closed itemset lattices*, Journal of Information systems, pp. 25–46

[Peltola és tsa., 1984] Hannu Peltola, Hans Söderlund, Esko Ukkonen (1984): *SEQAID: a DNA sequence assembling program based on a mathematical model*, Nucleic Acids Research, Volume 12, Number 1

[Pei és tsa., 2000] Jian Pei, Jiawei Han, Runying Mao (2000): *CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets*, ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 21–30

[Pei és tsa., 2001] Jian Pei, Jiawei Han, Laks V. S. Lakshmanan (2001): *Mining Frequent Item Sets with Convertible Constraints*, ICDE, pp. 433–442

[Pijls és Bioch, 1999] Wim Pijls AND Jan C. Bioch (1999): *Mining frequent itemsets in memory-resident databases*, Proceedings of the Eleventh Belgium /Netherlands Articial Intelligence Conference (BNAIC 99), pp. 75-82

[Petridis és Kehagias, 1990] V. Petridis, A. Kehagias (1998): *Predictive modular neural networks: applications to time series*, The Springer International Series in Engineering and Computer Science, Volume 466, Springer Netherlands

[Pitelis és Tefas, 2012] Nikolaos Pitelism, Anastasios Tefas (2012): *Discriminant Subspace Learning Based on Support Vectors Machines*, Machine Learning and Data Mining in Pattern Recognition, Lecture Notes in Computer Science, Volume 7376/2012, Springer, pp. 198–212

[Porter, 1998] Jim Porter (1998): *Disk/Trend Report*, Proceedings of the 100th Anniversary Conference on Magnetic Recording and Information Storage, Santa Clara Univerity

[Quinlan, 1986] J. R. Quinlan (1986): *Induction of Decision Trees*, Machine Learning, Volume 1, Number 1, pp. 81–106, Kluwer Academic Publishers, Hingham, MA, USA

[Quinlan, 1987] J. R. Quinlan (1987): *Simplifying decision trees*, Int. J. Man-Mach. Stud., Volume 27, Number 3, pp. 221–234, Academic Press Ltd., London, UK

[Quinlan, 1993] J. Ross Quinlan (1993): *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA

[Radovanović és tsa., 2010a] Milos Radovanović, Alexandros Nanopoulos, Mirjana Ivanović (2010): *Hubs in Space: Popular Nearest Neighbors in High-Dimensional Data*, The Journal of Machine Learning Research, Volume 11, pp. 2487–2531

[Radovanović és tsa., 2010b] Milos Radovanović, Alexandros Nanopoulos, Mirjana Ivanović (2010): *Time-Series Classification in Many Intrinsic Dimensions*, Proceedings of the 10th SIAM International Conference on Data Mining (SDM), pp. 677–688

[Radovanović, 2011] Milos Radovanović (2011): *Representations and Metrics in High-Dimensional Data Mining*, Izdavacka knjizarnica Zorana Stojanovica, Sremski Karlovici, Novi Sad

[Ratanamahatana és Keogh, 2004] C.A. Ratanamahatana, E. Keogh (2004): *Everything You Know about Dynamic Time Warping is Wrong*, SIGKDD International Workshop on Mining Temporal and Sequential Data.

[Ratanamahatana és Keogh, 2005] C.A. Ratanamahatana, E. Keogh (2005): *Three Myths about Dynamic Time Warping Data Mining*, Proceedings of SIAM International Conference on Data Mining (SDM)

[Rätsch és tsa., 2001] G. RĂtsch, T. Onoda, K.-R. Müller (2001): *Soft Margins for AdaBoost* Machine Learning, Volume 42, Number 3, pp. 287-320

[Read és Cressie, 1988] T. R. C. Read, N. A. C. Cressie (1988): *Goodness-of-Fit Statistics for Discrete Multivariate Data*, Springer Series in Statistics, Springer-Verlag, New York

[Reiz és Csató] B. Reiz, L. Csató (2008): *Tree-like bayesian network classifiers for surgery survival chance prediction*, Int. J. of Computers, Communications and Control, Volume 3, pp. 470-474.

[Reuter és tsa., 2011] Timo Reuter, Philipp Cimiano, Lucas Drumond, Krisztian Buza, Lars Schmidt-Thieme (2011): *Scalable event-based clustering of social media via record linkage techniques*, Fifth International AAAI Conference on Weblogs and Social Media

[Rényi, 1968] Rényi Alfréd (1968): *Valószínűségszámítás*, Tankönyvkiadó, Budapest

[Rifkin és Klautau, 2004] Ryan Rifkin, Aldebaro Klautau (2004): *In Defense of One-Vs-All Classification*, The Journal of Machine Learning Research, Volume 5, pp. 101–141

[Roden és tsa., 2009] D.M. Roden, J.M. Pulley, M.A. Basford, G.R. Bernard, E.W. Clayton, J.R. Balser, D.R. Masys (2009): *Development of a Large-Scale De-Identified DNA Biobank to Enable Personalized Medicine*, Clinical Pharmacology & Therapeutics, Volume 84, pp. 362-369

[Romano és tsa., 2009] Lorenza Romano, Krisztian Buza, Claudio Giuliano, Lars Schmidt-Thieme (2009): *XMedia: Web People Search by Clustering with Machinely Learned Similarity Measures*, 2nd Web People Search Evaluation Workshop at World Wide Web Conference

[Rónyai és tsa.,1998] Rónyai Lajos, Ivanyos Gábor, Szabó Réka (1998): *Algoritmusok*, Typotex Kiadó

[Roiger, 2003] R. J. Roiger, M. W. Geatz: *Data mining: a tutorial-based primer*, Addison-Wesley, New York, 2003.

[Rózsa, 1991] Rózsa Pál (1991): *Lineáris algebra és alkalmazásai*, Tankönyvkiadó, Budapest

[Sahni és Gonzales, 1976] S. Sahni, T. Gonzales (1976): *P-complete Approxiamtion Problems*, JACM, Volume 23, pp. 555–566

[Sakoe és Chiba, 1978] H. Sakoe, S. Chiba (1978):*Dynamic Programming Algorithm Optimization for Spoken Word Recognition*, Acoustics, Speech and Signal Processing, Volume 26, Number 1, pp. 43–49

[Salakhutdinov és Mnih, 2008] R. Salakhutdinov, A. Mnih (2008): *Probabilistic matrix factorization*, Advances in neural information processing systems, Volume 20, pp. 1257–1264

[Sarda és Srinivas, 1998] Nandlal L. Sarda, N. V. Srinivas (1998): *An Adaptive Algorithm for Incremental Mining of Association Rules*, DEXA Workshop, 240–245

[Savasere, és tsa., 1995] Ashoka Savasere, Edward Omiecinski, Shamkant B. Navathe (1995): *An Efficient Algorithm for Mining Association Rules in Large Databases*, The VLDB Journal, pp. 432–444

[Schapire és tsa., 1998] R. E. Schapire, Y. Singer, A. Singhal (1998): *Boosting and Rocchio applied to text filtering*, Proc. of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval, pp. 215–223

[Schultz és tsa., 2001a] Matthew G. Schultz, Eleazar Eskin, Erez Zadok, Salvatore J. Stolfo (2001): *Data Mining Methods for Detection of New Malicious Executables*, IEEE Symposium on Security and Privacy, Proceedings, pp. 38–49

[Schultz és tsa., 2001b] Matthew G. Schultz and Eleazar Eskin and Salvatore J. Stolfo (2001): *MEF: Malicious Email Filter - A UNIX Mail Filter that Detects Malicious Windows Executables*, Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, pp. 245–252

[Sebastiani, 2002] , F. Sebastiani (2002): *Machine learning in automated text categorization*, ACM Computing Surveys, Volume 34, Number 1, March 2002, pp. 1–47

[Severance, 1974] Dennis G. Severance (1974): *Identifier Search Mechanisms: A Survey and Generalized Model*, ACM Comput. Surv., Volume 6, Number 3, pp. 175–194, ACM Press

[Silberschatz és Tuzhilin, 1995] Abraham Silberschatz, Alexander Tuzhilin (1995): *On Subjective Measures of Interestingness in Knowledge Discovery*, Knowledge Discovery and Data Mining, Proceedings, pp. 275-281, https://www.aaai.org/Papers/KDD/1995/KDD95-032.pdf

[Shen és Shen, 1998] Li Shen, Hong Shen (1998): *Mining Flexible Multiple-Level Association Rules in All Concept Hierarchies*, Extended Abstract, Database and Expert Systems Applications, pp. 786–795

[Shieh és Keogh, 2008] J. Shieh, E. Keogh (2008): *iSAX: Indexing and Mining Terabyte Sized Time Series*, Proceeding of the 14th International Conference on Knowledge Discovery and Data Mining (KDD), pages 623–631, ACM

[Shih, 1999] Y.-S. Shih (1999): *Families of splitting criteria for classification trees*, Statistics and Computing, Volume 9, Number 4, pp. 309–315, Kluwer Academic Publishers, Hingham, MA, USA

[Srikant és Agrawal, 1995] Ramakrishnan Srikant, Rakesh Agrawal (1995): *Mining Generalized Association Rules*, Proceedings of the 21st International Conference on Very Large Databases (VLDB), Zurich, Switzerland

[Srikant és Agrawal, 1996] Ramakrishnan Srikant, Rakesh Agrawal (1996): *Mining Sequential Patterns: Generalizations and Performance Improvements*, Proceedings of 5th International Conference Extending Database Technology (EDBT), Springer-Verlag, pp. 3–17

[Stańczyk, 2011] Urszula Stańczyk (2011): *Recognition of Author Gender for Literary Texts*, Man-Machine Interactions 2, Advances in Intelligent and Soft Computing, Volume 103, Springer-Verlag Berlin Heidelberg

[Syed és Chia, 2011] Z. Syed, C.-C. Chia (2011): *Computationally generated cardiac biomarkers: Heart rate patterns to predict death following coronary attacks*, SIAM International Conference on Data Mining.

[Symeonidis és tsa., 2010] P. Symeonidis, A. Nanopoulos, Y. Manolopoulos (2010): *A unified framework for providing recommendations in social tagging systems based on ternary semantic analysis*, IEEE Transactions on Knowledge and Data Engineering Volume 22, pp. 179–192

[Takács, 2008] Gábor Takács, István Pilászy, Bottyán Nemeth, Domonkos Tikk (2008): *Investigation of Various Matrix Factorization Methods for Large Recommender Systems*, IEEE International Conference on Data Mining Workshops (ICDMW '08)

[Tan és tsa., 2005] Pang-Ning Tan, Michael Steinbach, Vipin Kumar (2006): *Introduction to Data Mining*, Addison-Wesley

[Tenenbaum és tsa., 2000] Joshua B. Tenenbaum, Vin de Silva, John C. Langford (2000): *A Global Geometric Framework for Nonlinear Dimensionality Reduction*, Science, Volume 290

[Tikk, 2007] Tikk Domonkos (2007): *Szövegbányászat*, TypoTEX, Budapest

[Thomas, 2000] Lyn C. Thomas (2000): *A Survey of Credit and Behavioural Scoring; Forecasting financial risk of lending to consumers*, International Journal of Forecasting 16, pp. 149–172

[Thomas és Sarawagi, 1998] , Shiby Thomas, Sunita Sarawagi (1998): *Mining Generalized Association Rules and Sequential Patterns Using SQL Queries*, Knowledge Discovery and Data Mining, pp. 344–348

[Thomas és tsa., 1997] Shiby Thomas, Sreenath Bodagala, Khaled Alsabti, Sanjay Ranka (1997): *An Efficient Algorithm for the Incremental Updation of Association Rules in Large Databases*, Knowledge Discovery and Data Mining, pp. 263–266

[Toivonen, 1996] Hannu Toivonen (1996): *Sampling Large Databases for Association Rules*, The VLDB Journal, pp. 134–145

[Tomasev és tsa., 2011] N. Tomasev, M. Radovanovic, D. Mladenic, M. Ivanovic (2011): *The Role of Hubness in Clustering High-Dimensional Data*, PAKDD 2011, Part I. LNCS (LNAI), Volume 6634, Springer, pp. 183â195.

[Ullmann, 1976] J. R. Ullmann (1976): *An Algorithm for Subgraph Isomorphism*, J. ACM, Volume 23, Number 1, pp. 31–42

[Uhlmann, 1991] Jeffrey K. Uhlmann (1991): *Satisfying General Proximity/Similarity Queries with Metric Trees*, Inf. Process. Lett., Volume 40, Number4, pp. 175–179

[Wang és tsa., 2008] X. Wang, L. Ye, E. Keogh, C. Shelton (2008): *Annotating Historical Archives of Images*, Proceedings of the 8th ACM/IEEE-CS Joint Conference on Digital Libraries, ACM, pp. 341–350

[Xing és tsa., 2011] Z. Xing, J. Pei, P.S. Yu, K. Wang (2011): *Extracting Interpretable Features For Early Classification of Time Series*, SIAM International Conference on Data Mining

[Yang, 2001] Ying Yang, Geoffrey I. Webb (2001): *Proportional k-Interval Discretization for Naive-Bayes Classifiers*, EMCL '01: Proceedings of the 12th European Conference on Machine Learning, pp. 564–575, Springer-Verlag

[Yankov és tsa. 2007] D. Yankov, E. Keogh, J. Medina, B. Chiu, V. Zordan (2007): *Detecting Time Series Motifs under Uniform Scaling*, Proceedings of the 13th International Conference on Knowledge Discovery and Data Mining (KDD), ACM, pp. 844–853

[Yao, 1975] S. B. Yao (1975): *Tree structures construction using key densities*, Proceedings of the 1975 annual conference, pp. 337–342, ACM Press

[Yi és Faloutsos, 2000] B.K. Yi, C. Faloutsos (2000): *Fast Time Sequence Indexing for Arbitrary Lp Norms*, Proceedings of the 26th International Conference on Very Large Data Bases, pp. 385–394

[Yi és tsa., 1998] B.K. Yi, H.V. Jagadish, C. Faloutsos (1998): *Efficient Retrieval of Similar Time Sequences under Time Warping*, Proceedings of the 14th International Conference on Data Engineering (ICDE), pages 201–208, IEEE

[Wang és tsa., 2003] Jianyong Wang, Jiawei Han, Jian Pei (2003): *CLOSET+: Searching for the Best Strategies for Mining Frequent Closed Itemsets*, Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03), Washington, DC, USA

[Way és tsa., 2012] Michael J. Way, Jeffrey D. Scargle, Kamal M. Ali, Ashok N. Srivastava (2012): *Advances in Machine Learning and Data Mining for Astronomy*, CRC Press, Taylor & Francis Group

[Wiener és tsa., 1995] E. D. Wiener, J. O. Pedersen, A. S. Weigend (1995): *A neural network approach to topic spotting*, Proc. of the SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval, pp. 317–332

[Witten és tsa., 2011] Ian H. Witten, Eibe Frank, Mark A. Hall (2011): *Data Mining: Practical Machine Learning Tools and Techniques*, third edition, Morgan Kaufmann

[Wu és tsa., 2000] Y.L. Wu, D. Agrawal, A. El Abbadi (2000): *A Comparison of DFT and DWT Based Similarity Search in Time-Series Databases*, Proceedings of the 9th International Conference on Information and Knowledge Management, pp. 488–495, ACM

[Zaki, 2000] Mohammed Javeed Zaki (2000): *Sequence Mining in Categorical Domains: Incorporating Constraints*, CIKM, pp. 422–429

[Zaki, 2001] Mohammed J. Zaki (2001): *Efficiently Mining Frequent Trees in a Forest*, Technical Report, Computer Science Department, Rensselaer Polytechnic Institute, July 2001, Troy, NY, 12180

[Zaki, 2002] Mohammed J. Zaki (2002): *Efficiently mining frequent trees in a forest*, Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 71–80, Edmonton, Alberta, Canada, ACM Press

[Zaki és Gouda, 2003] Mohammed J. Zaki and Karam Gouda (2003): *Fast vertical mining using diffsets*, Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 326–335, Washington, D.C., ACM Press

[Zaki és Ogihara, 1998] Mohammed Javeed Zaki, Mitsunori Ogihara (1998): *Theoretical Foundations of Association Rules*, Proceedings of third SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'98), Seattle, Washington

[Zaki és Hsiao, 2002] Mohammed Javeed Zaki and Ching-Jui Hsiao (2002): *CHARM: An Efficient Algorithm for Closed Itemset Mining*, Proceedings of 2nd SIAM International Conference on Data Mining, Arlington, VA, USA

[Zaki és tsa., 1997] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, Wei Li (1997): *New Algorithms for Fast Discovery of Association Rules*, Proceedings of the third International Conference on Knowledge Discovery and Data Mining, AAAI Press, pp. 283–296