Adatbányászati algoritmusok

Bodon Ferenc

2005. szeptember 29.

Copyright © 2002-2005 Bodon Ferenc

Ezen dokumentum a Free Software Foundation által kiadott *GNU Free Documentation license* 1.2-es, vagy bármely azt követő verziójának feltételei alapján másolható, terjeszthető és/vagy módosítható. Nincs *Nem Változtatható Szakasz*, nincs *Címlap-szöveg*, nincs *Hátlap-szöveg*. A licenc magyar nyelű fordítása a `http://hu.wikipedia.org/wiki/A_GNU_Szabad_Dokumentációs_Licenc_szövege` oldalon található.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 (`http://www.gnu.org/copyleft/fdl.html`) or any later version published by the Free Software Foundation; with no *Invariant Sections*, no *Front-Cover Texts*, and no *Back-Cover Texts*. A copy of the license is included in the section entitled "GNU Free Documentation License".

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani **Rónyai Lajosnak**, a Budapesti Műszaki és Gazdaságtudományi Egyetem tanárának az egész munka során nyújtott segítségéért, hasznos ötleteiért, útmutatásaiért, de legfőképpen azért, mert megismertetett az adatbányászattal. Köszönöm **Molnár-Sáska Gábornak** és **Pintér Mártának**, az MTA-SZTAKI dolgozóinak valószínűségszámítással kapcsolatos tanácsaikat.

Külön köszönet illeti **Czibula Veronikát** a tanulmány többszöri, alapos átnézéséért és a felfedezett hibák kijavításáért. **Marx Dániel** rengeteg információval látott el a LaTeX, emacs, Xfig hatékony használatát illetően. Köszönöm neki a fáradozásait.

Friedl Katának, ifjabb Benczúr Andrásnak, Lukács Andrásnak, Maricza Istvánnak és Bereczki Tamásnak köszönöm az értékes észrevételeit, megjegyzéseit.

Értékes észrevételeik és konstruktív javaslataiért köszönet illeti a BME diákjait, többek között (névsorrendben) Hajnacs Zoltánt és Schlotter Ildikót.

Tartalomjegyzék

Előszó . . . 8

1. Bevezetés **10**
1.1. A tudásfeltárás folyamata . . . 11
1.2. Szabványok . . . 13
1.3. Adatbányászati rendszer architektúrája . . . 13
1.4. Legjelentősebb adatbányászati feladatok . . . 14
1.5. Sikeres alkalmazások . . . 16
1.6. Az adatbányászat feltételei . . . 17

2. Alapfogalmak, jelölések **19**
2.1. Halmazok, relációk, függvények, sorozatok . . . 19
2.2. Lineáris algebra . . . 20
2.3. Gráfelmélet . . . 21
2.4. Valószínűségszámítás . . . 21
2.4.1. Hoeffding-korlát . . . 22
2.4.2. Entrópia . . . 22
2.5. Statisztika . . . 23
2.5.1. Hipotézisvizsgálat . . . 23
2.5.2. Az F-próba . . . 23
2.5.3. A χ^2-próba . . . 24
2.5.4. Függetlenségvizsgálat . . . 24
2.6. Algoritmus-elmélet . . . 25
2.7. Adatstruktúrák . . . 25
2.7.1. Szófák . . . 25
2.7.2. Piros-fekete fák . . . 28
2.7.3. Hash-tábla . . . 28

3. Előfeldolgozás, hasonlósági függvények **29**
3.1. Előfeldolgozás . . . 29
3.1.1. Hiányzó értékek kezelése . . . 29
3.1.2. Attribútum transzformációk . . . 30
3.1.3. Mintavételezés . . . 30
3.2. Hasonlósági mértékek . . . 31

3.2.1. Bináris attribútum . . . 32
3.2.2. Kategória típusú attribútum . . . 32
3.2.3. Sorrend típusú attribútum . . . 33
3.2.4. Intervallum típusú attribútum . . . 33
3.2.5. Vegyes attribútumok . . . 34
3.2.6. Speciális esetek . . . 34
3.2.7. Dimenziócsökkentés . . . 35
3.2.8. Szinguláris felbontás (Fogaras Dániel) . . . 35

4. Gyakori minták kinyerése 41
4.1. A gyakori minta definíciója . . . 42
4.1.1. Hatékonysági kérdések . . . 43
4.2. További feladatok . . . 44
4.2.1. Nem bővíthető és zárt minták . . . 44
4.2.2. Kényszerek kezelése . . . 45
4.2.3. Többszörös támogatottsági küszöb . . . 46
4.2.4. Dinamikus gyakori mintakinyerés . . . 46
4.3. Az algoritmusok jellemzői . . . 47
4.4. Az APRIORI módszer . . . 47
4.4.1. Jelöltek előállítása . . . 48
4.4.2. Zárt minták kinyerése, az APRIORI-CLOSE algoritmus . . . 50
4.5. Sorozat típusú bemenet . . . 50
4.5.1. APRIORI . . . 51
4.5.2. Zaki módszere . . . 52
4.5.3. Mintanövelő algoritmusok . . . 54
4.5.4. Kétlépcsős technikák . . . 55
4.5.5. A zárt minták „törékenysége” . . . 59
4.5.6. Dinamikus gyakori mintabányászat . . . 59

5. Gyakori elemhalmazok 62
5.1. A gyakori elemhalmaz fogalma . . . 62
5.2. Az APRIORI algoritmus . . . 65
5.2.1. Jelöltek előállítása . . . 65
5.2.2. Jelöltek támogatottságának meghatározása . . . 65
5.2.3. A gyakori elemhalmazok tárolása . . . 68
5.2.4. A bemenet tárolása . . . 69
5.2.5. Utolsó fázisok gyorsítása: APRIORI-TID és APRIORI-HYBRID algoritmusok 69
5.2.6. Futási idő és memóriaigény . . . 70
5.2.7. Kételemű jelöltek számának csökkentése: a DHP algoritmus . . . 73
5.3. Az ECLAT algoritmus . . . 75
5.4. Az FP-growth algoritmus . . . 76
5.4.1. Az FP-growth* algoritmus . . . 77
5.5. További híres algoritmusok . . . 78
5.5.1. A $\mathcal{DF}$-APRIORI algoritmus . . . 78
5.5.2. patricia . . . 79
5.5.3. kdci . . . 79

5.5.4. lcm . . . 79
5.5.5. Mintavételező algoritmus elemzése . . . 79
5.6. Elemhalmazok Galois lezárja . . . 80
5.6.1. A zárt elemhalmazok fogalma . . . 80
5.7. Kényszerek kezelése . . . 82
5.7.1. ExAnte . . . 82
5.8. Többszörös támogatottsági küszöb . . . 83
5.8.1. MSApriori algoritmus . . . 83

6. Gyakori sorozatok, bool formulák és epizódok 85
6.1. Gyakori sorozatok kinyerése . . . 85
6.1.1. A Gyakori Sorozat Fogalma . . . 86
6.1.2. APRIORI . . . 86
6.1.3. Elemhalmazokat tartalmazó gyakori sorozatok . . . 87
6.1.4. Sorozat típusú minta általánosítása . . . 91
6.2. Gyakori bool formulák . . . 91
6.3. Gyakori epizódok . . . 92
6.3.1. A támogatottság definíciója . . . 92
6.3.2. APRIORI . . . 93

7. Gyakori fák és feszített részgráfok 96
7.1. Az izomorfia problémája . . . 96
7.2. A gyakori gráf fogalma . . . 98
7.3. gyakori gyökeres fák . . . 98
7.3.1. TreeMinerH . . . 100
7.3.2. TreeMinerV . . . 101
7.4. Gyakori részfák . . . 103
7.5. A gyakori feszített részgráfok . . . 103
7.5.1. Az AcGM algoritmus . . . 103
7.6. A gyakori részgráfok keresése . . . 105
7.6.1. Az FSG algoritmus . . . 105
7.6.2. gSpan . . . 107

8. Asszociációs szabályok 110
8.1. Az asszociációs szabály fogalma . . . 110
8.2. Hierarchikus asszociációs szabályok . . . 111
8.3. Maximális következményű asszociációs szabály . . . 113
8.3.1. Egzakt asszociációs szabályok bázisa . . . 113
8.4. Az asszociációs szabályok hibái . . . 114

9. Funkcionális és közelítő függőségek 122
9.1. Funkcionális függőség . . . 123
9.2. Közelítő függőség . . . 123
9.3. TANE Algoritmus . . . 124

10. Osztályozás és előrejelzés **130**
10.1. Bevezetés . . . 130
10.2. A klasszifikáció teljesítményének méréséről . . . 131
10.3. Döntési fák . . . 132
10.3.1. A döntési fa előállítása . . . 133
10.3.2. Az ID3 algoritmus . . . 134
10.3.3. Továbbfejlesztések . . . 134
10.3.4. Döntési fák ábrázolása . . . 135
10.3.5. Mesterséges neurális hálózatok . . . 135
10.3.6. Bayesi hálózatok . . . 137
10.3.7. Egyéb módszerek . . . 138

11. Klaszterezés **140**
11.1. Egy lehetetlenség-elmélet . . . 141
11.2. Hasonlóság mértéke, adatábrázolás . . . 143
11.3. A klaszterek jellemzői . . . 144
11.4. A klaszterezés „jósága” . . . 145
11.4.1. Klasszikus mértékek . . . 145
11.4.2. Konduktancia alapú mérték . . . 147
11.5. Klaszterező algoritmusok típusai . . . 148
11.6. Particionáló eljárások . . . 150
11.6.1. Forgy *k*-közép algoritmusa . . . 150
11.6.2. A *k*-medoid algoritmusok . . . 151
11.7. Hierarchikus eljárások . . . 152
11.7.1. Single-, Complete-, Avegare Linkage Eljárások . . . 152
11.7.2. Ward módszere . . . 153
11.7.3. A BIRCH algoritmus . . . 153
11.7.4. A CURE algoritmus . . . 154
11.7.5. A Chameleon algoritmus . . . 156
11.8. Sűrűség-alapú módszerek . . . 156
11.8.1. A DBSCAN algoritmus . . . 156

12. Szövegbányászat (Tikk Domonkos) **158**
12.1. Dokumentumok előfeldolgozása . . . 159
12.1.1. A dimenziószám csökkentése . . . 161
12.1.2. Hatékonyság mérése . . . 162
12.2. Osztályozás . . . 163
12.2.1. Osztályozás strukturálatlan kategóriák rendszerébe . . . 163
12.2.2. Hierarchikus osztályozás . . . 170
12.3. Dokumentumok csoportosítása . . . 173
12.3.1. Szövegklaszterezés jellemző feladatai és problémái . . . 173
12.3.2. Reprezentáció . . . 174
12.3.3. Hatékonyság mérése . . . 174
12.3.4. Szövegklaszterező eljárások . . . 175
12.3.5. Dokumentumgyűjtemények . . . 177
12.4. Kivonatolás . . . 177

12.4.1. Az összegzéskészítő eljárások felosztása . . . 178
12.4.2. A kivonatolás hatékonyságának mérése . . . 179
12.4.3. Mondatkiválasztásnál használt jellemzők . . . 180
12.5. A legfontosabb kivonatoló eljárások . . . 181
12.5.1. A klasszikus módszer . . . 181
12.5.2. TF-IDF alapú módszer . . . 182
12.5.3. Csoportosítás alapú módszerek . . . 182
12.5.4. Gráfelméleti megközelítések . . . 183
12.5.5. SVD használata a kivonatolásban . . . 184
12.5.6. Esettanulmány: böngészés támogatása kivonatolással kézi számítógépeken . 184
12.6. Egyéb szövegbányászati feladatok . . . 187
12.6.1. Információkinyerés . . . 187
12.6.2. Témakövetés . . . 187
12.6.3. Fogalomtársítás . . . 187
12.6.4. Szöveges információk vizualizálása . . . 188
12.6.5. Kérdés-megválaszolás . . . 188
12.7. Nyelvfeldolgozás és szövegbányászat . . . 189
12.7.1. Szövegbányászat magyarul . . . 189
12.8. Linkgyűjtemény . . . 190
12.8.1. Tesztkorpuszok . . . 190
12.8.2. Cikk- és linkgyűjtemények . . . 190
12.8.3. Szövegbányászati szoftverek . . . 191
12.8.4. Néhány magyar vonatkozású eredmény és projekt . . . 191

13. Webes adatbányászat 192
13.1. Oldalak rangsorolása . . . 192
13.1.1. Az egyszerű Page Rank . . . 193
13.1.2. Az igazi Page Rank . . . 196
13.2. Webes keresés . . . 196
13.2.1. Gyűjtőlapok és Tekintélyek – a HITS algoritmus . . . 196
13.2.2. A SALSA módszer (Jakabfy Tamás) . . . 200
13.2.3. Gyűjtőlapok, Tekintélyek és véletlen séták (Jakabfy Tamás) . . . 202
13.2.4. Automatikus forrás előállító - Gyűjtőlapok és Tekintélyek módosításai . . . 203
13.2.5. Gyűjtőlapok és Tekintélyek módszerének hátrányai . . . 203

14. Adatbányászat a gyakorlatban 205
14.1. Felhasználási területek . . . 205
14.1.1. Az ügyfél életciklusa . . . 205
14.1.2. Kereskedelem . . . 206
14.1.3. Pénzügy . . . 207
14.1.4. Biológia és Orvostudomány . . . 207
14.2. Az adatbányászat bölcsője: az elektronikus kereskedelem (e-commerce) . . . 209
14.3. Adatbányász szoftverek . . . 210
14.3.1. Adatbányászati rendszerek tulajdonságai . . . 211
14.3.2. Esettanulmányok röviden . . . 212

Függelék **214**
Függelék A 214

Előszó

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 mind inkább elterjedtek. Az egyszerű és olcsó tárolási lehetőségek a nyers, feldolgozatlan adatok tömeges méretű felhalmozását eredményezték, ezek azonban a közvetlen visszakeresésen és ellenőrzésen kívül nem sok egyéb haszonnal jártak. A ritkán látogatott adatokból „adat temetők” (data tombs) alakultak ki [63], amelyek tárolása haszon helyett költséget jelentett. Ekkor még nem álltak rendelkezésre olyan eszközök, amivel az adatokba ágyazott értékes információt ki tudtak nyerni. Következésképpen a fontos döntések a döntéshozók megérzésein alapultak, nem pedig az információ-gazdag adatokon. Jól jellemzi ezt a helyzetet John Naisbitt híres mondása, miszerint „We are drowning in information, but starving for knowledge” (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 immár 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 említés szintjén sem talált választ. Bizonyos káosz uralkodott, amiben látszólag mindenre volt megoldás, ám ezek a megoldások többnyire részlegesek voltak, tele a legkülönbözőbb hibákkal.

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 nehézségével, multidiszciplináris voltával a kutatni, gondolkodni és újszerű problémákat megoldani vágyó igényét töké-

[1] A tárolókapacitás növekedése még Moore jóslatát is jócskán felülmúlja. Az utóbbi 15 év alapján ugyanis a tárolókapacitás 9 hónaponként duplázódik meg [119]

letesen kielégíti.

Sorra születtek meg a színvonalas munkák, elemzések, összehasonlítások, mint tiszta irányvonalak rajzolódtak ki a káoszban. A megoldatlan, nyitott problémákra még mindig keressük a választ, így valószínűleg az adatbányászat diadalmenete még sokáig töretlen marad.

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 informatikus beállítottságú olvasóknak készült. Feltételezzük, hogy az olvasó tisztában van algoritmus- [87] és adatbázis-elméleti alapokkal, továbbá nem ismeretlen terület számára a valószínűségszámítás [9, 50] és a lineáris algebra [121] sem.

A jegyzet célja az, hogy az adatbányászati apparátus olyan megismerését nyújtsa, melynek segítségével az olvasó sikerrel oldja meg az egyre több területen felbukkanó újabb és újabb adatbányászati problémákat. Algoritmikus adatbányászatról írunk, ezért azon mesterséges intelligencia területéhez tartozó eszközök (mesterséges neurális hálózatok, genetikus algoritmusok és fuzzy rendszerek), amelyekről azt tartják, hogy az adatbányászatban is használhatók, kevés hangsúlyt kapnak.

A jegyzet legfrissebb változata letölthető a

`http://www.cs.bme.hu/~bodon/magyar/adatbanyaszat`

címen található oldalról.

A jegyzet nem végleges! Folyamatosan bővül, változik. Egyes részek kisebb súlyt kapnak, mások viszont jobban részletezettek. Örömmel fogadok bármilyen észrevételt, javaslatot akár helyesírási, stilisztikai vagy tipográfiai hibára vonatkozóan. Ezeket kérném, hogy a

`bodon@cs.bme.hu`

címre küldjék.

Az írás LaTeX-ben készült, eleinte a *kile*, későbbiekben az *emacs* szövegszerkesztő segítségével. Egyes ábrák *Xfig*-el, mások a *pst-node* csomaggal lettek rajzolva. Az egész munkához az UHU-linux operációs rendszer (`http://www.uhulinux.hu`) nyújtotta a stabil és biztonságos hátteret.

1. fejezet

Bevezetés

A számítógép, korunk legdicsőbb találmánya, rohamléptekkel hódít teret magának az élet minden területén. Egy generáció alatt nélkülözhetetlenné vált, amit szüleink még el sem tudtak képzelni, számunkra már elválaszthatatlanná vált munkánktól és szórakozásunktól egyaránt.

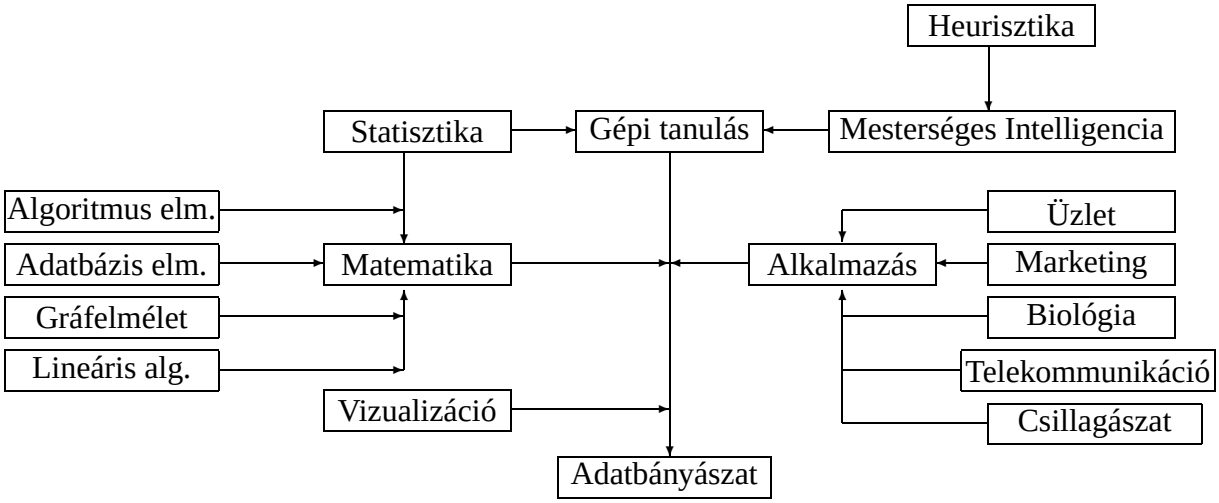
Az Internet elterjedésével még intenzívebben érzékelhető a számítógép térhódítása. A világon az egyik legnagyobb problémát, a távolságot hidalta át. Üzleti és magáncélú érintkezések váltak lehetővé rövidebb idő alatt és hatékonyabban, mint valaha. Adatok millióit kezelik és szállítják a számítógépes rendszerek. Az információkon alapuló döntéshozatal ideje lerövidült, hiszen a hozzáférés könnyebbé és gyorsabbá vált. Az üzleti élet szereplőinek élete is felgyorsult.

Ma a vállalatok léte múlhat az információk gyors és pontos begyűjtésén, elemzésén, a rugalmas fejlődésen, valamint az innováción. Egyre több felső vezető ismeri fel, hogy az Internet, az adatok elektronikus tárolása a vállalat szolgálatába állítható. Az adatok azonban önmagukban nem hasznosak, hanem a belőlük kinyerhető, a vállalat igényeihez igazodó, azt kielégítő információkra lenne szükség. Ez egy újabb szükségletet teremt: egy olyan eszköz iránti igényt, ami képes arra, hogy információszerzés céljából elemezze a nyers adatokat. Ez az új 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 végső soron érthető mintákat fedezünk fel az adatokban. Ezt gyakran megtehetjük különböző lekérdezések eredményeinek vizsgálatával, azonban ez a megoldás lassú, drága és nem elég átfogó. Nem is beszélve arról, hogy az emberi szubjektivitás sokszor hibás, továbbá az adatbázisok olyan nagyok lehetnek, hogy egyes lekérdezések elfogadhatatlanul lassan futnak le. 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”.

Az adatbányászatot az üzleti élet és a marketing keltette életre. Még ma is ezek az adatbányászat fő mozgató rugói. Szerencsére az adatbányászat lehetőségeit egyre több területen ismerik fel, melynek eredményeként az alapkutatásoknak is egy fontos eszköze lett. Alkalmazzák az orvosbiológiában, genetikában, távközlésben, csillagászatban, ...

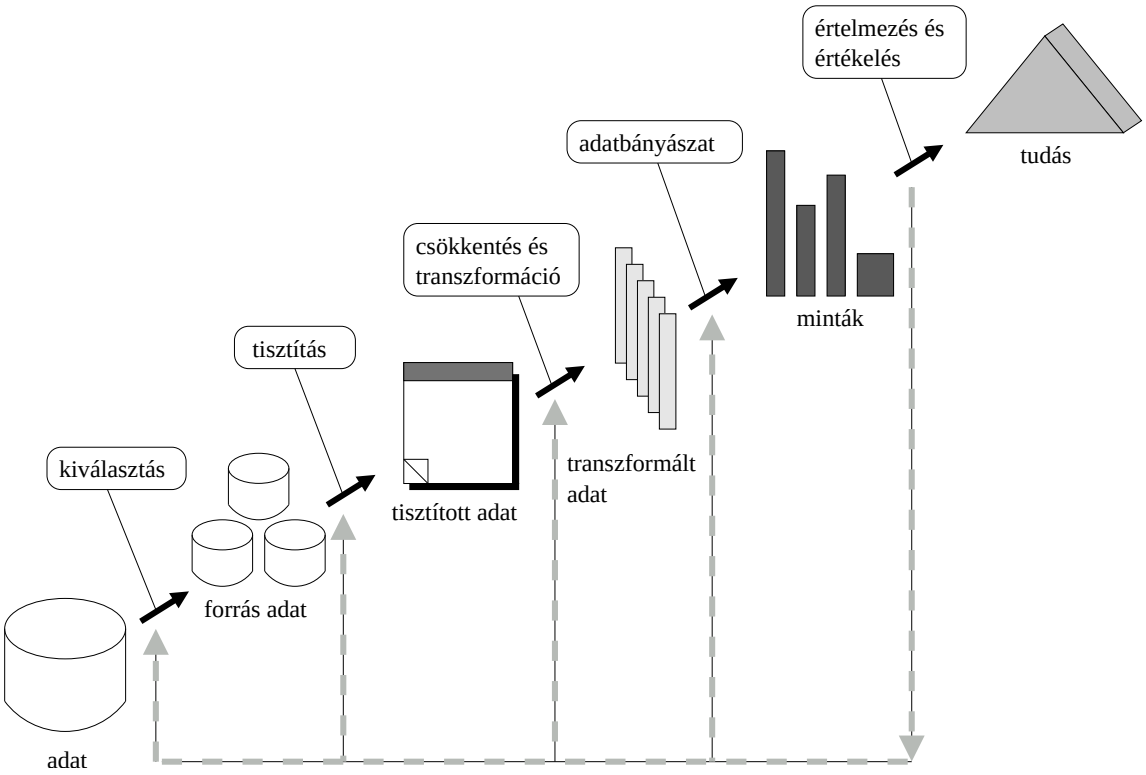
Az adatbányászat egy multi-diszciplináris terület. Az 1.1 ábrán látható, hogy mely tudományterületek eszközeit használja az adatbányászat. Az adatbányászat több hangsúlyt fektet az algoritmusokra, mint a statisztika, és többet a modellekre, mint a gépi tanulás eszközei (pl. neurális hálózatok). Mára az adatbányászat akkora területté nőtte ki magát, hogy szinte lehetetlen átlátni magas színvonalon az egészet.



1.1. ábra. Az adatbányászat kialakulása

1.1. A tudásfeltárás folyamata

A tudáskinyerés folyamata során 6-10 fázist szokás elkülöníteni [49, 63] attól függően, hogy mely lépéseket vonjuk össze (tekinthetjük például az 1.2 ábrát):



1.2. ábra. A tudásfeltárás folyamata

I. Az alkalmazási terület feltárása és megértése, fontosabb előzetes ismeretek begyűjtése, és a

felhasználási célok meghatározása.

II. Céladatbázis létrehozása: kiválasztani a használni kívánt adatbázist, (vagy annak csak egy részét), amiből a tudást ki akarjuk nyerni.

III. Adattisztítás, és előfeldolgozás: itt olyan alapvető operációkat értünk, mint a téves bejegyzések eltávolítása, hiányos mezők pótlása, zajok szűrése stb.

IV. Adatintegráció: a feldolgozás számára fontos, esetleg elosztott adatbázisok egyesítése.

V. Adattér csökkentés: az adatbázisból a cél szempontjából fontos attribútumok kiemelése.

VI. Adatbányászati algoritmus típusának kiválasztása: eldönteni, hogy a megoldandó feladat klaszterezés, vagy szabály-, illetve mintakeresés, esetleg osztályozás.

VII. A megfelelő adatbányászati algoritmus meghatározása. Előnyeinek, hátrányainak, paramétereinek vizsgálata, futási idő- és memóriaigény elemzése.

VIII. Az algoritmus alkalmazása.

IX. A kinyert információ értelmezése, esetleg visszatérés az előző lépésekhez további finomítások céljából.

X. A megszerzett tudás megerősítése: összevetés elvárásokkal, előzetes ismeretekkel. Eredmények dokumentálása és átadása a felhasználónak.

A sikeres adatbányászati projektekben az első 5 lépés teszi ki az idő- és pénzráfordítások legalább 80%-át. Ha a célok nem kellőképpen átgondoltak és a bányászandó adatok nem elég minőségiek, akkor könnyen előfordulhat, hogy az adatbányász csak vaktában dolgozik és a kinyert információnak tulajdonképpen 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.

Ez a jegyzet az 6. és 7. lépéseket veszi szemügyre: rendelkezésünkre áll egy adatbázis, tudjuk, milyen jellegű információra van szükségünk, és az adatbányász feladata, hogy ennek megoldására minél gyorsabb és pontosabb algoritmust adjon.

Általánosabban 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.

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

egyik legjobb 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.

Az adatok illetve a kinyert információk megjelenítésének módja legalább annyira fontos, mint az összefüggések meghatározása. A végfelhasználókat (akik általában vezetők) jobban megragadja egy jól elkészített ábra, mint különböző matematikai struktúrák nyers tálalása. A megjelenítés tehát fontos része az adatbányászatnak. Ezt jól 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). Ezeknél a rendszereknél az összefüggéseket, mintázatokat, közös tulajdonsággal rendelkező csoportokat maguk a felhasználók veszik észre. Az adatbányászati szoftverekről részletesebben a 14. fejezetben olvashatunk.

1.2. Szabványok

Kezdetben sok adatbányászati projektre jellemző volt, hogy az adatbányászok megkapták az adatokat és némi információt az alkalmazási területről és cserébe várták tőlük a kincset érő információkat. A szoros együttműködés hiánya azonban csak olyan információkhoz vezetett amelyekkel az alkalmazási terület embererei nem sok mindent tudtak kezdeni. Az adatbányászat elterjedésével (és a minőségbiztosítási elvárásokkal) fellépett az igény, hogy legyen egy szabvány, egy útmutató az adatbányászati projektek lebonyolításáról. Így született meg a CRISP-DM (CRoss Industry Standard Process for Data Mining) [29], amely adatbányászati eszköztől és felhasználási területtől függetlenül leírja, hogy miként kellene kinéznie egy adatbányászati projektnek, illetve ismerteti a kulcsfontosságú lépéseket, és a potenciális veszélyeket.

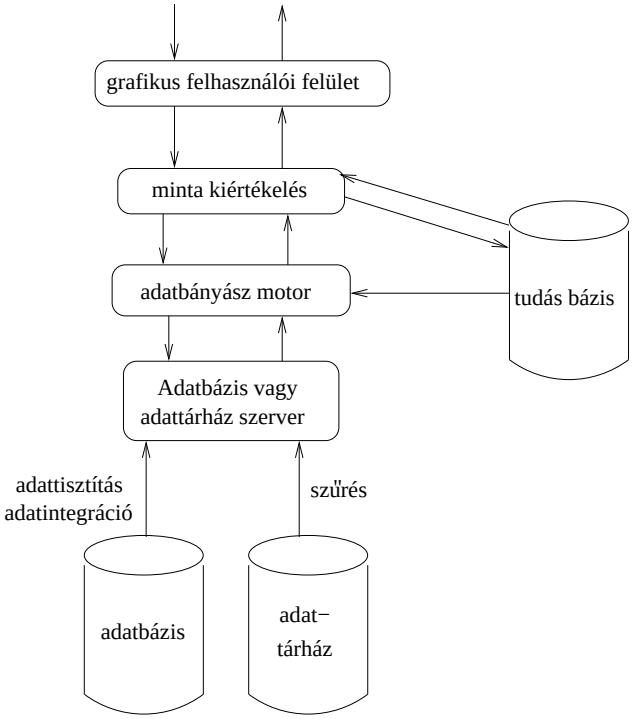
Az adatbányászati folyamat szabványosítása mellett egyre nagyobb az igény a folyamat egyes lépéseiben felmerülő megoldások, problémák, eszközök szabványosítására. Ezek közül a legismertebbek:

- 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 adat bányászati API (JDMAPI)

1.3. Adatbányászati rendszer architektúrája

Egy adatbányászati rendszernek kapcsolatban kell lennie az adatbázissal, a felhasználóval és esetleg valami tudásalapú rendszerrel. Ezek alapján egy tipikus adatbányászati architektúra az 1.3. ábrán látható.

Adatbázis, adattárház vagy más információ 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.



1.3. ábra. Tipikus adatbányászati rendszer architektúrája

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

Tudás bázis: A területre jellemző, valamilyen szinten 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.

Minta kiértékelő modul: Ez a modul felelős a kinyert minta vagy összefüggések kiértékeléséért a területre jellemző érdekességi mutatók alapján. Sokszor látni fogjuk, hogy minél jobban egybe tudjuk építeni az adatbányászatot a minta kiértékelésével, annál hatékonyabb és gyorsabb lehet 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, értelmes tálalása rendkívül fontos, hiszen ennek hiánya elriaszthatja a felhasználót az adatbányászattól.

1.4. Legjelentősebb adatbányászati feladatok

Feltehetjük, hogy az adatbázis valamilyen objektumok (ügyfelek, betegségek, vásárlók, telekommunikációs események, ...) különböző tulajdonságait írja le. A tulajdonság helyett gyakran használ-

juk majd az attribútum szót[1]. Az adatbányászat feladata a rejtett összefüggések, kapcsolatok felderítése. Az összefüggések típusa szerint a következő adatbányászati alapproblémákról beszélhetünk:

Gyakori minták kinyerése: Adott objektumok egy sorozata. Célunk megtalálni a gyakran előforduló (rész-) objektumokat. Az objektumok lehetnek elemhalmazok vagy sorozatok, esetleg epizódok (részben rendezések), gráfok stb.

Attribútumok közötti kapcsolatok: Gyakran hasznos, ha az objektumokra úgy tekintünk, mint az attribútumok megvalósulásaira és keressük az összefüggéseket az attribútumok között. Többféle összefüggés létezik. Ilyenek például az asszociációs-, 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 teszi. Leggyakrabban a modell egy döntési fa, de lehet if-then szabályok sorozata, valamilyen matematikai formula, vagy akár egy neurális hálózat stb. is.

Klaszterezés: Objektumokat előre nem definiált csoportokba (klaszterekbe) kell sorolnunk úgy, hogy az egy csoportba tartozó objektumok hasonlóak legyenek, míg a különböző csoportba kerültek különbözzenek egymástól. Két pont hasonlóságát egy előre megadott (távolságszerű) függvény segítségével szokás értelmezni.

Sorozatelemzés: A sorozatelemzésbe többféle adatbányászati feladat tartozik. Kereshetünk egymáshoz hasonlító (akár rész-) sorozatokat. Ezen kívül elemezhetjük a sorozat alakulását, és különböző regressziós módszerekkel próbálhatjuk megjósolni a jövőbeli valószínűleg előforduló eseményeket.

Eltéréselemzés: Azokat az elemeket, amelyek nem felelnek meg az adatbázis általános jellemzőinek, tulajdonságaik nagy mértékben eltér az általánostól *különc* pontoknak nevezzük. A legtöbb adatbányászati algoritmus az ilyen különc pontoknak nem tulajdonít nagy jelentőséget, legtöbbször zajnak vagy kivételnek kezeli őket. Azonban az élet egyre több területén merül fel az igény, hogy éppen az ilyen különc pontokat találjuk meg. Eltéréselemzés főbb alkalmazási területe a másolás-, koppintáskeresés továbbá a csalások, visszaélések, vírusok, hackertámadások kiszűrése.

Webes adatbányászat: Az Interneten óriási adattömeg található, így az Interneten alapuló információ-kinyerő algoritmusok is az adatbányászat területéhez tartoznak. A jegyzetben szó lesz intelligensebb keresésről, oldalak rangsorolásáról, illetve hasonló tartalmú oldalak megtalálásáról.

Előfordulhat, hogy az adatbányászati rendszer, 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,

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

ha felhasználható, azaz tudunk valamit kezdeni vele [137]. 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 egy 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 feladatát függvény approximációként is felfoghatjuk, aminek 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 egyes algoritmusok futási ideje olyan rövid legyen, hogy az eredmények a gyakorlatban elfogadható időn belül érkezzenek. Az alkalmazásokban nem ritkák a giga- sőt terabájt nagyságú adathalmazok. A [42] írásban 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 algoritmus rész). 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 karbantartása rész).

1.5. Sikeres alkalmazások

Az „adat bányászata” eredetileg statisztikusok által használt kifejezés, az adatok nem kellőképpen megalapozott felhasználására, amely során valaki helytelen következtetést von le. Igaz ugyanis, hogy tetszőleges adathalmazban felfedezhetünk valamilyen struktúrát, ha elég sokáig nézzük az adatot. Ismét utalunk a lehetséges következtetések nagy számából eredő veszélyre. A helytelen következtetésre az egyik leghíresebb példa az alábbi: 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 a képességét miután tudomást szerez róla.

A fenti példa ellenére mára az adatbányászat szó elvesztette jelentésének negatív tartalmát, a számos sikeres alkalmazásnak köszönhetően. A teljesség igénye nélkül felsorolunk belőlük néhányat.

– A bankok egyre gyakrabban alkalmaznak olyan automatikusan előállított döntési fákat, amelyek alapján egy program javaslatot tesz egy hitel megítéléséről. Ezt a kérelmezők személyes,

továbbá előzetes hitelfelvételi és törlesztési adatai alapján teszi (osztályozás) [143]. Tesztek például igazolták, hogy a hitelbírálat minősége javult az USA-ban, amikor a bankok áttértek a kötelezően alkalmazott, írásban rögzített szabályok alkalmazására [143]. Ezeket a szabályokat pedig az adatbányászat segítségével állították össze.

– A vásárlói szokások felderítése szupermarketekben, illetve nagy vevőkörrel rendelkező áruházakban hasznos lehet az áruház terméktérképének kialakításánál, akciók, eladáshelyi reklámok (Point of Sales, Point of Purchase), leárazások szervezésénél... (asszociációs szabályok).

– Az ember genotípusának elemzéséhez a gének nagy száma miatt szintén adatbányászati algoritmusok szükségesek. Az eddigi sikeres kísérletek célja olyan géncsoportok feltárása volt, amelyek a cukorbetegség bizonyos változataiért felelősek. A teljes emberi génrendszer feltárásával ez a terület egyre fontosabb lesz.

– Az on-line áruházak a jövőben egyre elfogadottabbak és elterjedtebbek lesznek. Mivel az on-line kereskedelemben nem használhatóak a megszokott személyes marketing eszközök a forgalom (és a profit) személyre szabott vásárlási ajánlatokkal növelhető. Az ajánlatokat az eddigi vásárlási adatok és a rendelkezésre álló demográfiai adatok elemzése alapján tehetjük meg (epizódkutatás, asszociációs szabályok).

– A csillagászatban az égitestek óriási száma miatt a hagyományos klaszterező algoritmusok még a mai számítási kapacitások mellett sem képesek racionális időn belül különbséget tenni galaxisok, közeli csillagok és más égi objektumok között. Az újabb, kifinomultabb algoritmusok futási ideje jóval kevesebb, ami lehetővé teszi a klaszterezést (klaszterezés).

– Utazás szervezéssel kapcsolatos minták kinyerésével hatékonyabban (és ennek következtében nagyobb nyereséggel) megszervezhetők a nagy költségfaktorú tényezők, pl. szállodai szobák, repülőjegyek leárazása, vagy áremelése (epizódkutatás, gyakori minta).

– A vírusölő programok az ismert vírusokat lenyomataik alapján detektálják, az ismeretleneket pedig többnyire valamilyen heurisztikus módon próbálják kiszűrni. Osztályozó algoritmusok felhasználásával az ismert vírusok tulajdonságai alapján olyan modellt lehet felállítani, ami jól leírja a vírusok tulajdonságait [129, 130]. A modellt sikeresen alkalmazták új ismeretlen vírusok kiszűrésére (osztályozás).

Néhány sikeres esettanulmányról a 14.3.2 részben olvashatunk.

1.6. Az adatbányászat feltételei

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

Nagy mennyiségű adat: A nagy mennyiségű adat 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 csak a véletlen eredménye. Sajnos sok adatot sokáig tart feldolgozni, sőt az algoritmusok egy jelentős része érzékeny arra, hogy az adatbázis elfér-e a memóriában.

Sok attribútum: Ha az objektumokat leíró attribútumok száma kicsi, akkor hagyományos eszközökkel (grafikonok, egyszerű táblázatok, kis dimenziós, forgatható, színes ábrák, ...) 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 módszereknek nincs esélyük.

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[2]) nevezik.

Torzítatlan adat: Az adatbányászat sikeressége múlhat az adatok nem megfelelő kiválasztásán. Ide tartozó fogalom az ún. BIBO (bias in, bias out[3]), amely arra hívja fel a figyelmünket, 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 csak kinyerik, 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.

A jegyzet fejezeteiben a legkevésbé ismert, de napjainkban egyre nagyobb teret nyerő területeket járjuk körül: a gyakori minták kinyerését, az attribútumok közötti összefüggések meghatározását, a sorozatelemzést, a klaszterezést és a webes adatbányászatot. Minden esetben az algoritmusok gyakorlati felhasználását példákon keresztül szemléltetjük; emellett megadjuk a problémák formális definícióit, és bemutatjuk a legismertebb, leghatékonyabb algoritmusokat is. A jegyzet további célja, hogy összefoglalja az eddig nem, vagy csak kis hatékonysággal megoldott problémákat, továbbá a jelenlegi kutatási területeket.

[2] szemét be, szemét ki
[3] torzítás be, torzítás ki

2. fejezet

Alapfogalmak, jelölések

Ebben a részben tisztázzuk a jegyzet során használt fogalmak jelentését. Célszerű akkor átnéznünk e fejezet egyes részeit, amikor az olvasás során olyan részbe ütközünk, ami nem teljesen tiszta.

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 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 lesz 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 mindenki tudja definiálni a lexikografikus rendezést 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 pedig az f *értékkészletének*. Azt a függvényt, amely 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 $a, b, c \in$ $\in H$ 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 invere, akkor *csoportról* beszélünk. Az a inverzére (a^{-1}) teljesüljön, hogy $a \star a^{-1} = a^{-1} \star$ $\star a = e$. 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 *disztributívak* egymásra nézve, azaz $(a + b) \star c = a \star c + b \star c$.

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

2.2. Lineáris algebra

Feltételezzük, hogy az olvasó tisztában van a *mátrix*, *vektor*, illetve a mátrix (vektor) *transzponáltjának* fogalmával. A hagyományoknak megfelelően az A mátrix i-edik sorából képzett vektort A^i-vel jelöljük, $||v||$-vel a v vektor euklideszi normáját ($\sqrt{\sum_i v_i^2}$) és $v^T w$-vel a v^T, w vektrok skaláris szorzatát ($\sum_i v_i^T w_i$).

2.3. 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. Ha $G\cong G$, akkor *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 mátrixat a gráfon kívűl az határozza meg, hogy melyik csúcsot hívjuk az elsőnek, másodiknak, ... A szomszédossági gráfot tehát a gráf és az $f : V\to\{1,\dots,|V|\}$ bijekció adja meg. Hurokél nélküli, 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) $\langle v_0,v_1,\dots,v_k\rangle$ sorozatát értjük, amelyre $u=v_0$, $u'=v_k$, és $(v_{i-1},v_i)\in E$ $(i=1,2,\dots,k)$. Egy út *egyszerű*, ha a benne szereplő csúcsok páronként különbözők. A $\langle v_0,v_1,\dots,v_k\rangle$ 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 1 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.

2.4. Valószínűségszámítás

Feltételezzük, hogy az olvasó tisztában van a *valószínűségi változó*, valószínűségi változó *eloszlásának, sűrűségfüggvényének, eloszlásfüggvényének* a valószínűségi változó *várható értékének* ($E[X]=\mu=\sum x\cdot p(x)$) és *szórásának* ($D^2[X]=\sigma^2=E[(X-\mu)^2]$) vagy általánosan az *n-edik centrális momentumok* fogalmával ($D^n[X]=E[(X-\mu)^n]$).

Kevésbé ismert a *ferdeség*, ami egy eloszlás asszimetriáját próbálja megadni. 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 a $\gamma_1 = \frac{D^3[X]}{(D^2[X])^{3/2}}$), de szokás még a $\beta_1 = \sqrt{\gamma_1}$-et is haszálni.

Szintén nem az alapfogalmak közé tartozik a *lapultság* fogalma, ami egy eloszlás csúcsosságát adja meg. A lapultságnak is több elfogadott definíciója létezik. Legelterjedtebb a $\beta_2 = \frac{D^4[X]}{(D^2[X])^2}$ (kurtosis proper), és a $\gamma_2 = \beta_2 - 3$ (kurtosis excess) értékek. 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.

2.4.1. Hoeffding-korlát

A Hoeffding-korlátra a mintavételzéssel kapcsolatos állítások alapja.

2.1. 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:*

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

2.4.2. Entrópia

Legyen X egy diszkrét valószínűségi változó, amely értékeit egy $\mathcal{X}$ halmazból veheti fel. Az $l_X = -\log_2 p(X)$ valószínűségi változót az X *entrópiasűrűségének* nevezzük. X entrópiáját – $H(X)$-et – ezen változó várható értékével definiáljuk:

$$H(X) = -\sum_{x\in\mathcal{X}} p(x)\log_2 p(x).$$

Az entrópia valamiképpen a változó *bizonytalanság*át fejezi ki. Ha $\mathcal{X}$ 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, $\log_2(|\mathcal{X}|)$-t.

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

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

vagy egy kicsit átalakítva kapjuk, hogy

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

Be lehet bizonyítani, hogy $\mathcal{H}(X|Y) = \mathcal{H}(XY) - \mathcal{H}(Y)$, 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:

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

2.5. Statisztika

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

Legyenek $X_1, X_2, \ldots, X_n$ független, azonos eloszlású valószínűségi változók. Ekkor a $\overline{X} = \frac{X_1+X_2+\cdots+X_n}{n}$ valószínűségi változót *empirikus középnek*, vagy *mintaátlagnak*, a $s_n^{*2} = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2$ valószínűségi változót pedig *korrigál empirikus szorásnégyzetnek* nevezzük.

2.2. definíció. *Legyenek $\xi_1, \xi_2, \ldots, \xi_n$ egymástól független, standard normális eloszlású valószínűségi változók. Ekkor az $\sum_{i=1}^{n}\xi_i^2$ valószínűségi változó eloszlását n paraméterű* χ^2 eloszlásnak *(χ_n^2) nevezzük.*

A fentiekből következik, hogy az $\frac{(n-1)s^{*2}}{\sigma^2}$ valószínűségi változó eloszlása χ_n^2, amenyiben a s^{*2} σ szórású, normális eloszlású valószínűségi változók korrigál empirikus szorásnégyzetét jelöli

2.3. definíció. *Legyenek X és Y két olyan valószínűségi változó, amelyek eloszlása rendre χ_n^2 és χ_m^2. Ekkor a $Z = \frac{X/n}{Y/m}$ valószínűségi változó eloszlását $F_{n,m}$ eloszlásnak hívjuk.*

2.5.1. Hipotézisvizsgálat

A hipotézisvizsgálat feladata mindig valamilyen 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 kísérletezgetések, minták állnak rendelkezésünkre. 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. A gyakorlatban egy paraméterrel (α) rögzítik az elsőfajú hiba elkövetésének megengedett valószínűségét. Az $1-\alpha$ értéket a *próba szintjének* hívjuk.

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 ún. *próbastatisztika* értéket, és ezt vetjük össze egy ismert eloszlással. Az α-nak célszerű kis (0.1 és 0.01 közötti) értéket választani[1].

2.5.2. Az F-próba

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 $\frac{(n_X-1)s_X^{*2}}{\sigma_X^2}$ és $\frac{(n_Y-1)s_Y^{*2}}{\sigma_Y^2}$ χ^2 eloszlásúak (n_X-1) illetve (n_Y-1) paramáterrel. Ha a nullhipotézis fennáll, akkor az

$$F = \frac{s_X^{*2}}{s_Y^{*2}}$$

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

próbastatisztika F-eloszlású (n_X-1, n_Y-1) paraméterrel. Azonban $\frac{1}{F}$ is F-elosszású (n_Y-1, n_X-1) paraméterrel, ezért a gyakorlatban $F^* = \max\{F, 1/F\} \geq 1$ statisztikát szokás használni.

2.5.3. A χ^2-próba

A χ^2 próbák az alábbi tételt használják fel.

2.4. 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.

A χ^2 eloszlás kvantiliseit függvény-táblázatokban megtalálhatjuk.

A χ^2-próba legfontosabb alkalmazási területei az (1.) illeszkedés-, (2.) függetlenség- és (3.)homogenitásvizsgálat. Témánkhoz a függetlenség-vizsgálat tartozik hozzá, így a továbbiakban ezt részletezzük. A χ^2 próba iránt érdeklődőknek a [73] magyar nyelvű irodalmat ajánljuk.

2.5.4. 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.

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

Ha az események valószínűségei adottak, akkor tiszta illeszkedés vizsgálati feladatról beszélünk, ahol

$$H_0 : \mathbb{P}(A_i \cap B_j) = p_i q_j$$

hiszen p_i, q_j értékek adottak. Jelölje k_{ij} az $A_i \cap B_j$ esemény bekövetkezésének számát. Ekkor ki kell számítanunk a

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

ún. 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 2.4-as tétel alapján χ^2_{rs-1} és az α paraméter határozza meg. Táblázatból keressük ki, hogy a χ^2_{rs-1} eloszlás hol veszi fel az $1-\alpha$ értéket. Amennyiben ez 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. kontingencia-táblázat. Ez egy többdimenziós táblázat, amely celláiban a megfelelő esemény bekövetkezésének száma található. Egy ilyen 2-dimenziós kontingencia-táblázatot láthatunk a következő ábrán.

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

Az $A_i \cap B_j$ megfigyelt értéke k_{ij}, várt értéke 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.

2.6. Algoritmus-elmélet

Terjedelmi okok miatt csak felsorolni tudjuk azokat az algoritmusokat, amelyeket az olvasónak ismernie kell. Ezek pedig: lineáris-, bináris keresés, mélységi-, szélességi bejárás, Kruskal algoritmusa minimális súlyú feszítőfa meghatározásához stb. Emellett feltételezzük, hogy az olvasó tisztában van az NP-teljesség (vagy általánosabban a bonyolultság) elméletének alapjaival.

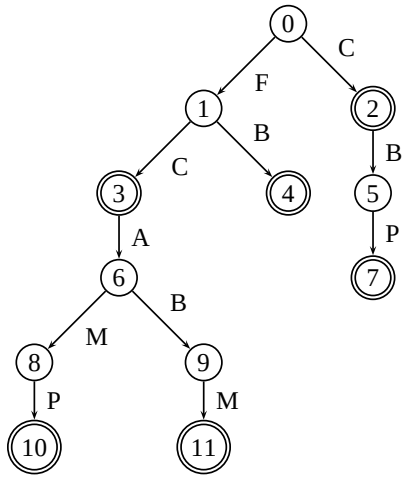
2.7. Adatstruktúrák

Feltételezzük, hogy az olvasó tisztában van a lista (vektor) és a tömb fogalmával. Az adatbányászatban további közkedvelt adatstruktúrái az ún. *szófa* (trie), vagy más néven prefix-fa (prefix-tree), a piros-fekete fa, illetve a hash-tábla.

2.7.1. Szófák

A szófát eredetileg szótár szavainak tárolásánál alkalmazták, annak érdekében, hogy gyorsan el lehessen dönteni, hogy egy adott szó szerepel-e a szótárban [36], [54]. A szavak az abc 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 (`retrieval`). A továbbiakban az alaphalmazt $\mathcal{I}$-vel, az alaphalmaz felett értelmezett, adott sorozatok halmazát szótárnak hívjuk. A 2.1 ábrán egy szófát láthatunk, mely az C, FC, FB, CBP, $FCAMP$, $FCABM$ sorozatokat tárolja.

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 $\mathcal{I}$-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.



2.1. ábra. Példa szófára

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) [54] minden ponthoz egy rögzített hosszúságú, mutatókat tartalmazó vektort veszünk fel. Az i-edik mutató mutat az i-edik elemhez tartozó él végpontjára. Ha a pontnak nincs ilyen címkéjű éle, akkor a mutató értéke NULL. A vektor hossza az $\mathcal{I}$ elemszámával egyezik meg.

A *láncolt listás implementációban* [36] 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 címke, mutató 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) $|\mathcal{I}| \cdot 4$ bájt, a listás implementációé $n \cdot 2 \cdot 4$ bánt, ahol n az adott pontból induló élek száma, amire igaz, hogy $0 \leq n \leq |\mathcal{I}|$. Ha a szófa pontjai olyanok, hogy kevés élük van, akkor a listás implementációnak lesz kevesebb memóriára szüksége, sok élnél azonban táblázatos implementáció a jobb megoldás. A két technikát ötvözhetjük akár egy adott szófán belül is [134], [161]: ha a pont éleinek száma meghalad egy korlátot (általában $\mathcal{I}/2$-t),

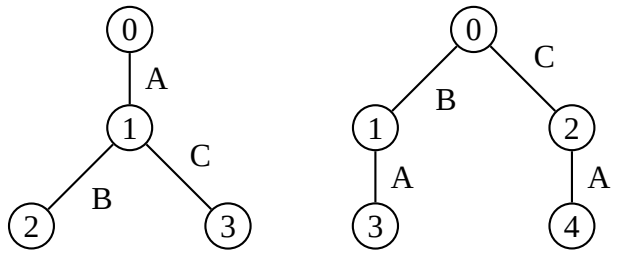
akkor táblázatos implementációt használunk, ellenkező esetben maradunk a listás megoldásnál.

Megemlítünk két szófa leszármazottat. Ezek a *nyesett szófák* (pruned trie) és a *PATRICIA 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, PATRICIA fáknál ez a korlátozás nem áll fenn.

Halmazokat tartalmazó szófák

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 ABC, ABD, ACE elemhalmazokat tárolják. Az első szófa az ABC szerint csökkenő sorrendet használ ($C \prec B \prec A$), míg a második ennek ellenkezőjét.



2.2. ábra. Példa: különböző rendezést használó szófák

Egy szófa memóriaigényét 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.

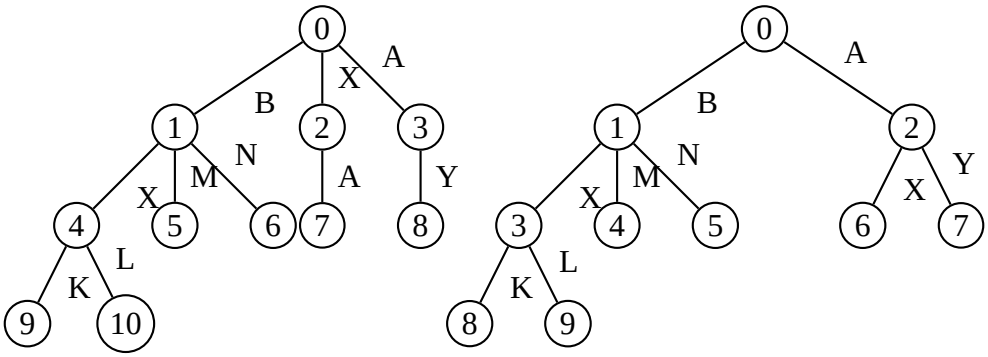
2.5. tétel. *A minimális szófa probléma NP-nehéz .*

Eredetileg a feladatot n-esekre bizonyították, de ebből következik, hogy halmazokra is érvényes. Legyen ugyanis az alaphalmaz $\mathcal{I}$. Ekkon 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 számána arányában csökkenő sorba vannak renzve. 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 sugalta.

Sajnos a fenti módon kapott szófa nem feltétlenül adja a legkevesebb pontot tartalmazó szófát. Ezt a legegyszerűbben egy ellenpéldával tudjuk bizonyítani. Legyenek a halmazaink a következőek: AX, AY, BXK, BXL, BM, BN. A B elem gyakorisága 4, az X elemé 3, A-é 2 és a többi elemé 1. Ha felrajzoljuk ezen gyakoriságok alapján kapott rendezés ($B > X > A > K > L > M > N$, de a K, L, M,

N elemek egymáshoz viszonyított sorrendje tetszőleges lehet) szerinti szófát, akkor a bal oldali szófát kapjuk. Ha az A és X elemek sorrendjét felcseréljük, akkor eggyel kevesebb pontot tartalmazó szófát kapunk (jobb oldal).



2.3. ábra. Ellenpélda arra, hogy az előfordulás szerinti csökkenő sorrend adja a minimális méretű szófát

2.7.2. 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:

I. Minden csúcsnak a színe piros vagy fekete.

II. Minden levél színe fekete.

III. Minden piros csúcsnak mindkét fia fekete.

IV. 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 (pl. [87]).

2.7.3. Hash-tábla

A hash-tábla magyar elnevezése hasító-tábla (☺), de mi ezt a szót nem fogjuk használni. A hash-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. A hash-táblákról önmagában fejezeteket lehet írni, ennyi bevezető azonban elég ahhoz, hogy megértsük a jegyzet további részeit.

3. fejezet

Előfeldolgozás, hasonlósági függvények

Ebben a fejezetben a legfontosabb előfeldolgozási műveleteket ismertetjük, majd rátérünk arra, hogy milyen elterjedt mértékek vannak elemek közötti hasonlóságra. Mindenek előtt azt kell tisztáznunk, hogy milyen típusú attribútumok léteznek matematikus szemmel.

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

I. A *kategória típusú* (*nominal*) attribútumnál az attribútum értékei között csak azonosságot tudunk vizsgálni. Tehát csak azt tudom mondani, hogy $a = a'$ vagy azt, hogy $a \neq a'$. A kategória típusú attribútum egy speciális esete a *bináris attribútum*, ahol az attribútum csak két értéket vehet fel.

II. 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.

III. Ha az eddigiek mellett meg tudunk adni egy + függvény, amivel az elemek csoportot alkotnak, akkor *intervallum típusú* (*interval scale*) attribútumról beszélünk.

IV. Ha egy intervallum típusú attribútumnál meg lehet adni zérus értéket, vagy pontosabban az attribútum elemei gyűrűt alkotnak, akkor az attribútum *arány skálájú* (*ratio scale*).

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

3.1. Előfeldolgozás

3.1.1. Hiányzó értékek kezelése

Az adatbányászati algoritmusok csak olyan elemeket tudnak kezelni, amelyeknek minden attribútuma adott. A való élet adatbázisainál ez nem mindig áll fenn, könnyen lehet, hogy bizonyos cellákat nem töltött ki az adatot begépelő személy. Ezeket a hiányzó cellákat fel kell tölteni valamilyen értékkel. Ez lehet egy alapértelmezett érték (default value), de szokás az átlagot, a mediánt vagy a maximális, minimális értéket választani olyan attribútum esetében amikor, ezeket definiálni tudjuk (pl.: intervallum típusúak esetén).

3.1.2. Attribútum transzformációk

Normalizálás

Normalizáláson azt értjük, hogy az attribútum elemeit a [0,1] intervallum elemeivel helyettesítjük úgy, hogy a helyettesített értékek eloszlása megegyezzen az eredeti értékek eloszlásával. Tegyük fel, hogy az A attribútum $a_1, a_2, \ldots, a_l$ értékeket vesz fel. Az a_j, $j=1, \ldots, l$ érték normáját a'_j-vel jelöljük. Normalizálásra két módszer terjedt el.

Min-max normalizálás: Itt egy sima lineáris transzformációt végzünk: $a'_j = \frac{a_j - min_A}{max_A - min_A}$, ahol min_A (max_A) jelöli az A attribútum legkisebb (legnagyobb) értékét.

Standard normalizálás (z-score normalization): $a'_i = \frac{a_i - \bar{A}}{\sigma_A}$, ahol $\bar{A}$ az A attribútum átlaga, σ_A pedig a szórása. A hagyományos szórás ($\sqrt{\frac{\sum_{i=1}^{l}(a_i - \bar{A})^2}{l}}$) helyett az abszolút szórást is használni szokták ($\frac{\sum_{i=1}^{l}|a_i - \bar{a}|}{l}$). Ennek előnye, hogy csökkenti az átlagtól távol eső pontok (különcök, outlier-ek) hatását.

3.1.3. Mintavételezés

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

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, asszociációs szabályok, χ^2 alapú függetlenségvizsgálatnál ez az eset áll fenn.

Tegyük fel, hogy elemek halmazából egy tetszőleges x elem előfordulásának valószínűsége p_x és az elemekből visszatevéses mintavételezéssel m mintát veszünk. A mintavételezés hibázik, amennyiben x relatív gyakorisága eltér p_x-től, pontosabban a mintavételezés hibája:

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

Jelölje X_i azt a valószínűségi változót, ami 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_x paraméterű bináris eloszlást követ. Ezt felhasználva:

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

A második egyenlőségnél kihasználtuk, hogy a binomiális eloszlás várható értéke $m \cdot p_x$. A binomiális eloszlás várható értékétől való eltérés valószínűségére több ismert korlát is létezik [138]. Az 1-es függelékben elolvashatjuk a Csernov-korlát egy általános formájának bizonyítását, itt csak a végeredményt adjuk meg, ami:

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

és

$$p\Big(Y \leq m \cdot (E[X] - \varepsilon)\Big) \leq e^{-2\varepsilon^2 m}$$

amiből megkapjuk, hogy:

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

Amennyiben a hibakorlátot δ-val jelölöm, akkor az alábbinak kell igaznak lennie:

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

Ha például azt szeretnénk, hogy a mintavételezés során tetszőleges elem minta, – illetve előfordulásának valószínűsége – 0.01-nál nagyobb eltérés valószínűsége kisebb legyen 1%-nál, akkor a minta mérete legalább 27000 kell, hogy legyen. A 3.1 táblázatban adott eltérés- és valószínűségkorlátokhoz tartozó minimális mintaméret található.

ε	δ	$\lvert\mathcal{M}\rvert$
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

3.2. Hasonlósági mértékek

Az adatbányászatban gyakran szükségünk lesz arra, hogy attribútumokkal leírt elemek között hasonlóságot definiáljunk. Természetesen elvárjuk, hogy ha minél inkább több azonos érték szerepel az attribútumaik között annál hasonlóbbak legyenek az elemek. A gyakorlatban hasonlósági mérték helyett *különbözőségi* mértékkel dolgozunk, amely a hasonlóság inverze (minél hasonlóbbak, annál kevésbé különbözők). Elvárjuk, hogy két elem különbözőségét ($d(x,y)$) ki lehessen fejezni egy pozitív valós számmal, továbbá egy elem önmagától ne különbözzön, szimmetrikus legyen ($d(x,y) = d(y,x)$), és teljesüljön a háromszög egyenlőtlenség ($d(x,y) \leq d(x,z) + d(y,z)$). Tehát a különbözőség metrika legyen. Két elem különbözősége helyett gyakran mondunk majd két elem *távolságát*.

A következőkben sorra vesszük, hogyan definiáljuk a távolságot különböző típusú attribútumok esetében, és azt, hogy miként lehet egyes attribútumok fontosságát (súlyát) megnövelni. Általánosan igaz az, hogy ha az attribútum elemein teljes rendezést tudunk definiálni (sorrend, intervallum, arányskálázott), akkor tetszőleges $a_1 \leq a_2 \leq a_3$ értékek esetén $d(a_1, a_3) \geq max\{d(a_1, a_2), d(a_2, a_3)\}$.

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 elemek hasonlóságát, ha azok m darab bináris attribútummal vannak leírva? Készítsük el a következő összefoglaló táblázatot.

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

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

Ez alapján definiálhatjuk az ún. invariáns és variáns hasonlóságot. Az invariáns hasonlóságot 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 hasonlóság az invariáns jelzőt, mert nem változik az értéke, ha valaki máshogy kódolja az attribútumokat (tehát kódolás invariáns). A legegyszerűbb invariáns hasonlóság 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 a ténynek, hogy valaki fertőzött beteg, mint annak, hogy nem az. A konvencióknak megfelelően 1-essel kódoljuk a lényeges (általában ritka) kimenetet. A legegyszerűbb variáns hasonlósági mérték a Jaccard-koefficiens 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 hasonlóság a nemegyezések relatív száma:

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

ahol m a kategória típusú attribútumok száma, u pedig azt adja meg, hogy 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.

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 ...

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 a $\frac{x-1}{M-1}$ művelettel. Így mindegyik egyenlő súllyal szerepel majd a végső hasonlósági mértékben. Ezután alkalmazhatjuk valamelyik intervallum típusú hasonlóságot.

3.2.4. Intervallum típusú attribútum

Az intervallum típusú attribútumokat általában valós számok írják le. Ilyen attribútumra példa egy ember súlya, magassága, egy ország éves átlaghőmérséklete stb. 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.

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 ($L_\infty(\vec{z}) = \max_i\{|z_i|\}$).

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 értékkészletű attribútumoknak nagyobb hatásuk van a hasonlóság értékére, mint a kis értékkészletűeknek. Jogos tehát az egyes attribútumok normalizálása, azaz transzformáljuk őket pl. a [0,1] intervallumba, majd ezen transzformált attribútumok alapján számítsuk a távolságokat (3.1.2 rész).

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 legyen $\sum_{i=1}^m w_i = 1$.

Előfordulhat, hogy olyan attribútummal van dolgunk, amely értékeit nemlineáris léptékben ábrázoljuk (nemlineáris növekedésű attribútumnak szokás hívni ezeket). Például a baktérium populációk növekedését vagy algoritmusok futási idejét exponenciális skálán érdemes ábrázolni. Az ilyen attribútumoknál nem célszerű közvetlenül intervallum alapú hasonlóságot alkalmazni, mert ez óriási különbözőségeket eredményez azokon a helyeken, ahol kis különbözőséget várunk.

Két megközelítés között szokás választani. Egyrészt használhatjuk az intervallum alapú hasonlóságot, de nem az attribútum eredeti értékén, hanem annak logaritmusán. Másrészt diszkretizálhatjuk az értékeket, és vehetjük csak a sorrendet a hasonlóság alapjául.

3.2.5. Vegyes attribútumok

Az előző részekben azt tekintettük át, hogyan definiáljuk a hasonlóságot két elem között adott típusú attribútumok esetén. Mit tegyünk akkor, ha egy objektum leírásánál vegyesen adottak a különböző típusú – intervallum, bináris, kategória stb. – attribútumok? Csoportosítsuk az egyes attribútumokat típusuk szerint, és határozzuk meg a két elem hasonlóságát minden csoportra nézve. A kapott hasonlóságokat képezzük a [0,1] intervallumba. Minden attribútumnak feleltessünk meg egy dimenziót a térben, így két elem hasonlóságához hozzárendelhetünk egy vektort a vektortérben. A hasonlóság 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 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 hasonlóságok nem ragadják meg jól két elem különbözőségét. A teljesség igénye nélkül bemutatunk két olyan esetet, amikor speciális távolságfüggvényre van szükség.

Elemsorozatok hasonlósága

Elemsorozaton egy véges halmazból vett elemek sorozatát értjük. Például a magyar nyelven értelmezett szavak elemsorozatok. Nézzük a $S = \langle abcde \rangle$ 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 hasonlóságot.

Egy elterjedt mérték az elemsorozatok hasonlóságára az ún. *szerkesztési távolság*. Két sorozatnak kicsi a szerkesztési távolsága, ha az egyik sorozatból kevés elem törlésével ill. beszúrásával megkaphatjuk a másikat. Pontosabban, két sorozat szerkesztési távolsága 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.

Bezárt szög alapú hasonlóság

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 távolságokkal. Emlékeztetőül a koszinusz-mérték pontos képlete:

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

3.2.7. Dimenziócsökkentés

Az adatbányászati alkalmazásokban az adathalmaz mérete általában nagy. Felmerül a kérdés, hogy lehet-e ezt a nagy adathalmaz egy kisebb méretűvel helyettesíteni úgy, hogy a kisebb adathalmaz valamilyen szempont szerint hűen reprezentálja a nagy adathalmazt. Természetesen az adatbányászati feladattól függ az, hogy mit jelent pontosan a hű reprezentáció.

Ebben a részben dimenzió-csökkentésről lesz szó, melynek 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 reprezentáló adathalmazt az $m \times n$-es M mátrixszal jellemezzük, az új leírást pedig az $m \times k$-s $\widehat{M}$ mátrixszal. Az n nagyon nagy lehet (az interneten együtt előforduló szópárok keresésénél például 10^9 körüli volt az értéke), 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 algorimusok futtatása, amelyek feltételezik, hogy az adatokat leíró mátrix a gyors elérésű memóriában található.

Két speciális feladatot tárgyalunk. Az elsőben az attribútumok valós számok és két objektum különbözőségén (hasonlóság inverze) 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.

3.2.8. Szinguláris felbontás

A szinguláris felbontás az elméleti szempontból egyik legtöbbet vizsgált, klasszikus lineáris algebrai eszközöket használó dimenzió-csökkentési eljárás. 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 lineáris algebrai programcsomag (Matlab, Octave, Maple) használata javasolt, míg nagyobb adatbázisoknál az adatok sajátosságát kihasználó szinguláris felbontó program (SVDPack) használata ajánlott.

Egy $U \in \mathbb{R}^{n \times n}$ mátrixot *ortogonálisnak* nevezünk, ha oszlopai ortogonális rendszert alkotnak, azaz $U^T U = I_n$, ahol I_n az $n \times n$ méretű egységmátrixot, és U^T az U transzponáltját jelöli. Másképpen mondva U invertálható és U^{-1} inverzére $U^{-1} = U^T$ teljesül. Mátrix ortogonalitásának szemléletes tárgyalásához szükségünk lesz a vektorok hosszának általánosítására, a norma fogalmára. Egy $v \in \mathbb{R}^n$

$$M_{m\times n} = \overbrace{\begin{pmatrix} | & & | \\ u_1 & \dots & u_m \\ | & & | \end{pmatrix}}^{U_{m\times m}} \cdot \overbrace{\begin{pmatrix} \sigma_1 & & & & & \\ & \ddots & & & & \\ & & \sigma_r & & & \\ & & & 0 & & \\ & & & & \ddots & \\ & & & & & 0 \end{pmatrix}}^{\Sigma_{m\times n}} \cdot \overbrace{\begin{pmatrix} — & v_1^T & — \\ & \vdots & \\ — & v_n^T & — \end{pmatrix}}^{V_{n\times n}^T}$$

3.1. ábra. A szinguláris felbontás sematikus vázlata.

vektor $\|v\|_2$-vel jelölt *2-normáját* a $\|v\|_2 = \sqrt{\sum_i v_i^2}$ egyenlőséggel definiáljuk. Egyszerűen látható, hogy $\|v\|_2{}^2 = v^T v$ teljesül. A 2-norma általánosítása a tetszőleges $M \in \mathbb{R}^{m\times n}$ mátrix esetén értelmezett $\|M\|_F$ *Frobenius-norma*, amelynek definíciója $\|M\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n M_{i,j}^2}$.

Visszatérve az ortogonalitás szemléletes jelentésére, egy ortogonális mátrix által reprezentált lineáris transzformációra úgy gondolhatunk, mint egy forgatásra, amely a vektorok hosszát nem változtatja. A 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 bal felső sarokból 45°-ban lefele elhelyezkedő $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > 0$ pozitív számokat csupa 0 követ 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 3.1. ábrán látható.

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*

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

ahol

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

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

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

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

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

$$V_r^T M^T M V_r = {\Sigma_+}^2.$$

Vezessük be az

$$U_r = M V_r {\Sigma_+}^{-1}$$

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 $\text{rang}(M_k) = \text{rang}(\Sigma_k) = k$. Sokkal mélyebb eredmény a következő, melynek bizonyítását mellőzzük.

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*

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

Továbbá a közelítés hibája a σ_i szinguláris értékekkel kifejezhető:

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

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 3.2. tétel alapján felülről becsülhető.

Minhash alapú lenyomat

Vizsgáljuk azt az esetet, amikor az M mátrix bináris és két sor (vektor) hasonlóságát a Jaccard-koefficiens adja meg. Emlékeztetőül:

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

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

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 attribútumok alapján vannak tárolva, azaz először kiolvashatjuk minden objektum első attribútumát, majd minden objektum második attribútumát, 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 8 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 ($r \ll m$), 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 az oszlopokat, majd válasszuk az egyes sorok hash értékének (h) azt a legkisebb indexet, ahol 1-es szerepel. A véletlen permutáció természetesen csak elméleti megközelítés, diszken található nagy adatbázis esetén túl lassú művelet. Ehelyett sorsoljunk ki minden oszlophoz egy véletlen hash értéket. Amennyiben feltehetjük, hogy a mátrix oszlopainak száma 2^{16}-nál kisebb, akkor a születésnapi paradoxon[1] elkerülése miatt válasszunk 32 bit szélességű egyenletes eloszlású véletlen számot. Az algoritmus tényleges implementálása során tehát egyesével olvassuk az oszlopokat, véletlen számot generálunk, és minden sornak folyamatosan frissítjük azt a változóját, ami megadja a legkisebb, 1-est tartalmazó indexet.

Mivel egy lenyomatnak k darab eleme van, ezért minden sorhoz 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 az 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 sorának ℓ-edik elemét jelöli.

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

3.3. észrevétel. *Tetszőleges* (i, j) *sorpárra igaz, hogy*

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

Bizonyítás: Csak akkor lehet a két lenyomat azonos, ha a legalább az egyik oszlopban az 1-est tartalmazó indexek közül olyan index 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:

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

[1] 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 sorokat. Definiáljuk X_l valószínűségi változót, ami 1 $\widehat{M}_{i,\ell} = \widehat{M}_{j,\ell}$ esetén, különben 0. Legyen $X = X_1 + \ldots + X_k$.

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

$$P\big(X > k(E[X] + \varepsilon)\big) \leq e^{-2\varepsilon^2 k}$$

Ebből a bal oldal átalakításával megkapjuk az állítást:

$$P\big(X > k(p+\varepsilon)\big) = P\big(\frac{X}{k} - p > \varepsilon\big) = P\big(\widehat{d_{ij}} - d_{ij} > \varepsilon\big)$$

Hasonlóan megkaphatjuk, hogy $P\big(d_{ij} - \widehat{d_{ij}} > \varepsilon\big) \leq e^{-2\varepsilon^2 k}$, amiből adódik, hogy

$$P\big(|\widehat{d_{ij}} - d_{ij}| > \varepsilon\big) \leq 2e^{-2\varepsilon^2 k}$$

■

4. fejezet

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 el juttatni 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* keresünk, ahol az elemek a termékeknek felel 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.

4.1. 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,).

4.1. 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 $x \preceq z \preceq y$.*

4.2. definíció. *Az $\mathcal{M}\ \mathcal{K} = (\mathcal{M}, \preceq)$ 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 mintakörnyezetet, 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.

4.3. 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 $f(y) \preceq_2 f(x)$.*

4.4. definíció. *A* gyakori minta kinyerésnek feladatában *adott egy $\mathcal{B}$ bemeneti (vagy feldolgozandó) adathalmaz, $\mathcal{M}\ \mathcal{K} = (\mathcal{M}, \preceq)$ 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 a $\langle\{A, D\}, \{A, C\}, \{A, B, C, D\}, \{B\}, \{A, D\}, \{A, B, D\}, \{D\}\rangle$ 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.

4.5. tulajdonság. *Gyakori minta minden részmintája gyakori.*

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 algoritmusnál (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.

4.6. definíció. *Legyen $\preceq$ a H halmazon értelmezett részben rendezés. A $\preceq'$ 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 $\mathcal{M}\ \mathcal{K} = (\mathcal{M}, \preceq)$ 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.

4.7. definíció. *Legyen $\mathcal{M}\ \mathcal{K} = (\mathcal{M}, \preceq)$ 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.

4.1.1. 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 naiv 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.

4.2. 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.

4.2.1. Nem bővíthető és zárt minták

4.8. 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.*

4.9. 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

4.10. tulajdonság. *Minden nem bővíthető minta zárt.*

Mégis mi célt szolgálnak a gyakori zárt minták? Ennek tisztázásához két új fogalmat kell bevezetnünk.

4.11. 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.

4.12. definíció. *Az $\mathcal{M}$ $\mathcal{K} = (\mathcal{M}, \preceq)$ mintakörnyezet a* zártságra nézve egyértelmű, *amennyiben minden $m \in \mathcal{M}$ minta lezártja egyértelmű.*

Látni fogjuk, hogy 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).

4.2.2. 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:

4.13. 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.

4.14. 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.*

4.15. 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.

4.16. definíció. *A* p *predikátum* erősen átalakítható, *amennyiben egyszerre prefix anti-monoton és prefix monoton.*

A 4.1 ábrán látható a kényszerek kapcsolata [115].

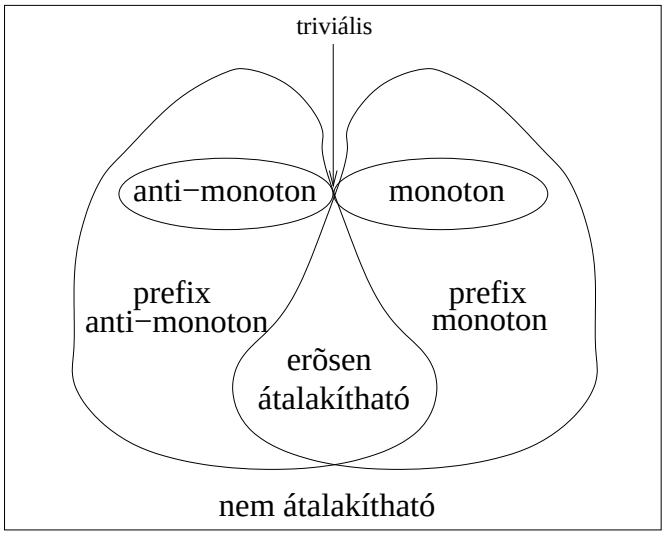
Sejthetjük, hogy az anti-monoton predikátumok lesznek a legegyszerűbben kezelhetők. Ilyen anti-monoton predikátumok például a következők:

– 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.



4.1. ábra. A kényszerek (predikátumok) osztályozása

4.2.3. 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 $min_supp : \mathcal{M} \rightarrow \mathbb{N}$ függvény, és az m akkor gyakori, ha $supp(m) \geq min_supp(m)$.

Többszörös támogatottsági küszöb esetén nem igaz a 4.5 tulajdonság. Hiába nagyobb ugyanis egy részminta támogatottsága, a részmintához tartozó támogatottsági küszöb még nagyobb lehet, és így a részminta nem feltétlenül gyakori.

4.2.4. 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.

4.3. 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.

Szélességi bejárást valósítanak meg azok az algoritmusok[1], 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 megtalálhatóak 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.

4.4. Az APRIORI módszer

Az eredeti APRIORI algoritmust gyakori elemhalmazok kinyerésére használták, és mint az AIS algoritmus [5] továbbfejlesztett változata adták közre. Rakesh Agrawal és Ramakrishnan Srikant [7] publikálták 1993-ban, de tőlük függetlenül, szinte ugyanezt az algoritmust javasolta Heikki Mannila, Hannu Toivonen és A. Inkeri Verkamo [99]-ben. Az 5 szerző végül egyesítette a két írást [6]. Kis módosítással az algoritmust gyakori sorozatok kinyerésére is (APRIORIALL, GSP algoritmusok), sőt, alapelvét bármely típusú gyakori minta (epizód, fa stb.) keresésénél is 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 4.5 tulajdonságot használjuk fel, ami így szólt: „Gyakori minta minden részmintája gyakori.”. Az állítást indirekten nézve elmondhatjuk, hogy egy minta biztosan nem

[1] A szélességi bejárást megvalósító algoritmusokat szintenként haladó (levelwise) algoritmusoknak is hívják.

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 az 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.

```
Bemenet:                B  :  bemeneti adat,
                min_supp  :  támogatottsági küszöb,
Kimenet:               GY  :  gyakori minták,
Átmeneti változók:     Jℓ  :  Az ℓ-elemű jelöltek,

ℓ ← 0;
Jℓ ← az üres minta;
while{ |Jℓ| ≠ 0 }
{
   támogatottság_meghatározás( B, Jℓ );
   for all j ∈ Jℓ do
      if( supp(j) ≥ min_supp )
      then GYℓ ← j;
   Jℓ+1 ← jelölt_előállítás( GYℓ );
   ℓ ← ℓ+1;
}
```

4.2. ábra. Az APRIORI módszer

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.

4.4.1. 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 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 $\subseteq$ 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.

```
Bemenet:                    GY_{ℓ-1}  : (ℓ-1)-elemű gyakori minták

Kimenet:                        J_ℓ  :  Az ℓ-elemű jelöltek,
Átmeneti változók:              Ĵ  : potenciális jelöltek halmaza

for all gy ∈ GY_{ℓ-1} do
  for all gy' ∈ GY_{ℓ-1}, gy ⪯ gy' do
    if( gy és gy' illeszthető)
    {
         Ĵ = minimális_valódi_felső_korlát(gy, gy');
         for all ĵ ∈ Ĵ
            if( minden_részhalmaz_gyakori(ĵ, GY_{ℓ-1}) )
               J_ℓ ← ĵ;
    }
```

4.3. ábra. Jelöltek előállítása

4.4.2. Zárt minták kinyerése, az APRIORI-CLOSE algoritmus

A zárt minták jelentőségét a 4.2.1 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 [112–114] algoritmust, amely már eleve csak a zárt mintákat határozza meg. Mindkét módszerhez az alábbi észrevételt használjuk fel:

4.17. 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.

4.5. 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[2]. Az illeszkedést egy illeszkedési predikátummal adhjuk 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

[2] Ha csak a matematikai defi níciókat 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.

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.

4.5.1. 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 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 [23] 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 az 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 fordulnak 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 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.

4.5.2. Zaki módszere

Zaki módszere [167] szintén jelölteket használ a keresési tér bejárásához, de a bejárás típusa – szemben az APRIORI-val – mélységi. A $\mathcal{M}\ \mathcal{K} = (\mathcal{M}, \preceq)$ 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 $I' \prec \prec {}'I''$, 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:

I. Az illeszkedési listából könnyen megkapható a támogatottság.

II. 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.

```
Bemenet:                B  :  bemeneti adat,
                 min_supp  :  támogatottsági küszöb,
Kimenet:               GY  :  gyakori minták,
Átmeneti változók:      J  :  jelöltek,
                   ILL(J)  :  jelöltek illeszkedési listája,


J ← 1-elemű minták;
ILL(J) ← ILL_felépítés( B, J );
for all j ∈ J do
  if( |ILL(j)| ≥ min_supp )
  then GY_1 ← j;
zaki_segéd( GY, ILL(GY), min_supp );      // GY = [∅]^+
```

4.4. ábra. Zaki módszere

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 a 4.5 ábrán látható.

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. Az 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

```
Bemenet:                  [P]^+ :  P prefixű, P-nél eggyel nagyobb gyakori minták
                 ILL([P]^+) :  [P]^+-beli minták illeszkedési lisátja,
                  min_supp :  támogatottsági küszöb,
Kimenet:                 GY :  P prefixű összes gyakori minta,
Átmeneti változók:        J :  jelöltek,
                     ILL(J) :  jelöltek illeszkedési listája,

for all m ∈ [P]^+ do
{
   for all m' ∈ [P]^+, m ≺ m' do
   {
      J, ILL(J) ← minimális_valódi_felső_korlát( m, m', ILL(m, m') );
      for all j ∈ J do
         if( |ILL(j)| ≥ min_supp )
         then GY' ← j;
   }
   zaki_segéd( GY', ILL(GY'), min_supp );
   GY ← GY ∪ GY';
}
```

4.5. ábra. zaki_segéd eljárás

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 az 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.

4.5.3. 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, értel-

mezhető 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).

4.18. definíció. *Az $\mathcal{M}$ $\mathcal{K} = (\mathcal{M}, \preceq)$ 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 $\mathcal{M} \setminus \overline{m}$-el azt a mintateret, amit úgy kapunk $\mathcal{M}$-ből, hogy töröljük azon mintákat, amelynek m részmintája ($\overline{m}$). Ha az m minta támogatottsága $\mathcal{S}$-ben $supp_{\mathcal{S}}(m)$ és az $m' \in \mathcal{M} \setminus \overline{m}$ támogatottsága $\mathcal{S}|m$-ben $supp_{\mathcal{S}|m}(m')$, akkor $m + m'$ támogatottsága is $supp_{\mathcal{S}|m}(m')$. A módszer pszeudokódja a 4.6 ábrán látható.

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.

```
Bemenet:                B  :  bemeneti adat,
                  min_supp  :  támogatottsági küszöb,
                         M  :  mintatér,
Kimenet:                GY  :  gyakori minták,
Átmeneti változók:      J1  :  egyelemű jelöltek,

J1 ← 1-elemű minták;
támogatottság_meghatározás( B, J1 );
GY1 ← gyakoriak_kiválogatása( J1, min_supp );
B̃ ← szűrés(B);

forall gy ∈ GY^1
{
  GY' ← Mintanövelő módszer(B̃|gy, min_supp, M\ḡy);
  forall gy' ∈ GY'
     GY ← gy + gy';
}
```

4.6. ábra. Mintanövelő módszer

4.5.4. 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 [126, 150] 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 79.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 [126] 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 naiv mintavételező algoritmus nagy hátránya, hogy még csökkentett min_freq mellett sem lehetünk biztosak abban, hogy nem vesztettünk el gyakori mintát. Toivonen algoritmusa [150] az adat-

bázist egyszer olvassa végig, és ha jelenti, hogy minden mintát megtalál, akkor bizonyítható, hogy ez igaz. Az algoritmus nem más, mint az naiv 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 $m \in \in MVFK(M)$)? 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.

4.19. 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 $GY' \cup \cup MVFK(GY')$-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 {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.

4.5.5. A zárt minták „törékenysége”

A zárt mintákon alapuló memóriacsökkentés tagadhatatlanul 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.

4.5.6. 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 [13, 30, 31, 108, 125, 145], ami felhasználja az adatbázis előző állapotára vonatkozó információkat és így gyorsabban ad eredményt, mint egy nulláról induló, hagyományos algoritmus.

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) [30] 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 adatbázis-

ban mindkét csoport elemeinek támogatottságát meg kell határozni, a régi adatbázisban azonban elég a második csoport elemeit vizsgálni.

A FUP az alábbi trivialitásokat használja fel.

I. 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.

II. 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 a 4.7-es ábrán látható.

```
Bemenet:                 S  :  régi bemeneti adat,
                         S' :  új bemeneti adat,
                      GY^S  :  régi gyakori minták,
                 min_freq   :  gyakorisági küszöb,
Kimenet:          GY^<S, S'> :  gyakori minták,
Átmeneti változók:       J^1 :  1-es csoportbeli jelöltek,
                         J^2 :  2-es csoportbeli jelöltek,

ℓ ← 0;
J^1_ℓ ← GY^S_ℓ;
J^2_ℓ ← {üres minta}\ GY^S_ℓ;
while{ |J^1_ℓ|+|J^2_ℓ| ≠ 0 }
{
   támogatottság_meghatározás( S', J^1_ℓ ∪ J^2_ℓ );
   J*_ℓ ← gyakoriak_kiválogatása( J^2_ℓ, min_freq );
   if (|J*_ℓ| ≠ 0) támogatottság_meghatározás( S, J*_ℓ );
   GY_ℓ ← gyakoriak_kiválogatása( J^1_ℓ ∪ J*_ℓ, min_freq );
   J**_ℓ+1 ← jelölt_előállítás( GY_ℓ );
   ℓ ← ℓ+1;
   J^1_ℓ ← J**_ℓ ∩ GY^S_ℓ;
   J^2_ℓ ← J**_ℓ \ GY^S_ℓ;
   delete(J**_ℓ);
}
```

4.7. ábra. FUP algoritmus

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 [31]).

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

A [145] cikkben Toivonen algoritmusában használt minimális valódi felső korlátokat használják annak érdekében, hogy csökkentsék a nagy adatbázist átolvasásának számát. 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. A 4.19-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. fejezet

Gyakori elemhalmazok

Gyakori mintákat először elemhalmazok formájában kerestek. Szupermarketek vásárlási adatait kellett elemezni, abból a célból, hogy a profitot növelni tudják. A nagy profitot elsősorban a gyakran vásárolt termékek, termékhalmazok jelentik, így ezek kinyerése jelentette az első lépést a feladat megoldásához.

A gyakori elemhalmazok bányászata nagyon népszerű kutatási terület lett. A publikált algoritmusokkal könyveket lehetne megtölteni. Ebben a jegyzetben csak a leghíresebb algoritmusokat és ötleteket ismertetjük.

5.1. A gyakori elemhalmaz fogalma

A fejezet során végig feltételezzük, hogy az olvasó tisztában van az előző fejezetben definiált fogalmakkal és az ismertetett algoritmusokkal. A gyakori elemhalmazok bányászata a gyakori mintabányászat azon népszerű családjába tartozik, amelynél a bemenet minták sorozataként adott és a támogatottságot tartalmazás szerinti illeszkedési predikátum alapján definiáljuk, vagyis az m minta illeszkedik a bemeneti mintára, ha az m-nek része. Tulajdonképpen a feladatot pontosan leírjuk, ha megadjuk a mintakörnyezetet.

Legyen $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ elemek (items) halmaza. A bemenet ($\mathcal{T}$) az $\mathcal{I}$ hatványhalmaza felett értelmezett sorozat ($\mathcal{T} = \langle t_1, \ldots, t_n \rangle$, ahol $t_j \subseteq \mathcal{I}$), a mintakörnyezet pedig $\mathcal{M}\,\mathcal{K} = (2^{\mathcal{I}}, \subseteq)$, ahol $\subseteq$ a hagyományos részhalmaz relációt jelöli. Ebben a mintakörnyezetben a $|\ |$ függvény a halmaz elemeinek számát adja meg. A $\mathcal{T}$ elemeit *tranzakcióknak*, vagy *kosaraknak* hívjuk. 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. Azt mondjuk, hogy az I elemhalmazt *tartalmazza* a t tranzakció, ha $I \subseteq t$. Ezek alapján a támogatottság és a gyakoriság definíciója egyértelmű. Például a $\langle ACD, BCE, ABCE, BE, ABCE \rangle$ sorozatban $supp(AC) = 3$, $freq(BE) = 4/5$.

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 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

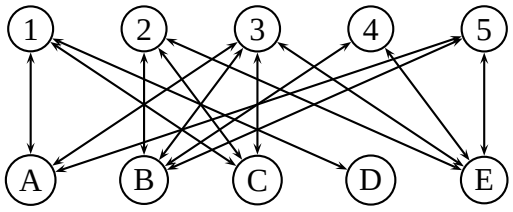
tranzakció	elem
1	C
2	A
2	B
2	C

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.

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,

5.2 ábrán látható.



5.1. ábra. Gráfos ábrázolási mód

	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, 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. Gondoljuk meg például, hogy egy 10^7 tranzakciót tartalmazó adatbázis csak 120 MB helyet kíván, amennyiben a tranzakciók átlagos mérete 6 elem. Legtöbbször tehát

alkalmazhatók azok az algoritmusok, amelyek feltételezik, hogy a bemenet (vagy a szűrt tranzakciók) elférnek a memóriában.

Mielőtt bemutatjuk az APRIORI módszer 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 eddig feldolgozott tranzakcióban 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 IO 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 az életben előforduló 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 a memóriából, amennyiben a számlálók 4 byte-osak. A számlálókat mindenképpen a memóriában szeretnénk tartani, hogy elkerüljük a folyamatos swappelést, 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.

A fenti algoritmus kis módosítását javasolták [11]-ban. 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 egyszerű algoritmusokat sem kell megprogramoznunk, mert egy teljesen 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
```

5.3. ábra. SQL utasítás gyakori elempárok kinyeréséhez

Látnunk kell, hogy a fenti parancs az összekapcsolás (FROM mezőben két tábla) művelet miatt nem fog működni, ha az adatbázis mérete túl nagy.

5.2. Az APRIORI algoritmus

Az APRIORI algoritmus az egyik legelső gyakori elemhalmazokat kinyerő algoritmus. Az általános módszert már ismertettük 4.4 részben, most már csak azt kell megnéznünk, hogy a három főeljárást ((1) jelölt-előállítás, (2) támogatottság meghatározása, (3) ritka minták törlése) hogyan kell elvégezni elemhalmazok esetében.

5.2.1. Jelöltek előállítása

Emlékeztetőül: egy elemhalmaz akkor lesz jelölt, ha minden részhalmaza gyakori. A jelölt-előállítás két részre oszlik.

Az első részben előállítunk $(\ell-1)$-elemű gyakori elemhalmaz párokból – mint generátorokból – ℓ-elemű potenciális jelölteket. A potenciális jelöltek a generátor párok elemeinek minimális valódi felső korlátja. Elemhalmazok esetében a minimális valódi felső korlátot megkaphatjuk, ha a két generátor unióját képezzük.

Két különböző generátorpár elemeinek lehet azonos az uniójuk, így ez a módszer még nem ismétlés nélküli. Könnyen azzá tehetjük, ha definiálunk az elemek halmazán egy teljes rendezést. A $\subseteq$ rendezés egy mérettartó lineáris kiterjesztését kaphatjuk, ha az azonos méretű elemhalmazok között a teljes rendezés szerinti lexikografikus rendezés alapján definiálunk egy sorrendet. Ezt a teljes rendezést felhasználva ismétlés nélküli jelölt-előállítást kaphatunk: csak azokat a párokat tekintsünk generátoroknak, amelyek elemeinek közös az $(\ell-2)$-elemű prefixük. Ez azt jelenti, hogy csak akkor vesszük két elemhalmaz unióját ha csak a legnagyobb elemükben térnek el. Könnyű belátni, hogy ez a jelölt-előállítás ismétlés nélküli és teljes, azaz minden minimális valódi felső korlátot előállít.

A második részben az előállított elemhalmaz mind az ℓ darab $(\ell-1)$-elemű részhalmazát ellenőrizzük, hogy gyakori-e. Csak akkor lesz az elemhalmaz jelölt, ha nincs ritka részhalmaza.

Példaként nézzük meg, hogy milyen 4-elemű jelölteket állít elő az APRIORI algoritmus. Legyenek a 3-elemű gyakori elemhalmazok a következők:

$$GY_3 = \{ABC, ABD, ACD, ACE, BCD\}$$

amiből egyetlen jelöltet hozunk létre: $J_4 = \{ABCD\}$. Nyilvánvaló, hogy ACD és ACE uniója nem lesz jelölt, hisz van olyan részhalmaza (ADE), ami nem gyakori (Érdemes végiggondolni, hogy az APRIORI algoritmus elődje, az AIS algoritmus [1]mennyi jelöltnek foglalt volna le helyet a memóriában, ha egy olyan kosárral találkozik, amely mind az 5 elemet tartalmazza.).

5.2.2. Jelöltek támogatottságának meghatározása

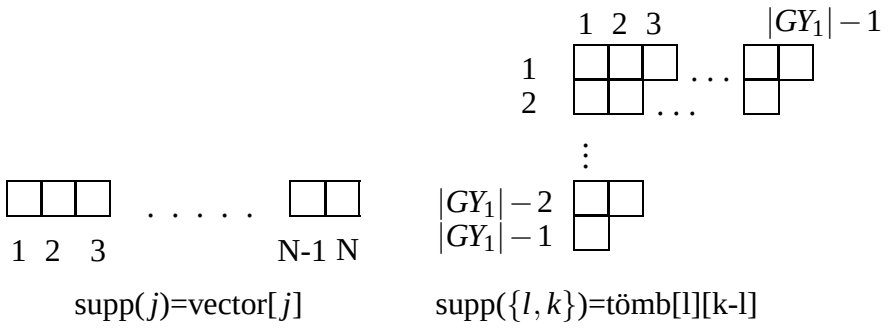
A jelöltek előfordulásait össze kell számolni. Ehhez egyesével vizsgáljuk a kosarakat, és azon jelöltek számlálóit növeljük eggyel, amelyeket tartalmaz a kosár.

[1]Az AIS algoritmusban az adatbázis ℓ-edik végigolvasása során (tehát amikor ℓ méretű gyakori elemhalmazokat keresünk) állítjuk elő, illetve számoljuk meg az ℓ méretű jelölteket. Amennyiben egy kosár tartalmaz $\ell-1$ méretű gyakori elemhalmazt, akkor felvesszük jelöltnek az összes olyan 1 termékkel bővített változatát, ahol az új termék eleme a kosárnak. Ha már korábban felvettünk ilyen jelöltet, akkor növeljük annak számlálóját.

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 (emlékeztetőül GY_j-vel jelöljük a j-elemű gyakori mintákat). Az l és k-adik elemekből álló pár támogatottságát a tömb l-edik sorának $k-l$-adik eleme tárolja (az általánosság megsértése nélkül feltehetjük, hogy $l < k$).



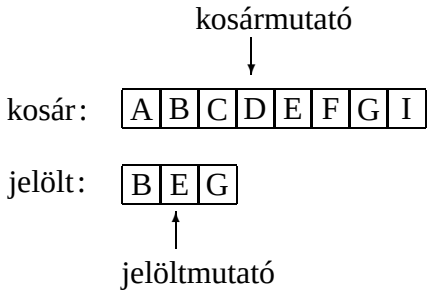
5.4. ábra. Adatstruktúrák az 1- és 2-elemű jelöltek támogatottságának meghatározásához.

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 [109]. A párok támogatottságának meghatározása kevesebb memóriát fog igényelni, de ezzel együtt lassabb is lesz.

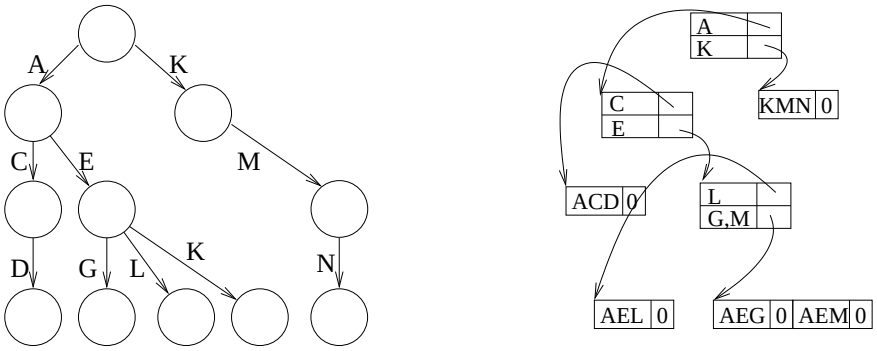
Nagyobb elemhalmazok támogatottsága

A kosarakban található jelölthalmazok meghatározásának egyszerű módja, ha minden egyes jelölthalmazra meghatározzuk, hogy tartalmazza-e azt a kosár. Rendezett halmazban rendezett részhalmaz keresése elemi feladat (szimultán bejárás):

Vegyünk fel két mutatót, amelyek a kosár, illetve a jelölt elemein fognak végighaladni. Kezdetben mutasson mindkét mutató a 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, ha sok jelölt van, akkor lassú, hiszen annyiszor kell a kosár elemein végighaladni, amennyi a jelöltek száma. A gyorsabb működés érdekében az egyes jelölteket szófában (trie) vagy hash-fában (hash-tree) célszerű tárolni. A szófát szokás prefix-fának vagy lexikografikus fának is hívni [3]. 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 [106]-ban írtak, ahol a szófát alkalmazó APRIORI algoritmust SEAR-nek nevezték el. A továbbiakban a szófában való keresést ismertetjük (a szófák felépítéséről a 2.7.1 részben már szóltunk). A következő ábrán egy szófát és egy hash-fát láthatunk, melyek az ACD, AEG, AEM, AEL, KMN elemhalmazokat tárolják.



5.5. ábra. Ugyanazon jelölteket tartalmazó szófa és hash-fa

A k-elemű tranzakcióban a ℓ-elemű jelölteket úgy találjuk meg, hogy a gyökérből kiindulva bejárjuk a fa bizonyos részfáit. Ha egy d szintű belső ponthoz a kosár j-edik elemén keresztül jutunk, akkor a kosár olyan j' elemein keresztül lépünk eggyel mélyebb szintre, amelyekre fennáll, hogy $j < j' \leq k - \ell + d$ (ugyanis $\ell - d$ elemre szükségünk lesz még). Azon éleken haladunk lejjebb, amelyek címkéi megegyeznek a kosár valamely j' elemével. Ez a rekurziós lépés két rendezett listában található közös elemek meghatározását jelenti. Ha eljutunk egy ℓ szintű ponthoz, az azt jelenti, hogy a pont által reprezentált elemhalmazt a vizsgált kosár tartalmazza, így ennek a pontnak az előfordulási mutatóját kell növelnünk eggyel.

A szófa nagy előnye a gyors támogatottság-meghatározás mellett, hogy a jelölteket is hatékonyan lehet előállítani. Tudjuk, hogy két gyakori elemhalmazból akkor állítunk elő jelöltet, ha a legnagyobb sorszámú elemük elhagyásával ugyanazt az elemhalmazt kapjuk, vagy más szavakkal, a két gyakori elemhalmaz prefixe megegyezik. Ha a gyakori elemhalmazokat (vagy legalábbis a legnagyobb méretű gyakori elemhalmazokat) szófában tároljuk, akkor ez azt jelenti, hogy csak közös szülővel rendelkező (azaz testvér) pontok generálhatnak jelöltet reprezentáló új pontokat.

A **??** részben már volt arról szó, hogy az elemeken definiált rendezés milyen hatással van a szófa alakjára. 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 szófa hatékony megvalósításának részleteit és további gyorsítási ötleteket a [19, 21, 51] írások-

ban találhatunk. Egy – kutatási célokra szabadon felhasználható – szófa alapú APRIORI implementáció letölthető a

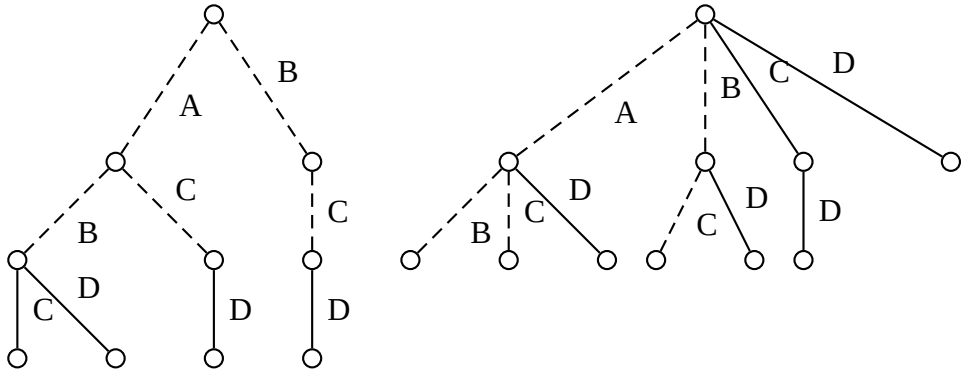
`http://www.cs.bme.hu/~bodon/en/apriori`

oldalról.

5.2.3. A gyakori elemhalmazok tárolása

A jelöltek előállításának második lépésében a potenciális jelölt minden eggyel kisebb méretű részhalmazáról meg kell tudnunk állapítani, hogy gyakori-e. A hatékonyság miatt célszerű a gyakori elemhalmazokat olyan adatstruktúrában tárolni, amellyel ez gyorsan megtehető. A szófa megfelel a célnak, főleg, ha arra gondolunk, hogy a gyakori elemhalmazokban nagyon sok a közös elem, sőt egy gyakori elemhalmaz minden részhalmaza gyakori. A szófa a közös prefixeket csak egyszer tárolja.

Memória kihasználtság szempontjából nem túl jó megoldás, ha a jelölteknek és a gyakori elemhalmazokat is külön szófában tároljuk. Ennek illusztrálásához tekintsük a következő ábrát.



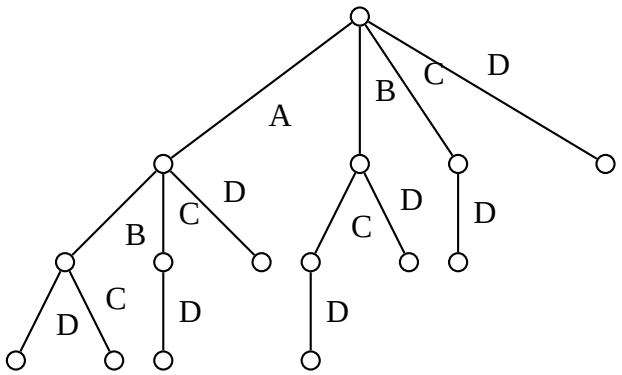
5.6. ábra. Példa: jelölteket és a gyakori elemhalmazokat tároló szófák

A két szófának van egy nagy közös része (azok az utak, amelyek a jelöltek eggyel kisebb prefixeit reprezentálják; az ábrán szaggatott vonallal jelöltük), így memóriát spórolhatunk meg, ha a két szófát összevonjuk. Az összevont szófát mutatja a következő ábra.

A fenti szófa minden információt tartalmaz, de memóriaigénye jóval kevesebb, mint a két szófa memóriaigényének az összege (az első két szófának összesen 21, míg a harmadiknak 15 csúcsa van). Hátránya, hogy – ebben a formában – a jelöltek támogatottságának meghatározásakor felesleges utakat járunk be. Ha például a kosár tartalmazza a C, D elemeket, akkor a gyökérből tovább fogunk lépni a C és D éleken keresztül, annak ellenére, hogy ezek nem vezetnek jelöltekhez, hiszen nem vezetnek olyan ponthoz, amelyek mélysége 3.

A problémát könnyen orvosolhatjuk, ha minden ponthoz tároljuk a pontból kiinduló leghosszabb út hosszát. Az ℓ-elemű jelöltek támogatottságának meghatározása során csak akkor lépünk egy d mélységi pontba, ha a pontból indul ki $\ell - d$ hosszú út.

Ennél még jobb megoldás, ha az éleket megkülönböztetjük. Legyenek például szaggatottak azok az élek, amelyek jelölthöz vezető úton vannak; a többi pedig legyen normális vonal. A csúcsokhoz tartozó éleket tároljuk külön listákban, és csak a szaggatott éleket nézzük a jelöltek támogatottságának meghatározása során.



5.7. ábra. Példa: jelölteket és a gyakori elemhalmazokat tároló szófa

5.2.4. 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.

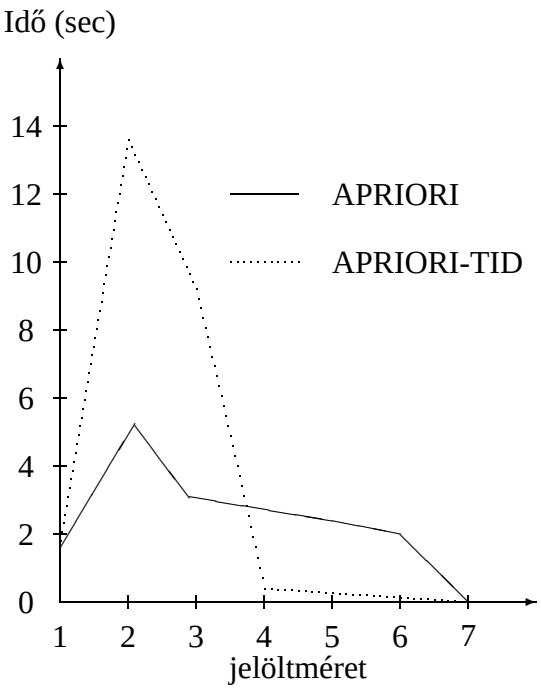
A szűrt tranzakciókat célszerű olyan adatstruktúrában tárolni, amit gyorsan fel lehet építeni (azaz gyorsan tudjuk beszúrni a szűrt tranzakciókat) és az gyorsan végig tudunk menni a beszúrt elemeken. Alkalmazhatunk erre a célra egy szófát, de tesztek azt mutatják, hogy egy piros-fekete fa (kiegyensúlyozott bináris fa), amelynek csúcsaiban egy-egy szűrt tranzakció található, még jobb megoldás, mert jóval kisebb a memóriaigénye.

5.2.5. Utolsó fázisok gyorsítása: APRIORI-TID és APRIORI-HYBRID algoritmusok

Az APRIORI-TID [7] a SETM [66] algoritmus továbbfejlesztett változata. Főbb különbség az APRIORI-hoz képest, hogy a támogatottság meghatározásánál nem a kosarakat használja, hanem egy segédtáblát. Az ℓ-edik lépésben rendelkezésünkre álló segédtábla tárolja a tranzakciók azonosítóit és az egyes tranzakciókban található ℓ-elemű jelölteket. Kis ℓ-eknél ez a segédtábla jóval nagyobb lehet, mint az eredeti adatbázis, de nagy ℓ-eknél gyorsabb támogatottság-meghatározást várhatunk. A kételemű jelöltekhez tartozó segédtáblát a bemeneti adatból és a gyakori elemből állítjuk elő, a 2-nél

nagyobb méretű jelöltekhez tartozó segédtáblát az előző segédtáblából és az utoljára meghatározott gyakori elemekből kapjuk. Természetesen új segédtábla előállítása után a régit törölhetjük, hiszen arra többé nem lesz szükség.

A jobb oldali ábra mutatja, hogy egy példaadatbázisban mennyi idő alatt találták meg a különböző méretű gyakori elemhalmazokat az APRIORI illetve az APRIORI-TID algoritmusok. Látható, hogy a kezdeti fázisokban (1, 2, 3-elemű gyakori elemhalmazok megtalálásában) az APRIORI gyorsabb, míg a későbbiekben több nagyságrenddel az APRIORI-TID. A két módszer ötvözése az ún. APRIORI-HIBRID [7] algoritmus, amely a kezdeti lépésekben az APRIORI-t a későbbiekben az APRIORI-TID-et használja.



Nagy kérdés az APRIORI-HIBRID algoritmussal kapcsolatban, hogy hol váltsunk át az egyik algoritmusról a másikra. Léteznek különböző heurisztikák és becslések arra nézve, hogy mekkora elemszámú jelölteknél váltsunk, de egyik sem garantálja, hogy a váltás nem történik idő előtt, azaz nem futunk ki a memóriából.

5.2.6. Futási idő és memóriaigény

A GYEK 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 előállításához 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.

Az algoritmus 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. 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.

5.2. tétel. *Ha*

$$|GY_\ell| = \binom{m_\ell}{\ell} + \binom{m_{\ell-1}}{\ell-1} + \cdots + \binom{m_r}{r}$$

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 $s = r-1$.

A fenti tétel a Kruskal–Katona tétel következménye, ezért a tételben szereplő felső korlátot a továbbiakban $KK_\ell^{\ell+p}(|GY_\ell|)$-el jelöljük.

5.3. tétel. *A 5.2. 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.

5.4. 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:

5.5. 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.

5.6. következmény. *Az összes lehetséges ℓ-nél nagyobb méretű jelölt száma legfeljebb*

$$KK_\ell^{\text{összes}}(|GY_\ell|) = \sum_{p=1}^{\mu_\ell(|GY_\ell|)} KK_\ell^{\ell+p}(|GY_\ell|).$$

[2] A H-t az $\mathcal{I}$ feletti *halmazcsaládnak* nevezzük, amennyiben $H \subseteq 2^{\mathcal{I}}$.

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 5.2. 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 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:

$$KK^*_{\ell,p}(GY_\ell) = \begin{cases} \binom{|GY_\ell|}{p+1} & \text{, ha } \ell = 1 \\ \min\{KK_\ell^{\ell+p}(|GY_\ell|), \sum_{i\in I} KK^*_{\ell-1,p}(GY_\ell^i)\} & \text{, ha } \ell > 1. \end{cases}$$

A definícióból következi, hogy $KK^*_{\ell,p}(GY_\ell) \le KK_\ell^{\ell+p}(|GY_\ell|)$, továbbá

5.7. tétel. $|J_{\ell+p}(GY_\ell)| \le KK^*_{\ell,p}(GY_\ell)$.

Bizonyítás: A bizonyítás teljes indukción alapul, az $\ell = 1$ eset triviális. Tulajdonképpen csak az kell belátni, hogy

$$|J_{\ell+p}(GY_\ell)| \le \sum_{i\in I} KK^*_{\ell-1,p}(GY_\ell^i)$$

Az egyszerűség kedvéért vezessük be a következő jelölést: $H \cup i = \{I \cup \{i\} | I \in H\}$, ahol H egy I feletti halmazcsalád. Vegyük észre, hogy $GY_\ell = \sum_{i\in I} GY_\ell^i \cup i$ és $GY_\ell^i \cap GY_\ell^j = \emptyset$ minden $i \ne j$ 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

$$J_{\ell+p}(GY_\ell) \subseteq \bigcup_{i\in I} J_{\ell-1+p}(GY_\ell^i) \cup i.$$

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:

$$\begin{aligned} |J_{\ell+p}(GY_\ell)| &\le |\bigcup_{i\in I} J_{\ell-1+p}(GY_\ell^i) \cup i| \\ &= \sum_{i\in I} |J_{\ell-1+p}(GY_\ell^i) \cup i| \\ &= \sum_{i\in I} |J_{\ell-1+p}(GY_\ell^i)| \\ &\le \sum_{i\in I} KK^*_{\ell-1,p}(GY_\ell^i), \end{aligned}$$

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

$$KK_{\text{összes}}^*(GY_\ell) = \sum_{p=1}^{\mu_\ell^*(GY_\ell)} KK_{\ell+p}^*(GY_\ell).$$

5.8. következmény. *$maxsize(GY_\ell) \leq \mu_\ell^*(GY_\ell) \leq \mu_\ell(|GY_\ell|)$.*

5.9. 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.

5.2.7. Kételemű jelöltek számának csökkentése: a DHP algoritmus

Ha pusztán matematikai szemmel nézzük a gyakori elemhalmazokat, akkor elmondhatjuk, hogy azok száma exponenciálisan nőhet (pontosabban a k- eleműek száma $\binom{m}{k}$ lehet, ahol m az elemek száma). Akár a teljes elemhalmaz is lehet gyakori, ami azt jelentené, hogy az APRIORI algoritmus ennek mind a 2^m részhalmazát megtalálná, ami az elemek számával exponenciálisan növekvő futási időt és tárolási igényt jelentene.

Nem kívánunk magunknak feldolgozhatatlan adatmennyiségeket előállítani, ezért a támogatottsági küszöböt úgy szokás beállítani, hogy a maximális méretű gyakori elemhalmaz ne legyen 10-15-nél nagyobb. A gyakorlat azt mutatja, hogy az értelmes beállítások mellett a kételemű (ritkán a három-, négyelemű) jelöltek száma lesz a legnagyobb, és ahogyan nő az elemhalmazok mérete, úgy csökken az ilyen méretű gyakori elemhalmazok száma.

A fenti ábra jól mutatja, hogy mind az APRIORI, mind az APRIORI-TID algoritmusok a gyakori elempárok megtalálásával töltik a legtöbb időt. Több különböző adatbázisra lefuttatott tesztek

is hasonló karakterisztikát mutattak. Ennek oka az, hogy általában a gyakori elemek száma nagy, és gyakori elempárok száma az elméleti maximumnál ($\binom{|Gy_1|}{2}$) jóval kevesebb. Így a kételemű jelöltek rengetegen, a három eleműek pedig kevesen vannak. A gyakori elemek nagy száma miatt sok hamis kételemű jelölt áll elő. A DHP algoritmus célja, hogy csökkentse a hamis jelöltek számát. A technikáját bármilyen nagyságú hamis jelöltek kizárására alkalmazhatjuk, a legnagyobb sebességnövekedést várhatóan a kételeműeknél hozza. Sőt nagyobb elemszámú jelölteknél a hamis jelöltek kizárására fordított erőforrások egyáltalán nem biztos, hogy megtérülnek.

Mi történik az adatbázis első végigolvasása során az APRIORI algoritmusnál? Összeszámoljuk az egyes elemek előfordulásait (kezdetben minden elem jelölt). Például tízezer jelölt esetén $10000 \cdot 4$ byte-t fogyasztunk el a memóriából (ha feltesszük, hogy egy elem sem található meg több mint $2^{32} \approx$ ≈ 4.2 milliárd tranzakcióban). A mai memóriák mérete mellett ez nem túl sok, így felmerül a kérdés, hogy a szabad memóriát fel tudnánk-e használni újabb hamis jelöltek kizárására.

A DHP (Direct Hashing and Pruning) algoritmus [111] az APRIORI továbbfejlesztése. Ötlete az, hogy miközben az elemek előfordulását számoljuk, gyűjtsünk információt az elempárokról is. Előfordulhat, hogy ezen információ hamis kételemű jelölteket szűrhetünk ki, és nem veszünk fel a jelöltek közé, annak ellenére, hogy mindkét elemük gyakori. Készítsünk hát egy hash-táblát, és hash-eljük bele az összes elempárt, ami előfordul valamely tranzakcióban. Egy elempár hash-elésénél növeljük annak a vödörnek a számlálóját, amibe hash-eltük. Ugyanabba a vödörbe természetesen több különböző elempárt is hash-elhetünk. Magukat a vödröket ne tároljuk a memóriában: elég a számlálókat tartalmazó vektort felvennünk. A vektor i-edik eleme fogja megadni az i hash-értékkel rendelkező vödör számlálóját. Könnyen látható, hogy az i-edik számláló értéke meg fog egyezni azon elempárok támogatottságának összegével, amelyek hash-értéke i. Azaz egy elempár nem lehet gyakori, ha azt olyan vödörbe hash-eltük, amelynek számlálója kisebb min_supp-nál.

Ezek szerint egy elempár az adatbázis első végigolvasása után akkor lesz jelölt, ha egyrészt mindkét eleme gyakori, másrészt gyakori vödörbe hash-eltük. Az adatbázis végigolvasása után nincs szükségünk a vödrök számlálóira, csak arra az információra, hogy melyik vödör gyakori. Éppen ezért helyettesítsük a hash-táblát egy bittérképpel, aminek i-edik pozíciójában 1-es áll, ha az i-edik vödör gyakori, és 0, ha nem. Ebből a bittérképből, továbbá a gyakori ℓ-elemű elemhalmazokból már elő tudjuk állítani az $(\ell+1)$-elemű jelölteket.

Minél nagyobb a hash-tábla, annál kevesebb az ütközés, és annál kevesebb az esélye, hogy egy ritka elemhalmaz azonos vödörbe kerül egy gyakorival (vagy sok ritka egy vödörbe kerül, aminek következtében a vödör gyakori lesz), tehát annál kevesebb hamis jelölt lesz. A hash-tábla azonban nem lehet túl nagy, hiszen a hash-táblának, a jelöltek és azok számlálóinak el kell férniük a memóriában.

Az 5.8 ábra a DHP algoritmus működésének első fázisára mutat példát (minimális támogatottsági küszöbnek 0,5-öt adtunk meg). Érdemes összehasonlítanunk a DHP által előállított kételemű jelölteket az APRIORI algoritmus által létrehozott jelöltekkel. Míg a DHP csak 4 jelöltet (amelyek között egy hamis jelölt sincs), addig az APRIORI algoritmus 6-ot állít elő.

Abban az esetben is használják a DHP ötletét, amikor a gyakori elemek száma olyan nagy, hogy a kételemű jelöltek nem férnek el a memóriában. Ekkor az adatbázis első végigolvasása után nem állítjuk elő a kételemű jelölteket, hanem inkább újra végigolvassuk az adatbázist és egy másik hash függvényt alkalmazó hash táblát készítünk. Mivel a gyakori elemeket ismerjük, ezért elég a gyakori elemekből álló párokat hash-elni. Az új hash táblával újabb hamis jelölteket szűrhetünk ki. Egy elempár abban az esetben lesz jelölt, ha a második hash-elésnél is gyakori vödörbe került.

Adatbázis

k_id	elemek
100	A C D
200	B C E
300	A B C E
400	B E

Elemek sorszámai

Elem	sorszám
A	1
B	2
C	3
D	4
E	5

GY_1

elem halmaz	támogatottság
{A}	2
{B}	3
{C}	3
{E}	3

H_1 hash-tábla h(i_1,i_2)=(10 · sorszám(i_1)+ sorszám(i_2)) mod 7

elempárok:	{C,E}						{A,C}
	{A,D}	{A,E}	{B,C}		{B,E}	{A,B}	{C,D}
számláló:	3	1	2	0	3	1	3
hash-érték:	0	1	2	3	4	5	6

J_2

Jelöltek
{A,C}
{B,C}
{B,E}
{C,E}

5.8. ábra. Példa DHP algoritmusra

5.3. Az ECLAT algoritmus

Az ECLAT algoritmus [167] Zaki módszerét követi, amit a 4.5.2 részben ismertettük. Elmondtuk, hogy a jelöltek előállításának módszere megegyezik az APRIORI jelölt-előállítási módszerének első lépésével, amit szintén részleteztünk már ebben a fejezetben (5.2.1 rész).

Egy dologgal vagyunk adósak, mégpedig az illeszkedési lista definíciójával. Elemhalmazok esetében az illeszkedési listát *TID-listának* nevezzük. Az I halmaz TID-listája megegyezik azon bemeneti elemek sorszámának halmazával, amelyek illeszkednek I-re[3]. Az illeszkedési listával kapcsolatos két elvárásunk teljesül. Ugyanis (1.) egy elemhalmaz támogatottsága megegyezik TID-listájának elemszámával, továbbá (2.) egy jelölt TID-listáját megadja generátorai TID-listájának metszete. Ha például az A elem TID-listája {2, 5, 7, 13, 14}, a C elemé pedig {2, 6, 13, 16}, akkor az AC halmazé {2, 13}.

Javíthatunk az algoritmus hatékonyságán, ha nem a jelöltek TID-listáit tároljuk, hanem a jelölt

[3] A TID a Transaction Identification rövidítése

és prefixe TID-listájának különbségét. A prefix támogatottságából 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) [164]. A különbségi listát használó algoritmusok nagy fölénnyel verik a többi algoritmust, amennyiben a bemenet sűrű, és nagy méretű gyakori minták is vannak.

5.4. Az FP-growth algoritmus

Az FP-growth algoritmus [64] a legelső mintanövelő algoritmus (mintanövelő módszerek lásd 4.5.3 rész). Az algoritmus pontos megadásához azt kell tisztázni, hogy miként definiáljuk a növesztés műveletet, és hogyan állítjuk elő a vetített bemenetet.

Mi sem egyszerűbb, mint növelni egy elemhalmazt: adjunk hozzá elemeket. Tehát a növelés az unió képzésének, a csökkentés az elemek törlésének felel meg.

Az FP-growth a szűrt bemenetet egy speciális adatstruktúrában tárolja, amelyből hatékonyan lehet előállítani a vetített bemenetet. Az FP-fa (Frequent Pattern) egy kibővített szófa. Minden belső ponthoz tartozik egy számláló. Ez a számláló adja meg, hogy a ponthoz tartozó elemhalmaz hány szűrt tranzakcióban fordul elő[4]. Ezen kívül össze vannak kötve azon pontok, amelyekbe ugyanazon címkéjű él vezet. Ha tehát a fában egy elem m darab él címkéjén található, akkor a fában megtalálható $m-1$ darab kereszt irányú él is, amely a következő olyan pontba mutat, amelybe ilyen címkéjű él vezet. Az FP-fát kiegészíti egy fejléc tábla, amely gyakori elem–mutató párokból áll. Az i elemhez tartozó mutató adja meg, hogy hol található az első olyan pont, amelybe i címkéjű él vezet.

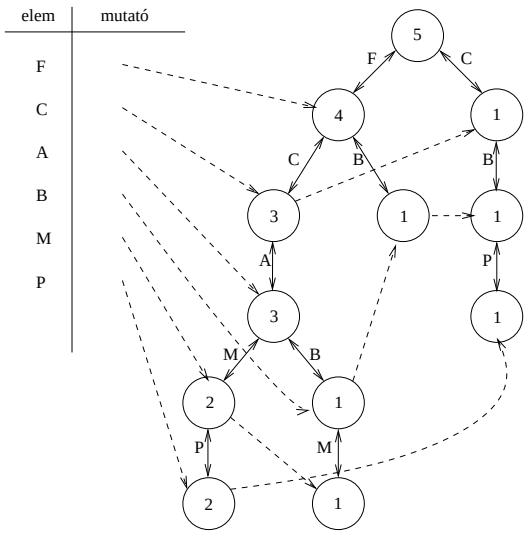
Az FP-fa mérete kritikus kérdés. Ha nem fér el a memóriába, akkor az algoritmus nagyon lassú lesz. Lévén az FP-fa egy keresztélekkel bővített szófa, ezért igaz rá, hogy alakja (és ezzel együtt a mérete is) függ az elemeken definiált rendezéstől. A **??** részben már elmondtuk, hogy a minimális méterű szófa megadása NP-nehéz feladat. A gyakorlat azt mutatja, hogy azon egyszerű heurisztika, hogy az elemeket támogatottságaik szerint csökkenően rendezzük, szinte mindig a legjobb megoldást adja. Ezért az FP-fa esetében ezt a megoldást célszerű követni.

A 5.9. ábrán a $\langle ACDFGIMP, ABCFLMO, BFHJO, BCKSP, ACEFLMNP\rangle$ bemenethez tartozó FP-fa látható. Ha támogatottsági küszöbnek 3-at választunk, akkor a szűrt, gyakoriság szerint csökkenően rendezett tranzakciók a következők: $\langle FCAMP, FCABM, FB, CBP, FCAMP\rangle$. A keresztéleket szaggatott vonallal jelöltük. Figyelem! A pontokhoz írt számok nem a pontok azonosítóját jelölik, hanem a számlálók értékét.

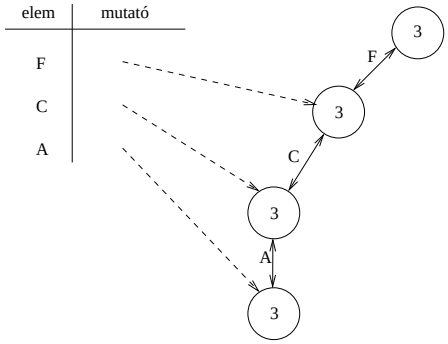
A fejléc mutatókból kiindulva és a keresztéleket követve megkaphatjuk a vetített bemenetet és meghatározhatjuk a vetített bemenetben gyakori elemeket. Az adott tranzakciók előfordulása megegyezik a keresztélek által mutatott pontok számlálójával. Ezek alapján a vetített bemenetet szűrhetjük és belőle egy újabb FP-fát építhetünk fel. Ezt a fát vetített FP-fának hívunk. A következő ábrán az M elemhez tartozó vetített és szűrt bemenet FP-fáját láthatjuk.

A hatékonyság szempontjából rendkívül fontos az a egyszerűsítési ötlet, hogy amennyiben egy vetített FP-fa egyetlen útból áll, és az utolsó állapotának számlálója nagyobb min_supp-nál, akkor ne folytassuk a rekurziót. A gyakori elemhalmazok pontosan az út által reprezentált halmaz részhalmazai.

[4]Ebben a tekintetben az FP-fa különbözik az APRIORI bemenetének tárolásához használandó szófától. Ott ugyanis egy ponthoz tartozó számláló azt adná meg, hogy a ponthoz tartozó elemhalmaz mennyi szűrt tranzakcióval *egyezik meg*.



5.9. ábra. példa: FP-fa



5.10. ábra. példa: vetített FP-fa

Ha prefix anti-monoton kényszert is kezelnie kell az algoritmusnak, akkor ez a rendezés eltérhet a kényszer által definiált rendezéstől. Választanunk kell ilyenkor, hogy a kis memória-fogyasztás fontosabb, vagy a kényszer által elérhető mintatér csökkentése.

5.4.1. Az FP-growth* algoritmus

2003 novemberében megszervezték az első gyakori elemhalmaz-kinyerő algoritmusok versenyét [58]. 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 szervezők végül annak adták a fődíjat (egy sört és egy pelenkát!), aki az FP-growth* algoritmust [60] küldte be.

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).

5.5. További híres algoritmusok

A három óriás (APRIORI, eclat, FP-growth) mellett léteznek olyan kevésbé ismert algoritmusok is, amelyek nagyon hatékonyan tudják megtalálni a gyakori elemhalmazokat. Ezeket ismertetjük ebben a részben.

5.5.1. A $\mathcal{DF}$-APRIORI algoritmus

A $\mathcal{DF}$-APRIORI [118] mélységi bejárást (Depth-First search) valósít meg. Helytelenül illették a szerzők az APRIORI jelzővel, hiszen az APRIORI módszer lényegét a jelöltek előállításának módja adja: csak az legyen jelölt, amiről tudjuk, hogy minden részmintája gyakori. Ez nem áll fenn ennél az algoritmusnál!

Pszeudokódja azonban teljesen megegyezik az APRIORI módszerével, egyedül a jelölt-előállítás módja más. Első lépésben meghatározza a gyakori elemeket. Rendezzük ezeket támogatottságaik alapján csökkenő sorba és jelöljük ezeket $i_1, i_2, \ldots, i_m$-el. Az algoritmus ezután pontosan $m-1$ lépést hajt végre, ahol minden lépés jelölt-előállítás, támogatottság meghatároz és a ritka elemek törléséből áll. Ha az l-edik lépésig bezáróan ($1 \leq l < m$) a megtalált gyakori elemek a $gy_1, gy_2, \ldots$ elemek, akkor az $l+1$-edik szint jelöltjei az $i_{m-l} \cup gy_1, i_{m-l} \cup gy_2, \ldots$ elemhalmazok lesznek. Kiindulást az i_m elem adja.

Amennyiben a gyakori elemhalmazokat szófában tároljuk, akkor a jelölt-előállítás pofonegyszerű és villámgyors. Nem kell mást tennünk csak felvennünk egy új gyökeret, amiből egy i_{m-l} címkéjű élen keresztül lehet eljutni a régi gyökérhez. Látnunk kell, hogy ez az algoritmus jóval több hamis jelöltet fog létrehozni, mint az APRIORI. Nem ellenőrizzük, hogy gyakori-e a jelöltek összes részhalmaza, hiszen ez az információ nem áll rendelkezésünkre.

5.5.2. patricia

5.5.3. kdci

5.5.4. lcm

5.5.5. Mintavételező algoritmus elemzése

Az egyszerű mintavételező algoritmust bemutattuk a 4.5.4 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 3.1.3 részben megadtuk, hogy mekkora mintát kell választani, ha azt akarjuk, hogy a relatív gyakoriságok megegyezzenek az előfordulások valószínűségével. Használjuk most is a A 3.1.3 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:

$$p(\text{gyakoriság}(x,m) < min_freq') = p\Big(\frac{Y}{m} < min_freq'\Big)$$

egy adott küszöbnél (δ') kisebb kell legyen és tudjuk, hogy

$$p > min_freq$$

A fenti egyenletre alkalmazva a Chernov-korlátot azt kapjuk, hogy

$$p\big(\frac{Y}{m} < min_freq'\big) =$$

$$p\big(\frac{Y}{m} - p < min_freq' - p\big) <$$

$$p\big(\frac{Y}{m} - p < min_freq' - min_freq\big)$$

$$\leq e^{-2(\text{min_freq'-min_freq})^2 m}$$

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 mintaméretthez 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

5.6. Elemhalmazok Galois lezárja

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.

5.10. 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 [112–114, 166].

5.6.1. A zárt elemhalmazok fogalma

A zárt minta fogalmát már bevezettük a 4.2.1 részben. Ismételjük meg úgy, hogy a definíció elemhalmazokra vonatkozzon: *Az I elemhalmaz zárt, amennyiben nincs nála bővebb halmaz, amelynek támogatottsága megegyezik I támogatottságával.*

A zárt elemhalmazokra adhatunk egy másik definíciót is. Vezessük, be a $cover'$ függvényt:

5.11. 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:

$$\begin{array}{ll} (1)\ I_1 \subseteq I_2 \Rightarrow cover(I_1) \supseteq cover(I_2) & (1')\ T_1 \subseteq T_2 \Rightarrow cover'(T_1) \supseteq cover'(T_2) \\ (2)\ T \subseteq cover(I) \Longleftrightarrow I \subseteq cover'(T) & \end{array}$$

5.12. 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

$$\begin{array}{ll} (I)\ I \subseteq h(I) & (I')\ T \subseteq h'(T) \\ (II)\ h(h(I)) = h(I) & (II')\ h'(h'(T)) = h'(T) \\ (II)\ I_1 \subseteq I_2 \Rightarrow h(I_1) \subseteq h(I_2) & (III')\ T_1 \subseteq T_2 \Rightarrow h'(T_1) \subseteq h'(T_2) \end{array}$$

5.13. 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.

5.14. 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.

■

A 4.4.2 részben bemutattuk, 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 (CHARM [**?**], CLOSET [116], CLOSET+ [154], MAFIA [26]), ezek ismertetésétől eltekintünk.

5.7. 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.

5.7.1. ExAnte

Az ExAnte [88] 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.

Nézzünk egy példát. 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.

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

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.

5.8. 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, amit 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.

5.8.1. 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 van 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.

6. fejezet

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.

6.1. 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 mobiltelefon és új tok vásárlása után új előlapot is vásárol.

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, 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.

6.1.1. 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 a 4.5 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:

$$bemenet : \mathcal{S} = \langle S_1, S_2, \ldots, S_v \rangle,$$

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:

6.1. 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.

6.1.2. APRIORI

A fent definiált feladat a gyakori mintakinyerés egy speciális esete, így alkalmazhatók rá az általános algoritmusok, például az APRIORI. Az általános leírást megadtuk a 4.4 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 5.2.2 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.

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).

6.1.3. 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.

6.2. 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 [8].

APRIORIALL

Ismét APRIORI! De minek tőrjük 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 bulizni (☺). Ennél még kényelmesebb megoldást javasoltak az APRIORIALL kitalálói[1]. Visszavezették ezt a feladatot az előző részben bemutatott APRIORI megoldásra.

[1] Ez nem meglepő, hiszen sem az ismétlés nélküli jelöltelőállítás sem a támogatottság meghatározása nem triviális feladat. Érdemes elgondolkozni azon, hogy miért nem.

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.

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 – a marketingesek igénye keltette életre. A kapott eredmények azonban nem elégítették ki őket, újabb feladattal áltak elő [140] [165]!

I. **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.

II. **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.

A 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.

6.3. 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*

I. $I_j \subseteq \cup_{k=l_j}^{u_j} \hat{t}_k.t, 1 \leq j \leq m$,

II. $\hat{t}_{u_i}.TIME - \hat{t}_{l_i}.TIME \leq$ idő_ablak, $1 \leq i \leq m$,

III. $\hat{t}_{l_i}.TIME - \hat{t}_{u_{i-1}}.TIME >$ min_eltelt_idő, $2 \leq i \leq m$

IV. $\hat{t}_{u_i}.TIME - \hat{t}_{l_{i-1}}.TIME \leq$ max_eltelt_idő, $2 \leq i \leq m$

A fentiekből látszik, hogy a 6.1 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 algoritmus, 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 4.1 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: $\mathcal{T} = \langle (A, 1.0), (B, 2.0), (C, 3.0) \rangle$. 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 $\mathcal{M}\ \mathcal{K} = (\mathcal{M}, \prec)$ mintakörnyezet második tagját változtathatjuk meg.

6.4. 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:*

I. *M'-t megkaphatjuk M-ből I_1 vagy I_n törlésével.*

II. *M'-t megkaphatjuk I-ből egy legalább 2 elemű I_i valamely elemének törlésével.*

III. *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.

6.5. észrevétel. *A fenti tartalmazási relációra nézve a támogatottsági függvény rendelkezik a monotonitás tulajdonságával.*

Ha visszatérünk ahhoz a példához, amelyen bemutattuk, hogy az eredi 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
$\langle(A,B),(C)\rangle$	$\langle(A,B),(C,D)\rangle$	$\langle(A,B),(C,D)\rangle$
$\langle(A,B),(D)\rangle$	$\langle(A,B),(C),(E)\rangle$	
$\langle(A),(C,D)\rangle$		
$\langle(A,C),(E)\rangle$		
$\langle(B),(C,D)\rangle$		
$\langle(B),(C),(E)\rangle$		

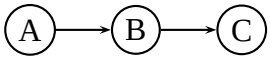
6.1. táblázat. Példa: GSP jelöltgenerálás

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.

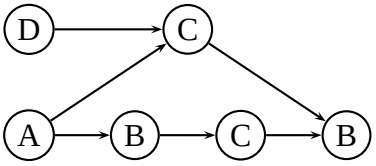
A jelöltek támogatottságának meghatározását nem részletezzük.

6.1.4. Sorozat típusú minta általánosítása

Tetszőleges elemsorozatot ábrázolhatunk egy gráffal. Például a $\langle A, B, C \rangle$ sorozat megfelelője a



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 a 6.1 ábrán látható.



6.1. ábra. Példa: sorozat általánosítása

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.

6.6. 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* $f : \{tid'_1, \ldots tid'_m\} \rightarrow \{tid_1, \ldots tid_n\}$ *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.

6.2. 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 [96], de hatékonysági okok miatt célszerű csak a diszjunktív normál formulákra szorítkozni.

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.

6.3. Gyakori epizódok

Az eddig 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[97, 98]. 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.

6.3.1. 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,

6.7. 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 $\mathcal{M}$ $\mathcal{K}$ 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.

6.8. 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 a 6.1 részben megadottal.

Két különböző támogatottsági definíció terjedt el.

6.9. definíció. *Legyen $\mathcal{S}$ bemeneti sorozat, $\mathcal{M}\ \mathcal{K} = (\mathcal{M}, \preceq)$ mintakörnyezet és τ anti-monoton illeszkedési predikátum. Az $m \in \mathcal{M}$ minta támogatottsága megegyezik*

I. S azon összefüggő részsorozatainak számával, amelyekre m minimálisan illeszkedik.

II. 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 $\mathcal{M}\ \mathcal{K} = (\mathcal{M}, \preceq)$ mintakörnyezet (esetleg w) és τ illeszkedési predikátum, célunk megtalálni a gyakori epizódokat.

6.3.2. 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 5.2 6.1.2 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.

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 az 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 az 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.

```
irégi.számláló ← irégi.számláló-1;
if( irégi.számláló = 0)
   forall æ in irégi.epizódjai
   {
      æ.hiányzás ← j.hiányzás+1;
      if( æ.hiányzás = 1) then
         æ.számláló ← j.számláló + æ.kezdeti_index-jelenlegi_index;
   }
```

6.2. ábra. régi elem kilépése

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 (6.3 ábra).

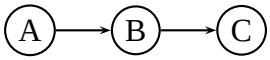
```
iúj.számláló ← iúj.számláló+1;
if( eúj.számláló = 1 )
   forall æ in iúj.epizódjai
   {
      æ.hiányzás ← æ.hiányzás-1;
      if æ.hiányzás=0 then
         æ.kezdeti_index ← jelenlegi_index;
   }
```

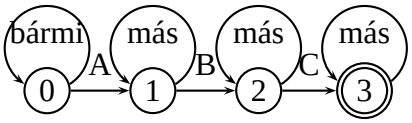
6.3. ábra. új elem belépése

Elemsorozatok támogatottságának meghatározása

Az elemsorozatok felismerése determinisztikus véges automatákkal történik, amelyek az egyes elemsorozatokat fogadják el. Az epizód alapján az automata előállítása egyszerű, az alábbi 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 a [83][89] [91][90][65] cikkekben olvashatunk.

7. fejezet

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 as sorozatokkal a támogatottságot megadó illeszkedési predikátumot a gráfoknál többféleképpen definiálhatjuk: részgráf, feszített részgráf, topologikus részgráf. Ez tovább bővíti a megoldandó feladatok körét.

7.1. 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 az 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. A két legismertebb részgráf izomorfiát eldöntő algoritmus Ullmanntól a backtracking [152] és B.D.McKaytól a Nauty [101].

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 naiv 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 $\prod_{i=1}^{k} |V_i|$-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.

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.

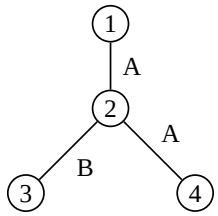
A gráf izomorfiából született probléma a gráfok *kanonikus kódolásának* problémája.

7.1. 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 a gráf szomszédossági mátrix oszlopai permutálásai közül kiválasztjuk azt, amely elemeit valamely rögzített sorrendben egymás után írva a legkisebbet kapjuk egy előre definiált lexikografikus rendezés szerint.

A szomszédossági mátrix alapú kanonikus kód előállításához szintén az invariánsokat célszerű használni. Ezáltal az oszlopok összes permutációjához tartozó kódok kiértékelése helyett egy oszlopot csak a saját csoportján belüli oszlopokkal kell permutálni.

Nézzük példaként a 7.1 ábrán látható csúcs- és élcímkézett gráfot (a csúcsokban szereplő számok a csúcsok azonosítói). Legyen $cimke(1) = e$, $cimke(2) = e$, $cimke(3) = e$, $cimke(4) = f$. A csúcsok címkéi szerint két csoportot hozunk létre. Ha figyelembe vesszük a fokszámot is, akkor a nagyobb csoportot két részre osztjuk ($\{1,3\}, \{2\}, \{4\}$). A 4!=24 kombináció helyett csak 2!=2 permutációt kell kiértékelnünk, ami alapján megkapjuk a kanonikus kódot: $\langle e000A0e0A00fBAABe\rangle$ lesz, ha a címkéken az abc szerinti rendezést vesszük és a 0 minden betűt megelőz.



7.1. ábra. Példa kanonikus kódolásra

7.2. 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 csa 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.

7.2. 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.

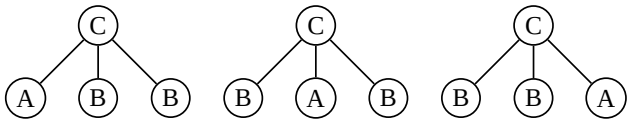
Mielőtt rátérnénk az általános eset tárgyalására nézzük meg, hogyan lehet kinyerni a gyakori címkézett fákat.

7.3. 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.

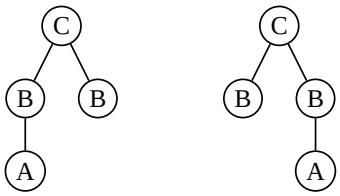
A gyakori fák kinyerése hasznos a bioinformatikában, a webelemzésnél, a félig strukturált adatok vizsgálatánál stb. Az egyik legszemléletesebb felhasználási terület a webes szokások elemzése. Gyakori elemhalmaz-kinyerő algoritmussal csak azt tudnánk megállapítani, hogy melyek a gyakran látogatott oldalak. 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.



7.2. ábra. Példa: rendezés nélküli, címkézett, gyökeres fák

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.

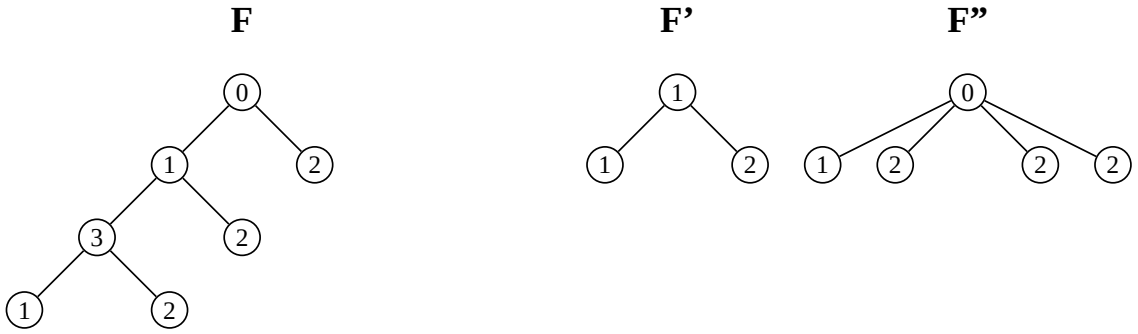


7.3. ábra. Példa: izomorf rendezett, gyökeres fák

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.

A 7.4 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 2.3 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.



7.4. ábra. Példa: gyökeres részfák tartalmazására

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.

7.3.1. TreeMinerH

A TreeMinerH [162] az APRIORI sémára épül (annak ellenére, hogy Zaki publikálta). Nézzük meg, hogyan állítjuk elő a jelölteket és hogyan határozzuk meg a támogatottságukat.

Jelöltek előállítása

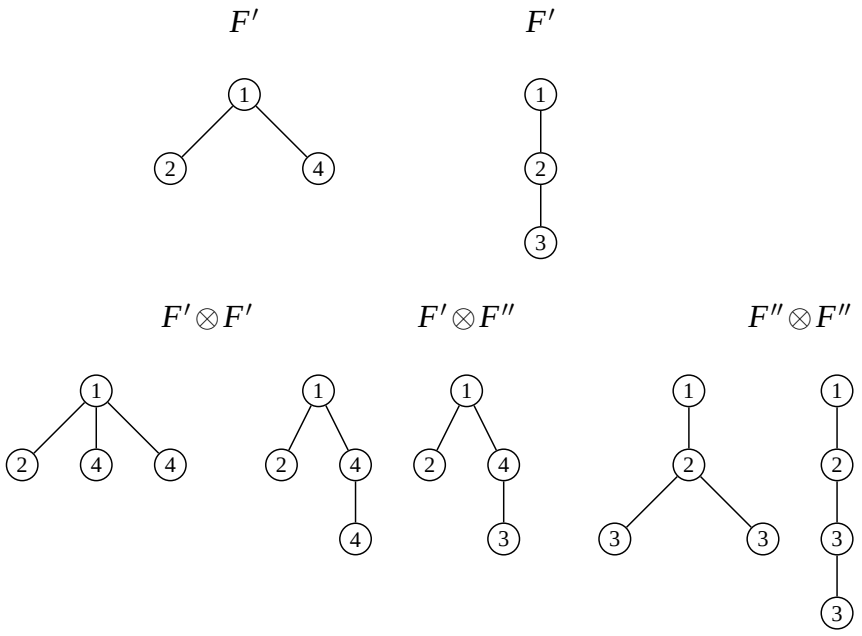
Egy ℓ-elemű jelöltet két $(\ell-1)$-elemű gyakori fa (F' és F'') illesztésével (jelölésben: $\otimes$) kapjuk meg. Hasonlóan az eddigiekhez a két $(\ell-1)$-elemű fa csak a legnagyobb elemükben különböznek, amely esetünkben azt jelenti, hogy ha elhagynánk a legnagyobb csúcsot (és a hozzá tartozó élt), akkor ugyanazt a fát kapnánk. Az általánosság megsértése nélkül feltehetjük, hogy $szulo_{F'}(\ell-1) \leq$ $\leq szulo_{F''}(\ell-1)$. A potenciális jelölt a G' 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 megkülönböztetnünk. 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.



7.5. ábra. Példa jelöltek előállítására

A kettőnél nagyobb elemszámú fák támogatottságának meghatározásánál szófa jellegű adatstruktúrát javasoltak. A fát 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.

7.3.2. TreeMinerV

A TreeMiner algoritmus [163] Zaki módszerét használja. A vertikális adatbázisból kiindulva előállítja a egyelemű fák illeszkedési listáit és a továbbiakban már csak ezen listákkal dolgozik.

Zaki módszerét ismertettük a 4.5.2 részben, a jelölt-előállítás pedig megegyezik a TreeMinerH jelölt-előállításának első lépésével. Csak azt kell tisztáznunk, hogyan határozzuk meg a jelöltek támogatottságát.

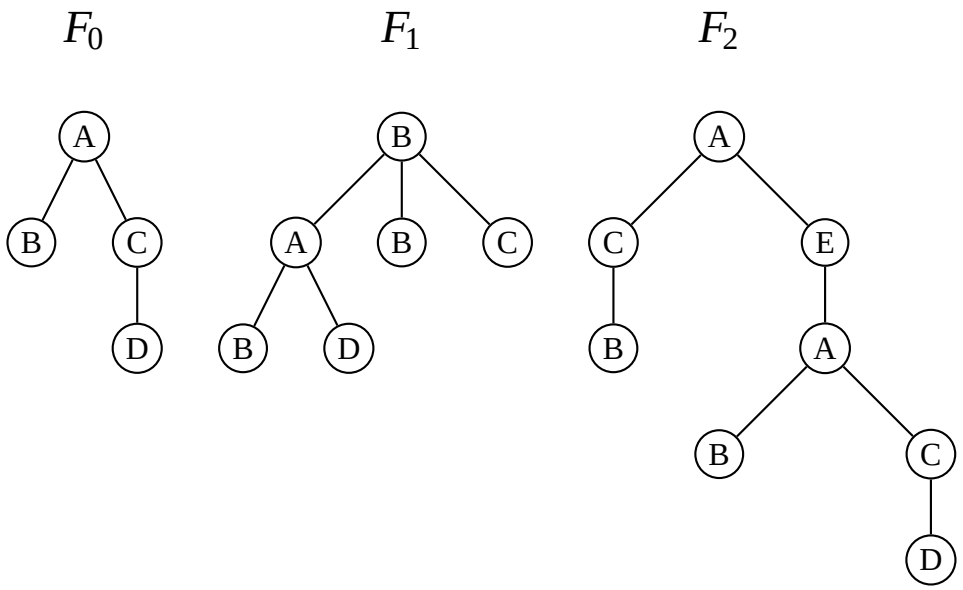
Jelöltek támogatottságának meghatározása

Az egyelemű jelöltek meghatározásához ismét elegendő egy lista, a kételemű jelöltekhez pedig egy tömb. A kettőnél nagyobb méretű jelöltek meghatározásának kulcsa az illeszkedési lista. A kiinduló illeszkedési listákat (amelyek az egyelemű gyakori fákhoz tartoznak) kételemű jelöltek meghatározása közben építjük fel.

Az a fő kérdés, hogy miként kell definiálni az illeszkedési listákat címkézett gyökeres fák esetében ahhoz, hogy teljesüljön a két elvárás (emlékeztetőül: a támogatottság egyértelműen meghatározható

legyen belőle, és a jelöltek illeszkedési listáit a generátoraiból elő tudjuk állítani).

A fogalmak szemléltetésére a 7.6 ábrán található fákat fogjuk használni. Jelöljük egy tetszőleges



7.6. ábra. Példaadatbázis: címkézett gyökeres fák

F gyökeres fa j csúcsából induló részfa legnagyobb sorszámát $MAX_F(j)$-vel. Például $MAX_{F_0}(0) = 3$, $MAX_{F_2}(4) = 7$,$MAX_{F_2}(1) = 2$.

Az ℓ-elemű F fa illeszkedési listájának minden eleme F-nek egy előfordulását rögzíti. Tegyük fel, hogy a F fa része az i-edik bementi fának (F_i-nek) és a tartalmazás injektív függvényét f-el jelöljük. Ekkor az illeszkedési lista ezen illeszkedését leíró eleme a következő 4-es: $\big(i, \langle f(0), f(1), \ldots, f(\ell - 1)\rangle, f(\ell), MAX_{F_i}(f(\ell))\big)$. Nyilvánvaló, hogy az illeszkedési listából a támogatottság és a súlyozott támogatottság is könnyűszerrel meghatározható. Már csak azt kell megnéznünk, hogy állítjuk elő a jelölt illeszkedési listáját, azaz hogyan illesztünk két illeszkedési listát.

Két illeszkedési lista illesztésének alapfeltétele, hogy a 4-esek első két tagjai megegyezzenek, hiszen azonos gráfban lévő illeszkedéseket keresünk (1. tag) és a generátorok prefixei azonosak (2. tag). Emlékszünk, hogy a F', F'' illesztésénél két esetet különböztettünk meg, attól függően, hogy az új csúcsot F' legnagyobb sorszámú elemének testvére lesz vagy gyereke. Jelöljük a két fa illeszkedési listáinak 3. és 4. tagját $f(\ell), MAX_{F_i}(f(\ell))$ és $f'(\ell), MAX_{F_i}(f'(\ell))$-el.

Az első típusú illesztés feltétele, hogy $MAX_{F_i}(f(\ell)) < f'(\ell)$. Ekkor a jelölt illeszkedési listája: $i, \langle f(0), f(1), \ldots, f(\ell-1), f(\ell)\rangle, f'(\ell), MAX_{F_i}(f'(\ell))$.

A második típusú illesztés csak akkor jöhet létre, ha $f(\ell) \leq f'(\ell)$ és $MAX_{F_i}(f(\ell)) \geq MAX_{F_i}(f'(\ell))$. A kapott jelölt illeszkedési listája az alábbi lesz: $i, \langle f(0), f(1), \ldots, f(\ell - 1), f(\ell)\rangle, f'(\ell), MAX_{F_i}(f(\ell))$

Nézzünk példákat. Illesszünk az $A \to C$ fát az $A \to D$ fával. gráffal. Legyen az első fa illeszkedési listája: $\langle(0, \langle 0\rangle$,2,3$), (2, \langle 0\rangle, 6, 7), (2, \langle 4\rangle, 6, 7)\rangle$, a másodiké: $\langle(0, \langle 0\rangle, 3, 3), (1, \langle 1\rangle, 3, 3), (2, \langle 0\rangle, 7, 7), (2, \langle 4\rangle, 7, 7)\rangle$. A generátorokból két jelöltet állítunk elő. Az elsőben a D címkéjű csúcs a C címkéjű csúcs testvére lesz, a másodikban a gyereke. Az első jelölt illeszkedési listája üres lesz, hiszen a 4 tagok egyenlőek az azonos bemeneti fákhoz tartozó

listákban. A második jelölt illeszkedési listája: $\langle(0,\langle 0{,}2\rangle{,}3{,}3),(2,\langle 0{,}6\rangle,7,7)$, amiből azonnal meg tudjuk állapítani, hogy a jelölt támogatottsága 2.

Amennyiben egy prefix csak egyszer fordul elő egy bemeneti fában, akkor két generátor illesztésénél elég csak a 4-esek első elemének egyenlőségét vizsgálni. Ha ezek egyeznek, akkor nyilvánvaló, hogy a prefixek ugyanott fordulnak elő. Memóriafogyasztás csökkentése érdekében ezért a második tagot csak akkor tároljuk, ha a prefix a bementi fában többször előfordul.

7.4. Gyakori részfák

FOLYT. KÖV.

Fák esetében a részfa izomorfia eldöntésére létezik polinom idejű (pontosabban $O(n\frac{k^{1.5}}{\log k})$, ahol k a mintafa, n a másik fa csúcsainak száma) algoritmus [135].

FOLYT. KÖV.

7.5. A gyakori feszített részgráfok

Ebben a részben bemutatjuk a legismerteb gyakori feszített részgráfokat kinyő algoritmust. A $\mathcal{M}\ \mathcal{K} = (\mathcal{G}, \preceq)$-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 fogalma lásd alapfogalmak 2.3 rész).

7.5.1. Az AcGM algoritmus

Az AcGM algoritmus [71] – ami az AGM javított változata [70] – 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 háromszögmátrixának 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:

$$X_\ell = \begin{pmatrix} T & x_1 \\ x_2^T & x_{l,l} \end{pmatrix}, Y_\ell = \begin{pmatrix} T & y_1 \\ y_2^T & y_{l,l} \end{pmatrix}.$$

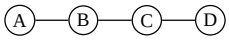
– T egy kanonikus szomszédossági gráf.

– 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 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 

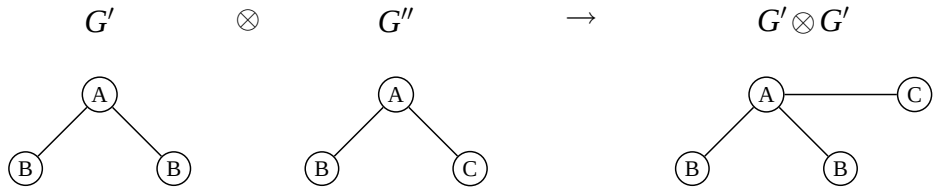
gráfot például a fenti jelölt előállítással nem lehetne kinyerni.

Nézzünk egy példát. A következő ábrán két gyakori 3 csúcsú gráfot láthatunk, amelyből a jelölt előállítás során a jobb oldalon látható gráfot hozzuk létre. Az első gráf szomszédossági mátrixa $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$, a másodiké $\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$, az illesztés során kapott szomszédossági mátrix pedig $\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & z \\ 0 & 1 & z & 0 \end{pmatrix}$.

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. A problémát az okozza, hogy az így kapott mátrix nem biztos, hogy normál formájú lesz. Az AcGM a következő módszerrel alakítja át a részmátrixot normál formájúvá.

FOLYT. KOV.



7.7. ábra. Példa jelöltek előállítására

támogatottságok meghatározása

A jelöltek előállítása után rendelkezésünkre fog állni egy nagy halom normál formájú szomszédossági mátrix. Ugyanannak a gráfnak több normál formájú szomszédossági mátrixa létezik ezért minden mátrixhoz hozzá kell rendelni az általa reprezentált gráf kanonikus kódját.

FOLYT. KOV. st23

Ha az azonos gráfot reprezentáló normál formájú szomszédossági mátrixok közül ki tudtuk választani a normál formájú szomszédossági mátrixot, akkor 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.

7.6. 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. Bemutatjuk a két legismertebb algoritmust az FSG-t és a gSpan-t.

7.6.1. Az FSG algoritmus

Az FSG algoritmus [86] 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.

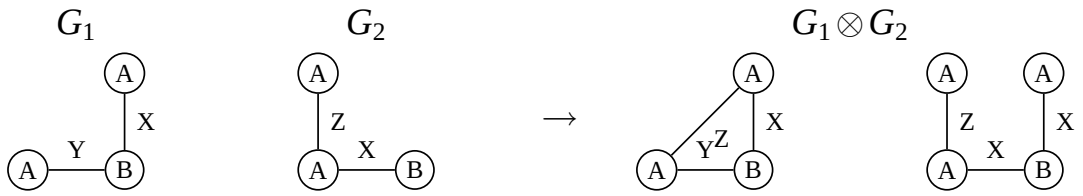
Megszokhattuk már, hogy a fő lépés a jelöltek előállítása.

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,

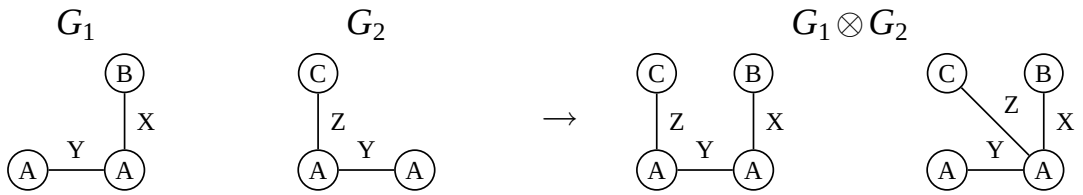
hogy minden gráfot önmagával is illesztünk. Két gráf illesztésénél – akárcsak két elemsorozatok esetében – több gráf jön létre. Jelöljük a G_2-nek a magba nem tartozó élét $e = (u, v)$-vel. Az előállított gráfok a G_1 bővítése lesz egy olyan $e' = (u', v')$ éllel, amelyre $u' \in V_1$, $e' \notin E_1$, $c_E(e) = c_E(e')$, $c_V(u) = c_V(u')$ 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 7.8 ábra.



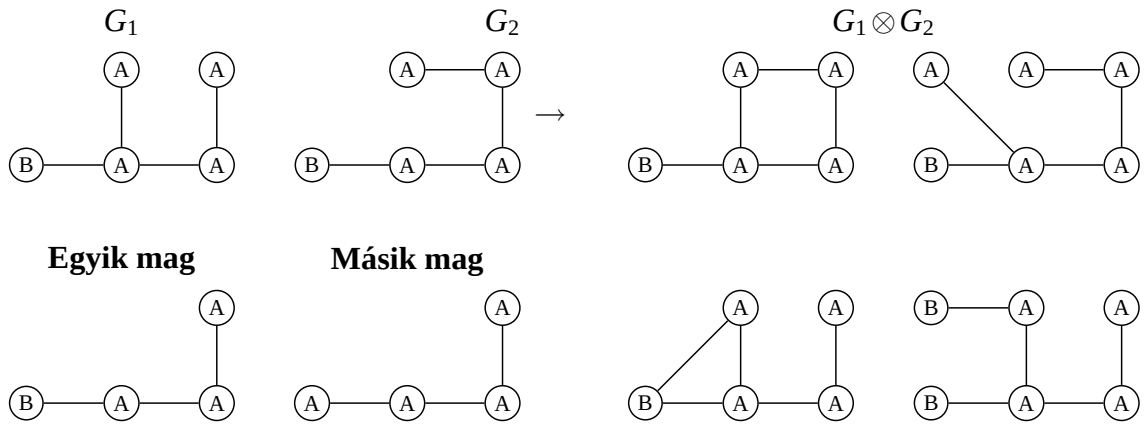
7.8. ábra. Példa: gráf illesztése

Az előállított potenciális jelöltek számát növeli az a tény is, hogy a magnak több automorfizmusa lehet, így az új élt több csúcshoz is illeszthetjük. Erre mutat példát a következő ábra.



7.9. ábra. Példa: gráf illesztése - mag automorfizusok

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 a 7.10 ábrán.



7.10. ábra. Példa: gráf illesztése - több közös mag

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-előállítsának 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 algoritmusok 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.

7.6.2. gSpan

A gráfok ábrázolására a gSpan a *DFS-kódokat*, illetve az abból előállított kanonikus kódolást használja. A mélységi kód előállításához ki kell választanunk egy gyökér csúcsot, majd ebből a csúcsból indulva bejárni a gráfot, mintha egy gyökeres fát járnánk be mélységi bejárás szerint. A bejárás szerint minden csúcshoz időcímkét rendelhetünk, amely megadja, hogy hanyadik lépés során látogattunk meg egy csúcsot. Mivel a gráf tartalmazhat köröket is, ezért előfordulhat, hogy egy csúcsot többször meglátogatunk. Ilyen esetben a csúcs időcímkéjét ne írjuk felül (és az idő számlálóját se növeljük). Hívjuk *előreélnek* azokat az éleket, amelyek még nem látogatt csúcsba vezetnek, a többit élt pedig *visszaélnek*.

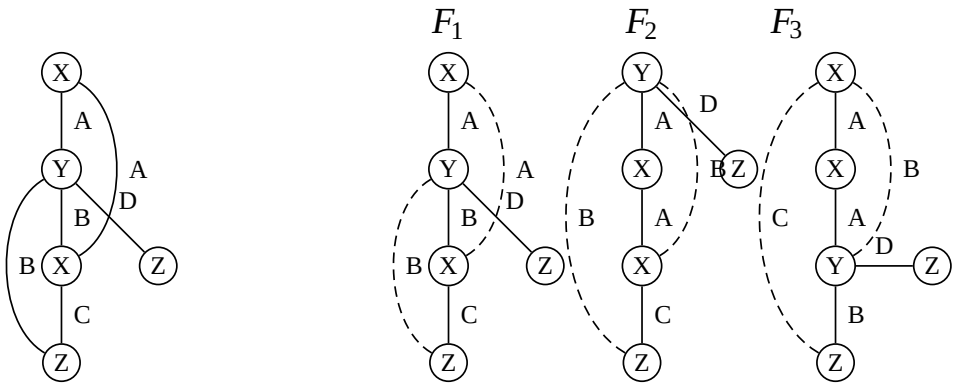
A gráf bejárása során minden lépésnek egy elem felel meg a DFS-kódban, azaz a kód hossza megegyezik a gráf éleinek a számával. Minden elem egy ötös, amelynek első két eleme az indulási és az érkezési csúcsok időbélyegét adja, a harmadik és ötödik elem ezen csúcsok címkéit és a negyedik elem az él címkéjét tárolja.

Természetesen egy adott gráfnak több DFS-kódja is lehet attól függően, hogy melyik csúcsot választjuk gyökének és milyen sorrendben vesszük egy csúcs gyermekeit. A 7.11 ábrán egy példagráfot, három különbözö mélységi bejárást és az azokhoz tartozó DFS-kódokat láthatjuk. A visszaéleket szagatott vonallal jelöltük.

A címkéken tudunk egy rendezést definiálni, ami alapján az ötösöket is rendezni tudjuk. Ezen rendezés szerint lexikografikusan rendezni tudjuk a kódokat is. Egy gráf kanonikus kódja legyen az a DFS-kódja, amely ezen rendezés szerint a legkisebb.

Legyen $\alpha = \langle a_0, a_1, \ldots, a_m \rangle$ egy DFS-kód. Ekkor a $\beta = \langle a_0, a_1, \ldots, a_m, b \rangle$-t az α *gyermekének* hívjuk, α-t pedig a β *szülőjének*. Ahhoz, hogy a β tényleg DFS-kód legyen a b címkéjű élnek az α által kódolt mélységi fa legjobboldali ágán kell elhelyezkednie. Erre a DFS-kódnövelésre láthatunk példát a következő ábrán.

Könnyű belátni, hogy amennyiben az új él visszaél, akkor csak a legjobboldalibb csúcsból indulhat.



él	F_1	F_2	F_3
0	(0,1,X,a,Y)	(0,1,Y,a,X)	(0,1,X,a,X)
1	(1,2,Y,b,X)	(1,2,X,a,X)	(1,2,X,a,Y)
2	(2,0,X,a,X)	(2,0,X,b,Y)	(2,0,Y,b,X)
3	(2,3,X,c,Z)	(2,3,X,c,Z)	(2,3,Y,b,Z)
4	(3,1,Z,b,Y)	(3,0,Z,b,Y)	(3,0,Z,c,X)
5	(1,4,Y,d,Z)	(0,4,Y,d,Z)	(2,4,Y,d,Z)

7.11. ábra. Példa: mélységi fák és mélységi kódok

7.12. ábra. Példa: mélységi kód

A szűlő-gyerek reláció megadásával definiálhatunk a DFS-kódfa fogalmát. A DFS-kódfa egy olyan fa, amelynek csúcsaiban DFS-kódok ülnek és minden szűlő-gyerek csúcs által reprezentált DFS-kódokra teljesül a fenti szűlő-gyerek kapcsolat és a fa redezett, azaz minden csúcs gyermeke a DFS-kód szerint növevő sorrendbe van rendezve.

Amennyiben egy kanonikus kódhoz hozzáveszek egy új élt úgy, hogy ez DFS-kódot eredményezzek, az nem jelenti azt, hogy ez a kód kanonikus kód lesz. A DFS-kódfában minden kanonikus kód megtalálható, de emellett számos nem kanonikus kód is szerepel. A rendezés azonban garantálja, hogy ha pre-ordes bejárás szerint bejárnánk a fát, akkor tetszőlges gráf első DFS-kódja egybe kanonikus kód is. A DFS-kódfát ezek szerint egyszerűsíthetjük, hogy kimetszük azon részfákat, amelyek csúcsai nem kanonikus kódokat tartalmaznak. A gSpan algoritmus tulajdonképpen ezt a gyakori gráfokat tároló egyszerűsített DFS-kódfát állítja elő. Mihelyt egy olyan DFS-kódot állít elő, amely nem minimális, a fát nem növeszti tovább ezen az ágon.

Könnyű belátni, hogy a $G = (V, E)$ gráfnak nem kell $|V| \cdot |E|$-nél többször törölni nem kanonikus DFS-kódját. A G gráfot csak $(|E| - 1)$-elemű részgráfjából származtathatjuk, amelyek száma legfeljebb $|E|$. A részgráf által nem tartalmazott élt, amennnyiben az előreél $|V| - 1$ féleképpen illeszthetjük a legjobboldalibb ághoz. Visszaél esetében pedig a legjobboldalibb csúcshoz kell tennünk, ezen lehetőségek száma pedig $|V| - 2$. Ez a korlát elég gyenge, hiszen csak annyit tett fel, hogy a legjobboldali úton található csúcsok száma kisebb $|V|$-nél. Az esetek többségében ez az út az élszámnál jóval kisebb, így a nemkanonikus kódok törlésének száma jóval kevesebb.

FOLYT KÖV.

8. fejezet

Asszociációs szabályok

A gyakori elemhalmazokat felhasználhatjuk arra, hogy gyakori elemhalmazokra vonatkozó szabályokat nyerjünk ki belőlük. Az $I_1 \to I_2$ 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 az értelme ezeknek a szabályoknak? Például az, hogy 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 akció hatására I_1 termék eladása nőni fog, ami I_2 termék eladásának növekedését okozza. Amit vesztünk a réven azt megnyerjük a vámon: összességében a profitunk nőni fog, és a leárazás reklámnak is jó volt.

Korunkra jellemző olcsó internetes üzletek is ilyen szabályok alapján dolgoznak. 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.

8.1. Az asszociációs szabály fogalma

Használjuk a 5.1 részben bevezetett definíciókat és jelöléseket (elemhalmaz, kosár, támogatottság, fedés, gyakori elemhalmaz stb.).

8.1. 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 $R : I_1 \rightarrow I_2$ 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*.

8.2. 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. 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 $I_1 \rightarrow I_2$ egy érvényes asszociációs szabály. A támogatottság anti-monoton tulajdonságát felhasználhatjuk annak érdekében, hogy ne végezzünk túl sok felesleges kettéosztást.

8.3. észrevétel. *Amennyiben I_1, I gyakori elemhalmazok a $\mathcal{T}$ bemeneti sorozatban, és $I_1 \subset I$, illetve $I_1 \rightarrow I \backslash I_1$ nem érvényes asszociációs szabály, akkor $I_1' \rightarrow I \backslash I_1'$ 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 $I_1' \rightarrow I \backslash I_1'$ szabály bizonyosságát, akkor

$$c' = \frac{\text{supp}(I)}{\text{supp}(I_1')} \leq \frac{\text{supp}(I)}{\text{supp}(I_1)} < min_conf$$

tehát $I_1' \rightarrow I \backslash I_1'$ sem érvényes asszociációs szabály.

■

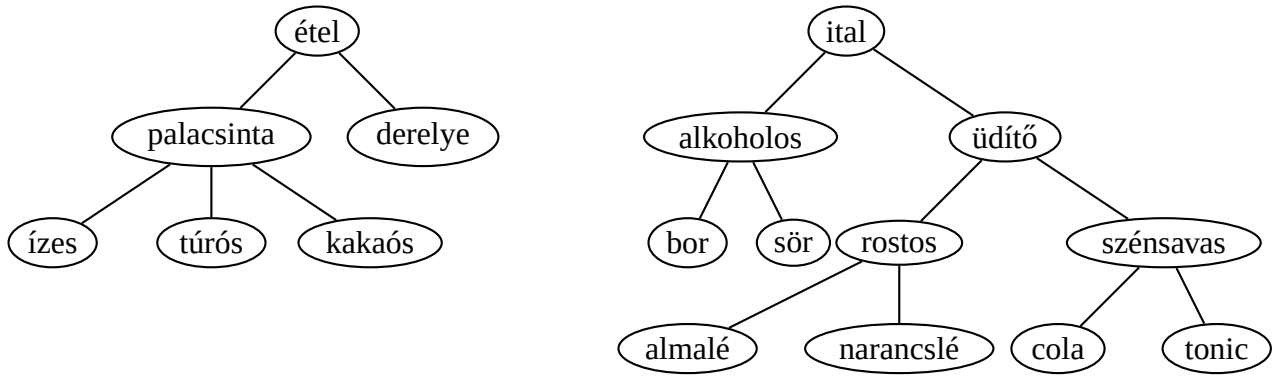
8.2. Hierarchikus asszociációs szabályok

Ebben a részben a hierarchikus asszociációs szabályokkal foglalkozunk, amelyek az asszociációs szabályok egyik általánosítás [53, 55, 62, 136, 139, 146]. 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 → chips támogatottsága lehet 5000, de ha 5 féle sör létezik, akkor termék szinten könnyen lehet, hogy mindegyik, sört tartalmazó, asszociációs szabály támogatottsága 1500 alatt lesz és nem lesz érvényes.

Egy üzletnek a kategória szintű asszociációs szabályok legalább annyira fontosak lehetnek, mint a termékeken értelmezett szabályok (pl.: 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. A fa leveleiben találhatók az egyes termékek, a belső csomópontokban pedig a kategóriák. Egy képzeletbeli büfé termék-taxonómiája az alábbi ábrán látható.



8.1. ábra. Példa: képzeletbeli büfé termék-taxonomiája

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 ß $\in I'$ -re vagy $i \in I$, vagy $\exists i' \in I$, hogy $i \in$ ős(i')[1]. 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 110. oldalon található definíció szerint minden $X \xrightarrow{100\%,s} \hat{X}$ szabály érvényes lenne, ha $\hat{X} \subseteq$ ős(X), és X gyakori termékhalmaz.

8.4. 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.*

[1] Gyökeres gráfoknál definiálhatjuk a szülő, gyermek, ős, leszármazott fogalmakat. Ezt az alapfogalmak gráfelmélet részében megtettük.

A támogatottság (s), és bizonyosság (c) definíciója megegyezik a 8.1. 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 elemhalmazokkal foglalkozzunk. Ne állítsunk elő olyan jelöltet, amely ilyen tulajdonságú [139].

A fentitől különböző megközelítést javasoltak a [55, 62]-ben. 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, részletek és futási eredmények találhatók [62]-ban.

8.3. 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.

Mindét állítást a támogatottság anti-monoton tulajdonságából közvetlenül adódik.

Ezek szerint minden asszociációs szabály levezethető a maximális következményrésszel rendelkező asszociációs szabályokból.

8.3.1. Egzakt asszociációs szabályok bázisa

A 100%-os bizonyossággal rendelkező asszociáció 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 bázis, amelyből minden egzakt asszociáció szabály levezethető. Ehhez a bázishoz a *pszeudó-zárt elemhalmazokon* keresztül jutunk.

8.5. definíció. *$I \subseteq \mathcal{I}$ pszeudo-zárt elemhalmaz, ha nem zárt, és minden $I' \subset I$, ahol I' pszeudo-zárt elemhalmaz 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.

8.6. definíció. *Legyen FP a pszeudo-zárt elemhalmazok halmaza $\mathcal{T}$-ben. Ekkor a Duquenne–Guigues-bázist a következőképpen definiáljuk:*

$$DG = \{r : I_1 \to h(I_1) \setminus I_1 | I_1 \in FP \wedge I_1 \neq \emptyset\},$$

ahol az I lezártját $h(I)$-vel jelöltük.

8.7. 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, amik 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 $I \in GY_i \setminus \setminus ZGY_i$ 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.

8.4. Az asszociációs szabályok hibái

Az asszociációs szabályok gyakorlati alkalmazása során az alábbi három súlyos probléma jelentkezett:

I. Az asszociációs szabályok száma túl nagy. Ha magasra állítjuk a 2 küszöbszámot, akkor kevés szabály lesz érvényes, azonban ekkor számos – amúgy érdekes – szabály rejtve marad. Ellenkező esetben azonban rengeteg szabályt jön létre, amelyek közül kézzel kiválogatni a fontosakat szinte lehetetlen feladat.

II. Legtöbb szabály érdektelen. 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.

III. 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 tekinthetünk úgy, 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 hamburger és hot-dogot egyszerre. Azok és csak azok vesznek majonézt, akik hamburgert esznek. Ezek szerint a „kosarak”66% tartalmaz hot-dogot és 50%-uk hot-dogot és majonézt is. Emiatt a hot-dog → majonéz érvényes asszociációs lehet. Felhasználva az asszociációs szabályok bevezetésénél bemutatott trükköt, 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 definíció összefüggések mutatószámához. 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 $p(I_2)$-nal, 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. De ilyen szabály rengeteg van!).

A fenti három problémát egyszerre oldanánk meg, ha valahogy definiálni tudnánk a szabályok érdekességi mutatóját. Sajnos ez nem olyan egyszerű feladat. Az utóbbi evtizedben rengeteg publikáció született különböző érdekességi mutatókról. Ha elég sokáig vizsgáljuk őket, akkor mindegyikről kiderül, hogy van valami hibája. Talán nem is létezik tökéletes megoldás?!? A következő részekben az érdekességi mutatókat tekintjük át.

Egy szabály „függetlensége”

Egy szabály nem érdekes, ha a feltétel és a következményrészek függetlenek egymástól. Valószínüségszámításbeli 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. Ez alapján egy szabály függetlenségi mutatóját, amelyet *lift*-tel jelölünk a következőképpen definiálják:

$$\text{lift}(I \to I') = \frac{freq(I \cup I')}{freq(I) \cdot freq(I')},$$

ahol $freq$ a gyakoriságot jelöli.

Ha ezek után egy adatbázisból a rejtett összefüggéseket asszociációs szabályok formájában akarjuk kinyerni, akkor a támogatottsági és bizonyossági küszöb mellett érdekességi 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 érdekességének meghatározásához minden adat rendelkezésünkre áll, így könnyedén megkaphatjuk az értékét.

Megjegyezzük, hogy a függetlenség mérésére használják még a $freq(I \cup I')$, $freq(I) \cdot freq(I')$ hányadosa ahelyett a $freq(I \cup I')$ és $freq(I) \cdot freq(I')$ különbségét is, továbbá az ún. *meggyőző értéket* (conviction) is. Ezt a $I \to I'$ implikáció logikai megfelelője alapján definiálják: $\frac{p(I) \cdot p(\overline{I'})}{p(I, \overline{I'})}$.

Az függetelenségi mutató gyengéje, hogy ha találunk egy érdekes szabály, 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 ($supp(I_1 \cup I_3) = supp(I_1) \cdot supp(I_3), supp(I_2 \cup \cup I_3) = supp(I_2) \cdot supp(I_3)$) 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 a $I_1 I_3 \to I_2$ is érvényes és érdekes asszociációs szabályok, hiszen

$$\begin{aligned}\text{intr}(I_1 I_3 \to I_2) =& \frac{supp(I_1 \cup I_2 \cup I_3)}{supp(I_1 \cup I_3) supp(I_2)} = \frac{supp(I_1 \cup I_2) supp(I_3)}{supp(I_1) supp(I_2) supp(I_3)} = \\ =& \text{intr}(I_1 \to I_2) \geq \text{min_intr},\end{aligned}$$

.

$$\frac{supp(I_1 \cup I_2 \cup I_3)}{supp(I_1 \cup I_3)} = \frac{supp(I_1 \cup I_2) supp(I_3)}{supp(I_1) supp(I_3)} \geq \text{min_conf}$$

Könnyű belátni, hogy amennyiben érdekességi mutató helyett a meggyőző értékeket használjuk, akkor ugyanerre a következtetésre jutunk. Ezek alapján, egy adatbázisból kinyert érdekes asszociációs szabályok között a többség haszontalan, amennyiben sok a nagy támogatottságú, más termékektől független termék.

Egy szabály „javítási” mutatója

A fenti esetet úgy is jellemezhettünk volna, hogy az $I_1 I_3 \to I_2$ szabály az $I_1 \to I_2$ szabály egy speciális esete, amely nem hordoz semmi többletinformációt. Ha elfogadjuk Occam borotvájának elméletét, akkor csak az általánosabb érvényű és egyszerűbb szabályt tartjuk meg. Ezt az elvet próbálták alkalmazni a [14] cikkben, amikor bevezették egy szabály „javítási” mutatóját (improvement).

Legyen egy szabály javítási mutatója az a minimális különbség, amely előfordulhat a szabály bizonyossága és egy részszabály bizonyossága között. Pontosabban:

$$impr(I_1 \to I_2) = \min_{I_1' \in I_1} \{conf(I_1 \to I_2) - conf(I_1' \to I_2)\}.$$

Amennyiben a javítási érték pozitív, akkor tetszőleges nem üres elemhalmaz eltávolítása a feltételrészből csökkenti a bizonyosságot legalább a javítási értékkel. Következésképpen egy nagy javítási értékel rendelkező szabály feltételrészében található elemek minden kombinációjának nagymértékben hatással van a következményrészre. A negatív javítási értékkel rendelkező szabályok a fölösleges szabályok, hiszen egy részszabálya nagyobb hatással van a következményre és általánosabb érvényű. Célszerű ezért bevezetnünk egy javítási küszöbszámot (*min_impr*) és csak az ennél nagyobb javítási értékkel rendelkező szabályokat kibányászni.

További érdekességi/függetlenségi mutatók

Valójában a függetlenségi mutató nem ragadja meg kellőképpen a két esemény (I és I' előfordulása) statisztikai függetlenségét. Tudjuk, hogy az I, I' események függetlenek, ha $p(I)p(I') = p(I, I')$,

amelyet átírhatunk $1 = p(I'|I)/p(I)$ 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 $f = 1$. A másodikban a tranzakciók száma négyezer, $supp(I) = 1992$, $c = 0.504$, amiből $f = 1.012$. 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 mérése alkalmasabb 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 mi esetünkben az egyik eseményrendszer az I elemhalmaz a másik az I' elemhalmaz előfordulásához tartozik, és mindkét eseményrendszernek két eseménye van[2] (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 könnyű pótolni lehet, hiszen például $k_{2,1} = supp(I) - supp(I \cup I')$.

A lift, χ-statisztika mutatók mellett még számos elterjedt mutatószám létezik függetlenség mérésére. A teljesség igénye nélkül felsorolunk néhányat

[2] Amennyiben mindkét eseményrendszer két eseményből áll, akkor az eredeti képletet módosítani szokás a Yates-féle korrekciós együtthatóval, azaz $\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{2} \left(\left|k_{ij} - \frac{k_{i.}k_{.j}}{n}\right| - \frac{1}{2}\right)^2 / \frac{k_{i.}k_{.j}}{n}$.

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 $I \rightarrow I'$ 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	$\frac{H(I'\|I)}{H(I)}$	

Megjegyezzük, hogy bináris változók esetén (tehát amikor a kontingenciatábla 2×2-es) a χ^2 statisztika értéke megegyezik az empirikus kovariancia négyzetének n-szeresével (ahol n a tranzakciók/kísérletek száma).

Szomszédosság alapú érdekességi mutató

A [39] cikkben egy érdekes ötlettel által elő a szerzők, melynek filozófiája a következő. Képzeljük el az asszociáció szabályokat, mint egy térképen elterülő, különböző magasságú hegyeket, ahol a hegy magassága a szabály érdekességével arányos. Mondhatnánk, hogy érdekes egy szabály, ha a hegy magassága egy bizonyos korlát felett van.

Gondjuk meg, hogy valóban érdekesebbek-e a Himalája környékén található magas hegyek, mint Észak-Amerika vagy a Japán legmagasabb hegye, annak ellenére az utóbbiak alacsonyabbak. Inkább igaz, hogy egy hegy érdekessége a magasságától és a környezetétől függ. Például a Japánban található Fuji hegy pont azért olyan érdekes, mert közel s távol nincs hasonló magasságú hegy. Ennek analógiára értelmezhetjük az asszociációs szabályok távolságát és környezetét, majd ez alapján definiálhatunk egy érdekességi mutatót.

8.8. definíció. *Legyenek $R_1 : I_1 \rightarrow I_2, R_2 : I_1' \rightarrow I_2'$ asszociációs szabályok. Két szabály elemhalmaz alapú távolságát (d_{iset}) a következőképpen definiáljuk:*

$$d_{iset}(R_1, R_2) = \delta_1 |(I_1 I_2) \ominus (I_1' I_2')| + \delta_2 |I_1 \ominus I_1'| + \delta_3 |I_2 \ominus I_2'|,$$

ahol $\delta_1, \delta_2, \delta_3$ *a felhasználó által megadott konstansok és* $\ominus$ *a szimmetrikus differenciát jelöli (*$A \ominus B = (A \setminus B) \cup (B \setminus A)$*).*

Például $d_{iset}(D \to BC, AD \to BC) = \delta_1 + \delta_2$, $d_{iset}(BC \to D, BC \to AD) = \delta_1 + \delta_3$. Tulajdonképpen mindkét távolságot az A elem jelenléte okozza.

A felhasználó döntheti el a három konstans értékét. Elképzeléseinknek megfelel, hogy a teljes differenciának van a legnagyobb szerepe, és a bal oldali differenciának nagyobb szerepe van a jobb oldali differenciánál. A szerzők alapértelmezésnek a $\delta_1 = 1$, $\delta_2 = \frac{n-1}{m^2}$, $\delta_3 = \frac{1}{m^2}$ értékeket javasolták, ahol m az összes elem számát jelöli. Ez a választás a következő jó tulajdonsággal rendelkezik. Amennyiben $R_1 : I_1 \to I_2, R_2 : I_1' \to I_2'$ olyan szabályok, hogy $I_1 I_2 = I_1' I_2'$, akkor $d_{iset}(R_1, R_2) < d_{iset}(R_1, R_3)$, bármely $R_3 : I_1'' \to I_2''$ szabályra, ahol $I_1 I_2 \neq I_1' I_2'$. Tehát tetszőleges szabályhoz közelebb vannak azok a szabályok, amik ugyanazon elemekből épülnek fel, mint azok, amelyek tartalmaznak a szabály valamelyik oldalán egy új elemet is.

Ezek után definiálhatjuk egy szabály *szomszédságát*. Az R szabály r-szomszédsága ($N(R, r)$) azon szabályok halmaza, amelyek R-től legfeljebb r távolságra vannak. Ha az alapértelmezett konstansokat használjuk, akkor igaz, hogy egy szabály 1-környezetében az azonos elemekkel rendelkező szabályok vannak. A szerzők kétféle érdekességi mutatót definiáltak.

Az első érdekességi mutató szerinte egy szabály érdekes, ha bizonyossága nagyon eltér a szomszédságában lévő szabályok bizonyosságától. Jelöljük az R szabály r-szomszédságába eső szabályok bizonyosságának átlagát $avr_conf(R, r)$-rel, szórását pedig $std_conf(R, r)$-el. Azt mondjuk, hogy az R szabály a *bizonyossága miatt érdekes*, ha bizonyossága az átlagos bizonyosságtól jóval nagyobb, mint a bizonyosságok szórása. Tehát az érdekességi mutatót a következőképpen adhatjuk meg:

$$intr_{\text{Dong 1}}(R) = \frac{|conf(R) - avr_conf(R, r)|}{std_conf(R, r)}$$

, és az R szabály érdekes, ha $intr_{Dong1}(R) > min_intr$, vagy $intr_{Dong1}(R) < \frac{1}{min_intr}$.

A második érdekességi mutató a szabályok elszigeteltségét próbálja megragadni. Ha egy szabály szomszédságában a lehetséges elemekhez mérten kevés érdekes szabály van, akkor a szabály izolált és egyben érdekes is. Legyen $R = X \to Y$ és jelöljük az r-szomszédságban található elemek elméleti maximalis számát $max_neighbor(R, r)$-el. Például egy 4-elemű halmaz kettébontásából kapott szabály 1-szomszédságában maximálisan $2^4 - 2$ elem lehetséges. Az elszigeteltség alapú érdekességi mutató

$$intr_{\text{Dong 2}}(R) = \frac{max_neighbor(R, r)}{|N(R, r)|}$$

.

Érdekességi mutatókkal szemben támasztott elvárások

Az érdekességi mutató formálisan próbálják megragadni, hogy milyen mértékben érdekes számunkra egy adott szabály. Az érdekességi mutatókkal szemben vannak elvárásaink, amelyeket szintén hasznos lenne formalizálni annak érdekében, hogy megállapítsuk az egyes mutatók hibáit és korlátait.

A három legfontosabb elvárás egy M érdekességi mutatóval szemben az alábbiak [117].

E1: $M = 0$, ha I és I' statisztikailag függetlenek.

E2: M monoton növő függvénye legyen $freq(I \cup I')$-nek, amennyiben $freq(I)$, $freq(I')$ változatlan.

E3: M monoton fogyó függvénye legyen $freq(X)$-nek, amennyiben $freq(I \cup I')$ és $freq(I \cup I' \setminus X)$ változatlan, ahol $X \in \{I, I'\}$.

Az érdekességi mutatók rendelkezhetnek még számos más tulajdonsággal. Ezek leírásához minden érdekességi mutatóra úgy tekintünk, mint egy O mátrixfüggvényre, amely az I, I' elemhalmazokhoz tartozó 2×2-es kontingencia-mátrixhoz rendel egy valós számot. Például a mátrix determinánsa egyenlő $supp(I \cup I')supp(\overline{I} \cup \overline{I'}) - supp(I \cup \overline{I'})supp(\overline{I} \cup I')$ kifejezéssel, amelyből egyszerűsítés után éppen a kovariancia n^2-szerese adódik. Az O mátrixfüggvény normált, amennyiben az értéktartománya megegyezik a $[-1,1]$ intervallummal.

Jelöljük S-sel a $[01;10]$ mátrixot. A következő tulajdonságokról beszélünk érdekességi mutatók kapcsán [142].

változószimmetria (T1): $O(K) = O(K^T)$, ami azt fejezi ki, hogy a mutató független attól, hogy melyik változó szerepel a feltételrészben és melyik a következményrészben.

sor/oszlop skála-invariancia (T2): Ha tetszőleges sort/oszlopot beszorzunk egy valós számmal, akkor mutató értéke ne változzon. Vannak esetek amikor jogos elvárás a skálainvariancia. Gondoljunk egy adatbázisra, amely diákok tanulmányi eredményeit tartalmazza. Az egyszerűség kedvéért tegyük fel, hogy a diákokat csak jó tanuló/rossz tanuló osztályokba soroljuk. Amennyiben az iskolát egy új, csak lány osztállyal bővítjük és az új lányok jó tanuló – rossz tanuló eloszlása megegyezik a meglévő lányok eloszlásával, akkor úgy gondoljuk, hogy nem kéne változnia semmi olyan mértéknek, amely a nem és a tanulmányi eredmények közötti kapcsolatot méri.

érték-invariancia (T3): Az értékinvariancia esetén mindegy, hogy melyik eseményt jelöljük 1-essel és melyiket 0-ával. Formálisan, $O(SM) = O(M)$ esetén beszélünk I szerinti érték-invarianciáról ($O(MS) = O(M)$ esetén pedig I' szerintiről).

inverzió-invariancia (T4): Ha mindkét attribútumnál felcseréljük az, hogy mit jelölünk 1-essel és mit 0-val és ennek következtében a mutató értéke nem változik, akkor a mutató inverzió-invariáns. Formálisan: $O(SMS) = O(M)$

Null-invariancia (T5): Ebben az esetben a mutató nem függ a kontingencia-táblázat 0,0-eleméhez tartozó értéktől.

A 8.1 táblázatban összefoglaljuk az ismert mértékekre vonatkozó tulajdonságokat.

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 3 féle 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:

$$\text{tej} \xrightarrow{80\%,4.8\%} \text{zabpehely}$$

$$\text{zsírszegény tej} \xrightarrow{80\%,1.2\%} \text{zabpehely}$$

név	tartomány	$E1$	$E2$	$E3$	$T1$	$T2$	$T3$	$T4$	$T5$
lift	$0 \cdots 1 \cdots \infty$		$\surd$	$\surd$	$\surd$				
empirikus korreláció	$-1 \cdots 0 \cdots 1$	$\surd$	$\surd$	$\surd$	$\surd$		$\surd$	$\surd$	
empirikus kovariancia	$-0.25 \cdots 0 \cdots 0.25$	$\surd$	$\surd$	$\surd$	$\surd$		$\surd$	$\surd$	
α	$0 \cdots 1 \cdots \infty$		$\surd$	$\surd$	$\surd$	$\surd$		$\surd$	
Yule Q	$-1 \cdots 0 \cdots 1$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	
Yule Y	$-1 \cdots 0 \cdots 1$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	$\surd$	
conviction	$0.5 \cdots 1 \cdots \infty$	$\surd$						$\surd$	
conviction*	$0.5 \cdots 1 \cdots \infty$	$\surd$		$\surd$				$\surd$	
Jaccard-koefficiens	$0 \cdots 1$		$\surd$	$\surd$	$\surd$				$\surd$
koszinusz mérték	$0 \cdots \pi$		$\surd$	$\surd$	$\surd$				$\surd$

8.1. táblázat. Érdekességi mutatók tulajdonságai

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.

9. fejezet

Funkcionális és közelítő függőségek

A *funkcionális függőség* az adatbázis-elmélészek által jól ismert fogalom. Kapcsolatot jelent egy relációs adatbázisban egyes attribútumok között. Informálisan az $X \to A$ funkcionális függőség azt jelenti, hogy az X oszlopok értékei egyértelműen meghatározzák az A oszlop értékét. Ilyen függőségre lehet példa amikor egy ember keresztneve egyértelműen meghatározza a nemét.

Ennek a fejezetnek a témája ilyen függőségek kinyerése. Képzeljünk el egy nagy adatbázist, amely különböző kémiai vegyületek és az azokkal végzett biológiai kísérletek eredményeit tartalmazza. Felbecsülhetetlen lehet olyan függőség kinyerése, ami azt mondja, hogy egy vegyület valamely minőségi paramétere –például rákkeltő hatása– függ a vegyület bizonyos struktúrális attribútumától. Az orvosi alkalmazások mellett egyre több területen merül fel az igény, hogy megvizsgáljuk, vannak-e kapcsolatok az attribútumok között. Funkcionális és közelítő függőségeket tipikusan sok attribútumot tartalmazó adatbázisokban keresünk.

Az adatbáziselmélet egyik alaptétele, hogy a funkcionális függőség sémaszintű fogalom. Ez azt jelenti, hogy a relációs séma és némi háttérinformáció segítségével megállapíthatóak a függőségek. Egy relációs séma pillanatnyi előfordulása alapján viszont soha nem tudjuk biztosan megállítani, hogy egy függőség fennáll-e, legfeljebb azt, ha nem.

Ezzel szemben mi a funkcionális függőségeket adatbányász szemmel nézzük; adott relációs tábla esetén meghatározzuk az összes olyan függőséget, ami fennállhat a relációban.

Mi a közös az egzakt asszociációs szabályokban és a funkcionális függőségekben? Már szóltunk arról, hogy a piaci-kosár modellnek megfelelő adatbázis felfogható egy, csak bináris értékeket tartalmazó relációs adatbázisnak. Az $X \to Y$ egzakt asszociációs szabály azt állítja, hogy amennyiben X értéke 1, Y értékét is tudjuk, szintén 1. Ha X értéke 0, akkor Y értékéről semmit sem tudunk. Egy $X \to Y$ funkcionális függőség ennél többet mond: ha X értékét tudjuk, akkor Y értékét is tudnunk kell, amennyiben volt már ilyen X értéket tartalmazó sor a relációban. A funkcionális függőségeket nem csak bináris táblákban kereshetjük, hanem tetszőleges típusú attribútumok között is.

A gyakorlati esetekben az adatbázisok tele vannak zajjal, kivételekkel, így az egzakt asszociációs szabályok száma általában nagyon kevés és szinte mindegyik annyira magától értetődő, hogy nem hordoz újdoságot. Sokkal érdekesebbek az 1-nél kisebb bizonyosságú szabályok. Hasonlót várunk a funkcionális függőségektől is, hiszen pusztán egyetlen sor tönkreteheti egy olyan függőség érvényességét, ami eddig több ezer soron keresztül érvényes volt. Ezért vezetjük be a *közelítő funkcionális függőség* fogalmát, ami alatt olyan funkcionális függőséget értünk, ami akkor állna fenn, ha a relációs tábla nem tartalmazna egyes sorokat. Egy függőség mértékének mérésére 3 mutatószám terjedt el, amiket a függőségek formalizálása után mutatunk be.

9.1. Funkcionális függőség

Legy R egy relációs séma, $X \subseteq R$ és $Y \in R$ és r az R felett értelmezett reláció. Jelöljük az r relációt ábrázoló tábla t-edik sorának X attribútumhalmazához tartozó elemeket $t[X]$-el, $\Pi(X)$-el az X oszlopaiban található elemek halmazát és legyen $c_X(x) = |\{t : t[X] = x\}|$ Azt mondjuk, hogy az $X \to Y$ funkcionális függőség *érvényes* (vagy fennáll) r-ben, ha minden $t, u \in r$ sorpárra ha $t[B] = u[B]$ minden $B \in X$-re, akkor $t[Y] = u[Y]$. Az $X \to Y$ funkcionális függőség *minimális*, amennyiben Y nem függ funkcionálisan X egyetlen valódi részhalmazától sem. A függőség *triviális*, amennyiben $Y \in X$.

9.2. Közelítő függőség

A közelítő függőségek a gyakorlati életben sokkal jellemzőbbek –és gyakrabban fordulnak elő–, mint a funkcionális függőségek, hiszen a gyakorlatban szinte mindig vannak kivételek, hibák, illetve az adatok valamilyen zajt tartalmazhatnak. A *közelítő függőség* egy olyan függőség, ami "majdnem" igaz r-ben. A "majdnem" persze nem matematikai fogalom, pontosan definiálnunk kell, mi az a kis hiba, amitől még eltekintünk függőség megállapításánál. A szakirodalomban egy szabály közelítőségének megállapítására több mérték is elterjedt.

A g_3 hiba: Egy függőség g_3 hibája azon sorok számán alapszik, amelyeket el kellene távolítani, hogy a megmaradt táblában a szabály funkcionális függőség lehessen. Formálisan: az $X \to Y$ függőség g_3-as hibája:

$$g_3(X \to Y) = 1 - \frac{\max\{|s| \big| s \subseteq r \text{ és } X \to Y \text{ funkcionális függőség } s\text{-ben}\}}{|r|}.$$

A g_3 hiba eléggé természetesnek tűnik: a hiba mértéke azon sorok aránya, amelyek kivételt jelentenek a függőség fennállásánál. Adott $0 \leq \varepsilon \leq 1$ hibaküszöb esetén, $X \to Y$ közelítő függőség, amennyiben $g_3(X \to Y)$ legfeljebb ε.

A τ' hiba: Legyen $pdep(Y)$ annak valószínűsége, hogy két véletlenszerűen megválasztott sornak megegyezik Y attribútuma, $pdep(Y|X)$ pedig ugyanez a valószínűség feltéve, hogy X attribútumaik megegyeznek. Könnyű végiggondolni, hogy $pdep(Y) = \sum_{y \in \pi(Y)} \frac{c_Y(y)^2}{|r|^2}$ és $pdep(Y|X) = \sum_{x \in \pi(X)} \sum_{y \in \pi(Y)} \frac{c_{X \cup Y}(x,y)}{c_X(x)} \frac{c_{X \cup Y}(x,y)}{|r|}$. A függőség τ értéke a két valószinűség különbségének normált értéke. Ez adja meg, hogy mennyire lehet megjósolni Y értékét X alapján. A τ' hiba ennek az értéknek a komplementerével egyezik meg.

$$\tau'(X \to Y) = 1 - \tau(X \to Y) = \begin{cases} 0 & \text{ha } |\Pi(Y)| = 1 \\ \frac{1 - pdep(Y|X)}{1 - pdep(Y)} & \text{különben.} \end{cases}.$$

Információ Függőségen alapuló hiba (IFD): Az entrópia és feltételes entrópia fogalmát már tisztáztuk a 2.4.2 részben. Megjegyezzük, hogy a feltételes entrópiát *információ függőség*nek is szokás nevezni. Felhasználva a $\mathcal{H}(Y|X) \leq \mathcal{H}(Y)$ tényt az információ függőségen alapuló hiba definiálható:

$$IFD(X \to Y) = \begin{cases} 0 & \text{ha } \mathcal{H}(Y) = 0 \\ \frac{\mathcal{H}(Y|X)}{\mathcal{H}(Y))} & \text{különben.} \end{cases}$$

Természetesen az entrópia kiszámításánál a valószínűségek helyett relatív gyakoriságokkal dolgozunk.

Mindhárom mértékre igaz, hogy 0 és 1 közé esik, továbbá az, hogy értéke 0, amennyiben $X \to Y$ egy érvényes funkcionális függőség. A mértékek közül nem lehet meghatározni, hogy melyik tükrözi legjobban a valóságot. Talán a g_3 hiba terjedt el leginkább, a fejezet további részében is ezt fogjuk használni. Matematikus szemmel az egyes hibákra vonatkozóan az alábbi állításokat lehet bebizonyítani.

- A $g_3(X \to Y)$ akkor és csak akkor veszi fel a maximumát, ha $|\Pi(X)| = 1$ és $|\Pi(Y)| = |r|$.
- Az $IFD(X \to Y)$ és $\tau'(X \to Y)$ akkor és csak akkor veszi fel 1 értéket, ha $(X \to Y)$ nem érvényes függőség továbbá X és Y a következő értelemben függetlenek. Minden $x \in \Pi(X)$ és $y \in \Pi(Y)$-ra $c_{X \cup Y}(x,y) = \frac{c_X(x)c_Y(y)}{|r|}$.
- Tetszőleges X, Y attribútumokra a $g_3(X \to Y)$, $\tau'(X \to Y)$, $IFD(X \to Y)$ hibák között a különbség -1 és 1 között bármilyen értéket felvehet. Ezt egyedül a $g_3(X \to Y) \leq \tau'(X \to Y)$ egyenlőtlenség korlátozza.

Az utolsó állítás azt mondja, hogy a hibák közötti különbségek tetszőleges nagyok lehetnek. A nagy különbségek e tétel bizonyításának céljából mesterkélten előállított relációkra igazak. A gyakorlatból vett adatok azt mutatják, hogy a hibaértékek általában csak kicsit különböznek egymástól.

Térjünk most vissza az eredeti célunkhoz és nézzük meg, hogyan lehet kinyerni egy adott relációból az érvényes funkcionális és közelítő függőségeket.

9.3. TANE Algoritmus

A TANE algoritmus [67, 68] két lépésből áll. Először partíciókat nyer ki, majd ezekből származtatja a függőségeket. Tisztázzuk, mit jelentenek a partíciók és milyen összefüggésbe hozhatók a függőségekkel.

Két sor, t és u az X attribútumhalmaz szerint *ekvivalens*, amennyiben $t[Z] = u[Z]$ minden $Z \in X$-re. Tetszőleges attribútumhalmaz a sorokat *ekvivalencia osztály*okba osztja. Jelöljük a t sor X szerinti ekvivalencia osztályát $[t]_X$-el ($[t]_X = \{u \in r | t[A] = u[A], \forall A \in X\}$). A $\pi_X = \{[t]_X | t \in r\}$ halmaz r-nek egy X szerinti *partíciója*. Tehát π_X a sorok diszjunkt halmazainak gyűjteménye. Jelöljük a π partíció ekvivalencia osztályainak számát $|\pi|$-vel!

Nézzük a következő táblázatban bemutatott relációt. Az A attribútum értéke 1 az első két sorban, így $[t_1]_{\{A\}} = [t_2]_{\{A\}}$, az A szerinti teljes partíció pedig $\pi_{\{A\}} = \{\{1,2\},\{3,4,5\},\{6,7,8\}\}$. A $\{B,C\}$-re vonatkozó partíció: $\pi_{\{B,C\}} = \{\{1\},\{2\},\{3,4\},\{5\},\{6\},\{7\},\{8\}\}$ és $\pi_{\{B\}} = \{\{1\},\{2,3,4\},\{5,6\},\{7,8\}\}$.

9.1. definíció. *A π partíció a π' partíció finomítása (vagy másként π finomítja π'-t), amennyiben minden ekvivalencia osztály π-ben részhalmaza π' valamely ekvivalencia osztályának.*

Tegyük fel, hogy π_X a π_Y finomítása és vegyünk egy tetszőleges $[t]_X$ ekvivalencia osztályt π_X-ből. Az ekvivalencia osztály definíciójából adódik, hogy mindazon sorok, amelyek az X attribútumot tekintve megegyeznek t-vel, $[t]_X$-ben vannak. A finomítás definíciójából adódik, hogy ezek egyben $[t]_Y$ ekvivalencia osztályában is benne vannak, tehát Y attribútum szerinti értékük megegyezik. Igaz tehát a következő lemma.

sor Id.	A	B	C	D
1	1	a	$	virág
2	1	A	@	tulipán
3	2	A	$	nárcisz
4	2	A	$	virág
5	2	b	@	liliom
6	3	b	$	orchidea
7	3	C	@	virág
8	3	C	#	rózsa

9.2. lemma. *Az $X \to Y$ funkcionális függőség akkor és csak akkor érvényes, ha π_X finomítása π_Y-nak.*

Nézzünk két példát! Az $\{B,C\} \to A$ érvényes, hiszen összehasonlítva $\pi_{\{B,C\}}$-t és $\pi_{\{A\}}$-t láthatjuk, hogy az előbbi finomítása az utóbbinak. Ezzen szemben a $\{A\} \to B$ nem érvényes, hiszen a $[t_3]_{\{A\}} = \{3,4,5\}$ ekvivalencia osztályt $\pi_{\{B\}}$ egyik ekvivalencia osztálya sem tartalmazza.

Hasonlóan könnyű bebizonyítani a következő lemmát.

9.3. lemma. *Az $X \to Y$ funkcionális függőség akkor és csak akkor érvényes, ha $|\pi_X| = |\pi_{X \cup \{Y\}}|$.*

Emlékszünk, hogy egy függőség g_3-as hibája azon sorok aránya az összes sorhoz, amelyeket törölni kellene, hogy a függőség érvényes legyen. A $g_3(X \to Y)$-t könnyű kiszámítani π_X és $\pi_{X \cup \{Y\}}$ alapján. π_X tetszőleges ekvivalencia osztálya $\pi_{X \cup \{Y\}}$ néhány ekvivalencia osztályának uniója. 1 osztály kivételével az összes osztály sorait törölni kellene ahhoz, hogy az $X \to Y$ érvényes legyen. Akkor kell a legkevesebb sort törölni, ha az 1 kivételes osztály az, amelyik a legtöbb sort tartalmazza. Ezek alapján:

$$g_3(X \to Y) = 1 - \frac{1}{|r|} \sum_{c \in \pi_X} \max\{|c'| \,\big|\, c' \in \pi_{X \cup \{Y\}} \text{ és } c' \subseteq c\}.$$

A továbbiakban a partíciók hatékony ábrázolásával és a g_3 hiba gyors közelítésével foglalkozunk. Ehhez meg kell ismerkednünk a *kulcs* és a *szuperkulcs* fogalmakkal. Ezeket az adatbáziselméleti szakemberek által jól ismert alapfogalmakat a megszokottól eltérően definiáljuk.

Az attribútumok egy halmaza *szuperkulcs* abban az esetben, ha nincs két olyan sor a relációban, ahol ezen attribútumhalmaz értékei páronként mind megegyeznek. Az X tehát akkor szuperkulcs, amennyiben π_X csak egyelemű ekvivalencia oszályokból áll. Az X attribútumhalmaz akkor *kulcs*, ha egyetlen valódi részhalmaza sem szuperkulcs.

Jelöljük $g_3(X)$-el azon minimális sorok arányát, melyeket eltávolítva X szuperkulcs lenne! Amennyiben $g_3(X)$ kicsi, akkor X egy közelítő szuperkulcs. A $g_3(X)$ könnyen számítható π_X ismeretében:

$$g_3(X) = 1 - \frac{|\pi_X|}{|r|}.$$

A partíciók kompaktabb reprezentációja érdekében helyettük az ún. *redukált partíciók*kal dolgozunk. Egy redukált partíciót az eredetiből úgy kaphatjuk meg, hogy töröljük abból az 1-elemű ekvivalencia osztályokat. A π_X redukált partícióját $\widehat{\pi}_X$-el jelöljük. A függőség bal oldalán található attribútumhalmaz partíciójának 1-elemű ekvivalencia osztályai azért nem jelentősek, mert egyetlen függőség érvényessége sem múlik azokon. Jogos tehát, hogy ezeket ne vegyük figyelembe.

Könnyű meggondolni, hogy az 1-elemű osztályok a finomítás relációra sincsenek hatással, így igaz a 9.2-es lemma. Ezzel szemben a 9.3-es lemma nem feltétlenül igaz, hiszen lehet, hogy $|\widehat{\pi}_X| = |\widehat{\pi}_{X\cup\{A\}}|$ annak ellenére, hogy $|\pi_X| \neq |\pi_{X\cup\{A\}}|$. Mivel $g_3(X) = g_3(Y)$ akkor és csak akkor ha $|\pi_X| = |\pi_Y|$ ezért a 9.3-es lemmát helyettesíthetjük az alábbival:

9.4. lemma. *Az $X \to Y$ funkcionális függőség akkor és csak akkor érvényes, ha $g_3(X) = g_3(X \cup \{Y\})$.*

A $g_3(X)$ értékét könnyen megkaphatjuk a redukált partíciókból a következő egyenlőség felhasználásával:

$$g_3(X) = \frac{||\widehat{\pi}_X|| - |\widehat{\pi}_X|}{|r|},$$

ahol $||\widehat{\pi}_X||$ az ekvivalencia osztályok méreteinek összegét jelöli.

A $g_3(X \to Y)$ számítása $O(|r|)$ időben végezhető, habár ez bizonyos esetekben elkerülhető, hiszen

$$g_3(X) - g_3(X \cup \{Y\}) \leq g_3(X \to Y) \leq g_3(X).$$

Amennyiben $g_3(X) - g_3(X \cup \{Y\}) > \varepsilon$ vagy $g_3(X) < \varepsilon$, akkor szükségtelen kiszámítani $g_3(X \to Y)$-t, hogy megtudjuk $X \to Y$ érvényes-e.

A TANE algoritmus az APRIORI sémára épül. A jól ismert „gyakori termékhalmaz minden részhalmaza gyakori” szabályhoz hasonlóan a mi esetünkben az igaz, hogy amennyiben $X \to Y$ nem érvényes, akkor $X' \to Y$ sem az, ahol $X' \subseteq X$. A függőségek bal oldalából hálót építhetünk fel. A háló élei és a nemtriviális függőségek között egyértelmű kapcsolatot vonhatunk: az X és az $X \cup \{Y\}$ halmazok között vezető él az $X \to Y$ függőséget reprezentálja. A hálóban egy szintenként haladó algoritmussal meghatározhatjuk azt a határvonalat, amely a minimális függőségeket jelképezi. A szokásos módon a keresést az egyelemű halmazokkal kezdjük majd egyre nagyobbakat vizsgálunk. Az algoritmus pszeudókódja a következőkben olvasható, az egyes segédfüggvények részletezése további előkészítést kíván.

```
Bemenet: r reláció,
Kimenet: minimális funkcionális függőségek halmaza.

  L_0 := {∅}
  C^+(∅) := R
  L_1 := {{A}|A ∈ R}
  i := 1
  while L_i ≠ ∅
     FÜGGŐSÉG_SZÁMÍTÁSA(L_i, C^+)
     TÖRLÉS(L_i, C^+)
     i := i+1
     L_i ← APRIORI_jelölt_generál(L_{i-1});  //Új szint
```

9.1. ábra. TANE algoritmus

Az APRIORI alapú algoritmusok ereje abban rejlik, hogy hatékonyan járják be a hálót: ha egy szint alapján következtetni lehet, hogy a következő szinten nincs érvényes szabály, akkor arra nem folytatjuk a keresést. Amikor a hálóban egy szintet feljebb lépünk, meg kell határoznunk az újonnan

vizsgált attribútumhalmazok partícióit. Ehhez nem kell végigolvasnunk a teljes adatbázist, mert az előző rétegek partícióiból ki tudjuk számítani a kérdéses partíciót.

Legyen π' és π'' partíciók *szorzata* az a legkevésbé finomított partíció, ami még finomítja π' és π'' partíciókat A szorzatpartíciót jelöljük $\pi' \cdot \pi''$-al. A következő lemma igaz.

9.5. lemma. *Tetszőleges $X, Y \subseteq R$ attribútumhalmazokra $\pi_X \cdot \pi_Y = \pi_{X \cup Y}$.*

A partíciók alapján meg tudjuk határozni az érvényes funkcionális függőségeket, hiszen $|\pi_X|$-ból kiszámítható a $g_3(X)$, ami a 9.4-as lemma alapján segítségünkre lehet egy függőség érvényességének eldöntésénél.

Amikor az X halmazt dolgozzuk fel, akkor a $X \setminus \{Y\} \rightarrow Y$ függőség érvényességét vizsgáljuk, ahol $Y \in X$. A TANE algoritmusnak nem célja az összes függőség kinyerése, csak a minimálisaké. Az $X \setminus \setminus \{Y\} \rightarrow Y$ függőség minimalitásának eldöntéséhez ellenőriznünk kell, hogy van-e olyan részhalmaza X-nek, amelyre érvényes az $X' \setminus \{Y\} \rightarrow Y$ függőség. Jelöljük $C(X)$-el azon attribútumokat, amelyek nem függenek X egyetlen valódi részhalmazától sem vagy nem elemei X-nek. A $C(X)$ halmazt *X jobb oldali jelölt*halmazának hívjuk, formálisan:

$$C(X) = \{Y \in X | X \setminus \{Y\} \rightarrow Y \text{ nem érvényes}\} \cup R \setminus X.$$

Ahhoz, hogy eldöntsük, $X \setminus \{Y\} \rightarrow Y$ függőség minimális-e, teljesülnie kell annak, hogy Y eleme minden $C(X')$-nek, ahol X' eggyel kisebb részhalmaza X-nek.

Nézzünk egy példát. Legyen $X = \{A, B, C\}$ és $\{C\} \rightarrow A$ érvényes függőség. Mivel $\{C\} \rightarrow A$ érvényes, ezért $A \notin C(\{A, C\}) = C(X \setminus \{B\})$, ami alapján $\{B, C\} \rightarrow A$ nem lehet minimális.

A következő, könnyen bizonyítható lemma figyelembe vételével a jobb oldali jelöltek halmazát tovább csökkenthetjük, anélkül, hogy minimális függőséget veszítenénk.

9.6. lemma. *Legyen $Z \in X$ és $X \setminus \{Z\} \rightarrow Z$ érvényes függőség. (1) Ha $X \rightarrow Y$ érvényes, akkor $X \setminus \{Z\} \rightarrow Y$ is az. (2) Ha X szuperkulcs, akkor $X \setminus \{Z\}$ is az.*

A fentiekből következik, hogy ha $X \setminus \{Z\} \rightarrow Z$ érvényes függőség, és X megjelenik egy másik függőség bal oldalán, akkor innen Z-t törölhetjük, a függőség érvényessége megmarad. További ötleteket felhasználva eljuthatunk a *redukált jobb oldali jelöltek* definíciójáig:

$$C^+(X) = \{Y \in R | \forall Z \in X, X \setminus \{Y, Z\} \rightarrow Z \text{ nem érvényes}\}.$$

A következő lemma azt mondja ki, hogy ezeket a redukált jobb oldali jelölthalmazokat felhasználhatjuk egy függőség minimalitásának eldöntéséhez.

9.7. lemma. *Legyen $Y \in X$ és $X \setminus Y \rightarrow Y$ érvényes függőség. Az $X \setminus Y \rightarrow Y$ akkor és csak akkor lehet minimális, ha minden $Z \in X$-re $Y \in C^+(X \setminus Z)$ feltétel teljesül.*

A fentiek alapján már megadhatjuk a `FÜGGõSÉG_SZÁMÍTÁSA` eljárás lépéseit. Az érvényesség eldöntéséhez a 9.4-as lemmát használjuk. A `FÜGGŐSÉG_SZÁMÍTÁSA` eljárás az $C^+(X)$ halmazokat is meghatározza és belátható (lásd a [67] cikk függelékét), hogy ezt helyesen teszi.

Ha a keresés során kulcsra bukkanunk, akkor további törlési lehetőségeink vannak. Tudjuk, hogy az $X \rightarrow \{Y\}, Y \notin X$ érvényességét az $X \cup \{Y\}$ feldolgozása során vizsgáljuk, hiszen szükségünk van $\pi_{X \cup \{Y\}}$-re. Ha X szuperkulcs, akkor $X \rightarrow Y$ mindig érvényes, így nincs szükségünk $X \cup \{Y\}$-ra.

Tegyük fel, hogy X szuperkulcs, de nem kulcs. Ekkor $X \rightarrow Y, Y \notin X$ nem lehet minimális, sőt, ha $Z \in X$-re $X \setminus \{Z\} \rightarrow Z$ érvényes, akkor a 9.6-es lemma miatt $X \setminus \{Z\}$ is szuperkulcs és nincs szükségünk

```
Bemenet:  L_l  l-edik szintu jelöltek,
Kimenet: l-edik szintu minimális funkcionális függőségek halmaza.

1   for all X ∈ L_l do C+(X) ← ∩_{Y∈ X} C+(X \ {Y})
2   for all X ∈ L_l do
    {
3      for all Y ∈ X ∩ C+(X) do
4         if X \ {Y} → Y érvényes then
5            kimenet X \ {Y} → Y
6            C+(X) ← C+(X) \ Y
7            C+(X) ← C+(X) \ (R \ X)
    }
```

9.2. ábra. A FÜGGŐSÉG_SZÁMÍTÁSA eljárás

```
Bemenet:  L_l  l-edik szintu jelöltek,
Kimenet:  L_l  csak a lényeges jelöltek,

   for all X ∈ L_l do
   {
      if C+(X) = ∅ then L_l ← L_l \ X
      if X (szuper)kulcs then
         for all Y ∈ C+(X) \ X do
            if Y ∈ ∩_{Z∈ X} C+(X ∪ {Y} \ {Z}) then kimenet X → Y
         L_l ← L_l \ X
   }
```

9.3. ábra. A TÖRLÉS eljárás

π_X kiszámítására ahhoz, hogy az $X \setminus \{Z\} \to Z$ érvényességét eldöntsük. Következésképpen az összes olyan szuperkulcsot törölhetjük a vizsgálandó elemek közül, amelyek nem kulcsok. A TÖRLÉS lépés ennek alapján felírható.

Az első feltétel szerint X-et akkor töröljük, ha $C^+(X) = \emptyset$. A második feltétel szerint pedig akkor, ha X kulcs. Ez utóbbi esetben előfordulhat, hogy érvényes minimális függőségekre bukkanunk. A következő lemma garantálja azt, hogy az algoritmus megtalálja ezeket a szabályokat.

9.8. lemma. *Legyen X szuperkulcs és $Y \in X$. Az $X \setminus \{Y\} \to Y$ függőség akkor érvényes és minimális, amennyiben $X \setminus \{Y\}$ kulcs és minden $Z \in X$-re fennáll, hogy $Y \in C^+(X) \setminus \{Z\}$.*

A TANE algoritmust könnyű adaptálni közelítő függőségek kinyerésére. Ehhez mindössze 2 sort kell megváltoztatni a FÜGGŐSÉG_SZÁMÍTÁSA eljárásban. Magától értetődő, hogy a

```
4          if X \ {Y} → Y érvényes then
```

feltétel helyett a

```
4'         if g_3(X \ {Y} → Y) ≤ ε then
```

feltételt kell alkalmazni. Ezen kívül a

7 $$C^+(X) \leftarrow C^+(X) \setminus (R \setminus X)$$

sort a

7' $$\texttt{if } g_3(X \setminus \{Y\} \rightarrow Y) = 0 \texttt{ then}$$
$$C^+(X) \leftarrow C^+(X) \setminus (R \setminus X)$$

sorra kell cserélni.

Az eddigiekhez hasonlóan elmondhatjuk, hogy legrosszabb esetben a TANE algoritmus futási ideje és memória szükséglete az attribútumok számával exponenciális, és a sorok számával lineáris. A gyakorlatban azonban a helyzet jóval kedvezőbb, mivel a függőségekben szereplő attribútumok száma kicsi, s így az alulról induló, szintenként haladó algoritmus gyorsan megtalálja a függőségeket.

10. fejezet

Osztályozás és előrejelzés

10.1. Bevezetés

Ismeretlen, előre nem megfigyelhető változók, attribútumok értékének előrejelzése más ismert, megfigyelhető változók, attribútumok ismeretében régóta aktív kutatás tárgyát képezi. A kérdés gyakorlati jelentőségét nehéz lenne túlértékelni. Ebben a fejezetben vázlatosan ismertetjük, hogy miként alkalmazhatók a statisztika és gépi tanulás területén kifejlesztett módszerek az adatbányászatban[1].

A megnevezések tisztázása érdekében előrebocsátjuk, hogy a tanulmányban akkor beszélünk előrejelzésről (predikcióról), 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. Fogalmaink szerinti előrejelzést és klasszifikációt a statisztikai irodalom általában regressziószámítás, valamint diszkriminancia elemzés és klasszifikáció néven illeti. A gépi tanulás területén az eljárásokat összefoglalóan felügyelt tanulásnak (supervised learning) nevezik.

Az adatbányászatban leggyakrabban alkalmazott előrejelző és klasszifikáló módszerek a következők:

I. Döntési fák

II. Lineáris és logisztikus regresszió

III. Mesterséges neurális hálózatok

IV. Naiv bayesi klasszifikáció és bayesi hálózatok

V. Genetikus algoritmusok

VI. Asszociáció szabályokra támaszkodó technikák

VII. Fuzzy következtetés

Mindegyik eljárásról elmondható, hogy (legalább) két lépcsőben működik. Először az ún. tanító adatbázison felépítjük a modellt, majd később azt alkalmazzuk olyan új adatokra, amelyeken a magyarázott változó értéke nem ismert, de ismerni szeretnénk. Amikor előrejelző, vagy klasszifikáló módszert választunk a következő tulajdonságait célszerű figyelembe venni:

[1] Ez a fejezet Sarlós Tamás és Bodon Ferenc közös munkája.

– Előrejelzés teljesítménye: Milyen értékes információt ad számunkra a modell a nem megfigyelhető magyarázó változóról (lásd 10.2 szakasz)?

– Gyorsaság: A modell előállításának és használatának időigénye.

– Robusztusság: Érzékeny-e a modell hiányzó, vagy outlier adatokra.

– Skálázhatóság: Használható-e a modell nagyon nagy adathalmazokra is?

– Értelmezhetőség: Kinyerhetünk-e az emberek számára értelmezhető tudást a modell belső szerkezetéből?

– Skála-invariancia: A klaszterzés lehetetlenség-elméletét adatpálva 11.1 skála-invariánsnak hívunk egy osztályozó eljárást, ha a módszer kimenete nem változik abban az esetben, ha tetszőleges intervallum típusú magyarázó változó helyett annak $\alpha > 0$-szorosát vesszük.

Az adatbányász közösség leginkább a korábban is ismert előrejelző 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 előrejelzők teljesítményének értékelésével foglakozunk, majd az eljárásokat ismertetjük. A hagyományos statisztikai módszerek (diszkriminancia analízis, lineáris regresszió, lásd. pl.: [73] ismertetésétől eltekintünk, helyettük inkább az „egzotikusabbakra” koncentrálunk: a döntési fák, a mesterséges neuronhálózatok, a Bayes-hálózatok, és négy további eljárás főbb jellemzőit mutatjuk be [77], [63], [56] és [103] írások alapján.

10.2. A klasszifikáció teljesítményének méréséről

Általános módon egy klasszifikáló vagy előrejelző módszer teljesítményét várható hasznosságával mérhetjük. Legyen a magyarázandó változó Y, a magyarázó változ(ók) pedig X, eljárásunkat jelöljük f-fel. Ekkor célunk $E\left[U\left(Y, f\left(X\right)\right)\right]$ maximalizálása, ahol $U(y, \widehat{y})$ jelöli az előrejelzett $\widehat{y}$ hasznosságát, miközben a valódi érték y. A feladatot fordítva, minimalizálásként is megfogalmazhatjuk, ha $U = -L$ valamilyen elkerült veszteséget mér. Mivel a várható érték változóiban additív és a konstanssal való eltolás nem változtat az optimalizáláson, ezért $L\left(y, y\right) = 0$ feltehető. A hibát a gyakorlatban egy távolságfüggvénnyel definiálják (lásd 3.2 rész). Amennyiben a magyarázandó változó intervallum skálán mérhető, akkor a lgelterjedtebb megoldás a négyzetes hiba alkalmazása. Bináris Y esetén *bináris osztályozásról* beszélünk.

Klasszifikáció esetén a fenti várható érték egyszerűen a téves döntések valószínűségekkel súlyozott összege. Ha a várható értéket meghatározó valódi eloszlásokat ismernénk, akkor megtalálható a legjobb előrejelző / klasszifikáló. Például (azonos kovarianciájú) többdimenziós normális eloszlásokat feltételezve egyszerű kvadratikus (lineáris) döntési szabályokat kapunk [144], [73]. Az eloszlás paramétereit általában még akkor is becsülnünk kell, ha feltételezhető / feltételezünk egy adott típusú eloszlás.

Az adatbányászat területén a normalitás nem reális feltevés (gondoljunk a sok nominális változóra). Az adatbányászati módszerek nem élnek feltevésekkel az eloszlással kapcsolatban.

Ugyanakkor a módszerek összetettségük folytán – ha hagyjuk őket – képesek nem csak a tanító adatbázis szabályszerűségeit, hanem a mintaadatokban lévő egyedi hibákat és torzításokat is megtanulni (ami kifejezetten káros). Így általában pusztán a tanító adatbázis segítségével nem megalapozott

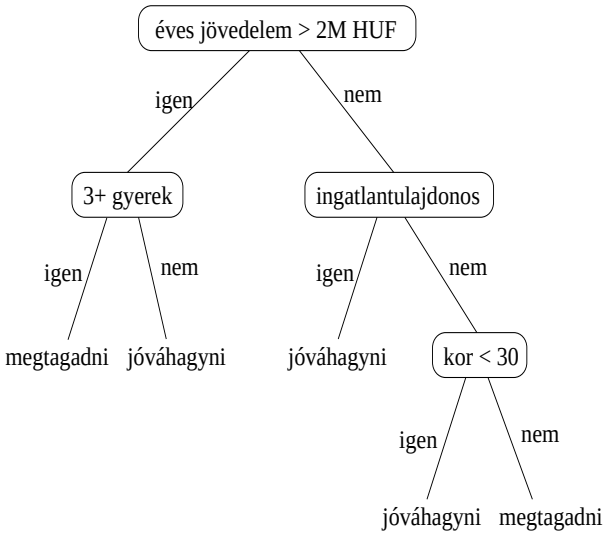
a várható haszon / költség nagyságát megbecsülni (vö. **??**. szakasz). A túlzott modellbonyolultság elkerülésére pl.: a regressziószámítás területén modellszelekciós kritériumok (módosított R^2, Akaike Schwartz, stb.), illetve heurisztikus eljárások (stepwise regresszió) állnak rendelkezésre.

Az adatbányászati modellekre – sajnos – ritkán állnak rendelkezésre olyan módszerek, amelyek segítségével az illeszkedés jóságáról statisztikai teszttel dönthetünk (a kivételeket lásd **??** szakaszban, ill. [20] cikkben). Egy lehetséges általános megközelítést Rissanen adott meg [2]. A „legrövidebb leírás”[2] elve szerint egy adathalmazt magyarázó elméletek közül az a leginkább elfogadhatóbb, amelynél összesen a legkevesebb bit szükséges a modell és az adatoknak a modell segítségével való leírásához.

Mégis a leggyakrabban alkalmazott módszer a következő. Adatainkat három részre osztjuk. Ugyanazon tanító adatokon több konkurens modellt építünk, majd az ellenőrző adathalmaz segítségével kiválasztjuk a legjobbat, amelyet alkalmazni fogunk. A végső modell teljesítményét, pedig egy – az előző kettőtől diszjunkt – teszt adatbázison mérjük. Ismételt mintavételezési technikákkal csökkenthetjük a fenti eljárás adatigényét, illetve több klasszifikáló eredményeinek kombinálásával is javítható az előrejelzés pontossága [63].

10.3. Döntési fák

A döntési fák alapötlete, hogy bonyolult összefüggéseket egyszerű döntések sorozatára vezet vissza. Egy ismeretlen minta 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 10.3. ábra.[3]



10.1. ábra. Döntési fa hitelbírálatra

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

[2] Minimum Description Length, MDL.

[3] Az ábrázolt döntési fa sem értékítéletet, sem valós hitelbírálati szabályokat nem tükröz, pusztán illusztráció.

azért előnyös, mert így a fák teljesítménye zaj jelenlétében sem romlik, 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.

Az is a döntési fák előnyős tulajdonsága, hogy a gyökérből egy levélbe vezető út mentén a feltételeket összeolvasva könnyen értelmezhető szabályokat kapunk a döntés meghozatalára, illetve hasonlóan egy laikus számára is érthető módon azt is meg tudjuk magyarázni, hogy a fa miért pont az adott döntést hozta. Továbbá a fák nagyméretű adathalmazokra is hatékonyan felépíthetők.

A döntési fák egyik fontos tulajdonsága, hogy egy csomópontnak mennyi gyermeke lehet. Nyilvánvaló, hogy egy olyan fa, amely pontjainak kettőnél több gyermeke is lehet mindig átrajzolható bináris fává. A legtöbb algoritmus ezért csak bináris fát tud előállítani.

10.3.1. A döntési fa előállítása

A fát a tanulóadatbázisból iteratívan állítjuk elő. Kiindulunk a teljes tanulóadatbázisból és egy olyan kérdést keresünk, aminek segítségével a teljes tanulóhalmaz jól szétvágható. 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. Egy csomópont leszármazottjaiban nem vizsgáljuk többé azt az attribútumot, ami alapján szétosztjuk a mintát.

A rekurziót akkor szakítjuk meg valamelyik ágban, ha a következő feltételek közül teljesül valamelyik:

- A csomópont elemei ugyanabba az osztályba tartoznak.
- Nincs több attribútum, ami alapján az elemeket tovább oszthatnánk. A csomóponthoz tartozó osztály ekkor az lesz amelyikhez a legtöbb tanítópont tartozik.
- Nem tartozik az adott csomóponthoz tanítópont.
- Az adott mélység elért egy előre megadott korlátot.

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.

A döntési fák előállítására a következő három fő algoritmus család ismert:

I. Interactive Dichotomizer 3 (ID3) család, jelenlegi változat C5.0[4]

II. Classification and Regression Trees (CART)[5]

III. Chi-squared Automatic Interaction Detection (CHAID)[6]

[4]Magyarul: Interaktív tagoló / felosztó
[5]Klasszifi káló és regressziós fák
[6]Khi-négyzet alapú automatikus interakció felismerés

10.3.2. Az ID3 algoritmus

Az ID3 az egyik legősibb és legismertebb osztályzó algoritmus. A tesztattribútum kiválasztásához az entrópia csökkenését alkalmazza. Ha Y egy l lehetséges értéket p_i $(i = 1, \ldots, l)$ valószínűséggel felvevő valószínűségi változó, akkor Y Shannon-féle entrópiáján a

$$H(Y) = H(p_1, \ldots, p_l) = -\sum_{j=1}^{l} p_j \log_2 p_j$$

számot értjük[7]. Az entrópia az információ-elmélet (lásd [33]) központi fogalma, és Y változó értékével kapcsolatos bizonytalanságunkat fejezi ki. Ha egy X változót megfigyelünk és azt tapasztaljuk, hogy értéke x_i, akkor Y-nal kapcsolatos bizonytalanságunk

$$H(Y|X = x_i) = -\sum_{j=1}^{l} P\left(Y = y_j|X = x_i\right) \log_2 P\left(Y = y_j|X = x_i\right)$$

nagyságú. Így ha lehetőségünk van X-t megfigyelni, akkor a várható bizonytalanságunk

$$H(Y|X) = \sum_{i=1}^{k} P(X = x_i) H(Y|X = x_i)$$

Eszerint X megfigyelésének lehetősége a bizonytalanság

$$I(Y, X) = H(Y) - H(Y|X)$$

csökkenését eredményezi, azaz X ennyi információt hordoz Y-ról. Az ID3 Y klasszifikálásakor olyan X attribútum értékei szerint ágazik szét, amelyre $I(Y, X)$ maximális, azaz $H(Y|X)$ minimális.

A bináris, döntési fákban a magyarázó X attríbutum tipusától függően kétféle feltételt hozunk létre. Intervallum és sorrend típus esetében $X \geq c$, ahol c egy olyan konstans, amit az X felvesz, kategória típus esetében $X \subseteq K$, ahol K az X által felvehető kategóriák halmazának részhalmaza. Az első esetben X felvett értékeivel lineárisan arányos feltételes entrópiát kell számítani, a másodikban pedig a felvett kategóriák számával exponenciális sokat (ugyanis egy n elemű halmaznak 2^n darab részhalmaza van).

10.3.3. Továbbfejlesztések

Míg az ID3 családba tartozó fák csak klasszifikációra, addig a másik kettő klasszifikációra és előrejelzésre is alkalmazható. A C5.0 és a CHAID fák kizárólag egyetlen attribútumra vonatkozó egyenlő, kisebb, nagyobb teszteket használnák a csomópontokban a döntésekhez, azaz a jellemzők terét téglatestekre vágják fel. A CART fák ferdén is tudnak vágni, attribútumok lineáris kombinációját is tesztelik. Míg a CART eljárás mindig bináris döntéseket használ a csomópontokban, addig egy nominális attribútumra egy C5.0 fa annyi felé ágazik, ahány lehetséges értéket az attribútum felvehet.

Talán a leglényegesebb különbség a különböző fák között, hogy mit tekintenek jó döntésnek, vágásnak. Nominális magyarázott változó esetén a CHAID eljárás – nevének megfelelően – a χ^2-tesztet használja. A CART metodológia a Gini-indexet minimalizálja. Ha egy n csúcs k lehetséges

[7] Az entrópia képletében $0 \cdot \infty$ megállapodás szerint 0-val egyenlő.

értékű attribútum szerinti szétvágásakor $c_1, \ldots, c_k$ gyerekei keletkeznek, akkor

$$Gini\,(n) = \sum_{i=1}^{k} p_i \left(1 - \sum_{j=1}^{l} p_{ij}^2\right)$$

ahol p_i az i. attribútum érték relatív gyakorisága az n szülő csúcsban lévő mintában, p_{ij} pedig a magyarázott változó j. lehetséges értékének relatív gyakorisága a c_i gyerekhez tartozó almintában. A Gini-index alapján tehát mindig olyan attribútumot keresünk, amely alapján a legnagyobb homogén osztály tudjuk leválasztani.

Ha a magyarázott 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 C5.0 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.

Tanácsos megvizsgálni, hogy nem fordul-e elő, hogy a generált C5.0 vagy CHAID fa egymás után ismételten kevés (2-3) 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.

Láttuk, hogy többféle mutatószám létezik a vágási kritérium kiválasztására. Ezek között nem létezik a legjobb. Bármelyikhez lehet készíteni olyan adatbázist, amelyet rosszul osztályoz a vágási kritériumot használó algoritmus.

10.3.4. Döntési fák ábrázolása

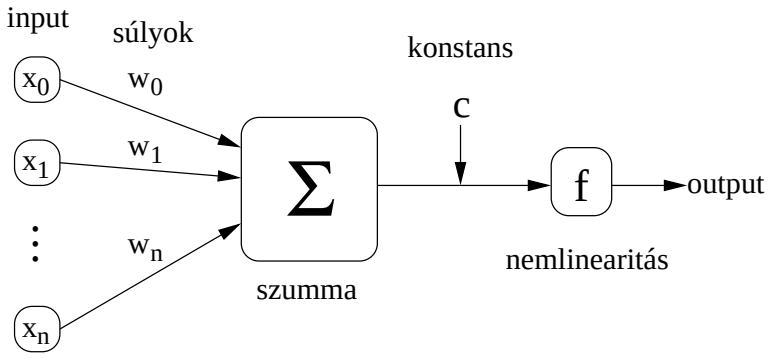
A döntési fa előállítása után két fontos kérdes szokott felmerülni. Egyrészt tudni szeretnénk, hogy melyik levélbe esik sok tanító pont, azaz melyek azok a szabályok, amelyek sok tanító pontra érvényesek. Másrészt látni szeretnénk, hogy a levelek mennyire jól osztályoznak; a tesztpontok közül (ha vannak 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 rendszzerében is), hogy minden levelet egy körcikkely reprezentál. A körcikkely nagysága arányos a levélhez tartozó tanító pontokkal, a szine pedig a levélhez tartozó szabály jóságát adja meg. Például minél sötétett a szín, annál rosszabb az osztályozás aránya. Egy ilyen ábrázolásra láthatunk pélfát a következő ábrán.

10.3.5. Mesterséges neurális hálózatok

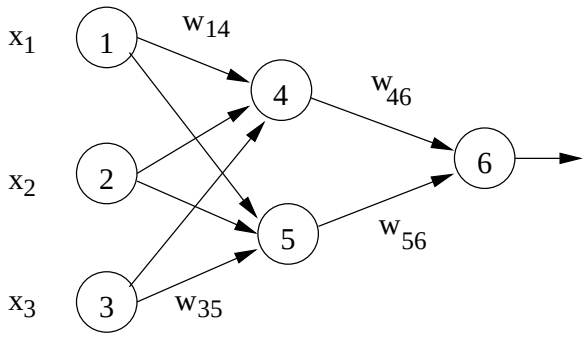
A hagyományos lineáris regressziós modellek nem igazán alkalmasak korlátos vagy diszkrét értékkészletű magyarázott változók előrejelzésére. Legegyszerűbb példaként tekintsük azt az esetet, ha

egy bináris változó értékét kell becsülnünk (csődbe megy-e a vállalat, vagy sem), vagy általánosabban, becsüljük meg a csődbemenetel valószínűségét. Az, hogy ez a valószínűség a magyarázó változók lineáris függvénye, nem túl valószínű. Ehelyett gondolhatunk arra, hogy a keresett valószínűséget egy eloszlásfüggvény értéke adja meg a magyarázó változók valamilyen lineáris kombinációjaként számított helyen. Azaz a lineáris modellünk kimenetét egy nemlineáris eloszlásfüggvényen vezetjük keresztül (lásd 10.2. ábra). Ha az eltérésváltozó eloszlásfüggvényének a log-Weibul eloszlást választjuk, akkor a logit (logisztikus, $\frac{P(Y=1)}{P(Y=0)} = \frac{1}{1+e^{-l(x)}}$), ha a normális eloszlást, akkor a probit ($f(x) = \phi(x)$) modellt kapjuk [25].



10.2. ábra. Logisztikus regresszió

A mesterséges neuronhálózatok – némileg az agyműködést utánzó biológiai analógiára is támaszkodva – a fenti gondolatot viszik tovább és több logisztikus regressziót kombinálnak, a logisztikus regresszió outputját másik logisztikus regresszió inputjaként felhasználva. A legnépszerűbb modell a többrétegű előrecsatolt neuronhálózat (lásd 10.3. ábra). Az első réteg csomópontjaiban (neuronok) az input (magyarázó változók, 1-3. neuronok) helyezkedik el, az outputot (magyarázott változókat) a legutolsó réteg kimenete (6. neuroné) 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. (A 10.3. ábrában 4-6. neuronok helyébe egy 10.3. ábra szerinti logisztikus regressziót kell képzeljünk.)



10.3. ábra. Többrétegű előrecsatolt neurális hálózat

Mind a logisztikus regresszió, mind a neurális hálózatok paramétereikben nem lineáris függvényapproximátornak tekinthetők. A tapasztalatok és az elméleti eredmények (lásd.: [56]) 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 módszerrel kereshetjük meg. A gradiens eljárások alapelve, hogy egy függvény maximum / minimum helyét úgy keresik meg, hogy egy kezdőpontból kiindulva a gradiens (derivált) irányában / a gradienssel ellentétes irányban mozdulunk el, majd az eljárást ismétlik.

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 visszaterjedés) eljárás szerint zajlik:

I. Az inputokból előrehaladva kiszámítjuk az outputok eredményét.

II. Az utolsó output rétegből rétegről rétegre visszafelé haladva a megfelelő gradiens szabály szerint módosítjuk w_{ij} értékeket.

A neuronhálók hátrányaként említhető, hogy a súlyok rendszere közvetlenül nem értelmezhető emberek számára. Nem tudjuk egyszerűen megindokolni, hogy mi alapján hozta meg a neuronháló a döntést. Egy hálózat tulajdonképpen egy fekete doboznak tekinthető a felhasználó szemszögéből. Sok területen nem elfogadható, ha egy módszer nem ad magyarázatot, ezért a neuronhálók alkalmazási köre erősen korlátozott. Ugyanakkor léteznek olyan eljárások, amelyek a neuronhálók súlyaiból emberek számára érthető, a döntéseket indokló szabályokat nyernek ki [63].

10.3.6. Bayesi hálózatok

A Bayes-tétel segítségével meghatározható az optimális (lásd 10.2. szakasz) klasszifikációs szabály. Jelöljük C_i-vel azt az eseményt a klasszifikálandó eset az i. osztályba tartozik. Az elemek megfigyelhető tulajdonságait X vektor írja le. Az egyszerűség kedvéért a tévedés költsége legyen minden esetben azonos. Ekkor egy ismeretlen, X tulajdonságú példányt abba az osztályba (i) érdemes (optimális) sorolni, amelyikre $P(C_i|X)$ maximális. A Bayes-szabály alapján

$$P(C_i|X) = \frac{P(X|C_i)\,P(C_i)}{P(X)}$$

Mivel $P(X)$ minden i-re konstans, ezért elegendő $P(X|C_i)\,P(C_i)$-t maximalizálni. $P(C_i)$ vagy a priori adott, vagy pedig a mintából a relatív gyakoriságokkal egyszerűen becsülhető. Így már „csak” $P(X|C_i)$-t kéne meghatározni. A naiv Bayes klasszifikáló feltételezése szerint egy osztályon belül az attribútumok eloszlása független, azaz

$$P\left((X_1, X_2, \ldots, X_l) = (x_1, x_2, \ldots, x_l)\,|C_i\right) = \prod_{j=1}^{l} P\left(X_j = x_j|C_i\right)$$

$P\left(X_j = x_j|C_i\right)$ valószínűségek szintén a mintából becsülhetők.

A Bayes-hálók (Bayesian belief 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

$$P\left((Z_1, Z_2, \ldots, Z_s) = (z_1, z_2, \ldots, z_s)\right) = \prod_{j=1}^{s} P\left(Z_j = z_j|par\left(Z_j\right)\right)$$

teljesül, ahol $par\left(Z_j\right)$ a Z_j csúcs szüleit (a gráfban közvetlenül belemutató csúcsok halmazát jelöli). Minthogy a háló struktúrája a teljes eloszlást leírja, ezért tetszőleges Z_j csúcsokat kijelölhetünk outputnak / előrejelzendőnek. Ha nincsenek rejtett változók, akkor a szükséges

$$P\left(Z_j = z_j | par\left(Z_j\right)\right)$$

valószínűségek közvetlen becsülhetők a mintából. Ha a háló rejtett változókat is tartalmaz, akkor a gradiens módszer egy változata alkalmazható. Végül olyan eljárások is ismertek, amelyek segítségével a hálózat topológiája a tanuló példákból kialakítható, nem feltétlenül szükséges azt előre megadni.

10.3.7. Egyéb módszerek

Az alább ismertetett módszereket ritkábban szokták adatbányászati céllal alkalmazni, mint a döntési fákat, mesterséges neuronhálózatokat vagy Bayes-i hálózatokat.

k-legközelebbi szomszéd módszere

A k-legközelebbi szomszéd módszere egy „lusta” klasszifikáló eljárás, amely nem épít modellt. Alapelgondolása, hogy a hasonló attribútumú objektumok hasonló tulajdonságokkal bírnak. A hasonlóságot (igazából a különbözőséget) a klaszterelemzésnél (lásd **??**. fejezet). is használt távolságfüggvénnyel mérjük. A tanuló adatbázist eltároljuk és amikor egy ismeretlen objektumot kell klasszifikálnunk, akkor megkeressük a távolságfüggvény szerinti k darab legközelebbi pontot, és az objektumot abba a kategóriába soroljuk, amely a legtöbbször előfordul a k szomszéd között (többségi szavazás). A módszer egyfajta lokális sűrűségfüggvény becslő eljárásnak is tekinthető. A magyarázó attribútumok számának növekedésével a (**??**. szakaszban tárgyalt) problémák léphetnek fel.

Genetikus programozás

A genetikus algoritmusok nem tekinthetők valódi klasszifikáló eljárásnak, hanem inkább egy módszernek arra, hogy az adatokat jól leíró modellt keressünk (optimalizációs eljárás) [56]. A különféle modellek egy populációt alkotnak, a modellek tulajdonságait / paramétereit ún. génekben kódoljuk és a biológiai evolúció mintájára olyan mechanizmusokat működtetünk, amelyek az életrevalóbb (adatokat jobban leíró) modellek túlélésének kedveznek (szelekció). A keresést a modellek kombinálása (génkeresztezés, crossover) és a paraméterek véletlenszerű változtatása (mutáció) teszi teljessé. Igazán nagyméretű feladatok megoldására az eljárás nagy számításigénye miatt nem alkalmas.

Szabályalapú technikák

Habár az asszociációs szabályok (lásd **??**. fejezet) nem feltétlenül fejeznek ki oksági kapcsolatokat, a tapasztalatok szerint klasszifikálók építésére is felhasználhatók [63]. Egy lehetséges megközelítés szerint olyan asszociációs szabályokat bányászunk, amelyek következményrésze a magyarázandó változó, majd a viszonylag sok speciális szabályból kevesebbet készítünk a hasonló feltételrészek összevonásával (ARCS eljárás). Más módszer szerint a magyarázott változó különböző értékeihez tartozó tanító esetekben külön-külön keresünk gyakori termékhalmazokat. Ha egy termékhalmaz gyakorisága gyökeresen eltér a két mintában, akkor azt feljegyezzük, mint jellemző tulajdonságot. Egy elem klasszifikálásakor pedig az elemben megfigyelhető jellemző termékhalmazok alapján döntünk. (CAEP módszer.)

Fuzzy logika

A fuzzy (életlen) logika célja a természetes nyelvekben mindennaposan használt bizonytalan fogalmak megragadása [56]. A hagyományos logika / halmazelmélet megközelítése szerint ha a jövedelmet három kategóriára (alacsony, közepes, magas) osztjuk fel, akkor egy adott jövedelem (pl.: havi X ezer forint) egy és csak egy kategóriába tartozik. Ugyanakkor minden ilyen felosztás meglehetősen önkényes, például ha 99000 Ft jövedelem alacsonynak számít, akkor a 100000 Ft miért közepes? A fuzzy logikában a kategóriák határa nem éles, a havi 99000 és 100000 Ft *valamilyen mértékben* egyszerre alacsony és közepes is. A fuzzy logika a klasszikus logika következtetési szabályait terjeszti ki ezen mértékekkel való számolásra és lehetővé teszi, hogy absztrakt fogalmakat kezeljünk.

11. fejezet

Klaszterezés

Klaszterezésen elemek csoportosítását értjük. Úgy szeretnénk a csoportosítást elvégezni, hogy a hasonló elemek ugyanazon, míg az egymástól eltérő elemek külön csoportba kerüljenek. 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 klasztereznék 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 kőrök és a többi lap). Klaszterezéskor tehát az adathalmaz mellett meg kell adnunk, hogy miként definiáljuk az elemek hasonlóságát, továbbá, hogy mi alapján csoportosítsunk (összefüggő alakzatokat keressünk, vagy a négyzetes hibát minimalizáljuk stb.).

A jóság egzakt definíciójának hiánya mellett nagy 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 ($\sum_{k=1}^{25}\mathcal{S}_{25}^{(k)} > 4\cdot 10^{18}$).

Szükség van azonban az elemek automatikus csoportosítására, így a problémákon túl kell lépni. Objektív definíciót kell adnunk az elemek hasonlóságának mértékére és a klaszterezés minőségére. Amennyiben megfelelő matematikai modellbe ágyaztuk a feladatot, lehetőség nyílik olyan algoritmusok megkeresésére, amelyek jól és gyorsan oldják meg a feladatot. Ezekről az algoritmusokról és a hasonlóság megállapításának módjáról szól ez a fejezet.

Klaszterezés során csoportokba, osztályokba soroljuk az elemeket, tehát osztályozást végzünk. Az eredeti osztályozási feladattól (lásd előző fejezet) az különbözteti meg a klaszterezést, hogy nincs megadva, hogy melyik elem melyik osztályba tartozik (tehát nincs egy tanító, aki helyes példákkal segíti a tanulásunkat), ezt nekünk kell meghatároznunk. Ezért hívják a klaszterezést *felügyelet nélküli tanulás*nak (unsupervised learning) is.

A klaszterezés az adatbányászat legrégebbi és leggyakrabban alkalmazott része. Számos helyzetben alkalmazzák, így csoportosítanak weboldalakat, géneket, betegségeket stb. Az egyik legdinamikusabban fejlődő terület azonban a személyre szabott szolgáltatásoké, ahol az ügyfeleket, ill. vásárlókat kategorizálják, és az egyes kategóriákat 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 fejezet további részében először egy meghökkentő kutatási eredményről számolunk be, majd a hasonlóság meghatározásáról beszélünk végül rátérünk a legismertebb klaszterező algoritmusokra.

11.1. Egy lehetetlenség-elmélet

A klaszterezés az egyik legnehezebben átlátható adatbányászati terület. Napról napra újabb és újabb cikkek jelennek meg különböző „csodaalgoritmusokról”, amelyek szupergyorsan és helyesen csoportosítják az elemeket. Elméleti elemzésekről általában kevés szó esik –azok is gyakran elnagyoltak, sőt hibásak–, viszont az algoritmust igazoló teszteredményekből nincs hiány. Mintha minden algoritmusnak illetve szerzőnek létezne a maga adatbázisa, amivel az eljárás remek eredményeket hoz.

Ebben a káoszban kincset érnek a helyes irányvonalak megvilágításai és a megalapozott elméleti eredmények. Egy ilyen gyöngyszem Jon Kleinberg munkája, amit az „An Impossibility Theorem for Clustering (A Klaszterezés Lehetetlenség-elmélete)” című cikkében publikált 2002-ben [81]. 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 csökkentem, illetve külön csoportban lévő elempárok távolságát növelem, akkor az újonnan kapott távolságok alapján működő eljárás az eredetivel megegyező csoportosítást adja.

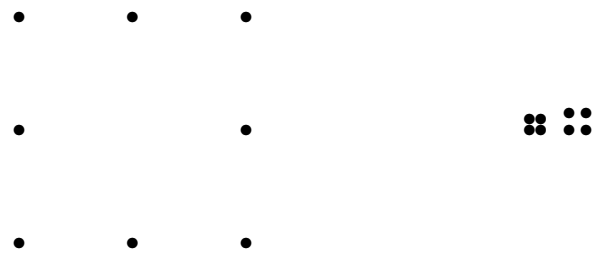
A fenti tulajdonságok teljesen természetesek, azt gondolnánk, hogy minden algoritmus ilyen. Ezért nem túl bíztató a következő tétel:

11.1. 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 távolsági függvénynek szemi-metrikának kell lennie, tehát a háromszög egyenlőtlenségnek nem kell teljesülnie

Kleinberg azt is bebizonyítja, hogy bármely két tulajdonsághoz létezik klaszterező eljárás, amely rendelkezik a választott tulajdonságokkal. Például a single-linkage eljárás (lásd 11.7.1. 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. Nézzük a következő ábrát. Bal oldalon láthatjuk az eredetileg megadott pontokat, jobb oldalon pedig az átmozgatás során kapottakat.

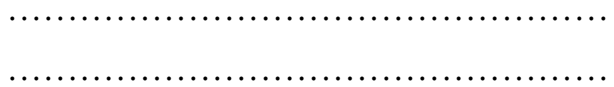


Legtöbben a bal oldali pontokat egy csoportba vennék (nagy négyzetet reprezentáló pontok), a jobb oldalon láthatókat viszont két külön csoportba sorolnák (két kis négyzethez tartozó pontok). A klaszteren belüli távolságokat tehát csökkentettük, a klaszterezés mégis megváltozott, azaz klaszterezési eljárásunk nem rendelkezik a konzisztencia tulajdonsággal.

Sajnos Kleinberger 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.

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 különbség 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 következő 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 tartoznak, és különbözőbe, ha más absztrakt objektum részei. A klaszterezés nehézsége pont abban rejlik, hogy automatikusan kell felfedezni objektumokat az elemek alapján, ami ráadásul nem egy egyértelmű feladat (például Rubin vázájának esete).

Ha a klaszterezés során az absztrakt objektumokat összefüggő alakzatok formájában keressük (pl. vonal, gömb, amőba, pálcikaember stb.) akkor van esély jol megoldani a feladatot. Összességében tehát tökéletes klaszterezés nem létezik, ugyanakkor a lehetetlenség elmélet nem zárja ki az összefüggő alakzatokat felfedező eljárás létezését.

11.2. Hasonlóság mértéke, adatábrázolás

Adott n elem (vagy más néven objektum, egyed, megfigyelés stb.). Tetszőleges két elem (x,y) között értelmezzük a *hasonlóság*ukat. Mi a hasonlóság helyett annak inverzével, a *különbözőség*gel dolgozunk ($d(x,y)$). $d(x,y)$-től elvárjuk (amellett, hogy $d(x,y) \geq 0$) azt, hogy

I. egy elem különbözősége önmagától 0 legyen: $d(x,x) = 0$,

II. szimmetrikus legyen: $d(x,y) = d(y,x)$,

III. és teljesüljön a háromszög-egyenlőtlenség: $d(x,z) \leq d(x,y) + d(y,z)$,

tehát a különbözőség metrika (távolság) legyen[2]. A továbbiakban elemek különbözősége helyett gyakran mondunk elemek *távolságát*.

A klaszterezés legáltalánosabb esetében minden egyes elempár távolsága előre meg van adva. Az adatokat ekkor egy ún. távolság mátrixszal reprezentáljuk:

$$\begin{bmatrix} 0 & d(1,2) & d(1,3) & \cdots & d(1,n) \\ & 0 & d(2,3) & \cdots & d(2,n) \\ & & 0 & \cdots & d(3,n) \\ & & & \ddots & \vdots \\ & & & & 0 \end{bmatrix},$$

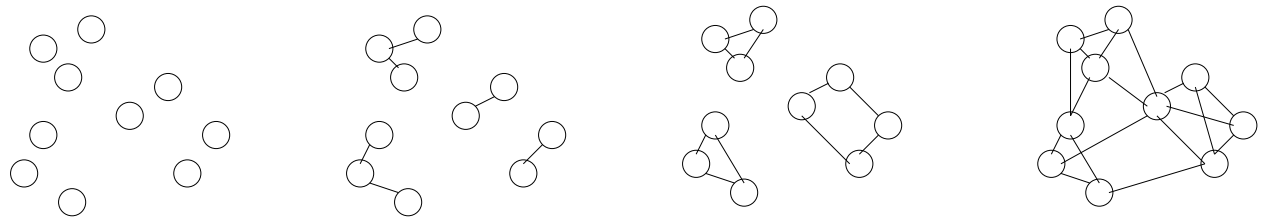
ahol $d(i,j)$ adja meg az i-edik és a j-edik elem különbségét.

A gyakorlatban az n elem (vagy objektum) attribútumokkal van leírva, és a különbözőséget az attribútumok alapján definiálhatjuk valamilyen *távolsági függvénnyel*. Ha megadjuk a távolsági függvényt, 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 távolsági függvény. Az n értéke nagy lehet, így nem tehetjük fel, hogy az adatok elférnek a memóriában.

Sokszor fogjuk a klaszterezést gráfparticionálási feladatként vizsgálni. Az elemekre tekinthetünk úgy, mint egy $G = (V,E)$ súlyozott, irányítatlan, teljes gráf pontjaira, ahol az éleken található súlyok a távolságot, vagy éppen a hasonlóságot adják meg. Az $(u,v) \in E$ él súlyát $w(u,v)$-vel jelöljük.

[2]Megjegyzés: Ha a 3. tulajdonság nem teljesül, akkor szemi-metrikáról beszélünk, ha az erősebb $d(x,y) \leq \leq \max d(x,z), d(y,z)$ tulajdonság áll fenn, akkor pedig ultrametrikusról (más néven nem-archimédeszi).

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 egyik pontja a másik pont k legközelebbi pontjai között szerepel. Az alábbi ábrán ilyen gráfokat láthatunk:



k=0 k=1 k=2 k=3

11.1. 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.

11.3. 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 a elemek közötti átlagos, illetve a maximális távolság:

$$D_{avg}(C) = \frac{\sum_{p \in C} \sum_{q \in C} d(p,q)}{|C|^2},$$

$$D_{max}(C) = \max_{p,q \in C} d(p,q).$$

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 (11.7.3. 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: $d_{avg}(C_i, C_j) = \frac{SS_{C_i} + SS_{C_j} - 2\vec{LS}_{C_i}\vec{LS}_{C_i}^T}{|C_i||C_j|}$.

11.4. A klaszterezés „jósága”

Mint már említettük, a klaszterezés jóságára 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.

11.4.1. 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-medián 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)[79]. A feladat NP-nehéz marad, ha a gráf Euklideszi térbe képezhető, sőt, konstans szorzó erejéig közelítő megoldást adni, még ilyenkor is, nehéz feladat [102]!

k-center probléma: Ez a feladat a k-medián módosítása, csak itt a legnagyobb hibát kell minimalizálni.

k-klaszter probléma: Célunk itt a klaszteren belüli távolságösszegek ($\sum_{i=1}^{k} \sum_{p,q \in C_i} d(p,q)$) minimalizálása. A feladat (és konstans szorzó erejéig annak közelítése) NP-nehéz $k \geq 2$ ($k \geq 3$) esetén [122].

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 minimalizálják.

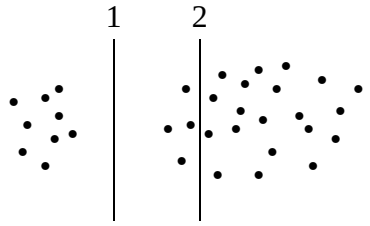
Azok az algoritmusok, amelyek a fenti célfüggvényeket minimalizálják, az elemeket kis kompakt felhőkbe csoportosítják. Ez valamennyire elfogadhatónak tűnik, azonban ezeknek a megközelítéseknek számos súlyos hátránya van.

I. Legfontosabb, hogy csak elliptikus klasztereket generál, tehát tetszőleges amőba alakú, de kompakt klasztert felvág kisebb kör alakú klaszterekre.

II. 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.

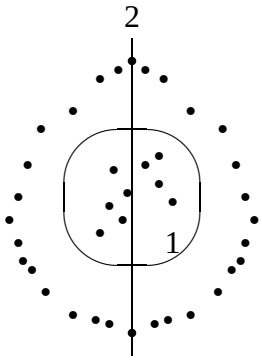
A négyzetes hibaösszeget minimalizáló eljárások további hibája, hogy érzékeny a távol eső (outlier) pontokra, hiszen egy távoli pont a klaszter középpontját nagyon „elhúzhatja”.

Elrettentő példaként nézzük a következő két ábrán látható pontokat. 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 klasztereken 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.).



11.2. ábra. Hibás klaszterezés: eltérő méretű klaszterek esetén

A 11.3. ábrán látható pontokat a 2-medián problémát megoldó algoritmusok a 2-es egyenes szerint csoportosítanák.



11.3. ábra. Hibás klaszterezés: egymást tartalmazó klaszterek esetén

11.4.2. Konduktancia alapú mérték

A klasszikus mértékek igen közkedveltek a matematikusok körében, köszönhetően az egyszerűségüknek és annak, hogy remekül elemezhetők. Rengetek írás született, amelyek matematikus szemmel kitűnőek, a gyakorlatban azonban –ahogy azt az előző két példa is illusztrálta– haszontalanok. 30-40 évig mégis ezek a problémák álltak a középpontban. A kutatók szép eredményeket értek el, a hasznosság igénye azonban sokáig kimondatlan maradt.

Az adatbányászat népszerűsödésével egyre fontosabb szerepük lett a "Mi haszna van?", "Mi a jó klaszterezés?" kérdéseknek. Hamar kiderült, hogy a klasszikus mértékek a gyakorlati esetek többségében egyszerűen rossz eredményt adnak. Új mértékek, megközelítések születtek. Ezek közül talán a legígéretesebb a konduktancia alapú mérték [78].

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 $|C|$-val, $E(S) = E(V \backslash S)$-el (edge(S)) 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 helyett 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) megadja az összes vágás közül azt, amelyiknél a legkisebb az arány a vágás súlya és a vágást alkotó két ponthalmaz közül a kisebbik elemszáma között. 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).

11.2. 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 11.2 és a 11.3 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.

11.3. definíció. *A $\{C_1, C_2, \ldots, C_k\}$ a (V, E) gráf egy (α, ε)-partíciója, ha:*

I. minden C_i klaszter konduktanciája legalább α,

II. 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.

11.5. Klaszterező algoritmusok típusai

A szakirodalomban jónéhány klaszterező algoritmus található. Nem létezik ideális klaszterező algoritmus, mivel az eredmények összehasonlítására nincs objektív mérték. Az egyes alkalmazások jellegétől függ, hogy melyik algoritmust célszerű választani.

A klaszterező algoritmusokat 5 kategóriába soroljuk.

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. Sajnos a gyakorlatban gyakran 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.

11.6. 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.

11.6.1. Forgy k-közép algoritmusa

A k-közép algoritmus (k-means algorithm) [52] az egyik legrégebbi (1965-ből származik) és leggyszerűbb 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 besorolás, új középpont választás iterációs lépéseket addig ismételjük, amíg történik változás.

Jancey 1966-ban Forgy-tól teljesen függetlenül ugyanezt az algoritmust javasolta egy apró módosítással [74]. Az új reprezentáns pont ne az új középpont legyen, hanem a régi és az új középpontot összekötő szakaszon, például a középponton. 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.

I. Lehet, hogy az algoritmus lokális optimumban áll meg, tehát az iterációk során nem változik semmi, mégis létezik olyan csoportosítás, ahol a négyzetes hiba kisebb.

II. 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.

III. Rendelkezik a négyzetes hibát minimalizáló algoritmusok minden hibájával (lásd a 11.4.1-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, 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.

11.6.2. 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.

A k-medoid algoritmusokban egy klasztert nem a középpont reprezentál, hanem a leginkább középen elhelyezkedő elem, a medoid. Továbbra is egy négyzetes hiba jellegű függvényt próbálunk minimalizálni, de a négyzetes hiba itt a medoidoktól való távolságok összegét jelenti (k-medián probléma, lásd 11.4.1 rész).

A PAM algoritmus

A PAM (Partitioning Around Medoids) algoritmus [80] menete teljesen megegyezik a k-közép menetével. Kezdetben választunk k véletlen elemet, ezek lesznek először a medoidok. Ezután elkezdődik az elemhozzárendelés medoidokhoz, új medoid választása iteratív folyamat. Egy elemet a legközelebbi medoidhoz rendelünk.

Abban az esetben választunk új medoidot egy klaszterben, ha ezzel csökken a négyzetes hiba. Határozzuk meg az összes nem medoid, medoid párra (x, x_m), 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 változása negatív, 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 [80] 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 [107] az eredeti adathalmazon dolgozik. Nem az összes lehetséges csere által eredményezett négyzetes hiba változását számítja ki, hanem csak egy, véletlenszerűen választott

párét. Ha a változás negatív (sikeres választás), akkor a pár elemei szerepet cserélnek, 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 többször futtassuk az algoritmust mindig 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.

A CLARANS nagy adathalmazokon való alkalmazhatóságával foglalkoznak a [43] cikkben. R*-fát 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 képes felfedezni.

11.7. 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.

11.7.1. Single-, Complete-, Avegare Linkage Eljárások

A legegyszerűbb egyesítő, hierarchikus eljárás az alábbi.

I. Kezdetben minden pont külön klaszterhez tartozik.

II. Keressük meg, majd egyesítsük a legközelebbi klasztereket.

III. 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 (1-2 Gbyte memóriával rendelkező gép teljesen hétköznapinak számít) ez azt jelenti, hogy az elemek száma 30-40 ezernél nem lehet több.

Amennyiben két klaszter távolságát a legközelebbi pontjaik 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 a klaszterek túl közel kerülnek egymáshoz, akkor hajlamos a single linkage összekötni őket, és esetleg a klaszteren belül egy vágást képezni.

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. Ha tudjuk, hogy olyan adathalmazt kell klasztereznünk, ahol a (tetszőleges alakú) csoportok jól elkülönülnek egymástól, továbbá nincsenek outlierek, akkor a single eljárás 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 minimális távolság helyett a maximális távolságot használjuk, akkor *complet linkage* eljárásról beszélünk, ha pedig az átlagos hasonlóságot, vagy az egyesített klaszter átmérőjét, akkor *average linkage* eljárásról.

11.7.2. 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.

11.7.3. 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.

11.7.4. A CURE algoritmus

A CURE (Clustering Using REpresentatives) algoritmus [61] 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.

I. A reprezentáns pontok alapján kiválasztja a két legközelebbi klasztert. Két klaszter távolságát reprezentáns pontjaik távolságának minimuma adja.

II. 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.

III. 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 a [61]-t.

A CURE algoritmus számos hibával rendelkezik:

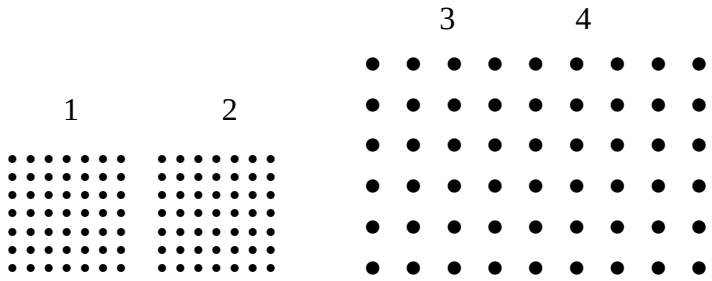
I. Az elemeknek vektortérben adottnak kell lenniük.

II. Minden klasztert rögzített számú reprezentáns pont jellemez. De miért jellemezzen ugynannyi pont egy kis kör alakú klasztert és egy nagy amőba alakút? Természetesebb lenne, ha a reprezentáns pontok száma függene a klaszter méretétől és alakjától.

III. 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 sosem 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.

IV. 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.

V. Rosszul kezeli az eltérő sűrűségű pontokat. Ezt leginkább az alábbi ábra illusztrálja. A CU-



11.4. ábra. Hibás klaszterezés: eltérő sűrűségű klaszterek esetén

RE az 1-es és 2-es klasztereket fogja egyesíteni (azok reprezentáns pontjai vannak egymáshoz legközelebb) a 3-as és 4-es klaszterek egyesítése helyett.

Megjegyezzük, hogy az algoritmust bemutató cikk hosszú bevezetőjében éppen arra hívta fel a figyelmet, hogy a mások által publikált algoritmusok mennyire rosszul kezelik a különböző méretű és amőba alakú klasztereket. Ennek ellenére a tesztekben bemutatott adathalmazban nagyságrendileg azonos méretűek voltak a klaszterek és alakjuk elliptikus volt.

11.7.5. 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. Nézzük meg, hogyan definiálja az algoritmus két klaszter relatív belső kapcsolódását és relatív távolságát.

Relatív távolság

Relatív belső kapcsolódás

11.8. 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.

11.8.1. A DBSCAN algoritmus

A DBSCAN a legelső sűrűség-alapú eljárás [44]. 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 $p_1 = p, p_2, \ldots, p_n = q$ 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:

11.4. definíció. *Az elemek egy C részhalmaza klaszter, amennyiben*

I. Ha $p \in C$ és q sűrűség-alapon elérhető p-ből, akkor $q \in C$ (maximalitás).

II. 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 eleme p 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 tetszőlege 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.

12. fejezet

Szövegbányászat

Az írástudó emberi civilizációk kialakulása óta a tudást szöveges dokumentumok formájában tárolják. Az ősi egyiptomiak is szöveges dokumentumokat hagytak az utókorra, azonban hieroglifikus írásuk megfejtése korántsem bizonyult könnyű feladatnak. A szöveg megértését végül az segítette elő, hogy a feliratok több nyelven szerepeltek ugyanazon a kövön, amelyek közül az egyik görög volt a másik kettő egyiptomi. Ezáltal a görög nyelv szolgált kulcsként a hieroglifák megfejtéséhez, ez segített a templomok és piramisok falán és a papirusz tekercseken talált szövegekben lévő tudás feltárásában. Az ősi egyiptomi hieroglifák megfejtéséből két dolgot tanulhatunk: egyrészt, hogy a szöveges dokumentumok az emberiség egyik ősi emlékezeti mechanizmusa, fontos megbízhatóan tárolni az adatokat és rendelkezni kell azzal a képességgel, hogy ha szükséges visszanyerjük ezeket a dokumentumokat. Másrészt azt, hogy a dokumentumok szimpla elérése nem elegendő, a tudás feltárása speciális gyakorlatot és erőforrást igényel.

Napjainkban, amikor a dokumentálási és adminisztrációs folyamatok túlnyomó része elektronikusan valósul meg — és ezáltal rendkívül nagy mennyiségű elektronikus dokumentum keletkezik —, megfigyelhető az a trend, hogy az adminisztratív munkát végzők munkaidejük egyre növekvő hányadát fordítják (elektronikus) dokumentumok kezelésére. Míg ez csupán 20%-ot tett ki 1997-ben, addigra 2003-ban már 30–40%-ra becsülték ezt az arányt az [18, 72] munkákban idézett Gartner Group tanulmányban. A Merill Lynch elemzői szerint az üzleti információk 85%-a *strukturálatlan adat* formájában van jelen, mint pl. e-mailek, emlékeztetők, üzleti és kutatási beszámolók, prezentációk, hírek, reklámanyagok, weboldalak, ügyfélszolgálati tevékenység jegyzetei, stb. [18].

Az adatbányászati módszerekkel az adatbázisokban *strukturáltan tárolt* adatokból nyerhetők ki összefüggések. Ezek a módszerek nem működnek a strukturálatlan, általános típusú, szöveges adatokra. Ezért a *strukturálatlan* szöveges adathalmazok hasonló célú feldolgozása más megoldásokat tesz szükségessé. Az ezzel foglalkozó szakterületet *szövegbányászat*nak nevezzük.

Az adatbányászat definíciójával analóg módon, a *szövegbányászatot* dokumentumokon végzett olyan jellegű feldolgozási és elemzési tevékenységként határozhatjuk meg, melynek célja a dokumentumokban rejtetten meglévő új információk feltárása, azonosítása.

A szövegbányászat alapvető problémája nyilvánvaló: a természetes nyelvek emberek közti — elsősorban szóbeli, majd később írásbeli — kommunikáció miatt keletkeztek és fejlődtek ki, és nem számítógépes feldolgozásra. Az emberek könnyedén felismerik és alkalmazzák a nyelvi mintákat, és általában nem okoz gondot nekik olyan, a számítógépek számára nehezen megoldható feladatok, mint pl. különböző helyesírási variációk kezelése, kontextus felismerés, vagy stilisztikai jelleg azonosítása. Tehát nyelvi tudásunk lehetővé teszi a strukturálatlan szövegek megértését, ugyanakkor nincs meg bennünk a számítógépeknek az a képessége, hogy a szöveget nagy mennyiségben, vagy nagy

12.1. ábra. A szövegbányászat általános modellje

sebességgel dolgozzuk fel. A szövegbányászat általános célja tehát az emberi nyelvi tudás ötvözése a számítógép nagy sebességével és pontosságával [48].

A szövegbányászat általános modellje a 12.1 ábrán látható. A kiinduló pont a dokumentumok halmaza, amin először előfeldolgozási lépéseket hajtunk végre (ld. 12.1. szakasz). Ezután hajtjuk végre a szövegbányászati módszereket, majd az eredményeket információkezelő rendszerben tároljuk. A felhasználó ebből tudja az igényeinek megfelelő tudást megszerezni.

Olyan problémákkal, amelyekre a szövegbányászat nyújthat megoldást az üzleti élet szereplői és az átlagos felhasználók egyaránt gyakran találkoznak. A nagy forgalmat lebonyolító ügyfélszolgálatoknál például hatalmas mennyiségű ügyféllel történő beszélgetés zajlik naponta. Ezek jellemző tartalma, fontosabb témái, az ügyfélkör igényeinek változása a szolgáltatónak fontos információt jelent, amellyel hatékonyan reagálhat a piac változásainak kihívásaira.

Szintén fontos információt hordozhat üzleti döntéshozók számára a konkurens cégekről, ill. termékekről szóló üzleti hírekről szóló automatikus értesítés.

Az átlagos felhasználók közül is mindenki szembesült már a kulcsszó-alapú keresés korlátaival. Ha többértelmű keresőkifejezést használunk — a tipikus példák: *jaguár* (állat, autómárka), *saturn* (bolygó, elektronikai cég, autótípus), *tus* (zuhany, írószer, vívás, zene)[1] —, akkor a keresés finomítására van szükség a kívánt információ elérésére. Ha a kontextus megadható lenne, vagy a keresett oldalak tematizáltan lennének tárolva, akkor az jelentősen megkönnyítené a keresést.

A keresők gyakran adnak eredményül nagyméretű, akár több száz oldalas dokumentumokat, amely nyilván több témát is tárgyal, és nem feltétlenül releváns a kereső számára. Ahhoz, hogy a felhasználó megtalálja a neki fontos információt bele kell mélyednie a szövegbe, ami rendkívül időigényes. Erre a problémára a szövegbányászat az összegzéskészítő módszereket kínálja megoldásként, amelyek automatikusan összefoglalják a dokumentum tartalmát, aminek alapján a felhasználó már könnyebben tájékozódhat.

Az eddig ismertettet példák csak ízelítőt nyújtanak a szövegbányászat már létező és jövőbeli felhasználásairól. Mielőtt a következő szakaszokban mélyebbrehatóan elkezdünk foglalkozni a témával, a 12.1 táblázatban összefoglaljuk a szövegbányászat alapvető ismérveit összehasonlítva az adatbányászattal.

12.1. Dokumentumok előfeldolgozása

Mint azt a 12.1 ábrán láttuk a szövegbányászati feladatok megoldásának első lépése a szövegek előfeldolgozása, aminek célja hogy megfelelő, egységes gépi reprezentációs alakra hozzuk őket. Egy teljesen általános szövegreprezentációs modellnek rendkívül széleskörű tudást kell magában foglalnia, többek között például a természetes nyelvtanokat is. Első megközelítésben azonban csak statisztikai elemzések elvégzésére alkalmas modellt keresünk, amelyben a gépi tanulás algoritmusai hatékony alkalmazhatók, mint pl. az adatbányászat esetében a korábbi fejezetekben ismertetett módszerek.

Mivel a szövegeket a számítógép nem tudja értelmezni, ezért szükség van egy olyan eljárásra, amely a szövegek tartalmát tömören reprezentálja, és amely természetesen bármely dokumentumra

[1] Érdekes, hogy a nemzetközi keresők erre a keresőszóra a nyomtatóval kapcsolatos cikkeket is találnak a *tűs* szó ékezetnélküli reprezentációja miatt. A példa jól mutatja: a hatékony szövegbányászati alkalmazások —bizonyos részben —nyelvfügg ők.

12.1. táblázat. Az adat- és szövegbányászat összehasonlítása ([72] felhasználásával)

	adatbányászat	szövegbányászat
az elemzés tárgya	numerikus és típusba sorolható	szabadformátumú szöveges dokumentum
az adatok jellege	strukturált	strukturálatlan
az adatok tárolási helye	relációs adatbázis	tetszőleges dokumentumgyűjtemény
feladat	összefüggések feltárása, jövőbeni szituációk előrejelzése	szövegelemzés, kategorizálás; összegzéskészítés; vizualizálás; csoportosítás, stb.
módszerek	neurális hálózatok, döntési fák, statisztikai modellek, klaszteranalízis, idősorok elemzése, stb.	dokumentum indexelés, speciális neurális hálózatok, számítógépes nyelvészeti eszközök, ontológiák
jelenlegi piacméret világszinten	100.000 elemző a nagy és közepes vállalatoknál	100.000.000 vállalati munkatárs és egyéni felhasználó
széleskörű piaci megjelenés ideje	1994-től	2000-től

alkalmazható. A továbbiak során — ha ettől eltérően nem jelezzük — a reprezentáció egységének a szavakat tekintjük. Egyes módszerek több szóból álló kifejezéseket is alkalmaznak, amely azonban jelentősen megnöveli a dokumentumok feldolgozásának (indexelésének) idejét, valamint a tárigényt.

Az információ-visszakeresés (information retrieval IR) területén a dokumentumokat leggyakrabban a *vektortér-modell* segítségével vannak reprezentálva [124]. A dokumentumokat szintaktikai szabályok segítségével felbontjuk *token*ekre (legegyszerűbb esetben a szóköz elválasztó karakter alkalmazásával; ekkor a tokenek szavak), és a tokeneket *szótövező* segítségével kanonikus alakra hozzuk, azaz a szótővel helyettesítjük (ld. még 12.7 szakasz). Az egyszerűség kedvéért a továbbiakban a kanonikus alakot *szó*nak nevezzük. A dokumentumgyűjteményben előforduló különböző szavak alkotják a *szótár*at, vagy más néven *lexikon*t.

Minden tengely egy szót reprezentál, a dokumentumokat pedig *vektorként* ábrázoljuk a szavak által kifeszített vektortérben. A dokumentumok gyűjteményét *szó–dokumentum mátrix*szal reprezentáljuk ($A \in \mathbb{R}^{M \times N}$), amelynek valamely a_{ij} elem az i-edik szó előfordulásait reprezentálja a j-edik dokumentumban, vagyis az i-edik tengelyhez tartozó szó relevanciáját, súlyát adja meg a d dokumentumra vonatkozóan. A sorok száma, M, megegyezik a szótár méretével, N pedig a dokumentumok száma. Mivel általában egy dokumentumban az egész szótárból kevés szó fordul elő, az A mátrix ritka. M rendkívül nagy is lehet, ebből adódóan a szövegek kezelésének egyik problémája a vektortér magas dimenziója. A dimenziószám csökkentésére vonatkozó módszereket a 12.1.1 pontban tekintjük át.

Az a_{ij} érték megválasztására több lehetőség van. A legegyszerűbb a *bináris reprezentáció*:

$$a_{ij} = \begin{cases} 1, & \text{ha } n_{ij} > 0 \\ 0, & \text{ha } n_{ij} = 0 \end{cases}, \tag{12.1}$$

ahol n_{ij} az i szó előfordulásának száma a j dokumentumban. Ezt az értéket szintén választhatjuk az adott szó fontosságának megfeleltetéseként:

$$a_{ij} = n_{ij}.$$

A dokumentumokat reprezentáló vektorokat normálhatjuk, hogy hosszuk 1 legyen pl. az $\|\cdot\|_1$, $\|\cdot\|_2$ vagy $\|\cdot\|_\infty$ norma szerint. Ha $\|\cdot\|_1$-t választjuk, akkor az előbbi érték

$$a_{ij} = n_{ij}/n = f_{ij} \tag{12.2}$$

lesz, ahol f_{ij} a szó dokumentumbeli *gyakoriság*át jelöli (TF súlyozás).

A **??**tm:eq:TF) súlyozási séma azonos fontosságúnak kezeli az összes szótárbeli szót, holott nyilván a témaspecifikus szavak, mint pl. „adatbányászat” jellemzőbbek egy dokumentum tartalmára mint a névelők, határozók, névutók, stb., pl. „az”, „hogy”, „alatt”. Ha i szó n_i dokumentumban fordul elő, akkor n_i/N a szó ritkaságát, azaz fontosságát jellemzi a gyűjteményben. Az $IDF(i)=1+\log(n_i/N)$ *inverz dokumentum frekvencia*[2] értéke a vektortér-modell egyes tengelyeit különböző mértékben nyújtja meg. Így kaphatjuk meg a legnépszerűbb, ún. TFIDF[3] súlyozási sémát:

$$a_{ij} = f_{ij} \cdot IDF(i). \tag{12.3}$$

Ezen kívül más, bonyolultabb súlyozási sémák is ismertek, amelyek a dokumentumok hosszát, illetve az egyes szavak információ elméleti alapon számított entrópiáját is figyelembe veszik [1, 40, 123].

12.1.1. A dimenziószám csökkentése

Már kisebb dokumentumgyűjtemények esetén (néhány tízezer dokumentum) a szótár mérete jellemzően többszázezres nagyságrendű, amellyel általában az algoritmusok nagy része nem tud megbirkózni a nagy számítási és memória igény miatt. Ha csak 100.000 átlagosan 1000 különböző szót tartalmazó dokumentumunk van, annak hatékony tárolása is legalább $2 \cdot 10^8 = 0{,}2$ GB memóriát igényel. A szótár méretének a csökkentése tehát kiemelkedő fontosságú feladat, hiszen ezzel mind a gyűjtemény reprezentálásához szükséges memória[4], mind pedig az algoritmusok futási igénye csökkenthető. A dimenziószám csökkentése a mintafelismerés (pattern recognition) szakirodalmában jól ismert feladat. Az ismert eljárások két csoportba sorolhatók: jellemzők kiválasztása és újraparaméterezés. Szövegbányászati alkalmazásukat a [160] tanulmány tekinti át.

A jellemzők kiválasztása

A legegyszerűbb eljárások az alábbi két empirikus megfigyelésen alapulnak:

[2] IDF-nek több definíciója létezik. Egy alternatív verzió: $IDF(i) = \log(N/n_i)$.

[3] terminus frekvencia és inverz dokumentum frekvencia

[4] Bizonyos tanulási feladatokhoz, pl. kategorizálás esetén, az egész gyűjteményt a memóriában kell tárolni. Ha ez csak lapozófile alkalmazásával lehetséges, az jelentősen meghosszabbítja a futási időt.

– Minél több dokumentumban szerepel egy szó, annál kisebb mértékben jellemzi a dokumentumok tartalmát, azaz annál kisebb a megkülönböztető képessége és az információ tartalma.

– Minél ritkábban fordul elő egy szó az adott dokumentumgyűjteményen belül, annál kevésbé releváns.

A fenti megfigyelések alapján a *dokumentum frekvencia küszöbölő* (Document Frequency Thresholding) módszer elhagyja a θ_1 küszöbérték alatti frekvenciával rendelkező szavakat, és a θ_2 küszöbérték feletti n_i értékkel rendelkező szavakat. A módszer azzal a feltételezéssel él, hogy az első kategóriába eső szavak kicsiny információ tartalmúak, és nem befolyásolják jelentősen pl. a kategorizálás hatékonyságát, míg a második kategóriába eső szavak nem diszkriminatívok.

Ide sorolható még az ún. *funkció szavak elhagyása*[5] is, amelyek a szöveg tartalmára vonatkozóan nem bírnak jelentős információtartalommal. A funkció szavak listája nyilván minden nyelven más, de akár a dokumentumgyűjtemény témájától és általánosságától is függhetnek [57].

Kategorizálásnál használnak még a szavak és kategóriák együttes előfordulásából becsült információ elméleti és statisztikai mértékeket. Az információs nyereség módszer esetén minden szóra megvizsgálják, hogy van-e olyan kategória, amelyben az előfordulása vagy elő nem fordulása kiugró. Az így kapott értékeket összegezve a valamely küszöbérték alatti szavakat kevéssé diszkriminatívnak tekintve elhagyjuk. Hasonló elven működik a kategóriák és szavak közti függetlenség hiányát vizsgáló χ^2-statisztikán alapuló módszer [1].

Újraparametrizálás

Az újraparametrizálás során új jellemzőket állítunk elő a vektortér eredeti jellemzői (dimenziói) kombinációiként. A legismertett ilyen módszer a szinguláris értékfelbontáson (SVD) alapuló látens szemantikus indexelés (LSI) [16, 37].

Az LSI módszer feltételezi, hogy a dokumentumok szóhasználati mintázatában létezik egy elrejtett, azaz látens struktúra, és hogy ezt statisztikai módszerekkel közelíteni lehet. A sajátértékfelbontáson alapuló SVD segítségével kiválaszthatók azok a valamely legnagyobb K sajátértékhez tartozó jellemzők, melyek az A szó–dokumentum mátrixot kellően jól reprezentálják. A K értéke lényegesen kisebb M-nél. Az LSI azokat a dokumentumokat, amelyek sok hasonló szót tartalmaznak szemantikailag közelinek, és azokat, amelyek kevés közös szót tartalmaznak, szemantikailag távolinak értékeli. Ez az egyszerű módszer meglepően jól korrelál azzal, ahogy egy ember, aki a dokumentumot átnézi, besorolja az adott dokumentumgyűjteményt. Annak ellenére, hogy az LSI algoritmus algebrai módszert használ, tehát nem ért semmit a szavak jelentéséből, meglepően jó szemantikai következtetésekre ad lehetőséget, azaz „rendkívül intelligensnek tűnik”.

12.1.2. Hatékonyság mérése

Különböző jellegű szövegbányászati módszerek hatékonyságát más-más kiértékelő mértékkel vizsgáljuk. Természetesen a hatékonyság mérésére csak akkor van lehetőség, ha rendelkezésre áll a várt eredmény, ami csak bizonyos szövegbányászati módszerek esetén lehet adott, pl. kategorizálásnál. Ekkor ugyanis, ha olyan dokumentumgyűjteményen teszteljük a módszert, ahol az egyes dokumentumok kategóriája ismert, akkor könnyen ellenőrizhetjük a módszer helyességét. Ezzel szemben pl. kivonatolás esetén nehéz egy optimális eredményt összehasonlítási alapnak tekinteni, amely bárki

[5] Angol szakirodalomban *function words* vagy *stopwords*.

szerint az adott szöveg legjobb kivonata, mivel a kivonat emberi megítélése szubjektív. Ezért ilyen esetekben csak tapasztalati, heurisztikus módszerek vannak a hatékonyság mérésére, illetve a különböző eredmények összehasonlítására, rangsorolására. A különböző feladattípusokhoz tartozó mértékeket az adott szakaszon belül fogjuk tárgyalni.

12.2. Osztályozás

Adatbányász módszerek jellemzően relációs adatbázisokon működnek, ahol az adatok strukturáltan, oszlopokba és sorokba rendezve vannak tárolva. Hasonló módon lehetőség van a strukturálatlan adatok hierarchikus struktúrába, ún. *taxonómiá*ba való rendszerezésére is. A taxonómia úgy működik, mint egy számítógépes könyvtárstruktúra, ami kézenfekvő és intuitív eszközt ad a navigálásra és az információk elérésére, keresésére [18].

A dokumentumok tartalmuk alapján *tematikus kategóriarendszer*be (kategóriákba) történő besorolását *osztályozás*nak (más néven *kategorizálás*nak) nevezzük, ami az egyik legalapvetőbb szövegbányászati feladat. A kategóriák rögzítettek és előre adottak. A kategóriák egymáshoz való viszonya alapján beszélhetünk *egyszerű osztályozás*ról — ilyenkor nincs semmilyen összefüggés az egyenrangú kategóriák között, illetve *hierarchikus osztályozás*ról — amikor a kategóriák egy strukturált rendszert, általában fát, vagy körmentes irányított gráfot alkotnak. Ebben az értelemben a taxonómia tehát kategóriák hierarchikus rendszere.

Az üzleti alkalmazásokban azonban két ok miatt még nem túl elterjedt a dokumentumok taxonómiába rendezése. Egyrészt a taxonómia megalkotása és fenntartása nehéz feladat. Olyan szakértőt kíván, aki átlátja az egész cég üzleti szervezetét, és rendszerező képességgel bír. A taxonómia mérete igen nagy lehet, akár több ezer kategóriát is tartalmazhat.[6] Egy cég profiljának, termékeinek változása a taxonómia változtatásának szükségességét is magával vonja, ami szintén időigényes és költséges feladat. Egyébként az automatikus taxonómiakészítő módszerek csak az utóbbi években kezdtek megjelenni a piacon (Verify, Stratify, Inxight, Autonomy). A másik nagy akadályt a piacon lévő szoftverek osztályozási szempontból gyenge hatékonysága jelenti. Ez részben annak tudható be, hogy viszonylag egyszerű algoritmusokat alkalmaznak az általános feladat nehézségéhez képest, részben pedig annak, hogy ezek az algoritmusok jelentős számú tanulóadatot igényelnek, és amelyre nem szán időt az üzleti felhasználó.

12.2.1. Osztályozás strukturálatlan kategóriák rendszerébe

Az osztályozási feladatok között a dokumentum–kategória reláció jellegétől függően az alábbi megkülönböztetést tesszük:

- *Bináris osztályozás*nak nevezzük, amikor csak *egy* kategória adott, és a dokumentumokról azt kell eldönteni, hogy ebbe beletartoznak-e vagy sem.
- *Egycímkés osztályozás* (multi-class) esetén *több* kategória adott, és minden dokumentumok *legfeljebb egy* kategóriához tartozik.
- *Többcímkés osztályozás* (multi-label) esetén szintén *több* kategória adott, de minden dokumentum *több* kategóriába is beletartozhat.

[6] Erre jó példa a nemzetközi szabadalmi hivatal által kifejlesztett IPC taxonómia: `http://www.wipo.org/classifications/fulltext/new_ipc/index.htm`

12.2. ábra. Rocchio osztályozó működése

– *Többszintű osztályozás* (multi-level) esetén szintén *több* kategória adott, és egy dokumentumnak lehetnek elsődleges, másodlagos stb. kategóriái.[7]

Az automatikus osztályozás tipikus *felügyelt tanulási feladat* (supervised learning), amikor megadott tanuló példák alapján az osztályozót képessé tesszük arra, hogy felismerje az egyes osztályokba tartozó dokumentumok jellegzetességeit. Adott tehát egy *tanuló dokumentumhalmaz*, ahol a dokumentumok a kategóriájukkal fel vannak címkézve. Az algoritmus először ez alapján megtanulja a kategóriák jellemzőit, majd ismeretlen kategóriájú dokumentumok címkéjére ad becslést.

A felügyelt tanuláshoz a dokumentumgyűjteményt két diszjunkt halmazra bontjuk, *tanuló-* és *teszthalmaz*ra: $\mathcal{D}_{\text{Train}} \cap \mathcal{D}_{\text{Test}} = \emptyset$, and $\mathcal{D}_{\text{Train}} \cup \mathcal{D}_{\text{Test}} = \mathcal{D}$. A tanuló halmaz egy részét gyakran a módszerek megfelelő paramétereinek beállításához használjuk, ezt *validációs halmaz*nak nevezzük. Legyen adott továbbá K számú kategória, $\mathcal{C} = \{c_1, \ldots, c_K\}$, és minden c kategóriához egy $\mathcal{D}_c$ tanuló dokumentumhalmaz. Egy kategóriához $N_j = \left|\mathcal{D}_{c_j}\right|$ dokumentum tartozik. Az egész tanulóhalmaz tehát $N = \sum_{j=1}^{K} N_j = |\mathcal{D}_{\text{Train}}|$ dokumentumot tartalmaz. A feladat egy ismeretlen $\vec{d} = (d_1, \ldots, d_M) \in$ $\in \mathcal{D}$ dokumentum kategorizálása. A következőkben ismertetett módszerek általában az első három feladattípus megoldására alkalmasak, ettől eltérő esetben ezt külön jelezzük.

Rocchio-eljárás

A *Rocchio-eljárás* klasszikus módszernek számít az információ-visszakeresés területén. Osztályozási feladatra először a [69] munkában adaptálták, és azóta is sok kutatás foglalkozott vele (ld. [132]. Minden c kategóriához megalkotunk egy *prototípusvektor*t, amit a $\mathcal{D}_c$ tanulópéldák átlagaként számítjuk ki (centroid), és ehhez hasonlítjuk az ismeretlen dokumentum vektorát. Az osztályozandó dokumentum és egy kategória prototípusvektorának távolságát koszinusz- vagy más távolságmértékkel számolhatjuk.

A módszernek kicsiny a számításigénye, ezért a tanulás nagyon gyors. Hátránya viszont, hogy ha egy kategóriához több különböző csoportbeli szöveg is tartozik — pl. *Sport* alá fociról és vitorlázásról szólók — akkor az osztályozó a legtöbbjét helytelenül sorolja be, mert a dokumentumok centroidja kívül eshet a csoportokon ld. 12.2 ábra. Ez a jelenség abból adódik, hogy a Rocchio-osztályozó a *lineáris osztályozó*k közé tartozik, amelyek a dokumentumok terét lineárisan osztják fel.

A módszer hatékonysága lényegesen javítható, ha a prototípusvektorok megalkotásánál a negatív tanulóadatokat is figyelembe vesszük. Ekkor a

$$\vec{c} = \beta \cdot \sum_{j \in \mathcal{D}_c} \vec{d}_j - \gamma \cdot \sum_{j \notin \mathcal{D}_c} \vec{d}_j \tag{12.4}$$

képlettel számítható a c prototípusvektora[8]. Ha a második tagban nem az összes negatív tanulópéldát, hanem csak a majdnem pozitív tanulópéldák átlagát vesszük — ezek ugyanis azok, amelyektől a legnehezebb megkülönböztetni a pozitív tanulóadatokat, akkor további lényeges hatékonysági javulás érhető el [128, 157].

[7] A többszintű osztályozás esetén a feladat általában hierarchikus kategóriarendszerrel párosul, ezért —bár strukturálatlan kategóriarendszer esetén is értelmezhető a probléma —ezt a 12.2.2 pontban tárgyaljuk.

[8] A dokumentumok centroidjaként számolt prototípusvektort a $\beta = 1$, $\gamma = 0$ paraméterek mellett kapjuk meg.

Naiv Bayes-módszer

A naiv Bayes-módszer (pl. [75]) valószínűség számítási alapon működő osztályozó [94]. A tanulóhalmaz alapján egy besorolandó dokumentumhoz a Bayes-tétel alapján megbecsüli az egyes kategóriákhoz való tartozás valószínűségét,

$$P(c_j|\mathbf{d}) = \frac{P(c_j)P(\mathbf{d}|c_j)}{P(\mathbf{d})}, \tag{12.5}$$

ahol a nevező mindig ugyanaz, tehát elhagyható. A módszer elnevezésében a naiv jelző arra — az egyébként általában nem helytálló — feltételezésre utal, hogy a változók (szavak) feltételesen függetlenek, ha a kategória adott. Így a $P(\mathbf{d}|c_j)$ értékének becslése — amely nagy számú tanulóadat esetén bonyolult feladat — lényegesen leegyszerűsödik, és ezért a **??**eq:Bayes) kifejezés az alábbiak szerint írható fel:

$$P(c_j|\mathbf{d}) = P(c_j)\prod_{i=1}^{M} P(d_i|c_j).$$

A $P(c_j)$ valószínűség a tanuló példák gyakorisága alapján megbecsülhető:

$$\hat{P}(C = c_j) = \frac{N_j}{N},$$

valamint

$$\hat{P}(d_i|c_j) = \frac{1+N_{ij}}{M+\sum_{k=1}^{M} N_{kj}},$$

ahol N_{ij} az i-edik szó előfordulása a $\mathcal{D}_{c_j}$ dokumentumokban.

Érdekes módon annak ellenére, hogy a szavak független előfordulására vonatkozó kiinduló feltételezés általában nem igaz, a módszer igen jó eredményt ad, amit elméleti eredmények is alátámasztanak [38]. Sőt, ha bonyolultabb, s ezáltal nagyobb számításigényű valószínűségi modellt használunk [85], akkor sem javul jelentékenyen a hatékonyság.

Legközelebbi szomszédokon alapuló osztályozó (k-NN)

Egy adott dokumentum besorolásakor e módszer valamilyen távolságfogalom segítségével megvizsgálja, hogy a tanuló adatok közül melyik k dokumentum vektora hasonlít legjobban a vizsgált $\vec{d}$ vektorhoz. Ezen vektorokhoz tartozó kategóriák távolság arányában történő súlyozásából felállítható a dokumentumhoz tartozó kategóriáknak rangsora. A hasonlóság megállapítására általában a koszinusz- vagy az euklideszi-távolságot használják. A módszer az ún. *lusta tanuló* eljárások közé tartozik, vagyis a tanulóhalmazt nem dolgozza fel előre, hanem csak az adott dokumentum feldolgozása során végez döntést.

A k paraméter beállítását, ami része az osztályozó megalkotásának, tapasztalati úton végzik a *validációs adatok*on. A vizsgálatok azt mutatták ki [158], hogy $30 \leq k \leq 45$ értékek adják a legjobb eredményt. A k-NN módszer *nem lineáris osztályozó*, ezért a Rocchio-eljárásnál ismertetett problémák nem jelentkeznek.

Az eredmények azt mutatják (12.2.1 szakasz), hogy a módszer elég hatékony. A legfőbb hátránya, a futási időben jelentkező magas számítási igény, hiszen egy dokumentum osztályozásához az egész tanulóhalmazt rangsorolni kell, ami lényegesen bonyolultabb, mint pl. a lineáris osztályozóknál egy szorzás végrehajtása.

Döntési fa alapú szövegosztályozók

Döntési fán alapuló szövegosztályozó egy olyan fa, amelyben a közbenső csomópontok szavak (szótári elemek), a csomópontokból kiinduló ágakat az adott szó teszt dokumentumbeli előfordulásának súlya határozza meg, a levelek pedig kategóriákkal vannak címkézve. Az osztályozás a $\vec{d}$ tesztdokumentumban a döntési fa csomópontjaihoz tartozó szavak súlyának rekurzív vizsgálata alapján történik, a dokumentumhoz végül a levél kategóriacímkéjét rendeljük hozzá. A döntési fa alapú szövegosztályozók általában bináris reprezentációt használnak, így a döntési fa is bináris.

A legtöbb szövegosztályozó standard döntési fa tanuló csomagot használ, mint az ID3, a C4.5, a C5, ill. CHART vagy CHAID. Általánosságban a c kategóriához tartozó döntési fa megtanulása az „oszd meg és uralkodj” stratégia két lépéséből áll: (1) annak ellenőrzése, hogy minden tanuló dokumentumnak ugyanaz-e a címkéje (c vagy $\overline{c}$); (2) ha nem, akkor egy olyan d_j szó kiválasztása, amely a tanulóhalmazt úgy particionálja, hogy az egyes osztályokban a d_j értéke megegyező legyen, és ezek az osztályok különböző részfába tartozzanak be. A módszer addig folytatódik rekurzívan, amíg az egyes a levelekbe csak azonos kategóriába tartozó tanulóadatok vannak. A túltanulást a döntése fa csonkolásával lehet megakadályozni. A témát részletesen a [104, 3. fejezet] tárgyalja.

Neurális hálózat alapú módszerek

A szövegosztályozást olyan neurális hálózattal valósítják meg, ahol a bemeneti réteg neuronjai a szavaknak felelnek meg, a kimeneti réteg a kategóriákat reprezentálja, a rétegek közti súly pedig a függőségi relációt jellemzi. Egy dokumentum osztályozása esetén a bemeneti neuronok értéke a dokumentum vektora lesz, és hálózat kimenete határozza meg a osztályozási döntést. A hálózat tanítása visszacsatolt módszerrel történik: ha egy szöveget rosszul kategorizál a hálózat, akkor a hibát visszacsatolva módosítjuk a súlyok értékét, ily módon minimalizálva a hibát.

A neurális hálózat alapú szövegosztályozó az *inkrementális módszer*ek közé tartozik, azaz az első néhány tanulóadat alapján felépített kezdeti osztályozót az újabb tanulódokumentumok vizsgálata során módosíthatja. Ez az adaptivitás előnyös lehet, ha a kategóriák tartalma módosul, vagy ha nem áll a tanulás kezdetén rendelkezésre az összes tanulóadat.

Az egyik legegyszerűbb esete ennek a *perceptron* algoritmus [34, 131, 157]. Kiinduláskor a bemeneti súlyok értékét azonosra állítjuk. A bináris reprezentációval (12.1) reprezentált $\vec{d}$ dokumentumot a már felépített osztályozóval kategorizáljuk. Ha ez sikeres, akkor semmit nem módosítunk rajta, viszont, ha nem, akkor az alábbi módon változtatjuk a súlyokat. A perceptron *additív súlybeállítás*t használ: ha $\vec{d}$ a c kategóriára pozitív példa, akkor „aktív” ($d_j = 1$) szavak súlyát $\alpha > 0$ *tanulási rá*val növeljük; ellenkező esetben pedig α-val csökkentjük. A tanulás végén a kicsiny súlyú szavak negatív példákat jelentenek a kategóriára vonatkozóan, így ki lehet őket hagyni a szótárból, ezzel is csökkentve a vektortér dimenzióját (vö. 12.1.1 szakasz) [34].

*Multiplikatív súlybeállítás*t alkalmaznak a különböző verziójú *Winnow* algoritmusok [34], ahol $\alpha_1 > 1$, ill. $0 < \alpha_2 < 1$ konstansokkal való szorzással a szavak súlyát rendre növelik, ill. csökkentik. A *kiegyensúlyozott* Winnow algoritmusa minden szóhoz két súlyt rendel, amiket a pozitív, ill. negatív példák külön szabályoznak. Az utóbbi esetben egy súly értéke negatív is lehet.

Az eddig ismertetett neurális hálózat alapú módszerek lineáris osztályozók, mivel a hálózat kimenete lineárisan függ a bemenettől. Egyszerűségük ellenére a leghatékonyabbak eljárások közé tartoznak. Több munka megvizsgálta a nemlineáris neurális hálózatok alkalmazását is egy vagy több rejtett réteget illesztve a hálózatba. Ez a módosítás azonban az osztályozó hatékonyságára vonatkozóan semmilyen [131] vagy csak igen csekély [157] javulást eredményez.

12.3. ábra. Optimális $\vec{w}$ kiválasztása lineárisan szeparábilis esetben

Support Vector Machine (SVM)

A számos más alkalmazási területen is jó eredményeket adó SVM eljárás egyike a leghatékonyabb szövegosztályozási módszereknek [76]. Csak bináris osztályozási feladat megoldására alkalmas, ezért egyszerű vagy általános osztályozás esetén ilyenek kombinációit alkalmazzuk.

Az SVM egy $\vec{d}$ vektorhoz az alábbi kifejezés alapján rendel 1 vagy -1 értéket:

$$s = \mathbf{w}^T \phi(\mathbf{d}) + b = \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{d}, \mathbf{d}_i) + b$$

és a kérdéses kategóriához való hozzárendelést az alábbi egyenlőtlenség adja meg:

$$y = \begin{cases} 1, & \text{ha } s > s_0 \\ -1, & \text{egyébként} \end{cases},$$

ahol $\vec{d_i}$ a tanulóhalmaz elemei, $y_i \in \{-1,1\}$ értéke pedig a vizsgált kategóriába való tartozást jelöli. A $K(\mathbf{d}, \mathbf{d}_i)$ kernel (mag) kifejezés értékét gyakran egy polinom határozza meg:

$$K(\mathbf{d}, \mathbf{d}_i) = (\mathbf{d}^T \mathbf{d}_i + 1)^d$$

Az SVM tanítása azon $\vec{w}$ vektor meghatározásából áll, amely maximalizálja a tanulóadatok két osztálya (bele, ill. nem bele tartozó) közötti távolságot. Fontos megjegyezni, hogy a legjobb $\vec{w}$ kiválasztásában csak a tanulóadatok egy része játszik szerepet, az ún. *tartóvektor*ok (support vectors) (ld. 12.3 ábra).

Az optimalizálást attól függően végezzük, hogy a kategóriához tartozó és nem tartozó vektorok lineáris szeparabilitása az $M-1$ dimenziós térben feltételezhető-e vagy sem. Ez utóbbi esetben némileg módosított eljárást kell alkalmazni [159], ami valamelyest jobb megoldást ad mint a lineárisan szeparálható eset [76].

A módszer jelentőségét tovább növeli, hogy nagy adathalmazok esetén is alkalmazható. Ez annak a tulajdonságának köszönhető, hogy a végső SVM-t a tanuló adatok kisebb részhalmazaira megalkotott SVM-ek kombinációiként is elő lehet állítani. A [41] közleményben egy olyan tanulóalgoritmust alkalmaztak, amely az SVM módszer tanulási sebességét a Rocchio-eljárásával összemérhetővé teszi.

Szavazásos osztályozás

Egy vagy több kiválasztott módszernek más-más tanulóhalmazon elvégzett eredményeit kombinálja a szavazásos osztályozás. Az osztályozó felépítése az alkalmazott módszerek (az osztályozók együttesét *bizottság*nak, elemeit *tagok*nak nevezik) és azok eredményének súlyozásától függően különböző lehet. A bizottság tagjainak kiválasztásánál általában azt a szempontot követik, hogy a tagok lehetőleg minél függetlenebbek legyenek, azaz különböző elven működjenek [151]. Az eredmények kombinációjára számos eljárás létezik [95], amelyek eltérő mértékben és módon veszik figyelembe a tagok hatékonyságát.

Az eredeti tanulóhalmazból kialakított ideiglenes tanulóhalmazok megvalósításától függően is több verziója létezik a szavazásos osztályozásnak.

Az egyik módszer [22] esetén az eredeti N elemű tanulóhalmazból *ismétléses módon* véletlenszerűen kiválasztunk N elemet, így az új tanulóhalmaz az eredetiből bizonyos elemeket többször, másokat egyszer sem tartalmaz. Az eredetiből kivett elemek gyakoriságát diszkrét Poisson-eloszlással modellezzük. Ezt R-szer elvégezve ugyanennyi különböző dokumentumgyűjteményhez jutunk, amire az osztályozó bizottságban résztvevő eljárásokat lefuttatva R eredményt kapunk. A vizsgált $\vec{d}$ dokumentumot ahhoz a kategóriához rendeljük hozzá, amelyikre a legtöbb tag „szavaz”:

$$y(\mathbf{d}) = \arg\max_{y} \sum_{r:\, f_r(\mathbf{d})=y} 1,$$

ahol f_r $(r = 1, \ldots, R)$ a bizottság tagjait jelöli.

Az *AdaBoost* eljárás verziói [127, 128] ugyanazt az osztályozót alkalmazzák egymás után különböző tanulóhalmazzal. Az egyes tanulóadatok súlyát a következő tanulóhalmazban adaptív módon attól függően változtatják, hogy milyen eredményt adott az előző osztályozásoknál. Egy dokumentum súlyát növelik, ha osztályozás sikertelenül volt, csökkentik, ha sikeres. A végső osztályozó az R-edik osztályozó eredményeként áll elő.

A bizottságokat osztályozók albizottságaiként összeállítva [133], illetve a bizottságok döntési fákkal való kombinációját [155] alkalmazva tovább lehet javítani a *boosting* típusú módszerek hatékonyságán.

Hatékonyságmérés

Az osztályozási módszerek hatékonysága a szokásos információ-visszakeresésben alkalmazott mértékek segítségével mérhető. Az egyszerűbb feladatok (bináris, egycímkés és többcímkés osztályozás) esetén ezek a mértékek közvetlenül alkalmazhatók. Tekintsük először az alábbi mennyiségeket egy kategóriára vonatkozóan:

a, a kategóriához helyesen hozzárendelt dokumentumok száma
b, a kategóriához helytelenül hozzárendelt dokumentumok száma
c, a kategóriához helytelenül nem hozzárendelt dokumentumok száma
d, a kategóriához helyesen nem hozzárendelt dokumentumok száma

Ezek segítségével a következő mértékeket definiáljuk:

$$\begin{aligned}
\textit{felidézés}\ (\text{recall}) = R &= \frac{a}{a+c} \\
\textit{pontosság}\ (\text{precision}) = P &= \frac{a}{a+b} \\
\textit{szabatosság}\ (\text{accuracy}) = A &= \frac{a+d}{a+b+c+d} \\
\textit{hiba}\ (\text{error}) = E = 1-A &= \frac{b+c}{a+b+c+d}
\end{aligned} \tag{12.6}$$

Ezek közül felidézés és pontosság mértékek együttesét alkalmazzák leggyakrabban. A szabatosságot szövegosztályozási feladatoknál ritkábban használják, ugyanis a rendszerint nagy nevező miatt ez a mérték kevésbé érzékeny az $a+d$ számláló változására, mint az előbbi kettő [132, 34. o.][158]. Mivel az R és P értékek maximalizálása egymással ellentétes feladat, ezért egy módszer értékeléséhez mindkettőre egyaránt szükség van. Ezt az ún. *egyensúlyi pont* meghatározásával érjük el, amire $P \approx R$.

Az egyensúlyi pontot az adott módszer paramétereinek változtatásával kaphatjuk meg. Itt problémát jelenthet az, hogy egyes módszereknél esetleg nincs ilyen paraméterbeállítás, illetve hogy a két érték azonossága nem feltétlenül kívánatos cél [93].

Másik lehetőség a két mennyiség parametrikus kombinációja [153],

$$F\text{-}mérték = F_\beta = \frac{(\beta^2+1)PR}{\beta^2 \cdot P + R}, \qquad \beta \geq 0 \tag{12.7}$$

ahol a $\beta = 1$ esetén a két mennyiség azonos súllyal szerepel. Ez a manapság leggyakrabban használt mérték az osztályozási módszerek kiértékelésére[9]

Az eddig tárgyalt mértékek egy kategóriára vonatkoztak, tehát a bináris feladat kiértékelésére alkalmasak. Könnyen lehet őket azonban átlagolással adaptálni egy- és többcímkés osztályozáshoz is (többszintű osztályozás kiértékelését a 12.2.2 pont megfelelő részében tárgyaljuk). Az átlagolást kétféleképpen lehet elvégezni: *mikró-átlagolt* mértékek esetén az összes dokumentumra külön kiszámolják az adott értéket, és azokat átlagolják; *makró-átlagolt* esetben pedig kategóriákra számolják a mértékeket, majd ezeket átlagolják. Tehát a mikro-átlagolás a dokumentumokhoz, míg a makro-átlagolás a kategóriákhoz rendel azonos súlyt.

Ha olyan osztályozó hatékonyságát mérjük, amelyik a tesztdokumentumokhoz kategóriák megbízhatósági szinttel ellátott rangsorát adja eredményül, akkor egy másik IR mértéket, a *11 pontos átlagos pontosság*ot használjuk. Ekkor egy tesztdokumentumra a felidézést a

$$R = \frac{\text{A listában szereplő helyes kategóriák száma}}{\text{Az összes helyes kategóriák száma}} \tag{12.8}$$

képlettel számoljuk. A **??**tm:eq:11pt:r) hányados 11 rögzített értékére $(0, 0{,}1, \ldots, 0{,}9, 1)$ meghatározzuk, hogy hány elemét kell a listának figyelembe venni (azaz a számláló mérete mekkora legyen), hogy a kívánt felidézés értéket érjük el. Evvel az értékkel számoljuk a pontosság értékét:

$$P = \frac{\text{A listában szereplő helyes kategóriák száma}}{\text{A listában szereplő összes kategória száma}}. \tag{12.9}$$

Végül pedig az így kapott 11 értéket átlagolva megkapjuk a módszert jellemző hatékonyságot egy dokumentumra vonatkozóan. A teljes $\mathcal{D}_{\text{Test}}$ halmazra vonatkozó globális hatékonyság értéke a fenti módon dokumentumonként kiszámolt értékek átlagaként határozható meg.

Kiértékelés és példák

A módszerek összehasonlítását standard dokumentumgyűjtemények segítségével végzik. A módszerek korrekt összehasonlításához teljesülnie kell, hogy

I. egyazon gyűjteményen, ugyanazokkal a dokumentumokkal és kategóriákkal teszteljünk;

II. ugyanazt a tanuló és tesztgyűjteményt használjuk;

III. ugyanazt a hatékonysági mértéket alkalmazzuk rögzített paraméter beállítással.

[9] Ezt a mértéket használták pl. a 2005-ös KDD kupára beadott eredmények értékelésére is: `www.acm.org/sigs/sigkdd/kdd2005/kddcup.html`

Bár a fenti irányelveket nem tartották mindig szem előtt a kutatások elvégzésekor, ennek ellenére a egyszerű szövegosztályozásnál leggyakrabban használt Reuters-21578[10] gyűjtemény, illetve annak különböző verziói a legalkalmasabbak az összevetés alapjának [132]. Ez a gyűjtemény SGML formátumú híranyagokat tartalmaz, amelyek 135 gazdasági jellegű kategóriába sorol be. A gyűjteménynek többféle tanuló- és tesztadatokra történő felosztása létezik, a legtöbben az Apté által javasolt felosztást [12] (9603 tanuló, 3299 teszt dokumentum), illetve ennek bizonyos módosításait, szűréseit használják. Egyes dokumentumok több, akár 14 kategóriába is tartoznak, mások akár egybe sem. A tanuló dokumentumok eloszlása is egyenetlen, a legnagyobb elemszámú kategóriának 2709 tanuló dokumentuma van, de a kategóriák feléhez kevesebb mint 10 dokumentum tartozik.

A legátfogóbb összehasonlítás a [132] cikkben található, ami alapján a következő megállapításokat tehetjük:

– A legjobb hatékonyságú osztályozók a boosting technikát alkalmazó bizottságok, az SVM, valamint a k-NN módszert alkalmazó algoritmusok.

– A neurális hálózat alapú módszerek szintén jó teljesítményt nyújtanak, bár az előző csoportba sorolt eljárásoknál valamivel rosszabb eredményt adnak. Speciális átmeneti függvény használatával azonban ez a módszer is az előző csoporthoz hasonló vagy akár jobb eredményeket tud elérni [147].

– A harmadik csoportba a Rocchio-eljárás és a naiv Bayes-alapú módszerek tartoznak, ezeknek a leggyengébb az osztályozó képességük. Itt fontos megemlíteni, hogy az előbbi a majdnem pozitív tanulóadatok alkalmazásával a **??**tm:eq:genro) képletben lényegesen javítható. Ide sorolhatók még a döntési fa alapú módszerek is, amelyek alapesetben szintén a legkevésbé hatékony eljárások közé tartoznak, de módosításokkal ez lényegesen javítható [41].

12.2.2. Hierarchikus osztályozás

Egyszerű szövegosztályozás esetén a dokumentumok számának növekedése, és a lefedett témakörök sokfélesége átláthatatlan méretű kategóriarendszert eredményezhet. Ezt a problémát a kategóriák hierarchizálásával, azaz *taxonómiá*ba rendezésével könnyen át lehet hidalni. Ennek bevezetése az osztálystruktúra átláthatósága mellett algoritmikusan hatékonyabb eljárások alkalmazását is lehetővé teszi.

A teljes taxonómián való osztályozási problémát az algoritmusok kisebb osztályozási feladatokra bontják, úgy, hogy a taxonómia minden belső csomópontjához rendelnek egyet. Általában a *mohó algoritmus*t vagy annak valamilyen gyengített változatát használják. Ez az algoritmus egy adott csomópontban megvizsgálja, hogy az aktuális dokumentum annak melyik gyerekkategóriájába tartozik leginkább, majd e kiválasztott kategóriából kiindulva rekurzívan folytatódik és terminál, ha levélhez ér.

A naiv Bayes-módszert alkalmazza hierarchikus osztályozásra a [100] munka, ahol a kevés tanulóadattal rendelkező levélkategóriák paramétereit (szóelőfordulások aránya) az ún. *shrinkage* (apadás) statisztikai simító eljárás segítségével határozza meg a szülő kategóriák megfelelő adatait felhasználva. A módszer segítségével a mohó algoritmus egyik jellegzetes hibája — ti. hogy a taxonómia felső szintjén elkövetett osztályozási hibát már nem lehet korrigálni — nagy részben kiküszöbölhető.

[10] `http://www.daviddlewis.com/resources/testcollections/reuters21578/`

A neurális hálózatok architektúrájának és a taxonómiáknak strukturális hasonlósága kézenfekvővé teszi a neurális hálózatok alkalmazását hierarchikus osztályozás esetén. A HITEC[11] [148] osztályozó egy ismeretlen d dokumentum kategorizálásánál a taxonómia gyökeréből indulva szintenként határozza meg a legvalószínűbb kategóriát, azaz minden szintet a neurális háló egy rétege reprezentál. A végeredményt a levélkategóriák szintjén kapjuk. Az eljárás két paraméter alkalmazásával bővíti az egy szinten kiválasztott kategóriák körét, hogy a mohó jellegű következtetés előző bekezdésben jelzett hibáját kiküszöbölje. Az egyikkel a kiválasztott kategóriák száma adható meg, a másikkal pedig az, hogy a kiválasztott kategóriáknál legjobbtól való mekkora eltérés engedhető meg.

Hatékonyságmérés

Taxonómiába való osztályozáskor több lehetőség van a hatékonyság mérésére a taxonómia kialakításától függően. Amennyiben a dokumentumok csak levélkategóriákba vannak besorolva, akkor az egyszerű osztályozásnál ismertetett mértékeket lehet alkalmazni a levélkategóriák összességére (ld. 12.2.1. szakasz). Ez azonban némileg félrevezető eredményt is adhat, hiszen általában „kevésbé rossz” az az osztályozási következtetés, amely egy levélkategória helyett annak testvérét találja meg (tehát a szüleik közösek), mint az amelyik a kategóriarendszer teljesen más ágához rendeli a dokumentumot. Ha egy dokumentumot nemcsak a levélkategóriához tartozónak tekintünk, hanem annak összes szülőjéhez is hozzárendeljük[12], akkor pontosabb képet kaphatunk az osztályozás értékelésekor, feltéve ha a teljes — tehát nem csak levélszintű kategóriákra — taxonómiára számoljuk a pontosság, felidézés, F-mérték értékeit. Különösen indokolt ez akkor, ha vannak olyan dokumentumok, amelyek a taxonómia közbenső csomópontjaihoz vannak rendelve.

Valamelyest leegyszerűsítve az algoritmusok általában a hierarchikus osztályozást a taxonómia csomópontjaira dekomponált egyszerű osztályozási feladatok sorozataként oldják meg. Ezért a taxonómiába egyre lejjebb jutva, az osztályozási hibák összeadódnak, és egyre kevésbé lesz pontos az eredmény. Ez a tendencia jól megfigyelhető, ha a szokásos mértékeket (pontosság, felidézés, F-mérték) *szintenként* számítjuk ki.

Hierarchikus osztályozás esetén gyakran találkozunk a többszintű osztályozás problémájával, amikor tehát egy dokumentumnak vannak elsőrendű, másodrendű stb. kategóriái. Itt a kétszintű osztályozás esetével foglalkozunk[13]. A szakirodalom az egyszerű osztályozástól eltérő mérőszámokat javasol a hatékonyság mérésére erre az esetre, amelyeket célzottan a későbbiekben ismertetésre kerülő szabadalmi teszt-dokumentumgyűjteményhez alakítottak ki (ld. 12.4. ábra).

I. **Top:** Az osztályozó által legnagyobb konfidenciaértékkel meghatározott kategóriát hasonlítja a dokumentum elsődleges kategóriájához.

II. **Top 3:** Az osztályozó által javasolt három legnagyobb konfidenciaértékkel bíró kategóriát hasonlítja a dokumentum elsődleges kategóriájához. Ha a három közül valamelyik talál, akkor az osztályozás sikeresnek számít.

III. **Any:** Az osztályozó által legnagyobb konfidenciaértékkel meghatározott kategóriát hasonlítja a dokumentumhoz tartozó összes (elsődleges, másodlagos) kategóriákkal. Ha valamelyikkel megegyezik, akkor az osztályozás sikeresnek számít.

[11] `http://categorizer.tmit.bme.hu`

[12] A kategóriák sorozatát ekkor *kategóriaösvény*nek nevezzük.

[13] Természetesen a taxonómia szintjeinek számára nem teszünk megkötést.

12.4. ábra. Magyarázat a többszintű osztályozásnál alkalmazott mértékekhez (mc – fő kategória; ic – egyéb kategória; A – osztályozó eredménye; B – eredeti érték) [46]

Kiértékelés és példák

Mivel a hierarchikus osztályozással csak a 90-es évek végétől kezdtek el foglalkozni, ezért sokáig nem volt olyan dokumentumgyűjtemény, amelyen a különböző módszereket össze lehetett volna hasonlítani. A kutatók ezért a különböző korpuszokon tesztelték algoritmusaikat, pl. Reuters-gyűjtemény kategóriáit[14] rendezték különböző taxonómiákba [27, 35, 85, 156] és ezen végezték méréseiket. Ezek az eredmények azonban csak hozzávetőlegesen hasonlíthatóak össze, hiszen a 12.2.2. pontban ismertetett irányelvek nem teljesültek, sőt még a kategóriák halmaza (taxonómia) is többnyire eltért.

A szabadalmi hivatalokban több feladathoz is nagy segítséget jelenthet a hierarchikus osztályozók alkalmazása. A szabadalmak feldolgozása során a beadványokat emberi munkával elemzik és továbbítják a megfelelő szakcsoporthoz, akik a szabadalom szakmai elbírálását és besorolását elvégzik. A szakcsoportok meghatározása automatikussá tehető, vagy felügyelt félautomatikus módon is végezhető, mivel a osztályozó eljárások pontossága itt elegendő. Az osztályozó rendszer további segítséget adhat a szakértőknek is, amennyiben javaslatokat ad a beadványok kategóriájának a meghatározásához.

Természetesen más intézmény is hatékonyan alkalmazhatja ezeket a módszereket, hiszen a bejövő dokumentumok rendszerezése általános feladat akár állami, önkormányzati, vagy ipari intézményekben is. Mindazonáltal a szabadalmi hivatalok esetében rendelkezésre állnak a szükséges előfeltételek: a jól definiált, rögzített taxonómia és a nagy számú tanulóadat.

Ennek az érdekeltségnek is köszönhető, hogy az első, kimondottan hierarchikus osztályozás algoritmusok validálására alkalmasa teszt-dokumentumgyűjteményt a WIPO (World Intellectual Property Organization – Nemzetközi Szellemi Tulajdonok Szervezet) bocsátotta közzé 2002 végén [46], amely angol nyelvű szabadalmi szövegeket tartalmazott, majd nem sokkal később német nyelvű gyűjteményt is közzétettek [45]. Az angol gyűjtemény mintegy 75000 XML formátú dokumentumból áll, amely összesen 3 GB adat, a német gyűjtemény összesen 110 ezer XML dokumentumot tartalmaz. A gyűjtemények fel vannak osztva tanuló- és tesztadatokra. A dokumentumok az IPC (Internatial Patent Classification – Nemzetközi Szabadalmi Osztályozás) kategóriarendszerének[15] felső négy szintjébe (osztály, szekció, alszekció, főcsoport) vannak besorolva, amely kb. 5000 kategóriát jelent összesen. Minden dokumentumnak pontosan egy elsőrendű (fő) kategóriája és tetszőleges számú, átlagosan 4–5 másodrendű kategóriája van.

Ezen a gyűjteményen végzett átfogó összehasonlítást a 12.4. ábrán látható mértékekkel egy nemzetközi kutatócsoport [47]. Munkájukban a naiv Bayes-eljárás, a legközelebbi szomszédok módszer, az SVM, és a Winnow egy-egy hierarchikus osztályozásra specializált verzióját hasonlították össze különböző tanulási halmazok mellett. Az módszerek hatékonyságát szekció és alszekció szintjén vizsgálták, az eredményeket a 12.2. tartalmazza. Ugyanezen a gyűjteményen a neurális hálózat alapú HITEC-et is tesztelték, és lényegesen jobb eredményeket kaptak: a taxonómiában egy szinttel lejjebb volt képes a HITEC a többi módszer által egy szinttel feljebb elért eredményre [149]. Ez alapján megállapítható, hogy a taxonómia topológiáját kihasználó neurális hálózati architektúrán működő algoritmus kedvezőbb eredményeket szolgáltat.

[14] Gyakran csak egy kisebb részhalmazt.

[15] `http://www.wipo.org/classifications/fulltext/new_ipc/index.htm`

12.2. táblázat. A WIPO-alpha angol nyelvű szabadalmi dokumentumgyűjteményen elért eredmények összehasonlítása a legalacsonyabb konfidenciaszinten (A módszerek nevének rövidítése: NB – Naïve Bayes, SVM, *k*-NN – legközelebbi szomszédok módszere)

Módszer/ forrás	Mérték	IPC szint		
		szekció	alszekció	főcsoport
HITEC	Top	66.41	54.63	38.38
[47]	Top	55.00 NB, SVM	41.00 SVM	–
HITEC	Top3	89.41	79.48	59.64
[47]	Top3	79.00 NB	62.00 *k*-NN	–
HITEC	Any	76.46	66.36	50.90
[47]	Any	63.00 NB	48.00 SVM	–

Másik nagyméretű dokumentumgyűjtemény a Reuters Corpus Volume 1 (RCV1)[16], amely mintegy 800 ezer híranyagot tartalmaz, és három különböző taxonómiába vannak az XML dokumentumok besorolva (téma szerint, ipari kód szerint, és területi kód szerint). A kategóriák száma azonban itt sokkal kisebb mint a szabadalmi korpuszok esetében, mindössze 103 téma, 364 ipari és 366 területi kódú kategóriát tartalmaz. Bár gyűjtemény egyes részeit már többen feldolgozták, teljes körű vizsgálat készítése még várat magára.

12.3. Dokumentumok csoportosítása

Ahogy azt az adatbányászati rész vonatkozó fejezete is kiemeli (ld. a **??**. szakaszt a **??**. oldalon), a *csoportosítás*, avagy *klaszterezés* sokban hasonlít az osztályozáshoz, ugyanakkor két alapvető eltérést mutat, ekkor ugyanis nem ismert

I. a dokumentumok címkéje, továbbá a feladat elvégzése után sem jellemezhetők általában a csoportok automatikusan címkékkel;

II. hogy a dokumentumhalmaz hány csoportot alkot.

Összefoglalóan: többnyire nincsen olyan referenciaadat amihez hasonlítani lehetne a csoportosítás eredményét, vagyis tanulási szempontból a klaszterezés *felügyelet nélküli tanuló módszer*. A csoportosító algoritmusokat ezért akkor alkalmazzuk, amikor nem áll rendelkezésre rögzített kategóriarendszer (taxonómia) a hozzátartozó tanulóadatokkal.

12.3.1. Szövegklaszterezés jellemző feladatai és problémái

A csoportosító eljárások tehát hasonló típusú feladatok megoldására alkalmasak mint az osztályozók. Bár a kezdeti motivációt az információ-visszakereső rendszerek hatékonyságának növelése

[16] http://about.reuters.com/researchandstandards/corpus/

jelentette [153], az utóbbi években inkább az internetes és intranetes keresési feladatok támogatása vált a jellemző céllá. Szövegklaszterező eljárást alkalmaztak dokumentumgyűjtemények böngészésének támogatására [**?**], illetve internetes keresések eredményeinek csoportokba szervezésére [**?**]. Szintén gyakori probléma dokumentumok hierarchikus klaszterekbe rendezése [85], az internetes dokumentumokhoz automatikus taxonómia generálása[17], továbbá már meglévő taxonómia osztályok dokumentumainak további csoportosítása, amelyet aztán fel lehet használni a taxonómia finomítására.

Ha a feladat nem numerikus, hanem szöveges adatok csoportosítása, akkor ebből adódóan a következő jellegzetességeket kell kezeli [**?**]:

– Az adatok dimenziószáma legalább 10 000-es nagyságrendű. Mivel a dokumentumokat reprezentáló vektorok viszont rendkívül ritkák, a módszereknek ezt a dichotómiát tudnia kell kellően kezelni.

– A dokumentumgyűjtemények nagy mérete (különösen a világháló esetében) miatt a módszereknek hatékonyan kell működnie, és skálázhatónak kell lennie.

– A klaszterek neveinek érthetőnek kell lennie, mivel ezek tájékoztatják a felhasználót (pl. böngészés során) a csoportba tartozó dokumentumok tartalmáról.

Szövegklaszterezés általános feladata ezek alapján nagy méretű dokumentumhalmaz elemeit csoportokba rendezni úgy, hogy azonos csoportba kerüljenek a hasonló témával foglalkozó dokumentumok.

12.3.2. Reprezentáció

A dokumentumok reprezentálására a szokásos vektortér-modellt alkalmazzuk (12.1. szakasz). A dokumentumokat általában szavak szintjén dolgozzuk fel, a szótárba pedig a nemtriviális szavak kanonikus alakjai kerülnek. A szövegklaszterező módszerek a dokumentumok tartalmi hasonlóságát a bennük szereplő szavak együttes előfordulásai alapján határozzák meg. A vektortér-modellben ez a feladat a dokumentumvektorok távolságának hasonlósági mértékek segítségével való meghatározását jelenti. Mivel dokumentumvektorokban tárolt értékek folytonosak, ezért a **??**. pontban ismertetett mértékek alkalmasak a hasonlóság, ill. különbözőség vizsgálatára — szövegklaszterezés esetén az euklideszi- (ld. (**??**)) más néven koszinusz-távolságot használjuk leggyakrabban.

12.3.3. Hatékonyság mérése

A csoportosítás minőségének vizsgálatát két típusú mértékkel lehet vizsgálni. Az első típusba az ún. *belső mértékek* tartoznak, amelyek nem használnak fel külső tudást a csoportosítás jóságának meghatározására. A második típusba a *külső mértékek* tartoznak, amelyeket akkor lehet alkalmazni, ha rendelkezésre állnak a dokumentumok osztálycímkéi, ekkor ezeket hasonlítjuk össze a címkéket a klaszterező által meghatározott csoportokkal.

A belső mértékek például a *csoportok belüli közelség* és a *csoportok közti távolság* mértékek különböző típusai, amelyeket a **??**. pont ismertet. Külső mértékek közül az *entrópiá*t és az *F-mérték* csoportosításnál alkalmazott verzióját tárgyaljuk, amelyeket a 12.3.4. szakaszban a módszerek kiértékelésénél használunk.

[17] A `www.yahoo.com`-hoz hasonló könyvtár-struktúra automatikus felépítése.

Az entrópia [**?**] mértéknél először az osztályok adatelosztási értékét számoljuk ki, azaz minden j csoportra meghatározzuk annak a p_{ij} valószínűségét, hogy e csoport eleme az i osztályba tartozik. A p_{ij} érték segítségével a j klaszter entrópiáját a

$$E_j = -\sum_{i}^{C} p_{ij} \log(p_{ij}) \tag{12.10}$$

kifejezés adja meg, ahol $c_i\, i \in [1, C]$ jelöli a kategóriákat. Végül a csoportosítás entrópiáját a **??**tm:eq:entropy) értékek csoportméret szerint súlyozott átlagaként kapjuk meg:

$$E = -\sum_{j=1}^{K} \frac{n_j E_j}{N}, \tag{12.11}$$

ahol K a csoportok száma, n_j a j-edik csoport elemszáma, N pedig a dokumentumok száma. Egy módszer annál jobb minél kisebb az entrópiája.

Az F-mértéket csoportosításnál az alábbi módon számoljuk [**?**]. Legyen adott a j csoport, és az i osztály. Ekkor a j csoporthoz tartozó felidézés és pontosság a

$$R(i, j) = n_{ij}/n_i \qquad P(i, j) = n_{ij}/n_j \tag{12.12}$$

képletekkel számolható, ahol n_{ij} az j csoportban lévő i osztálybeli elemek száma. A j csoportra vonatkozó F-mértéket a két mennyiség (12.7) kifejezés szerinti kombinációjaként kapjuk: $F_1(i, j) = (2R(i, j)P(i, j)) \,/\, (R(i, j) + P(i, j))$, az összesített F-mérték pedig súlyozott átlagként áll elő:

$$F_1 = \sum_{i}^{C} \frac{n_j}{N} \max_{j \in [1,K]} \left(F_1(i, j)\right). \tag{12.13}$$

12.3.4. Szövegklaszterező eljárások

Ebben a szakaszban a szöveges adatok csoportosítása alkalmazott hierarchikus és particionáló eljárásokat tekintjük át.[18]

A módszerek összehasonlításánál körültekintően kell eljárni, és csak akkor lehet valamely eljárást egy másiknál jobbnak tekinteni, ha különböző mértékek és korpuszok esetén a legtöbb esetben jobb eredményt ad.

Hierarchikus klaszterezők

A [**?**] tanulmányban három *egyesítő hierarchikus klaszterező*t hasonlítanak össze nyolc különböző korpuszokon (ld. 12.3.5. pontot is); a módszerek csak az egyesítendő párok kiválasztásában különböznek. A vizsgált eljárások a *centroid kapcsolódás*, *centroid–egyszerű kapcsolódás*,[19] és az *UPMGA* módszer [**?**]. Ez utóbbi a $s(x, y) = \frac{\sum_{\vec{d}_1 \in x, \vec{d}_2 \in y} \cos(\vec{d}_1, \vec{d}_2)}{n_x, n_y}$ hasonlósági mértéket alkalmazza.

A módszerek közül az UPGMA adja a legjobb eredményt az F-mérték szerint az összes vizsgált gyűjtemény esetén, bár a másik két módszer sem ad lényegesen rosszabb értékeket. Entrópia mérték

[18] Természetesen ezen kívül még sok más eljárás is ismert, többek közt valószínűségi és fuzzy alapú módszerek, de ezek ismertetése meghaladják e könyv kereteit.

[19] Először minden csoportra kiszámolják a csoporton belüli hasonlóságot, majd azt a két csoportot vonják össze, ahol a $s(z) - (s(x) + s(y))$ érték a legkisebb. Itt x és y összevonásából keletkezik z csoport.

szerint a UPGMA és a centriod-egyszerű (CE) kapcsolódás közel azonos eredményeket ad, míg a centroid kapcsolódás a másik kettőnél lényegesen rosszabb. Megfigyelhető, hogy a kezdeti fázisban még hasonló eredményeket ad mindhárom módszer, de később a CE kezd több hibát véteni [**?**]. Ebből megállapítható, hogy a vizsgált eljárások közül az UPGMA bír a legkedvezőbb tulajdonságokkal.

K-átlag klaszterezők

A particionáló algoritmusok egyik fajtája a k-átlag típusú klaszterező (ld. **??**. pont). Először ennek egy szövegcsoportosításra hatékonyan alkalmazható módosítását, a *kettészelő k-átlag* (bisecting k-means) eljárást ismertetjük, majd összehasonlítjuk az eredeti k-átlag eljárással.

Az algoritmus a teljes dokumentumhalmazból indul ki, és a következő lépésekből áll:

I. Válasszunk ki egy felosztandó klasztert.

II. Osszuk pontosan két részre a k-átlag eljárás segítségével (kettészelő lépés).

III. Végezzük el a 2. lépést *i*-szer[20], és válasszuk ki azt a vágást, amelyik a legnagyobb csoporton belüli közelséget adja.

IV. Ismételjük meg a fenti 3 lépést, ameddig a szükséges csoportszámot nem érjük el.

Az első lépésben több módon választhatjuk ki a felosztandó klasztert; ez lehet pl. legnagyobb méretű csoport, vagy a legkisebb csoporton belüli közelséggel bíró csoport.

A kettészelő k-átlag módszer előnye, hogy mind hierarchikus mind elkülönülő csoportokat lehet vele generálni, tehát szigorúan véve az eljárás *felosztó hierarchikus klaszterező*nek tekinthető. A módszernél lehetőség van a csoportok finomítására is[21], ha az eredményül kapott klaszterekből kiindulva a k-átlag eljárást lefuttatjuk. Az eljárás időigénye — finomítással is — lineáris a dokumentumok számának függvényében.

A módszert a [**?**] közleményben összehasonlították az eredeti k-átlag eljárással és a UPGMA egyesítő hierarchikus klaszterezővel F-mérték és entrópia tükrében, amely alapján az alábbiak állapíthatók meg:

– A kettészelő k-átlag módszer mind a k-átlag, mind az UPGMA módszernél jobb a vizsgált 8 korpusz legtöbbjén (mindkét mérték szerint).

– Az UPGMA eredményeinek k-átlag módszerrel történő finomítása lényegesen javít mindkét mérték szerint az eredményeken.

– Az eredeti k-átlag módszer jobb eredményeket ad, mint a alap és a finomított UPGMA eljárás.

– Noha a két k-átlag alapú eljárás eredményei több futás átlagaként álltak elő, ezeknek a többszörös futási ideje sem éri el az egyesítő hierarchikus UPGMA futási idejét, mivel egy futáson a különbség mintegy 80–100-szoros.

Az egyesítő hierarchikus algoritmusok szövegklaszterezésen való gyenge teljesítményére a magyarázat a dokumentumok jellegzetességében rejlik. Az osztályozott szövegek alapján minden osztályhoz rendelhető egy szótár, amely a tipikus szavakat tartalmazza. Ugyanakkor valamely osztályba

[20] Különböző centroidokból kiindulva, mindig más és más lesz a két csoport.

[21] Nem csak ebben az esetben, hanem az összes hierarchikus klaszterezőnél, pl. UPGMA.

eső dokumentum nemcsak osztályának szótárából tartalmaz szavakat, ráadásul ezek az osztályszótárak a többértelmű szavak, vagy tematikusan közeli kategóriák esetén át is fedhetnek.

Egy szavak dokumentumonkénti eloszlásának jellege miatt, gyakran előfordul, hogy egy dokumentum legközelebbi szomszédja másik kategóriába tartozik. Az ilyen legközelebbi szomszédok aránya a vizsgált korpuszok esetében a 5 és 30% között volt! Minél távolabbi szomszédokat tekintünk, ez az arány természetesen annál nagyobb lesz.

Az egyesítő hierarchikus algoritmusok működésének jelegéből adódóan, a módszer során elkövetett hiba nem korrigálható később. A k-átlag módszerrel történő finomítás ezért javítja lényegesen az eredményeket, mert ott lehetőség van dokumentumok csoportok közti mozgatására is.

A k-átlag módszerek ezen tulajdonságuknál fogva nem érzékenyek a hamis közeli szomszédok jelenségére, és ezért jobb eredményt szolgáltatnak dokumentumokra.

A kettészelő k-átlag módszer hatékony működésének az az oka, hogy ha az 1. lépésben mindig a legnagyobb elemszámú csoportot választjuk felosztásra, akkor a keletkező csoportok mérete hasonló lesz. Mivel jellemzően a kis csoportok jobb minőségűek, viszont a kiértékelő függvényekben a nagyobb méretű csoportokat minősége nagyobb súllyal szerepel, ezért a k-átlag módszer — amely nagyon különböző méretű csoportokat gyárt — általában rosszabb eredményt ad.

12.3.5. Dokumentumgyűjtemények

A 12.3. táblázatban klaszterezési algoritmusok elemzésére használt dokumentumgyűjtemények találhatók. A RE0 és RE1 korpuszok a már ismertetett Reuters-adatok részhalmazaként állt elő (ld. 170. oldal). A TR31 és TR45 a TREC gyűjteményben találhatóak[22], a kategóriák címkéi pedig az ugyanott megadott fontossági értékek alapján adhatók meg[23]. Szintén TREC gyűjtemény az FBIS, illetve az LA1 és LA2, amelyek rendre a Foreign Broadcast Information Service és a Los Angeles Times kollekciók adatait tartalmazza. Ez utóbbi esetben az osztálycímkéket a cikkek rovatai alapján határozták meg. Végül a WAP gyűjtemény a WebACE projekt [**?**] keretében a Yahoo! taxonómiából összegyűjtött felcímkézett dokumentumokat tartalmaz.

Szintén több kutató használta a CLASSIC3 tesztkorpuszt, amely 1400 repülésügyi rendszereket (CRANFIELD) tárgyaló, 1033 orvosi témájú (MEDLINE), és 1460 információ-visszakereséssel foglalkozó (CISI) dokumentumot tartalmaz [24].

12.4. Kivonatolás

Internetes keresés esetén szinte mindenki találkozott már azzal a problémával, hogy a keresőmotorok által talált honlapok legalább egy része nem felel meg a felhasználó információigényének. A felhasználó részéről a keresőszolgáltatás által adott rövid cím és pár soros leírás alapján annak eldöntése, hogy egy adott dokumentum releváns-e számára szintén nem egyszerű feladat. Ehhez olykor a teljes dokumentumot le kell tölteni és át kell futni, azaz időigényes munkát jelent. A keresőszolgáltatások és/vagy a tartalomszolgáltatók (honlap/dokumentum készítői) részéről szintén nem várható el, hogy automatizálás nélkül emberi és anyagi erőforrásokat állítson a cél érdekébe.

[22] TREC: Text REtrieval Conference. `http://trec.nist.gov`

[23] Részleteket ld. [**?**]; forrás `http://trec.nist.gov/data/qrels_eng/index.html`

[24] `ftp://ftp.cs.cornell.edu/pub/smart`

12.3. táblázat. Klaszterező eljárások elemzésére használt dokumentumgyűjtemények adatai (a jelölések feloldását ld. a szövegben) [**?**]

Név	Forrás	Dokumentumok száma	Kategóriák száma	Min osztály-méret	Max osztály-méret	Átlagos osztály-méret	Szótár mérete
RE0	Reuters-21578	1504	13	11	608	115.7	11465
RE1	Reuters-21578	1657	25	10	371	66.3	3758
WAP	WebAce	1560	20	5	341	78.0	8460
TR31	TREC	927	7	2	352	132.4	10128
TR45	TREC	690	10	14	160	69.0	8261
FBIS	TREC	2463	17	38	506	144.9	2000
LA1	TREC	3204	6	273	943	534.0	31472
LA2	TREC	3075	6	248	905	512.5	31472

Ebben a szakaszban olyan szövegbányászati módszereket vizsgálunk, amelyek ezt a feladatot, tehát a dokumentumok összegzését automatikusan elvégzik. Ezeket a módszereket összefoglalóan *összegzéskészítő eljárások*nak nevezzük.

12.4.1. Az összegzéskészítő eljárások felosztása

Ezeket a módszereket összefoglalóan *összegzéskészítő eljárások*nak nevezzük, amelyeket a szakirodalom az összegzés előállítása alapján két alapvetően különböző csoportba oszt: *kivonatolás*nak (extraction) hívjuk az olyan eljárást, amelynek eredménye kizárólag az eredeti szövegből vett részeket tartalmaz, míg ezzel szemben az *összefoglalás-készítő* (abstraction) módszerek által előállított szöveg, olyan elemeket is tartalmaz, ami nem része a feldolgozott dokumentumnak.

Az emberi gondolkodás és információfeldolgozás modellezése — így az összegfoglalás-készítésé is — bonyolult feladat. Az összefoglalás függ a készítő személyétől, szaktudásától, különbözhet méretben, nyelvezetben, stílusban és részletezettségben. Az összegfoglalás-készítés folyamatának matematikai vagy logikai formulákkal való leírása rendkívül komplex feladat [**?**]. Az utóbbi években a nyelvtechnológiai eszközök fejlődése azonban lehetőséget adott olyan rendszerek megalkotására amelyek képesek szövegek szemantikai feldolgozására is. Ilyen eszközök segítségével, a szövegben található frázisok és lexikai láncok meghatározásával, majd azok összefűzésével, lehetőség van *összegfoglalások* automatikus *generálására*. Ennél lényegesebben egyszerűbb a kivonatoló eljárások működése, ahol az eredeti szövegben meglévő, azt leginkább jellemző szövegegységek (mondatok, bekezdések, stb.) kiválasztása a cél.

A kivonatoló eljárások hátránya:

– Az ily módon kiválasztott mondatok jellemzően az átlagosnál hosszabbak (ld. 12.4.3. pont). Mivel az ilyen mondatoknak egyes részei gyakran nem tartalmaznak lényegi információt, az feleslegesen kerül be a kivonatba.

– A dokumentumokban lévő fontos információegységek általában az egész dokumentumban elszórtan vannak jelen, és ezt a kivonatoló módszerek nem képesek feldolgozni.

– A szövegben szereplő ellentmondó információkat a kivonat nem dolgozza fel megfelelően.

Az összefoglaló eljárások hátránya:

– A felhasználók jobban kedvelik a kivonatolással készült összegzést, mint a generált összefoglalókat [**?**]. Ennek oka, hogy a kivonat a szerző eredeti kifejezéseit, szóhasználatát tartalmazza, valamint esetlegesen lehetőséget nyújt a sorok közötti információk olvasására is.

– A mondatszintézis területe jelenleg még angol nyelvre is gyerekcipőben jár, ezért az automatikusan generált szövegekben gyakran még mondaton belül is ellentmondás, van, így az egész szöveg könnyen összefüggéstelenné válik. Kivonat esetén inkoherencia csak a mondatok határainál fordul elő.

Mivel a legtöbb működő alkalmazás a kivonatolás módszerét alkalmazza, ezért a továbbiakban erre fókuszálunk.

A felhasználási cél alapján az összegzéskészítő eljárásokat az alábbi szempontok szerint lehet rendszerezni [**?**].

– Részletezettség: *indikatív* vagy *informatív*. Az indikatív összegzés azt tartalmazza, hogy a szövegnek mi a témája, míg az informatív összegzés ugyanannak egy speciális részletét tárgyalja.

– Tartalom: *általános* vagy *kérdés-vezérelt*. Az összegzés lehet egy dokumentum tartalmának általános leírása, vagy kiemelheti a tartalomnak a felhasználó által megadott kérdéssel kapcsolatos részét.

– Megközelítés: téma, ill. típus specifikus vagy független. A tapasztalatok azt mutatják, hogy különböző típusú (pl. rövidhír, tudományos publikáció) dokumentumokban a lényegi információ más helyen található.

12.4.2. A kivonatolás hatékonyságának mérése

Elsőként megvizsgáljuk, hogy milyen módszereket és mértékeket alkalmaznak a kivonatolás eredményének kiértékelésére, hogy ezáltal könnyebben érthető legyen, melyek az egyes módszerek előnyei és hátrányai. Egy összegzés megítélése személyenként változó lehet, függetlenül attól, hogy automatikus vagy ember által készített anyagról van szó.

A kivonatoló technikák kiértékelésére az 1960-es években Edmundson által javasolt mértéket [**?**] használják még ma is a leggyakrabban. Az automatikusan generált kivonatokat szakértők által mondatkiválasztással elkészített kivonatokkal vetik össze meghatározva a megegyező mondatok számát. Ezután a szokásos IR mértékekkel — pontosság, felidézés — jellemzik a kivonatolás minőségét. Ennek a módszernek a hátránya, hogy szükséges hozzá emberi előfeldolgozás, ugyanakkor ebben rejlik az erőssége is, hiszen ha egy módszer ezen mérték alapján valamely tanulóadat-halmazon jól teljesít, akkor várhatóan ismeretlen szövegeken is jól működik, a felhasználó számára jól érthető, hasznos kivonatokat generál.

A szakirodalom a fentieken kívül még az alábbi szempontokat tekinti iránymutatónak egy kivonat hasznosságának és teljességének megítélésében [**? ?**]:

I. Meg tudja-e válaszolni a felhasználó mindazokat a kérdéseket a kivonat elolvasása után, amelyekre az egész szöveg elolvasása esetén képes lenne?

II. Mi a tömörítési aránya a kivonatnak az eredeti szöveghez képest?

III. Van-e a kivonatolt szövegben ismétlődés, redundancia?

Ugyanakkor a kivonatok egyéb jellemzőit, pl. intelligencia, kohézió, összefüggés, olvashatóság sokkal nehezebb értékelni.

A kivonatolás minőségére vonatkozóan megkülönböztetnek belső és külső mértékeket [**?**], aszerint, hogy csak a kivonat tulajdonságait veszi-e figyelembe az adott mérték, vagy a kivonat minőségét valamely más cél elvégzésében nyújtott támogatás hatékonyságának tükrében vizsgálják. A felsorolt mértékek közül a második az előbbi, míg az első az utóbbi kategóriába tartozik. Kizárólag a tömörítési arány nem megfelelő jellemzője a kivonat minőségének, hiszen pl. a redundanciát, vagy az információ hasznosságát nem veszi figyelembe.

12.4.3. Mondatkiválasztásnál használt jellemzők

A mondatkiválasztással működő kivonatoló technikák úgy működnek, hogy a dokumentum minden egyes mondatához hozzárendelnek egy heurisztikus módon meghatározott értéket, és a legmagasabb pontszámmal rendelkező mondatokat teszik bele a kivonatba. A mondatokhoz rendelt értéket az alábbi tényezők növelik:

- **Kulcsszó-előfordulás:** Azok a mondatok, amelyekben a szöveg leggyakoribb szavai szerepelnek, általában jól reprezentálják a dokumentumot.
- **Cím-kulcsszó:** A címben szereplő szavak általában utalnak a dokumentum tartalmára is, ezért az olyan szövegközi mondatok amelyekben címszavak szerepelnek általában az átlagosnál jobban jellemeznek egy dokumentumot.
- **Előfordulási hely heurisztika:** Újsághírek esetén többnyire az első mondat, technikai–tudományos szövegeknél az összefoglalás utolsó mondatai, illetve a konklúzió tartalma jól jellemzi az adott dokumentumot.
- **Utaló frázisok:** Az olyan kulcsszavakat tartalmazó mondatok, mint pl. „ez a cikk”, „a tanulmány”, „jelen munkánkban” az átlagosnál több információt hordoznak a szöveg egészéről.
- **Nagybetűs szavak:** Rövidítéseket, vagy tulajdonneveket tartalmazó mondatok általában nagyobb információ tartalommal bírnak.

A mondatokhoz rendelt értéket az alábbi tényezők csökkentik:

- **Rövid mondatok kiszűrése:** A kivonatban jellemzően nincsenek rövid, néhány szavas mondatok.
- **Névmások:** Személyes, vonatkozó, birtokos, stb. névmásokat tartalmazó mondatok csak akkor kerülnek be a kivonatba, ha meghatározható, hogy mire utalnak. Ekkor az utalt szó kerül a kivonatba kerülő mondatban a névmás helyére.
- **Informális és pontatlan szavak:** A gyakori és sok jelentéssel bíró, vagy pontatlan szavak negatív tényezők a mondat kiválasztásnál.
- **Idézésre utaló szavak:** Angol nyelvű híreknél jellemző idézésre utaló szavak szintén negatív faktorok: *adding, said, according*, stb.

– **Redundancia-csökkentés**: Ezt a pontszámot olyan eljárásokban alkalmazzák, ahol egyenként határozzák meg a kivonatba kerülő mondatokat. Az értéket minden új mondat kiválasztásánál újraszámolják, megelőzendő azt, hogy a kiválasztott mondat valamelyik már korábban a kivonatba került mondathoz hasonlítson, pl. úgy, hogy arányosan csökkentik a még nem beválasztott mondatok pontszámát aszerint, hogy mennyire hasonlítanak az aktuális kivonathoz [**?** **?**].

Az alkalmazott jellemzők jellege szerint megkülönböztethetünk nyelvi, statisztikai, ill. információelméleti, és végül kombinált módszereket.

12.5. A legfontosabb kivonatoló eljárások

12.5.1. A klasszikus módszer

Bár az automatikus összegzés készítő eljárások csak az Internet és a keresőmotorok széleskörű elterjedésével kerültek a kutatások homlokterébe, az első eredmények e témában már az 60-es évek elején megszülettek [**?**]. Edmundson korai munkájában összefoglalta az akkor ismeretes eljárásokat, és lerakta a kivonatolási technikáknak mind a mai napig érvényben lévő alapjait [**?**]. Módszere az alábbi lépésekből áll:

I. Emberek által készített kivonatok tanulmányozásának segítségével határozzuk meg azokat az automatikusan generált kivonatok esetén elvárt jellemzőket.

II. Készítsünk ennek megfelelő kivonatokat emberi munkával.

III. Tervezzünk olyan matematikai és logikai formulákat a mondatok pontozására és rangsorolására, hogy a kívánt (manuálisan gyártott) eredmény kapjuk.

IV. A pontozási-kiválasztási rendszert finomítása mellett addig ismételjük a módszert, amíg a manuálisan és automatikusan generált kivonatok nem lesznek azonosak.

Edmundson rendszere az alábbi jellemzőket vette figyelembe a pontozási-kiválasztási rendszer paraméterezése során. Az ún. funkció szavak kiszűrése és szótövesítés (ld. még 12.7. szakasz) elvégzése után az következő tényezőket vizsgálta:

– Utaló szavak és frázisok;

– Gyakori és egyben informatív szavak (kulcsszavak).

– Cím-kulcsszavak.

– Előfordulási hely heurisztika.

Ezek után minden i mondatra meghatározta az alábbi S_i kifejezés értékét:

$$S_i = w_1 \cdot C_i + w_2 \cdot K_i + w_3 \cdot T_i + w_4 \cdot L_i \tag{12.14}$$

ahol C_i, K_i, T_i, és L_i rendre a mondatban szereplő utaló frázisok, kulcsszavak, címszavak száma, illetve az előfordulási heurisztika által meghatározott érték. A w_i $(i = 1, \ldots, 4)$ együtthatók az egyes tényezőkhöz rendelt fontossági vagy súlyfaktor.

Az ilyen módon megállapított S_i értékek alapján vagy a k legmagasabb értékkel rendelkező, vagy egy meghatározott küszöbértéknél nagyobb pontszámmal bíró mondatokból alkotjuk a kivonatot.

A módszer időtállóságát mutatja, hogy még manapság is vannak ilyen alapon működő kivonatolók [**?**]. Egyetlen komoly hiányossága, hogy nem veszi figyelembe a kiválasztott mondatok hasonlóságát, és nem alkalmaz redundancia-csökkentő módszereket.

12.5.2. TF-IDF alapú módszer

A TD-IDF módszer gyakorlatilag az IR paradigma alkalmazása kivonatolási célra [**?**]. A módszer elsősorban kérdés-vezérelt kivonat előállítására alkalmas, de megfelelő módosítással általános kivonat létrehozására is alkalmassá tehető.

A dokumentum szavaiból mondat szinten készített kumulált halmaz (ún. zsák) alapján a szokásos TF-IDF[25] modell segítségével a mondatokhoz frekvencia vektorokat rendelünk. Ezeket azután a kérdés szavaiból képzett vektorral valamilyen hasonlósági mértéket (pl. koszinusz távolság) alkalmazva összehasonlítjuk, és a leginkább hasonló mondatokat kiválasztjuk. Általános összegzés létrehozásához a dokumentum leggyakoribb (informatív) kulcsszavaiból képezzük a kereső vektor szavait. Mivel elvileg ezek reprezentálják leginkább a dokumentum témáját, a hasonló mondatokból összeállított kivonat a szöveg általános összegzésének tekinthető.

Ennek az eljárásnak több gyenge pontja is van. Egyrészt a felhasználói szokások alapján megadott kérdés is legtöbbször általános kivonatot eredményezhet, hiszen ha olyan tematikus szavakkal keresünk egy szövegben mint pl. „information retrieval”, ami a szöveg tágabb témája, akkor nem jutunk specifikus információhoz. Másrészt, mivel a módszer csak olyan mondatokat választ ki, amelyekben a kereső szavak szerepelnek, biztosan kimarad néhány nagyon fontos és informatív mondat a kivonatból, ami pl. már az adott dokumentum témáját részletesebben ismerteti. Ugyancsak emiatt a kivonat rendkívül redundáns lesz.

12.5.3. Csoportosítás alapú módszerek

A jól megírt dokumentumokra általában igaz az a *szerkesztési elv*, hogy egy tágabb területhez tartozó témákat tárgyalnak egymás után. Ez alapján (ténylegesen vagy implicite) szakaszokra bonthatók szét. A dokumentum tematikus szerkezetét a kivonatnak is tükröznie kell, hiszen az összefoglalásban szerepelnie kell a szövegben tárgyalt témáknak. Ezt a szerkesztési elvet egyes kivonatolók csoportosító (klaszterező) eljárások alkalmazásával valósítják meg. Itt jegyezzük meg, hogy e módszerrel nem csak egyes dokumentumok, hanem dokumentumgyűjtemények kivonatolása is elvégezhető.

Az MMR módszer

Az MMR (Maximum Marginal Relevance — maximális szélső relevancia) módszer [**? ?**] mind statisztikai, mind nyelvi jellemzőket felhasznál a mondatok kiválasztása során, vagyis egyaránt figyelembe tudja venni a kulcs- és címszavak előfordulását, időrendi sorrendet, kérdéshez/területhez való hasonlóságot (tehát általános és kérdés-vezérelt kivonatot is képes előállítani), redundancia-csökkentést, és névmások előfordulásának büntetését. A mondatokat az alábbi formula szerint pontozza:

$$MMR(\text{mondat}_i) = \lambda \sum_{s \in S} w_s (Q_s \cdot S_i) + (1-\lambda) \sum_{l \in L} w_l (L_l \cdot S_i), \tag{12.15}$$

[25] Itt a dokumentum szint helyett —ld. (12.3) kifejezés —a mondat szinten van megvalósítva

ahol S a statisztikai, L a nyelvi jellemzők halmaza, Q a kérdés, w pedig az egyes jellemzőkhöz tartozó súlyok. A súlyok állításával lehet szabályozni az előállítandó kivonat típusát.

Az MMR módszer a kivonatot inkrementálisan állítja össze, mindig azt a mondatot választva ki, amely leginkább hasonlít a kérdéshez vagy a dokumentum témájához[26], és leginkább különbözik a már kiválasztott mondatoktól[27]. Ez a módszer lehetővé teszi azt is, hogy tetszőleges méretű kivonatot generáljunk, hiszen a kivonat bővítése bármikor befejezhető.

A módszer a feldolgozás elején csoportosítja a dokumentum (vagy dokumentumgyűjtemény) mondatait valamilyen hasonlósági mérték alapján. A kivonatoló minden csoportból a csoport központjához legközelebbi mondatot, mint a csoporthoz tartozó mondatok témáját leginkább reprezentálót választja ki a kezdeti fázisban. Ezek a mondatok általában a csoportban lévők közül a leghosszabbak. A mondatok klaszterezése **??**. fejezetben ismertetett módszerek segítségével könnyen elvégezhető.

A módszernek fontos paramétere a kiinduláskor meghatározott hasonlósági küszöbérték, θ. Az algoritmus sémája a következő: kezdetben minden mondat magában önálló csoportot alkot. Két csoportot egyesítünk, ha a köztük lévő hasonlóság, $sim(C_i, C_j) \geq \theta$, ahol C_i az i csoportban lévő mondatok középpontja. Minden egyesítés után újraszámoljuk a keletkezett csoport középpontjának értékét. Az eljárást addig folytatjuk, amíg van olyan csoportpár, amelyek elegendően hasonlóak az egyesítéshez.

A θ paraméter értéke nagy hatással van a csoportok megalkotására, tehát ez eljárás nem túl robosztus. Ez a tulajdonság javítható, ahogy arra a [**?**] tanulmány rámutat, ha nem a távolsági mérték alapján képezzük a csoportokat, hanem a leggyakoribb kulcsszavak mondatokban történő előfordulása alapján.

A MEAD módszer

A MEAD módszer [**?**] dokumentumgyűjtemények kivonatolására alkalmas. Bemenete a dokumentumok TF-IDF súlyozási sémával való csoportosításának eredménye. Minden klasztert egy külön témának lehet tekinteni, amit a témára vonatkozó legnagyobb (TF-IDF) frekvenciájú szavak reprezentálnak.

A hírarchívumok kivonatolása esetén a mondatkiválasztás három tényezőt vesz figyelembe. Elsőként a klaszter közepétől való távolságot (C_i), a mondatnak a dokumentumon belüli előfordulását[28] (L_i), és a dokumentum első mondatával való hasonlóságot (F_i). Ezen mennyiségek lineáris kombinációjaként áll elő egy mondat pontszáma, a korábban ismertetett módszereknél bemutatott módon (ld. (12.14) és (12.15) kifejezéseket). Itt az F tényező szerepe megfelel a (12.14)-ben található T faktorának. A különbség az, hogy a MEAD módszer esetén a mondat pontszámát redundanciacsökkentés érdekében újraszámolják az új mondatok bevétele után.

A módszer hátránya, hogy a TF-IDF súlyozási sémát alkalmazza, amely nem a leghatékonyabb kivonatolási technikák esetén. Másik gyenge pontja, hogy hírarchívumok feldolgozására alkalmas, hiszen a három tényezőből kettő is (F_i és L_i) erősen a dokumentumok elején lévő mondatokat favorizálja, ezért más típusú dokumentumok esetén nem alkalmazható hatékonyan.

12.5.4. Gráfelméleti megközelítések

Ahogy az előző szakaszban láttuk, a kivonatolás első lépése több módszer esetén is a dokumentum mondatainak, vagy a dokumentumoknak a tematikus csoportosítása. A mondatok gráfelméleti repre-

[26] A **??**eq:MMR)-ben az összeg első tagja.

[27] A képlet második tagja

[28] Minél közelebb van egy mondat a dokumentum elejéhez, annál nagyobb ez az érték.

zentációja alkalmas eszköz témák meghatározására [**?**]. A szokásos előfeldolgozási lépések után a mondatokat egy irányítatlan gráf csomópontjaival reprezentáljuk, és a csomópontok között éleket a mondatokban előforduló közös szavak számával súlyozzuk. Az élek súlyára vonatkozóan minimális küszöbértéket is meghatározhatunk.

Ennek a reprezentációnak két eredménye van: a gráfpartíciók, vagy -klikkek egy témához tartozó mondatokat azonosítanak, és így csoportbarendezést generálnak. A klikkek erősségét, tehát az egy csoportba tartotó mondatok kohézióját a küszöbérték növelésével emelhetjük, és ezáltal egyúttal a témák számát is szabályozhatjuk. Ez a reprezentáció egyaránt lehetőséget nyújt általános és kérdésvezérelt kivonatok létrehozására; az előbbi esetben minden gráf klikkből egy-egy mondatot választva lefedjük az egész dokumentum(gyűjtemény) tématerületét, míg az utóbbi esetben elegendő a kérdéssel egy klikkben lévő mondatok közül kiválasztani néhányat.

A másik fontos eredmény, hogy a nagyszámú éllel rendelkező csomópontok a dokumentum(gyűjtemény) fontos mondatait is meghatározzák, amelyeknek ezáltal nagyobb esélye van a kivonatba kerülésre. A grafikus megközelítés könnyen hasznosítható dokumentumon belüli és közötti összefüggések vizuális megjelenítésére is.

12.5.5. SVD használata a kivonatolásban

A kivonatolásnál is jól felhasználható a szinguláris értékfelbontás [29] (SVD) módszer azon tulajdonsága, képes többdimenziós adatok ortogonális dimenzióinak megtalálására. Az LSI-t dokumentum-szó mátrixokra alkalmazva képes olyan mondatok közötti szemantikus összefüggések felfedésére is, amelyek nem tartalmaznak közös szavakat [**?**]. Azok a szavak, amelyek többnyire azonos kontextusban szerepelnek ugyanazon szinguláris dimenzióban helyezkednek el. Az LSI módszer nagy előnye, hogy a fogalmi (vagy szemantikus) összefüggéseket automatikusan az emberi agy által reprezentált módon képes megragadni [**?**]. Az LSI jól használható témákra jellemző szavak, illetve mondatok meghatározására egyaránt. Mivel az SVD fontossági sorrendben határozza meg a kölcsönösen ortogonális szinguláris irányokat a mondat-vektorok terében, ezért ha ezekből a dimenziókból választjuk a ki a reprezentatív mondatokat, akkor egyrészt biztosítva lesz a dokumentum teljes tematikájának lefedettsége, másrészt az ortogonalitás garantálja a redundancia-mentességet [**?**]. Egyetlen megszorítás, hogy csak eredendően tematikus egységekbe rendezett szövegekre alkalmazható hatékonyan, ám a legtöbb dokumentum ilyen szerkesztési elvet követ.

12.5.6. Esettanulmány: böngészés támogatása kivonatolással kézi számítógépeken

A kivonatoló eljárásokat a szakasz bevezetőjében tárgyalt internetes keresés/böngészés segítésén kívül még számos más területen is hatékonyan fel lehet használni, pl. összehasonlító táblázatok készítésére, többnyelvű információkinyerés támogatására, biográfiai profilok készítésére, strukturált adatbázis-építésre dokumentumok tartalmának automatikus feldolgozásával, stb.

Itt most a kivonatolás egyik speciális és kézenfekvő felhasználási területét ismertetjük részletesebben: a kisképernyős (kézi számítógép, PDA; mobiltelefon) tartalomszolgáltatás támogatását. Az Internet vezeték nélküli használata a felsorolt eszközök segítségével manapság egyre elterjedtebbé válik. A távolkeleten (Japán, Korea) az Internet használat jelentős részét a felhasználók a mobiltelefonjuk segítségével végzik. Az információigény jelentős része olyan szituációkban adódik — utazás,

[29] Szövegbányászati kontextusban látens szemantikus indexelésnek (LSI) nevezik.

vásárlás közben, tárgyalások, illetve beszélgetések esetén — amikor vezetékes Internet nem elérhető. A kézi számítógépek és a mobiltelefonok elvben ideális eszközök az ilyen esetekben adódó információigény kielégítésére, azonban a kisméretű kijelzők gyakran akadályt jelentenek az Internet kényelmes használatában [**?**], ugyanis a honlapok a kijelző méretéből adódóan többnyire nehezen áttekinthetőek. További problémát jelent az adatbevitel nehézkessége, valamint az a tény, hogy rádióhullámokon keresztül történő letöltési sebesség, még mindig sokkal lassabb, mint vezetékes kapcsolat esetén.

Ezen problémák egy részére az internetes tartalomszolgáltatás kivonatoláson keresztül, több lépésben történő megvalósítása az egyik lehetséges megoldás. A felhasználók ugyanis általában nem teljes Internet-oldalak tartalmára kíváncsiak, különösen a PDA-n és mobiltelefonon való böngészésre jellemző helyzetekben, hanem csak egy töredékére, amin a releváns információ megtalálható, és ezek többnyire tényszerű adatok vagy linkek.

A továbbiakban a Buyukkokten és munkatársai által javasolt megoldást ismertetjük [**? ?**], amely a weboldalakat a fokozatosan, a felhasználó igényétől függően jeleníti meg. Ezzel a módszerrel jelentősen csökkenthető mind a letöltött adatmennyiség, s ezzel párhuzamosan a letöltési idő is, mind pedig a keresett információ megtalálásához szükséges navigálási műveletek száma, valamint a böngészésre fordított idő.

Weboldalak kivonatolásának speciális kérdései

Első lépés az eredeti weboldal tartalmának feldarabolása ún. *szemantikus szövegegységek*re. A feldarabolás az oldal szerkezetét követi, amely az oldal (HTML, XML, PHP, stb.) forrását feldolgozva a tartalomból szövegegységek hierarchikus struktúráját állítja elő. A szövegegységek a weboldalt alkotó részegységek, pl. bekezdések, listák és elemeik, táblázatok, képek, stb. Ezekből a szöveges módon megjeleníthető egységeket dolgozzuk fel a továbbiakban, a képeket, illetve a túl nagy méretű táblázatok elhagyjuk.

A szövegegységek kivonatolása felvet néhány problémát. Mivel itt nem teljes dokumentumok, hanem azok kisebb egységeire kívánunk kivonatolót alkalmazni, ezért nehezebb feladatot jelenthet a kulcsszavak, ill. -mondatok meghatározása, mivel a szövegegységek terjedelme jellemzően rövid. Másik különbség az, hogy a hagyományos kivonatoló módszerek nem támogatják a fokozatos megjelenítést: egy dokumentum (itt: szövegegység) feldolgozásánál először az egészet beolvassák, majd statikusan kiválasztják annak egyes részleteit.

Szintén megfontolást igényel a hiperlinkek ábrázolása is (megjelenítés, aktivitás, hossz, fontosság a tartalmazó mondatra vonatkozóan).

Végül problémát okoz a kivonatolásnál használt statisztikák elkészítése, hiszen a legtöbb módszer szóelőfordulások és -frekvenciaértékek alapján határozza meg egy adott mondat jelentőségét szövegegységen belül. Mivel jelen esetben a dokumentumgyűjtemény az egész világháló tartalma, azon előfordulási statisztikákat készíteni lehetetlen.

Szövegegységek fokozatos megjelenítésének alternatívái

A szövegegységek fokozatos megjelenítésére az alábbi megoldásokat tesztelték:

– **inkrementális:** három lépésben: egy sor, három sor, egész szövegegység.

– **összes:** rögtön az egész szövegegység megjelenik, nincs fokozatosság.

– **kulcsszó:** első lépésben a szövegegységben azonosított kulcsszavak jelennek meg, a következő fokozatban az első három sor, majd végül az egész szöveg látható lesz.

– **összegzés:** itt csak két lépcső van: a legfontosabb mondat, majd a teljes szöveg megjelenítése

– **kulcsszó/összegzés:** ez az előző két módszer kombinációja, ahol először a kulcsszavak, majd a kiemelt mondat, végül az egész szöveg jelenik meg.

A hiperlinkek minden esetben aktívan megjelennek, kivéve a kulcsszavak fázist. Amennyiben egy link nem fejeződik be a sor végén, a látható fragmense akkor is aktív.

Kulcsszavak és összegzés meghatározása

A kulcsszavak a szövegegységben szereplő egyes szavak kiértékelése alapján határozhatók meg. A TF-IDF formula kiszámolásához (ld. (12.3)) szükséges a korpuszban előforduló összes szó ismerete, ami természetesen nem megvalósítható, így közelítő módszer alkalmazására van szükség. Ezt egy webrobot alkalmazásával elkészített szótár segítségével lehet megbecsülni, amely az interneten gyakorta előforduló szavakat tartalmazza.

Egy szövegrészlet feldolgozása során minden szóra szótövesítést alkalmazunk, majd a szótár, illetve az adott weboldalon való előfordulási gyakoriság alapján meghatározzuk a szóhoz tartozó TF-IDF értéket. A szótárban nem szereplő szavak esetén a szótárban szereplő legkisebb gyakorisági értékkel számolnak. Egy küszöbérték elérése esetén a szó kulcsszavak közé kerül. Lehetőség van a speciális szedésű (félkövér, dőlt, stb.) szavak erősebb súlyozására.

A kivonat meghatározására a 12.5. szakaszban ismertetett bármelyik módszer alkalmazható. Az ismertetett tanulmány egy nagyon egyszerű és könnyen implementálható, Luhn nevéhez fűződő [**?**] korai módszer módosított verzióját használták a szövegegység legjellemzőbb mondatának meghatározására.

A megjelenítő módszerek összehasonlítása

A fent ismertetett fokozatos megjelenítő heurisztikákat egy 15 főből álló, internetes böngészésben jártas csapat segítségével tesztelték. Tíz tipikusan vezeték nélküli internetezés közben felmerülő feladatot tűztek ki a tesztelőknek, pl. link megkeresése adott oldalon, nyitvatartási idő megkeresése, filmmel, tudományos konferenciával, ill. tanulmánnyal kapcsolatos adat, valamilyen termék árának és egyéb paraméterének meghatározása, stb., úgy, hogy a kiinduló oldalak adottak voltak. A teszt eredményei azt mutatták, hogy böngészési időt tekintve az összegzés, ill. kulcsszó/összegzés fokozatokat használó megjelenítési forma a legkézenfekvőbb a felhasználóknak, míg az inkrementális és az összes módszer a legkevésbé hatékony. A navigálási műveletek számát tekintve még erőteljesebb az említett két módszer dominanciája, esetenként 97%-kal csökkent az egér, ill. billentyűzet használat mértéke. Itt egyértelműen a kombinált kulcsszó/összegzés módszer bizonyult a legjobbnak.

Vizsgálták még a letöltött adat mennyiségének csökkenési arányát. Az összegzés, kulcsszó és a kombinált módszerek esetén az alapértékként tekintett (HTML tag-ektől, képektől és táblázatoktól mentes) adatmennyiséghez képest némi pluszt jelent, hogy a kulcsszavak, illetve az összegzés elejét és végét jelző indexértéket is továbbítani kell a rendszernek a protokollban az átvitel során. Ez azonban mindössze rendre 4%, 24%, ill. 28% volt. A letöltött adatmennyiség a „legdrágább” esetben is átlagosan 87%-kal kevesebbnek bizonyult, ami alátámasztja az kivonatoláson alapuló módszer hatékonyságát a kisképernyős böngészés támogatására.

12.6. Egyéb szövegbányászati feladatok

Ebben szakaszban röviden bemutatunk olyan további szövegbányászati feladatokat, amelyek részletes ismertetése — terjdelemi okok miatt — meghaladja e könyv kereteit.

12.6.1. Információkinyerés

Az *információkinyerés* (information extraction – IE) az egyik legalapvetőbb számítógépes szövegfeldolgozási feladat. Az IE algoritmusok általában mintafelismerési eljárások segítségével azonosítják a szövegben a fontos kifejezéseket és a közöttük lévő kapcsolatokat. Legyen a példamondatunk „A Washington Post szombati híre szerint, a Katrina hurrikán pusztítását követő káosz miatt egyre többen, köztük New Orleans polgármestere, Ray Nagin is, Busht és a szövetségi kormányt okolják”[30]. Az információkinyerő szoftvereknek a mondatban azonosítaniuk kell Ray Nagint és Busht mint személyeket, New Orleanst mint helyszínt, szombatot mint dátumot, a Washington Postot és a szövetségi kormányt pedig mint médiát (céget), illetve intézményt; további feladatuk a személyek, helyek és időpontok közti összefüggésekre való következtetés elvégzése is. Ez a módszer nagyban segítheti a felhasználókat a nagy adathalmazok gyors és hatékony feldolgozásában. Napjaink szövegbányász szoftverei mind tartalmazzák ezt a funkciót [48].

12.6.2. Témakövetés

A *témakövető* rendszerek felhasználói profil vagy érdeklődés alapján a következtetnek a felhasználó számára érdekes más dokumentumokra. Jó példa erre a Yahoo! által ingyenesen üzemeltetett témakövető eszköz[31], amely a felhasználó által megadott kulcsszavak alapján értesítést küld, ha a témában új hír jelenik meg. A piacon lévő eszközök többsége csak kulcsszó alapú keresést végez, aminek következtében gyakran előfordul az az anomália, hogy a felhasználó eredeti érdeklődésétől különböző dokumentumokat kap. Pl. ha a „text mining” kifejezést adjuk meg kulcsszóként, akkor többször kapunk bányászattal, mint szövegbányásza kapcsolatos híreket. Ennek kiküszöbölésére egyes fejlettebb témakövető szoftverek esetén a felhasználó által az érdeklődési körét a rendszer által karbantartott taxonómiából kiválasztott kategóriák segítségével határozhatja meg. Még intelligensebb módszerek pedig erre automatikusan következtetnek a felhasználó által látogatott oldalak és a klikkelési szokások alapján.

Az üzleti világban is jól alkalmazható ez az eszköz. Lehetőséget ad pl. konkurens vagy saját cégek, ill. termékek figyelésére az írott elektronikus médiában. Hasonlóan fontos lehet bármely — pl. orvosi, oktatási, tudományos — szakmában a felhasználó szűkebb szakterületéről szóló információk naprakész követésében.

12.6.3. Fogalomtársítás

A *fogalomtársító* (concept linkage) eszközök feladata, hogy dokumentumokban meglévő olyan közös fogalmakat azonosítson, amely esetleg a felhasználó elől hagyományos keresési módszerekkel rejtve lennének. Leginkább olyan területeken lehet hasznosan alkalmazni őket, mint pl. az orvostudomány, ahol a rendkívül nagymennyiségű szöveges dokumentum elolvasása vagy átböngészése

[30] `http://www.index.hu`
[31] `www.alerts.yahoo.com`

lehetetlen feladat. Kedvező esetben a fogalomtársító eljárások olyan kapcsolatokat is felfedhetnek betegségek és kezelési módok között ily módon, amit az ember nem képes megtalálni.

A fogalomtársító eszközök működését jól szemlélteti az a módszer, ahogy D. Swanson két egymástól bibliográfiailag távol álló, ám logikailag összefüggő kutatási terület összekapcsolásával azonosította a magnézium szerepét a migrén kialakulásában [141]. A kutató azokban a közlemények előforduló gyakori kifejezéseket vizsgálta, amelyek címükben a „migrén” szót tartalmazták. Az egyik ily módon azonosított kulcsszó az „tovaterjedő kérgi gátlás” volt. Ezután hasonló keresést végzett ebből a kifejezésből kiindulva, s így találta meg többek közt a „magnézium elégtelenség” terminust. A két fogalom közti tényleges összefüggést elemző kutatásaiban kimutatta, hogy a korábban még nem vizsgált magnézium elégtelenség a migrén kialakulásában komoly szerepet játszik.

A Swanson által használt módszer alkalmazó automatikus eszközök jól alkalmazhatók szövegbányászatban [59]. Várhatóan az ilyen felhasználások a közeljövőben nagyban segíthetik a orvosi felhasználókat új kezelési módok felfedezésében.

12.6.4. Szöveges információk vizualizálása

Nagyméretű szöveges források/gyűjtemények esetén a vizuális hierarchiában vagy térképpel történő böngésző lehetőséggel kiegészített képi megjelenítés nagymértékben segítheti a felhasználót a keresett téma és az hozzá tartozó dokumentumok könnyebb azonosításában.

A szöveges adathalmazok képi megjelenítését végzi az Informatik V DocMiner[32] terméke. Ennek segítségével a felhasználó interaktív tartalom elemzést végezhet a vizualizált adatokon. A böngészést zoomolás, skálázás és résztérképek készítésének lehetősége is támogatja.

12.5. ábra. Az Informatik V Doc Miner szoftverének felhasználói felülette [48]

12.6.5. Kérdés-megválaszolás

Ez az alkalmazási terület már nagyrészt átfed a következő szakasz témájával, hiszen a kérdés-megválaszolásban (Question Answering – QA) nagy szerepet játszanak a nyelvtechnológiai eszközök. A feladat általánosan természetes nyelvű kérdések többnyire természetes nyelven történő megválaszolása adatbázis vagy a világháló segítségével.

Nyelvtechnológiai projektek keretében főleg angol nyelvű Kérdés-megválaszoló rendszerek ismertek, melyek közül például az MIT fejlesztett START[33] projekt az Internetről összegyűjtött információk alapján válaszol. Hasonló módon dolgozik az Answerbus[34] és az AskJeeves[35] kereső is. A természetes nyelv elemzésének bonyolultsága és nyelvtechnológiai eszközök jelenlegi fejlettségi szintje azonban behatárolja a kérdés-megválaszoló rendszerek hatékonyságát, amint azt az alábbi példa is jól szemlélteti. A *When does the Siam Cuisine Restaurant open?* kérdésre az alábbi válaszokat kaptuk:

– START: Unfortunately, I wasn't told when Siam Cuisine Restaurant opens.

[32] `http://www-i5.informatik.rwth-aachen.de/lehrstuhl/projects/DocMINER/`
[33] `http://www.ai.mit.edu/projects/infolab/`
[34] `http://www.answerbus.com/index.shtml`
[35] `http://www.ask.com/`

– ANSWERBUS: Siam Orchids Authentic Thai Cuisine Restaurant was opened on February 5, 2003.

– ASKJEEVES: This Center City location is open for lunch and dinner seven days a week.

12.7. Nyelvfeldolgozás és szövegbányászat

A szövegbányászati alkalmazásokban valamilyen mélységű nyelvi feldolgozás alkalmazására szinte mindig szükség van. A dokumentumok feldolgozása során leggyakrabban *szótövező* algoritmusokat használunk, amely a bemeneti szónak megadja a szótövét[36] Angol nyelvű szövegek esetén a Porter-algoritmust [120] használják leggyakrabban[37]. Amennyiben nemcsak szavak szintjén végezzük el a szövegek statisztikai feldolgozását, hanem a gyakoribb kifejezéseket is indexeljük a dokumentumokban és tároljuk a szótárban, akkor a kifejezéseket adatbányászati algoritmusokkal határozhatjuk meg, pl. *Apriori* [4] (ld. még **??**. szakasz).

Ha a szövegbányászati feladat statisztikák készítésénél részletesebb nyelvi feldolgozást — pl. szintaktikai vagy szemantikai elemzést — kíván, akkor szükség van legalább egy *szófajcímkéző* eszközre[38], vagy egy teljes morfológiai elemezőre. Ilyen feladatok pl. az információ-kinyerés, illetve az automatikus kérdés megválaszolás. Alapvetően statisztikai jellegű problémák esetén (osztályozás, csoportosítás) is tettek kísérletet nyelvtechnológiai eszközök bevetésére, de ezek szinte egyáltalán nem javították az algoritmusok minőségét, ugyanakkor a jelentős erőforrástöbbletet igényeltek.

Hatékony szintaktikai és szemantikai elemzéshez, illetve a mondatokon belüli ún. *névelem*ek azonosításához szükség van olyan téma- és nyelvspecifikus adattárakra és/vagy tezauruszokra, ontológiákra. Az adatbázisok a különböző típusú névelemeket (személy, helyszín, intézmény, cégnév, stb.), névszókat, igéket jellemző vonzataikkal (vonzatkerettár) tartalmazzák. A tezauruszok, ill. ontológiák akkor nyújthatnak többek között segítséget, ha egy adott terminus nincs benne a nyelvi adatbázisokban. Ekkor ugyanis a terminust valamelyik szinonimájával, vagy vele valamilyen ontológiai relációban lévő elemmel lehet az elemzés során helyettesíteni. A névelemek automatikus azonosítására vannak felügyelt tanulási sémát alkalmazó eljárások, de ezek hatékonysága nagyban függ a tanulóadatoktól.

12.7.1. Szövegbányászat magyarul

Mivel a szövegbányászat témaköre viszonylag fiatalnak tekinthető, ezért a főbb kutatások fókuszában főleg az elektronikus dokumentálás legfontosabb nyelve, az angol állt, utána messze lemaradva a többi nagy világnyelv, nem beszélve a világviszonylatban marginálisnak mondható magyar nyelvről[39]. Az utóbbi időszakban azonban — részben a hazai nyelvtechnológiai kutatások eredményeinek köszönhetően — felélénkült a számítógépes magyar nyelvfeldolgozás területe, és ez lökést adott a magyar nyelvre vonatkozó szövegbányászati alkalmazásoknak. Mivel az alapvető algoritmusok tekintélyes része nyelvfüggetlen, ezért ezeknél a magyar vonatkozást a szövegfeldolgozási lépés-

[36] Bizonyos esetekben, pl. a *palánk* szónál, több szótő is lehetséges, ennek kezelése azonban bonyolult szövegértelmezési feladat; a példa esetében végre kell hajtani a szótő egyértelműsítését. Mivel ez a jelenség viszonylag ritka, ezért általában feltételezzük, hogy a szótő egyértelmű.

[37] Az algoritmus különböző programnyelven írt implementációi letölthetők innen: `http://www.tartarus.org`. Itt a legtöbb európai nyelvhez is található szótövező.

[38] Part of Speech (POS) tagger

[39] A magyar nyelvtan bonyolultsága és egyedisége szintén nem kedvezett a korai alkalmazásoknak.

nél találunk, ami többnyire valamely szótövező eljárás alkalmazását jelenti. Az olyan bonyolultabb feladatoknál viszont, mint a kérdés-megválaszolás vagy az információkinyerés már lényegesen komolyabb szerepet kap a nyelvtechnológia. Összességében tehát megállapíthatjuk, hogy a magyar nyelvű szövegekkel kapcsolatos szövegbányászati alkalmazások olyan bonyolultságú feladatokkal képesek megbirkózni, amennyire fejlett nyelvtechnológiai eszközök jelenleg a piacon, illetve szabadon hozzáférhetően rendelkezésre állnak. A linkgyűjteményen belül külön részt szentelünk a magyar vonatkozású eredményeknek, projekteknek (ld. 12.8.4. pont).

12.8. Linkgyűjtemény

Az alábbi linkek és a fejezetben idézett irodalmi hivatkozások nagy része megtalálhatóak a szerző honlapján[40], ahol a hivatkozások érvényessége rendszeren ellenőrizve van.

12.8.1. Tesztkorpuszok

- `http://www.daviddlewis.com/resources/testcollections/`: Itt található meg a Reuters-21578 és egy korábbi verziója, az RCV-1, és a TREC-AP korpusz.
- `http://about.reuters.com/researchandstandards/corpus/`: A Reuters Corpus Volume 1 hivatalos honlapja.
- `http://trec.nist.gov/data.html`: Az egyik legnagyobb gyűjtemény, ahol szövegbányászati eljárások tesztelésére alkalmas adatok vannak. Itt található pl. az OHSUMED korpusz (filtering track), amelyet több osztályozó vizsgálatánál is használtak.
- `http://people.csail.mit.edu/jrennie/20Newsgroups/`: Szintén többször alkalmazott adathalmaz.
- `http://www.wipo.int/ibis/datasets/index.html`: csak regisztált felhasználók számára érhető el.

12.8.2. Cikk- és linkgyűjtemények

- `http://liinwww.ira.uka.de/bibliography/Ai/index.html`: Cikkgyűjtemény, ahol sok szövegbányászati témájú publikáció is található, különösen a kimondottan szövegosztályozással foglalkozó `automated.text.categorization.html` oldalon.
- `http://filebox.vt.edu/users/wfan/text_mining.html`: Szövegbányászattal kapcsolatos cikkek, termékek projektek linkgyűjteménye.
- `http://dmoz.org/Reference/Knowledge_Management/Knowledge_Discovery/Text_Mining/`: vegyes linkgyűjtemény.
- `http://www.text-mining.org/`: Szövegbányászattal foglalkozók közösségének honlapja.

[40] `http://categorizer.tmit.bme.hu/~domi/links`

12.8.3. Szövegbányászati szoftverek

- `http://registry.dfki.de/`: Nyelvtechnológiai és szövegbányász szoftverek gyűjteménye.
- `http://www.cs.uic.edu/~liub/LPU/LPU-download.html`: Ingyenesen letölthető szöveg-osztályozó szoftver.
- `http://www.intext.de/eindex.html`: Angol és német nyelvű szövegeken dolgozó szöveg-elemző program.
- `http://ka.rsten-winkler.de/hypknowsys/diasdem/index.html`: A Diasdam projekt által fejlesztett szemantikus szövegfeldolgozó szoftver honlapja.
- `http://software.wise-guys.nl/libtextcat/`: Nyelv- és karakterkódolás felismerő program, amely többek közt a magyar nyelvre is működik.
- `http://www.clearforest.com/Products/Platform.asp`: Szöveges adatokat is hatékonyan kezelni képes üzleti intelligenciai alkalmazás.
- `http://www.clearforest.com/Products/Tags.asp`: Szövegelemző és osztályozó eljárásokat tartalmazó programcsomag.
- `http://www.inxight.com/products/sdks/lx/`: Jó pár nyelvtechnológiai és szövegbányászati eljárást tartalmazó programcsomag (nyelv- és karakterkódolás felismerő, szótövező, tokenizáló, szófajcímkéző, és névszói kifejezésfelismerő). Összesen 31 nyelvet, köztük a magyart is támogatja.
- `http://www.inxight.com/products/sdks/sum/`: Az Inxight 19 nyelvet támogató összegzéskészítő szoftvercsomagja.

12.8.4. Néhány magyar vonatkozású eredmény és projekt

- `http://mokk.bme.hu/projektek/szoszablya`: A projekt létrehozta a Magyar Webkorpuszt — egy minden korábbinál nagyságrenddel nagyobb méretű magyar nyelvű tokenizált szöveggyűjteményt —, az ez alapján készítette Szószablya gyakorisági Szótárat, a *szabadon elérhető* HUNMORPH morfológiai elemzőt, a HUNSTEM szótövezőt és a HUNSPELL helyesírás-ellenőrzőt, valamint a programok által használt magyar helyesírási és morfológiai szótárat[41].
- `http://corpus.nytud.hu/mnsz`: A Magyar Nemzeti Szövegtár. 150 millió szót tartalmaz, morfológiai elemzéssel és automatikus szófaji egyértelműsítéssel (97,4%-os). Az egyértelműsítést statisztikai alapú eljárással érték el. Öt eltérő nyelvhasználatból származó szövegeket ölel fel: sajtó, szépirodalom, (tudományos) értekező próza, hivatali nyelvhasználat és személyes közlés.
- `http://www.inf.u-szeged.hu/projectdirs/hlt/nkfp2001.htm`: A projekt rövid üzleti jellegű hírekből történő releváns információ kinyerésével foglalkozott. Az információkinyerés célja tehát strukturált — gépileg lekérdezhető, feldolgozható — adathalmaz előállítása szöveges dokumentumok tartalmából.

[41] Letöltés innen: `http://magyarispell.sourceforge.net/`

13. fejezet

Webes adatbányászat

Az Internetről történő automatikus információkinyerő alkalmazások gombamód szaporodnak napjainkban. A terület mélyebb áttekintése túlmutat ezen írás keretein, ezért csak a talán legfontosabb két témát járjuk körül: a weboldalak rangsorolását és az intelligens Internetes keresést.

Az oldalak közötti megfelelő rangsor felállítása napjaink kritikus feladata. A keresőrendszerek mindenapos eszközökké váltak. Naponta milliók használják, így a helyes működésük mindenki érdeke.

Minden honlapkészítő álma, hogy az oldala elsőként jelenjen meg a keresők által visszaadott listában. Ez cégeknek sok látogatót és így sok potenciális ügyfelet jelent, másfelől a gyakran látogatott oldalakon elhelyezett reklámok is jó bevételt jelentenek. Központi szerepük miatt fokozott támadásoknak vannak kitéve. Egy rangsoroló algoritmus elkészítésekor ezért fontos megvizsgálni, hogy az milyen trükkökkel lehet azt becsapni, és ezek ellen hogyan kell védekezni.

13.1. Oldalak rangsorolása

Képzeljük el azt a ritkának nem mondható helyzetet, amikor egy keresőrendszer a feltett kérdésünkre rengeteg oldalt talál, olyan sokat, hogy kivitelezhetetlen feladat egyesével átnézni azokat és kiválasztani a fontosakat. Mégis tudjuk azt, hogy a talált oldalaknak valamilyen köze van a kérdésünkhöz: egyeseknek több, másoknak kevesebb.

Szükség van tehát az oldalak automatikus rangsorolására, aminek alapkövetelménye, hogy formálisan is tudjuk definiálni egy weboldal „fontosságát”. Felmerül a kérdés, hogy objektív-e a fontosság definiálása. A válasz egyszerű: nem. A kérdésre kiadott dokumentumok között ugyanis különböző emberek nem ugyanazt a sorrendet állítanák fel.

A feladatot meg kell oldani, akkor is ha tökéletes megoldást még elméletileg sem tudunk adni. Megelégszünk ezért a fontosság valamilyen heurisztikán alapuló közelítésével. Algoritmikusan szemlélve két algoritmuscsalád létezik, az egyik családba tartozó algoritmusok a *gráf összes pontját súlyozzák*, majd a súlyok rendezésével állapítják meg a sorrendet. Ezek a *globális algoritmusok*, míg a másik családot *kérdésfüggő rangsorolások*nak nevezhetjük, ami azt jelenti, hogy a rangsoroló algoritmus minden kérdésnél lefut, és ekkor csak egy *részgráf* csúcsait pontozza. Mindkét családnak megvan a maga előnye, az elsőnek csak egyszer kell lefutni, és utána csak memóriából való olvasás a keresőszoftver feladata, míg a második figyelembe tudja venni azt a tényt, hogy egy weboldal különböző témájú kereséseknél különböző módon szignifikáns.

13.1.1. Az egyszerű Page Rank

A Google keresőrendszerben 1998-ban implementált Page Rank (Brin-Page) algoritmus a gyakorlati alkalmazások során nagyon jó eredményt hozott [110]. A továbbiakban az ő módszerüket mutatjuk be.

Rendelkezésünkre áll N darab weboldal a hagyományos weboldalak minden tulajdonságával. Feladat lenne ezek között egyfajta fontossági sorrendet felállítani. Egy oldal fontosságát, hasznosságát jól tükrözi az oldalt meglátogató emberek száma. A legtöbb oldal készítői azonban a letöltések számát illetően semmilyen auditálást nem végez, így rangsoroló algoritmust nem alapozhatunk ezen információkra.

A linkeknek nagy szerepük van a fontosságban. Ha valaki saját oldalán egy másik oldalra mutató linket helyez el, akkor azt azért teszi, mert szerinte, a másik oldal hasznos információt tartalmaz, kapcsolódik az oldal témájához, valamilyen szempontból fontos. A Page Rank algoritmus (és minden kifinomult kereső rendszer) az oldalak közötti linkstruktúra alapján definiálja a fontosságot.

Egy oldal fontosságát az oldal rangja adja meg. Elképzeléseinknek megfelel az az állítás, hogy ha valahova sok link mutat, akkor az fontos oldal, továbbá, ha egy oldal fontos, akkor az általa mutatott lapok is azok. Informálisan egy rekurzív definíciót adnánk a fontosságnak: „egy oldal fontos, ha fontos oldalak mutatnak rá”.

A rang meghatározásához szükségünk van az oldalak közötti linkstruktúra ismeretére. Definiáljuk az N weblaphoz A $N \times N$-es sor-sztochasztikus mátrixot az alábbiak szerint: amennyiben az i. lapon n link található, akkor

$$A_{ij} = \begin{cases} \frac{1}{n} & \text{ha j-re mutat link i-ről} \\ 0 & \text{egyébként} \end{cases}$$

Az A mátrix (sor-)sztochasztikus, azaz $\forall i$-re $\sum_{j=1}^{N} A_{ij} = 1, A_{ij} \geq 0$. A sor-sztochasztikus mátrixokra igaz a következő tétel:

13.1. tétel. *Legyen A sor-sztochasztikus mátrix ($N \times N$-es), $j = (\frac{1}{N}, \ldots, \frac{1}{N})$. Ekkor*

$$p = \lim_{m \to \infty} jA^m$$

létezik és $pA = p$.

13.2. definíció. *A $p = (p_1, \ldots, p_N) \in \mathbb{R}_+^N$ vektor a lapok rang-vektora (tehát az i-edik lap rangja p_i).*

Az algoritmus menete a következő:

I. Készítsük el az A mátrixot az adott weblapok topológiájából.

II. Kezdetben minden oldal rangja $\frac{1}{N}$, tehát $p = (\frac{1}{N}, \ldots, \frac{1}{N})$,

III. végezzük el $p_{i+1} \leftarrow p_i A$ iterációt,

IV. ha teljesülnek a leállási feltételek akkor STOP, ellenkező esetben ugrás az előző utasításra.

Leállási feltétel lehetne az, hogy a p rang-vektor egy adott küszöbnél kisebbet változik. Az eredeti célunk azonban az egyes oldalak rangsorolása, nem pedig a pontos rangértékek meghatározása. Ezért sokkal ésszerűbb, ha akkor állítjuk le az iterációt, ha a rang-vektor alapján felállított sorrend nem változik egy adott számú iteráció után. A fent kimondott tétel a garancia arra, hogy az iteráció során

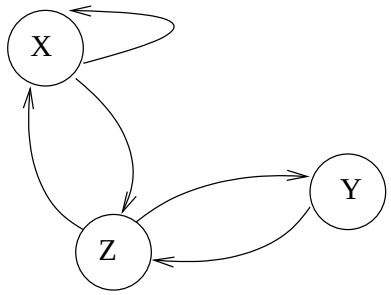
p rang-vektor egy vektorhoz konvergál, amiből következik, hogy az algoritmus minden esetben le fog állni.

Képzeljük azt, hogy kezdetben minden oldal fontossága $\frac{1}{N}$, és minden lap a következő lépést hajtja végre: a saját fontosságát egyenlő mértékben szétosztja az általa mutatott oldalak között. Könnyű végiggondolni, ha a fenti lépést hosszú időn keresztül folytatják, akkor minden lap fontossága meg fog egyezni a fent definiált rang-vektor laphoz tartozó rangértékével.

A fenti algoritmus elfogadásához egy másik intuitív magyarázat lehetne az alábbi: Tegyük fel, hogy a „sztochasztikus szörfölő” egy olyan, az Interneten barangoló lény, aki a kiindulási lapot, egyenletes eloszlás szerint, véletlenszerűen választja ki, valamint minden következő oldalt az aktuálisról elérhetők közül választja ki hasonlóan véletlenszerűen. Belátható, hogy annak a valószínűsége, hogy végtelen sok lépés után a szeszélyes szörfölő az i-edik lapra kerül, p_i.

Az algoritmus vitathatatlan előnye, hogy gyors ($N \cdot j \cdot A^M$ jól számítható) és könnyen programozható.

Nézzünk egy nagyon egyszerű példát az algoritmusra. 3 oldalt kell rangsorolnunk, amelyek linkstruktúrája a következő ábrán látható.



13.1. ábra. Példa az egyszerű Page Rank algoritmusra

A topológia alapján az A mátrix:

$$A = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 \\ \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix}$$

Az első három iteráció után a rang vektor N-szerese:

$$N \cdot p_1 = (1,1,1)$$

$$N \cdot p_2 = (1, \frac{1}{2}, \frac{3}{2})$$

$$N \cdot p_3 = (\frac{9}{8}, \frac{1}{2}, \frac{11}{8})$$

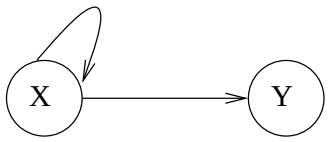
$$N \cdot p_4 = (\frac{5}{4}, \frac{11}{16}, \frac{17}{16})$$

Megmutatható, hogy $p = (\frac{6}{5}, \frac{3}{5}, \frac{6}{5})$

Ennek az egyszerű algoritmusnak két nagy hibája van, melyeket zsákutca, illetve pókháló problémának hívunk.

Zsákutca probléma

Zsákutcának nevezzük azt az oldalt, amiről nem mutat link semmilyen más lapra, de más lapról mutat rá. Amennyiben az oldalak között zsákutca van, akkor az A mátrix ehhez az oldalhoz tartozó sora csupa 0 elemet fog tartalmazni. Ekkor az A mátrix nem lesz sor-sztochasztikus, és oldalak fontossága „kiszivárog” a rendszerből. A probléma szemléltetésére nézzük a következő ábrán látható lapstruktúrát.



13.2. ábra. Példa zsákutcára

A hozzá tartozó mátrix: $A = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ 0 & 0 \end{pmatrix}$. Könnyen ellenőrizhető, hogy $A^2 = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} \\ 0 & 0 \end{pmatrix} = \frac{1}{2}A$, továbbá $A^m = \frac{1}{2^{m-1}}A$, amiből adódik, hogy a rangvektor a 0 vektorhoz fog tartani.

Pókháló probléma

Lapok olyan rendszerét, amelyben minden link csak e rendszerbeli lapra mutat, pókhálónak nevezzük. Jellemző rájuk, hogy az iteráció során magukba gyűjtik (esetleg az összes) a fontosságot. Ez komoly visszaélésekhez adhat alapot és SPAM-eléshez vezethet, hiszen linkek eltávolításával bárki alakíthat ki pókhálót, amennyiben van arra az oldalra mutató link.

Példaként térjünk vissza a 13.1 laptopológiához, csak most tegyük fel, hogy Y a Z-re mutató linkjét átállítja úgy, hogy ezentúl saját magára mutasson. Ekkor A mátrix a következőképpen módosul:

$$A = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 1 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix}$$

a rang vektor N-szerese az első négy iteráció során:

$$N \cdot p_1 = (1,1,1)$$

$$N \cdot p_2 = (1, \frac{3}{2}, \frac{1}{2})$$

$$N \cdot p_3 = (\frac{3}{4}, \frac{7}{4}, \frac{1}{2})$$

$$N \cdot p_4 = (\frac{5}{8}, 2, \frac{3}{8})$$

$$N \cdot p_5 = (\frac{1}{2}, \frac{35}{16}, \frac{5}{16})$$

belátható, hogy a rang vektor a $p = (0,1,0)$ vektorhoz fog tartani.

13.1.2. Az igazi Page Rank

A fenti két probléma kiküszöbölésére az oldalak megadóztatását javasolták. Ennek ötlete az, hogy szedjük be mindenkitől fontosságának bizonyos százalékát, majd a beszedett adót osszuk el egyenlően. Amennyiben ε-nal jelöljük a befizetendő adót, akkor a fentiek alapján A mátrix helyett a $B = \varepsilon \cdot U + (1-\varepsilon) \cdot A$ mátrixot használjuk, ahol $U = \begin{pmatrix} \frac{1}{N} & \frac{1}{N} \\ \dots\dots & \\ \frac{1}{N} & \frac{1}{N} \end{pmatrix}$.

Könnyen ellenőrizhető, hogy a B mátrix sor-sztochasztikus, így alkalmazhatjuk rá a 13.1-es tételt, ami ismét garantálja, hogy az algoritmus le fog állni.

Az igazi Page Rank algoritmusban az egyes lapok nem csak szomszédjaiknak osztják szét fontosságukat, hanem először befizetik az adót a királyi kincstárba, és csak a maradékot osztják szomszédjaiknak. Fontosságot pedig kapnak a rá mutató oldalak mellett a kincstárban található beszedett adóból is, egyenlő mértékben.

Amennyiben A mátrix helyett B mátrixot alkalmazzuk, a sztochasztikus szörfölőre nem lesz igaz az, hogy p_i annak valószínűsége, hogy i-edik oldalra lép. Igaz lesz viszont a „szeszélyes sztochasztikus szörfölőre”, akire ε valószínűséggel rájön a szeszély, és ilyenkor a következő állomását, egyenletes eloszlást követve, véletlenszerűen választja a lapok közül.

Az igazi Pagerank algoritmust a kezdeti Google(`http://www.google.com`) keresőrendszer használta a talált oldalak rangsorolásához. A keresőrendszerről részletesebb leírás található a [24] cikkben.

13.2. Webes keresés

Internetes keresés során egy keresőrendszertől két típusú kérdésre kérhetünk választ:

tág kérdés A választ tartalmazó, vagy a kérdéshez kapcsolódó oldalak száma nagy. Ilyen kérdés lehet, hogy információt szeretnénk a java nyelvről, vagy a gépkocsigyártókról.

szűk kérdés Ezen olyan specifikus kérdést értünk, amelyre a választ kevés oldal tartalmazza. Ilyen kérdés lehet, hogy „A *2001. Űrodüsszeia* hányadik percében hangzik el az első emberi szó?”

Szűk kérdésre a válaszadás automatikus módja jóval nehezebb feladat, mint tág kérdésre. Szűk kérdésnél annak veszélye fenyeget, hogy egyáltalán nem találunk választ pusztán hasonló szavakon alapuló kereséssel. Tág kérdéseknél ezzel szemben a probléma éppen a válaszhoz kapcsolódó lapok túl nagy száma lehet. Ebben a részben arra keresünk választ, hogy miként tudjuk kiválasztani a tág kérdésre kapott nagy mennyiségű oldalból a kérdéshez leginkább kapcsolódó oldalakat.

13.2.1. Gyűjtőlapok és Tekintélyek – a HITS algoritmus

Az 1999-ben Jon Kleinberg által publikált Gyűjtőlapok és Tekintélyek (Hubs and Authorities) módszere [82] a lapok linkstruktúráját használja fel. A linkstruktúra mellett számos információ állhat rendelkezésünkre, amelyek segítségünkre lehetnek az oldalak fontosságának meghatározásában. A látogatások számát már említettük. Probléma vele, hogy az oldalak elenyésző részét figyelik auditáló szoftverek.

Az oldalon elhelyezett metaadatok, kulcsszavak, az oldal leírása, de ezenkívül a szövegben kiemelt szavak (dőlt betű, vastag betű, villogó betű ...) szintén segíthetnek a kérdéhez kapcsolódás

mértékének eldöntésében. A tanulmányban ezek szerepét nem vesszük figyelembe. Jelöljük σ-val a kérdést, amire a választ keressük. Az algoritmus fázisai a következők:

I. M_σ (mag)laphalmaz kiválasztása hagyományos keresővel.

II. M_σ bővítésével bázis lap-részgráf konstruálása. Jelöljük ezt a bázist B_σ-val.

III. A σ-hoz tartozó gyűjtőlapok és tekintélyek (szimultán) kiszűrése B_σ-ból.

A gyűjtőlapoknak és tekintélylapoknak nem adunk pontos matematikai definíciót. Minden oldalhoz egy gyűjtőlap- és egy tekintélyértéket fogunk rendelni. Minél nagyobbak ezek az értékek, annál inkább tekintünk egy oldalt az adott kérdéshez tartozó gyűjtő-, illetve tekintélylapnak. Intuitív definíciója a két fogalomnak a következő lehetne: gyűjtőlap az olyan lap, ami sok tekintélylapra mutat, tekintélylapok pedig azok, amire sok gyűjtőlap mutat. Ezek szerint a gyűjtőlapok a σ szempontjából értékes linkek gyűjteménye, a tekintélylapok pedig a σ kérdéshez kapcsolódó értékes információkat tartalmazó lapok. Például az AMS honlapja egy matematikai gyűjtőlap, Jeffrey D. Ullman adatbányászatról szóló jegyzetvázlata pedig tekintélylap, amennyiben $\sigma =$"adatbányászati algoritmusok". Amikor egy kérdést felteszünk, akkor elsősorban a válasz érdekel bennünket, nem pedig az olyan oldalak, amik sok hasznos oldalra mutatnak. Az eredmény szempontjából a tekintélyoldalak a fontosak. Ezek megtalálásához gyakran a gyűjtőoldalakon keresztül vezet az út, így érdemes őket együtt keresni. Most pedig nézzük részletesen az algoritmus egyes lépéseinek működését.

M_σ mag meghatározása

Az algoritmus kiindulását képező weboldalaknak egy hagyományos kereső által σ kérdésre kiadott első t darab lapját vesszük. Ez a kezdőkészlet azonban nem mentes a hagyományos keresőrendszerek által adott hibáktól. Egyrészről lehet, hogy fontos oldalak nincsenek benne a találati listában. A "gépkocsi gyártók" kérdésre például nem fogják kiadni a Honda honlapját, mert a lapon ilyen szóösszetétel nincsen. Másrészről sok olyan oldalt is generálni fog, amelyek nem kapcsolódnak a témához. Ennek több oka is lehet, például az, hogy a kérdésnek több értelme is van (gondoljunk itt a Java nevű szigetre), vagy az egyes oldalak „hazudnak”, azaz olyan tartalmat állítanak magukról, amelyek nem igazak(pl.:mp3, free holiday ...). A fenti hátrányok ellenére elmondhatjuk, hogy ennek a magnak a „környezete” már hasznos információkban gazdag lesz.

B_σ bázis létrehozása

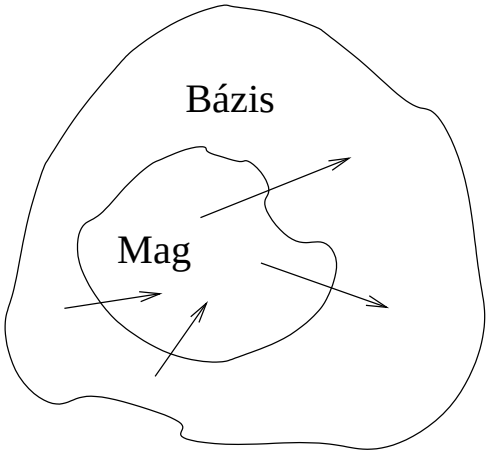
A gyűjtőlapokat és a tekintélyoldalakat a bázisból fogjuk kinyerni, így ezzel szemben az alábbi elvárásaink vannak:

I. Ne legyen túl nagy!

II. Legyen fontos lapokban gazdag!

III. Tartalmazza a σ-hoz releváns lapokat (vagy azok legtöbbjét)!

A tesztelés során kapott eredmények azt mutatták, hogy az alábbi egyszerű algoritmus a gyakorlatban jól működik. Induljunk ki az előző pontban definiált magból(azaz legyen $B_\sigma = M_\sigma$), majd adjuk hozzá az összes olyan oldalt, amelyre mutat link valamely B_σ-beli oldalról. Ezen kívül vegyük B_σ-hoz azokat az oldalakat, amelyekről mutat link valamely B_σ-beli lapra. Elképzelhető, hogy népszerű oldal



13.3. ábra. Bázis generálása a magból

is van B_σ-ban, amelyre rengeteg oldal mutathat, ezért egy oldal maximum egy előre meghatározott konstans (d) számú új lap felvételét „okozhatja”. Ezért ha egy lapra d-nél több lap mutat, akkor válasszunk ezek közül véletlenszerűen d darabot. Töröljük a bázisból a navigációt szolgáló éleket (pl.: vissza az előző oldalra) úgy, hogy csak a különböző hosztok közötti élek maradjanak. Itt azt a feltételezést tettük, hogy a hosztokat meg lehet különböztetni URL-jük alapján (Ez nyilván nem tökéletes megoldás, gondoljunk csak a unix alapú rendszerekre, ahol az egyes felhasználók honlapjának domainnevei megegyeznek. Nem könnyű kérdés az, hogy egy adott domaint mikor tekintsünk csak egy oldalnak, illetve mikor osszuk fel többre.

Kleinberg tapasztalata szerint a $t = 200, d = 50$ mellett a bázis mérete 1000 és 5000 között lesz.

Tekintélyek kinyerése

A tesztek alapján a bázis tartalmazni fogja a tekintélyek nagy részét. Hogyan leljük meg ezeket a több ezer oldal közül? Első ötlet lehetne, hogy a nagy be-fokú csúcsok reprezentálják a kereséshez kapcsolódó fontos oldalakat. Ez a megoldás azonban felemás eredményt ad: a jó oldalak mellett lesznek úgynevezett „univerzálisan népszerű” oldalak is. Ezekre jellemző, hogy σ-tól függetlenül a legtöbb kérdéshez tartozó bázisban megtalálhatóak. Például, ha $\sigma =$"java", akkor a B_σ-ban a legnagyobb be-fokú csúcsokhoz tartozó oldalak a

I. **www.gamelan.com**

II. **java.sun.com**

III. **amazon.com**

IV. karibi vakációkat hirdető oldal

Az utolsó két oldalt valamilyen automatikus módon ki kellene szűrni.

Kleinbergnek a következő szűrő ötlete támadt. A σ kérdéshez tartozó tekintélyeknek nagy befokon kívül jellemzője, hogy nagy az átfedés azokban a laphalmazokban, amik rájuk mutatnak. Ezekben benne lesznek a téma gyűjtőlapjai. A következő ábra szemlélteti a tekintélyek és az univerzálisan népszerű lapok közötti különbséget. A téma gyűjtőlapjai és tekintélyei általában egy sűrű páros gráfot

13.4. ábra. Topológiai különbség a tekintélyek és az univerzálisan népszerű lapok között

alkotnak, míg az univerzálisan népszerű lapokra szabálytalanul, összevissza mutatnak a linkek.

A sűrű páros gráf megtalálása a következőképpen történik. Legyen C a B_σ weblaphalmazhoz tartozó szomszédossági mátrix, tehát $c_{ij} = 1$ ha $i \to j$,0 különben. Ez hasonlít a Page Rank algoritmusnál ismertetett A mátrixra, azzal a különbséggel, hogy nincs sztochasztikusan skálázva. Rendeljünk minden laphoz egy gyűjtőlap, illetve egy tekintélylap értéket, tehát vezessük be a

$$g = (\ldots, g_i, \ldots), g_i \geq 0$$

$$t = (\ldots, t_i, \ldots), t_i \geq 0$$

gyűjtő-, illetve tekintély vektorokat, amelyek legyenek normált vektorok, tehát $||g|| = ||t|| = 1$. A két vektorra a tekintély és gyűjtőlap intuitív definíciója miatt legyen érvényes a következő két szabály:

$$g = \lambda C t$$

$$t = \mu C^T g$$

azaz egy lap gyűjtőértéke az általa mutatott tekintélyértékeinek összege- λ-val skálázva, és egy lap tekintélyértéke azon lapok gyűjtőértékeinek összege, amelyek rá mutatnak-μ-vel skálázva.

A két egyenletet egymásba írva:

$$g = \lambda\mu C C^T g$$

$$t = \lambda\mu C^T C t$$

Hasonlóan, mint az oldalak rangját a Page Rank algoritmusnál, a g és t vektorokat is iteratívan határozzuk meg. A lépések:

I. $t^{(0)} = g^{(0)} = \begin{pmatrix} \frac{1}{|B_\sigma|} \\ \vdots \\ \frac{1}{|B_\sigma|} \end{pmatrix}$

II. $\hat{t}^{(i+1)} \leftarrow C^T C t^{(i)}$ és $\hat{g}^{(i+1)} \leftarrow C C^T g^{(i)}$

III. $t^{(i+1)} \leftarrow \frac{\hat{t}^{(i+1)}}{||\hat{t}^{(i+1)}||}$ és $g^{(i+1)} \leftarrow \frac{\hat{g}^{(i+1)}}{||\hat{g}^{(i+1)}||}$

IV. ha teljesül a leállási feltétel, akkor STOP, ha nem GOTO 2

A leállási feltételről hasonló mondható el, mint a Page Rank algoritmusnál: nem g és t pontos értéke érdekel bennünket, hanem az első néhány, legnagyobb tekintélyértékkel rendelkező oldal. A tapasztalati eredmények azt mutatták, hogy 20 iteráció után a legnagyobb 5-10 tekintélyértékkel rendelkező oldal már stabilizálódik.

A kísérleti eredmények mellett mindig hasznos, ha matematikai tételek is igazolják azt, hogy az algoritmus véget fog érni, azaz $t^{(i)}$ és $g^{(i)}$ konvergálnak valahova. A következő tétel ezt a matematikai megalapozást nyújtja. A tétel bizonyítása a B függelékben található.

13.3. tétel. *A fent definiált $t^{(i)}$ és $g^{(i)}$ sorozatok konvergálnak nemnegatív értékű vektorokhoz.*

Kleinberg módszere igen jó eredményt ért el lényeges oldalak kiszűrésénél nagy találati halmazokból. Például a $\sigma =$"Gates"-re, a legfontosabb oldalnak a **http://www.roadahead.com**-ot találta, majd ezek után jöttek a Microsofthoz kapcsolódó oldalak. A győztes oldal Bill Gates könyvének hivatalos weblapja, amit az AltaVista csak a 123. helyre rangsorolt.

13.2.2. A SALSA módszer

Az algoritmus ([92], Stochastic Approach for the Link-Structure Analysis) a már megismert Mag és Bázis halmazokon dolgozik, és egy véletlen sétát valósít meg az alább definiált gráfokon, amely az eredeti gráf pontjainak Gyűjtőlap és Tekintély tulajdonságait emeli ki.

A G_t ill. G_g gráfok csúcsai legyenek az eredeti gráf csúcsai (a weboldalak), az i és j pont között pedig annyi él van, ahány olyan csúcs (Gyűjtőlap) van, amiből i-be és j-be is mutat link, ill. hány olyan csúcs (Tekintély) van, amibe i-ből és j-ből is van él.

Megjegyzésként elmondható, hogy a HITS algoritmusban egy (dupla) lépés alatt ezen gráfok összes élén továbbadtuk az induló csúcs pontszámát, míg a SALSA algoritmusnál nem az egészet, hanem figyelembe vesszük azt, hogy minden csúcs ugyanannyit továbbítson, így egy-egy Markov láncot definiálunk a gráfokon.

Az M_t és M_g Markov láncok formális definíciójához a $B(i) = \{k : k \to i\}$ mellett szükségünk lesz a $F(i) = \{k : i \to k\}$ jelölésre. Az előző bekezdés szerint a megfelelő átmenetvalószínűségek a következők:

$$P_t(i,j) = \sum_{k:k\in B(i)\cap B(j)} \frac{1}{|B(i)|}\frac{1}{|F(k)|} \text{ ill.}$$

$$P_g(i,j) = \sum_{k:k\in F(i)\cap F(j)} \frac{1}{|F(i)|}\frac{1}{|B(k)|}.$$

Az egyensúlyi súlyokat kiszámító iteráció indítása

$$[\underline{t}]_0 := [\underline{g}]_0 := \frac{1}{N}(1,\ldots,1)^T,$$

majd az iteráció lépése:

$$[\underline{t}(i)]_k := \sum_j P_t(j,i)\,[\underline{t}(j)]_{k-1}\,, \text{ ill.}$$

$$[\underline{g}(i)]_k := \sum_j P_g(j,i)\,[\underline{g}(j)]_{k-1}\,.$$

Feltéve egy pillanatra, hogy a Markov láncaink irreducibilisek, azaz a két fent definiált gráf összefüggő, az állítható, hogy az egyensúlyi eloszlásokban két pont tekintély ill. gyűjtőlap súlyának aránya megegyezik az eredeti gráfban vett be- ill. ki-fokszámainak arányával. Az állítás abból következik, hogy irreducibilis Markov láncnak egyértelmű a stacionárius eloszlása, és a fenti súlyarányokat feltéve az állítás ellenőrizhető a következőképpen:

Az irreducibilitás miatt egyértelmű stacionáris eloszlás ki kell elégítse, hogy

$$\forall i \; \underline{\mathbf{t}}(i) = \sum_j P_t(j,i)\underline{\mathbf{t}}(j).$$

Most B-vel az élek halmazát jelölve és feltéve az előzőek szerint, hogy

$$\forall i \; \underline{\mathbf{t}}(i) = \frac{|B(t)|}{|B|},$$

így számolhatunk:

$$\begin{aligned}
\underline{\mathbf{t}}(i) = \sum_j \underline{\mathbf{t}}(j)P_t(j,i) &= \sum_j \underline{\mathbf{t}}(j) \sum_{k\in B(j)\cap B(i)} \frac{1}{|B(j)|}\frac{1}{|F(k)|} = \\
&= \sum_j \frac{|B(j)|}{|B|} \sum_{k\in B(j)\cap B(i)} \frac{1}{|B(j)|}\frac{1}{|F(k)|} = \\
&= \sum_j \frac{|B(j)|}{|B|}\frac{1}{|B(j)|} \sum_{k\in B(j)\cap B(i)} \frac{1}{|F(k)|} = \\
&= \frac{1}{|B|}\sum_j \sum_{k\in B(j)\cap B(i)} \frac{1}{|F(k)|} = \\
&= \frac{1}{|B|}\sum_{k\in B(i)} \sum_{j\in F(k)} \frac{1}{|F(k)|} = \\
&= \frac{1}{|B|}\sum_{k\in B(i)} 1 = \frac{|B(i)|}{|B|}
\end{aligned}$$

Ennek megfelelően a leírt iteratív algoritmus lefuttatására tulajdonképpen nincs szükség, hiszen a stacionáris eloszlás az előbbi elméleti eredmény felhasználásával *közvetlenül számítható*. Ezzel együtt természetesen az is igaz, hogy az algoritmus könnyen becsapható, hiszen – az előző fejezetben leírtak szerint – az oldalunkra mutató linkek száma tetszőlegesen növelhető.

Itt jegyezzük meg, hogy a SALSA az egyenletes eloszlással indított HITS algoritmus első lépésének felel meg, ezután az első lépés után a SALSA stacionáris eloszlásának súlyai jelennek meg.

Több komponensből álló gráf esetén az algoritmus csak abban a komponensben dolgozik, ahonnan indult a séta. Mivel az indulás egyenletesen lett választva, ezért egy adott komponensből való indulás valószínűsége a komponens méretével arányos, azaz az alapgráfot G-vel, komponenseit G_k-val, az i csúcs komponensének indexét j-vel jelölve

$$a_i = \frac{|G_j|}{|G|}\frac{|B(i)|}{\sum_{\alpha\in G_j}|B(\alpha)|},$$

ahol a nevezőben levő összeg a komponens összes éleinek száma.

13.2.3. Gyűjtőlapok, Tekintélyek és véletlen séták

Mint láthattuk, a SALSA algoritmus ötlete az eredeti gráfból egyszerűen származtatható másik gráf(ok)on megvalósított véletlen séta volt. Láttuk továbbá, hogy az eredeti Gyűjtőlap és Tekintély algoritmusunk első lépése ekvivalens a SALSA-val. Jogosan kérdezhetjük tehát, hogy az eredeti algoritmusnak létezik-e véletlen séta analogonja, illetve átfogalmazva a kérdést, hogy az eredeti algoritmus is kapcsolatba hozható-e Markov láncok stacionáris eloszlásaival? A válasz persze igenlő, hiszen minden sztochasztikus vektorhoz létezik olyan Markov lánc, amelynek stacionáris eloszlása éppen az a vektor. A kérdés már csak az, hogy létezik-e olyan ezzel a tulajdonsággal bíró Markov lánc is, amelynek átmenetvalószínűségei az eredeti gráfból származtathatók?

A válasz – kissé meglepő módon – az, hogy az algoritmus összes közbülső eredménye előáll az eredeti gráfból származtatható Markov lánc stacionáris eloszlásaként, bár az, hogy az első fél lépés után már igaz ez (ott a SALSA súlyok jelennek meg), már előrevetíti az eredményt. Vezessük be a következő jelöléseket: egy B illetve egy F lépésnek egy a webgráfban levő link követését nevezzük hátra illetve előre irányban. Ezek kombinációit is definiáljuk, például $BFBF = (BF)^2$ egy négylépéses sétát jelent a webgráfban. Az i pontból a j pontba vezető $(BF)^n$ séták halmazát jelölje $(BF)^n(i,j)$, az i pontból induló $(BF)^n$ séták halmazát jelölje $(BF)^n(i)$, továbbá az összes $(BF)^n$ séták halmazára használjuk magát a $(BF)^n$ jelölést! Az $(FB)^n$ séták halmazai hasonlóan értendők.

Most definiáljuk a következő két Markov láncot: az állapotok halmaza az összes csúcs, amely magában az alapgráfban is benne volt, míg két csúcs között pontosan akkor van él, ha az alapgráfban van köztük legalább egy $(BF)^n$ illetve $(FB)^n$ séta. Az átmenetvalószínűségek pedig legyenek:

$$\begin{aligned} P_t(i,j) &:= \frac{|(BF)^n(i,j)|}{|(BF)^n(i)|}, \quad \text{illetve} \\ P_g(i,j) &:= \frac{|(FB)^n(i,j)|}{|(FB)^n(i)|}. \end{aligned}$$

A definíciókból az látható, hogy

$$\begin{aligned} |(BF)^n(i,j)| &= (C^TC)^n(i,j) \quad \text{és} \\ |(FB)^n(i,j)| &= (CC^T)^n(i,j), \quad \text{és ezekből} \end{aligned}$$

$$\begin{aligned} |(BF)^n(i)| &= \textstyle\sum_j (C^TC)^n(i,j) \quad \text{és} \\ |(FB)^n(i)| &= \textstyle\sum_j (CC^T)^n(i,j). \end{aligned}$$

Az eredeti HITS algoritmus n. iterációja után a pontszám vektorok normálás nélkül $(C^TC)^n\underline{\mathbf{1}}$ illetve $(CC^T)^n\underline{\mathbf{1}}$, azaz összeg normában ez ugyanaz, mint a megfelelő Markov láncok stacionáris eloszlása:

$$\begin{aligned} \underline{t}(i) &= \frac{|(BF)^n(i)|}{|(BF)^n|} \\ \underline{g}(i) &= \frac{|(FB)^n(i)|}{|(FB)^n|} \end{aligned}$$

Elmondható tehát, hogy az algoritmus végső pontszámarányai a csúcsokból induló hosszú BF illetve FB séták számainak arányától függ, aminek az a következménye, hogy nagyon erősen kötött alakzatok (teljes páros részgráfok) környékét az algoritmus kiemeli.

13.2.4. Automatikus forrás előállító - Gyűjtőlapok és Tekintélyek módosításai

Gyűjtőlapok és Tekintélyek alapú keresést sikerrel alkalmazták automatikus forrás előállítás során (automatic resource compilation, röviden ARC) [28]. A továbbiakban erről szólunk pár szót.

Általánosabb fogalmak keresésénél gyakran használunk előre szerkesztett hierarchikus fogalomtárakat. A legismertebb fogalomtárak a Yahoo! vagy az Infoseek oldalán találhatók. Ha például információkra van szükségünk a tangóról, akkor a Yahoo! főoldaláról a Recreation & Sports (kikapcsolódás és sport) linket választva eljuthatunk egy újabb oldalra. Itt már választhatjuk a dance (tánc) linket, majd a Ballroom-ot (társas) és végül a tangót. Innen már nem léphetünk tovább újabb alkategória kiválasztásával, hanem a tangóval foglalkozó legfontosabb weboldalak listáját láthatjuk. Mind a fogalomhierarchia felépítése, mind az egyes fogalmakhoz tartozó legfontosabb weboldalak megkeresése manuális úton történik, tehát emberek járják a világhálót és keresik az olyan oldalakat, amelyek tényleg hasznos információval szolgálnak a fogalomról.

Az ARC-nál a második lépést próbálták automatizálni: adott egy tág fogalom, keressük meg a hasznos információkat tartalmazó weboldalakat. Ehhez a Gyűjtőlapok és Tekintélyek keresést használták, két módosítással.

Egyrészről kétszer, nem pedig egyszer alkalmazták azt a lépést, amely során a Magból(M_σ) a Bázist(B_σ) előállították. Emiatt a Bázis mérete nőtt, viszont nem vesztünk el olyan oldalt, amely a hagyományos kereső által kiadott oldalaktól 2 link távolságra van.

Másrészről módosították az iteráció során használt C mátrixot is. Tudjuk, hogy a weboldalak HTML kódjában a <A HREF=""></A> tag jelent egy linket. Ha például egy oldalban a `<A HREF="http://www.mtnsms.com">ingyen sms</A>` tag található, akkor ha a szörfölő az *ingyen sms* szóra kattint, akkor a **www.mtnsms.com** oldalra kerül.

Megfigyelték, hogy nagyon gyakran a HREF tag környezetében az oldalt jellemző szavak találhatók. Ez nem meglepő, hiszen az oldalak készítői minél jobban próbálják segíteni az oldalt látogatóinak navigációját. A tag környezete tehát fontos, mert ha megtalálható ott a kérdéses fogalom, akkor várható, hogy a link egy hasznos oldalra mutat.

Szomszédossági mátrix helyett ezért olyan mátrixot javasoltak, amely elemei a következőképp számíthatók:

$$c_{ij} = \begin{cases} 1+n(f) & \text{ha j-re mutat link i-ről} \\ 0 & \text{egyébként} \end{cases}$$

ahol $n(f)$ a fogalom előfordulásának száma egy adott szélességen belül a HREF tagtől.

A szélességet kísérleti úton próbálták meghatározni: azt vizsgálták, hogy pár ismert oldalra mutató több ezer oldalban hol található meg az ismert oldalakat jellemző szó. A tesztek eredményeként megállapították, hogy ha az oldalon megtalálható a jellemző szó, akkor 97%-ban az a HREF 50 byteos környezetében is megtalálható.

Az algoritmust implementálták, és széleskörű felmérést készítettek, amelyben a megkérdezetteknek arra kellett válaszolniuk, hogy szerintük adott fogalmakra a három kereső(ARC, Infoseek, Yahoo!) közül melyik találta meg a legjobb oldalakat. A felmérésből kiderült, hogy a teljesen automatikus, emberi munkát nem igénylő ARC ugyanolyan jól teljesített, mint a másik két rendszer [28].

13.2.5. Gyűjtőlapok és Tekintélyek módszerének hátrányai

Vizsgálatok kimutatták, hogy a Gyűjtőlapok és Tekintélyek módszerének három hátránya van [17].

I. Előfordulhat, hogy egy hoszton található dokumentumhalmaz minden eleme egy másik hoszton található dokumentumra mutató linket tartalmaz. Ez növelni fogja a dokumentumhalmaz elemeinek gyűjtőlap értékét és a másik hoszton található dokumentum tekintélyértékét. Ennek ellenkezője is könnyen előfordulhat, nevezetesen: egy hoszton található dokumentum több olyan dokumentumra mutat, amelyek egy másik hoszton találhatóak. Látható, hogy ál hosztpárok létrehozásával a gyűjtőlap- és tekintélyértékek növelhetők, ami visszaélésre ad lehetőséget. Egy igazságos algoritmustól elvárjuk, hogy egyik hoszt se növelhesse túlzott mértékben mások fontosságát.

II. A weboldalakat gyakran automatikusan állítják elő valamilyen segédeszköz segítségével. Ezek az eszközök sokszor linkeket helyeznek el a generált oldalakon. Például a Hypernews rendszer USENET cikkeket konvertál weblapokká úgy, hogy a Hypernews honlapjára mutató linket szúr az oldal végére. Ezekre a linkekre nem igaz a fejezet elején elhangzott állítás, miszerint az oldal szerzője azért helyezi el a linket oldalán, mert a másik oldal a saját oldal témájára nézve hasznos információkat tartalmaz.

III. Bázis laphalmaz létrehozása során a Mag laphalmazhoz új oldalakat veszünk fel a linkstruktúra alapján. Az új oldalak között sok olyan lehet, amelyek nem kapcsolódnak a kérdéses témához. Amennyiben ezeket az oldalakat szoros linkstruktúra köti össze, akkor a „témasodródás” problémája mutatkozik: a legnagyobb tekintélyértékkel rendelkező oldalak csak tágabb értelemben fognak a témához kapcsolódni. Egy egyszerű teszt megmutatta, hogy a "jaguar *and* car" kérdésre a legjobb tekintélyoldalak (amelyek különböző autógyártó cégek honlapjai lettek) az általánosabb fogalomhoz (car) kapcsolódtak.

Az első esetben a problémát az okozza, hogy egy hoston elhelyezett több dokumentum összbefolyása túl nagy lehet: minél több dokumentum található egy hoszton, annál inkább képes növelni más hoszton található dokumentum tekintély- vagy gyűjtőlapértékét.

Ideális esetben azt várnánk, hogy egy hoszton található dokumentumhalmaznak összesen akkora befolyása legyen, mintha ezen a hoszton csak egyetlen dokumentum lenne található. Ehhez módosítanunk kell az iteráció során használt mátrixot: amennyiben egy hosztrol k darab dokumentum tartalmaz linket egy másik hoszton található dokumentumra, akkor a C mátrix ezen dokumentumaihoz tartozó értéke 1 helyett $\frac{1}{k}$ legyen.

Az [17] cikkben a másik két problémára is javasoltak megoldást. Szövegelemzés felhasználásával a Bázisban található oldalakhoz relevancia értéket társítanak, ami megadja, hogy az adott oldal mennyire kapcsolódik a témához. A relevancia értéknek több szerepe van. Egyrészt a témához kis mértékben kapcsolódó (kis relevancia értékű) lapokat töröljük a Bázisból, másrészt a tekintély- illetve gyűjtőlapérték meghatározásához a lap relevanciaértékét is figyelembe vesszük: a relevanciaértékkel arányosan nő egy lap tekintély- illetve gyűjtőlapértéke. A szövegelemzéssel bővített Gyűjtőlapok és Tekintélyek módszerét a továbbiakban nem tárgyaljuk, a részletek megtalálhatók a [17] cikkben.

A fejezetben bemutatott két fő algoritmusról pár összehasonlító tesztet találhatunk a [10] cikkben.

14. fejezet

Adatbányászat a gyakorlatban

Az eddigi fejezetekben matematikai modellekről, megoldandó feladatokról és algoritmusokról beszéltünk. E fejezet nem lesz ennyire tudományos: az adatbányászat legtipikusabb felhasználási területeit fogjuk átnézni. Azt vizsgáljuk, hogy milyen jellegű összefüggések után érdemes kutatni, és hogy ezen összefüggések felderítése milyen előnyökkel jár.

Az adatbányászat napjaink egyik legnépszerűbb területe. Nem meglepő, hogy napról napra új adatbányászati szoftver jelenik meg, hirdetve magáról azt, hogy a piac legjobb terméke. A fejezet második részében összefoglaljuk a jelenleg kapható adatbányászati szoftvereket, majd kitérünk arra, hogy milyen szempontokat vegyen figyelembe egy cég a megfelelő szoftver kiválasztásánál.

14.1. Felhasználási területek

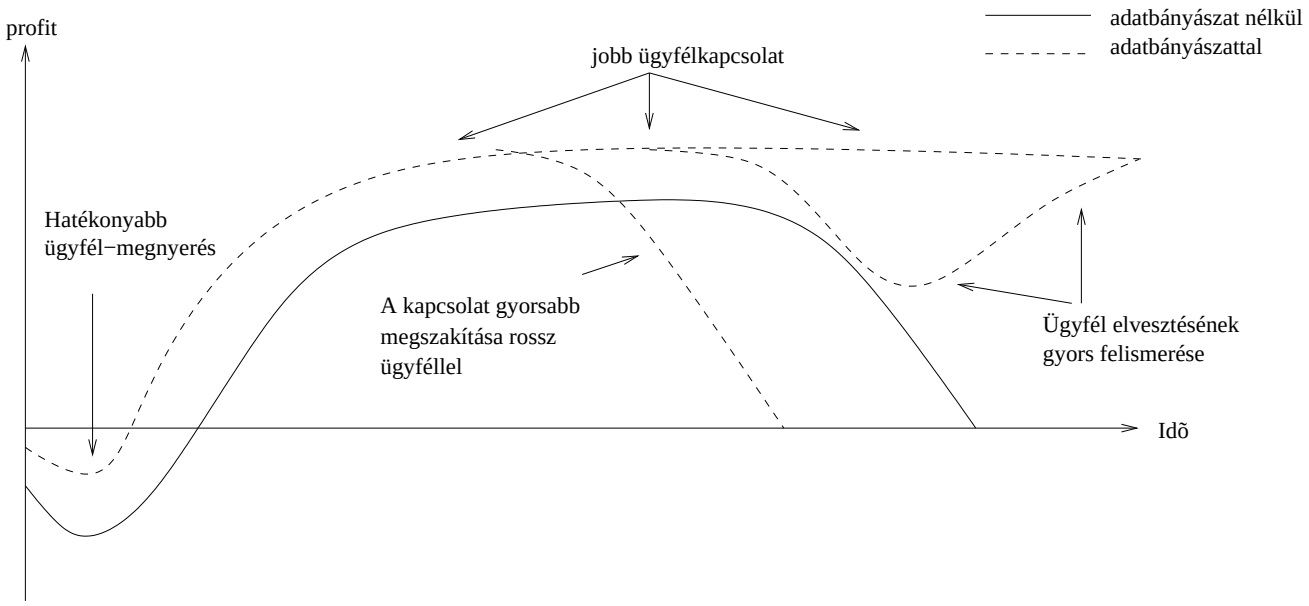
A sikeres alkalmazások hatására az adatbányászat egyre elfogadottabb tudományággá vált. Már szinte mindenhol fontos az adatok tárolása mellett azok feldolgozása és elemzése. A kinyert információ új tételek, törvényszerűségek felfedezését segítheti elő, vagy éppen fordítva: meglévő hipotéziseket cáfolhat meg. Ebben a részben 3 olyan területről szólunk, ahol az adatbányászat már mélyen gyökeret vert és az egyik legfontosabb eszközzé vált. Ezek a területek pedig: (1.) kereskedelem, (2.) pénzügy, (3.) biológia és orvostudomány. Az itt leírtakon túl számos esettanulmányt sikeres alkalmazásról szóló hírt lehet találni például a magyar adatbányászok honlapján (`http://www.datamining.hu`).

14.1.1. Az ügyfél életciklusa

A kereskedelemben és a pénzügyben is a profitot az ügyfelek termelik. A következő ábrán láthatjuk, hogy milyen változásokat képes hozni az adatbányászat az ügyfelek életciklusában.

Az adatbányászat segítségével hatékonyabban fel tudjuk ismerni a potenciális ügyfelek körét. Igy kevesebbet költünk azon ügyfelekre, amelyek nagy valószínűséggel nem lesznek ügyfeleink, azaz faragunk a költségeken.

Meg tudjuk különböztetni a jó és a rossz ügyfeleket. A rossz ügyfelektől hamarabb megszabadulhatunk, és az ezáltal megtakarított összegeket a jó ügyfelekre fordíthatjuk, ami még gyümölcsözőbb kapcsolatot eredményezhet. Azt is idejében észrevehetjük, ha egy jó ügyfelünknek növekszik az elégedetlensége velünk szemben.



14.1. ábra. Az ügyfél életciklusa

14.1.2. Kereskedelem

Többször hoztunk fel példát az adatbányászat kereskedelmi felhasználásáról. Magát az asszociációs szabályokat is egy kereskedelmi példán keresztül vezettük be. Ez nem véletlen, hiszen a vásárlói kosarak elemzésének igénye keltette életre ezt a területet.

A kereskedelemben ma már minden üzletben megtalálhatóak a működést segítő számlázó, raktárkészlet-kezelő, programok. Ezek egyre összetettebbek, a puszta vásárlások felsorolásánál és visszakeresésénél jóval többet tudnak: ha rendelkezésünkre áll valamilyen vevőazonosító, akkor a vevők teljes vásárlói történetét megkaphatjuk, de ezenfelül hitelekről, beszerzésekről, szállításokról is rögzíthetünk adatokat. A szervízekből érkező visszacsatolásoknak is fontos szerepe lehet egy termék sikerénél.

Az on-line áruházak elterjedésével a vevőkről begyűjtött adatok minősége tovább javul. Az elektronikus kereskedelemről külön részben szólunk bővebben.

Az adatbányászatnak a kereskedelem területén a következő céljai lehetnek.

– Vásárlói szokások elemzése az asszociációs szabályok, illetve az epizódok kinyeréséhez vezetnek. A szabályokat felhasználhatjuk eladást ösztönző akciók szervezésénél, áruházak térképének kialakításánál, eladáshelyi reklámeszközök kialakításánál, termékfejlesztésnél . . . .

– Klaszterezés és osztályozás segítségével vásárlói csoportokat hozhatunk létre. A célcsoportok pontosabb behatárolásával irányíthatóbb, emberközelibb, interaktív, egyedi és hatékonyabb reklám- és marketingstratégiát készíthetünk.

– A személyre szabott ügyfélszolgálat nagyban fokozza a vásárlók elégedettségét.

– A vásárlói szokások jobb megismerésével pontosabban tudjuk megjósolni az egyes termékek értékesítési adatait. Ha egy üzletnek pontos képe van a raktárkészletről, a fogyás üteméről és az igényekről, akkor hatékonyabban tudja megszervezni a beszerzéseket, a disztribúciós csatornák

típusát, nagyságát, a raktározás módját (pl.:just-in time). A hatékonyság növelésével csökkenteni tudjuk a költséget és jobban elosztani az erőforrásokat.

– Az új vevők toborzása mellett a régiek megtartása egyre fontosabb (CRM- Customer Relation Management). A vevők vásárlói sorozatainak elemzésével képet kaphatunk arról, hogy kinek csökkent a vásárlói kedve, így még azelőtt tehetünk ellene valamit (pl. hűségakció), hogy végképp elpártolna tőlünk.

14.1.3. Pénzügy

A bankok szolgáltatásai közül kiemelten fontosak a számlák, lekötött betétek vezetése, a hitelek nyújtása és egyéb pénzügyi tranzakciók lebonyolítása. Ezeken a területeken mára elismertté vált az adatbányászat szerepe.

– A bankok eredetileg azért jöttek létre, hogy mások értékeit megőrizzék. Az ügyfelek így biztonságban tudták a pénzüket, ami a kamatok miatt gyarapodott is, a bankok pedig nagy tőkékhez jutottak, amit be tudtak fektetni. Mára a bankok az ügyfeleket eltérően kezelik (lakossági, vállalati ügyfelek, számlaforgalomtól függően átlagos, fontos, kiemelt ügyfelek . . .). Az ügyfelek és a tranzakciók nagy száma miatt az ügyfélcsoportok manuális kialakítása lehetelen feladat. A klaszterezés és az osztályozás ezért ezen a területen kiemelten fontos eszköz.

– Hitelek nyújtása a kamatok és a rendszeres jövedelemforrás miatt jó befektetés a banknak. A kérelmező körülményeinek megvizsgálása nélkül osztogatni a hiteleket azonban kockázatos, mert lehet, hogy az ügyfél nem tudja visszafizetni. Ha az ügyfelekről sok adat áll rendelkezésre (nettó jövedelem, beosztás, családi állapot, korábbi banki tranzakciói . . .), akkor az osztályozást felhasználva olyan döntési fákat lehet létrehozni, amelyek nagy bizonyossággal megállapítják adott ügyfélről, hogy megbízható hitel szempontjából vagy nem. Ezt a módszert elsősorban olyan országokban alkalmazzák, ahol a hitel odaítélését nem kötik nagyon szigorú feltételekhez. Amerikában például elterjedt szokás, hogy Karácsony előtt a bankok előzetes megrendelés nélkül hitelkártyákat küldenek szét, amit a címzett nem köteles használni, de ha fizet vele, akkor a hitelt néhány hónapon belül vissza kell fizetnie. Nyilvánvalóan a megnövekedett vásárlói kedv ily módon való ösztönzése kiemelt jelentőségű lehet egy bank számára partnerkörének kibővítésénél, megtartásánál és természetesen a plusz generált pénzforgalom figyelembe vételénél, de ezen marketingakció kockázata igen magas, ha a hitelkártyát használó a későbbiekben nem fizeti vissza a bank pénzét.

– A bank/hitelkártyával történő fizetés és készpénzfelvétel a civilizáció nélkülözhetetlen eleme. A rengeteg biztonsági intézkedés ellenére a kártyás csalások még mindig sok kárt okoznak. Mivel a tranzakciók száma óriási, ezért manuális eszközökkel ebben az esetben is lehetetlen feladat kiszűrni a szokatlan viselkedést, ami a csalókra, kártyatolvajokra jellemző. Az eltéréselemzés az adatbányászat azon területe, ahol a szokásostól eltérő viselkedés, mintázat felfedezése a cél.

14.1.4. Biológia és Orvostudomány

A biológiában és az orvostudományban az adatok elemzéséből kapott törvényszerűségek értéke felbecsülhetetlen. Adatbányászat segítségével fejlesztenek új gyógyszereket, segít a rák elleni terápia

hatékonyabb kialakításában, különböző betegségek tüneteinek meghatározásában.... Ebben a részben két fontos alkalmazásról szólunk: a DNS láncok elemzéséről és a cukorbetegség kezelésének segítségéről.

DNS láncok elemzése

Az orvostudomány talán legnagyobb megoldatlan feladata a DNS láncok teljes megfejtése. Tudjuk, hogy minden élőlényt egyértelműen azonosít a DNS lánca, mintha egy genetikai kódot kaptunk volna a természettől! A DNS láncok a felelősek többek között a betegségekért, bizonyos emberi tulajdonságokért, hajlamokért, allergiákért.... Éppen ezért a kiemelt szerepért a DNS láncok fontosságát aligha lehet túlbecsülni.

Minden DNS lánc 4 építőkőből épül fel, ezek a nucleotidok: adenin, cytosin, guanin és thymin. Ez a négy nucleotid alkot egy hosszú láncot, ami leginkább egy spirál alakú létrára emlékeztet. Egy ember kb. 100 ezer génnel rendelkezik, egy gén pedig általában többszáz nucleotidból épül fel, ahol a nucleoidok sorrendjének fontos szerepe van. A DNS láncok elemzése nem pusztán epizódkutatásról szól. A tudás kinyeréséhez ötvözni kell a különböző adatbányászati technikákat!

– DNS láncokat gyakran kell összehasonlítani, ezért sorozatok hasonlóságának elemzése fontos módszer. Beteg és egészséges szövetekből vett minták összevetéséből megállapíthatjuk a kritikus eltéréseket. Először a két mintát külön vizsgálják és nyerik ki a gyakran előforduló mintázatokat. Később már csak ezeket a mintázatokat vetik össze. A beteg szövetben jóval gyakrabban előforduló mintázatok lehetnek a betegség genetikai tényezői. Vagy fordítva, az egészséges szövetben gyakrabban előforduló mintázatok adhatnak alapot a gyógyszer elkészítéséhez.

– A betegségekért általában nem csak egy gén felelős, hanem a gének egy kombinációja. Az asszociációs szabálykeresésnél megismert módszerekkel lehet feltárni a gyakran együtt előforduló esetleg felelős géneket beteg egyedek egy adott csoportjában.

– A gének és betegségek világát tovább bonyolítja, hogy a betegség különböző fázisaiban esetleg más-más gének aktívak. Ha egy adott betegségnél sikerülne ezt feltérképezni, akkor a különböző fázisokhoz elkészített gyógyszerek kifejlesztésével növelni lehetne a kezelés hatékonyságát.

Cukorbetegség

A cukorbetegség egy elterjedt és nem megfelelő kezelés esetén halált okozó betegség. Kezelésére a betegnek inzulint kell a szervezetébe juttatnia. Amennyiben a beteg túl sok inzulint kap, akkor szervezete tovább csökkenti a cukor termelését, és betegség tovább súlyosbodik. Ha viszont a beteg kevés inzulint kap, akkor szervezetében cukorhiány mutatkozik, aminek hatásaként szédülés, ájulás, sőt akár bénulás is előállhat.

Az inzulin megfelelő adagolása ezért kiemelten fontos a cukorbetegség kezelésében. A tudomány jelenlegi állása szerint nincs pontos képlet arra nézve, hogy egy adott paraméterekkel rendelkező beteg mekkora adagot kapjon. Habár egyre több eszköz jön létre a minél gyorsabb visszajelzésre, az adagok meghatározása még mindig ösztönszerűen, az orvos le nem írt, konkrétan meg nem fogalmazott tapasztalatai alapján történik.

Az inzulin megfelelő adagjának kiválasztása rengeteg paramétertől függ (testsúly, kor, nem, betegségre jellemző adatok stb.). A különböző mérő- és figyelőeszközök piacra kerülésével egyre több adat gyűlik össze, így lehetővé válik ezek elemzése. Az osztályozás, korrelációanalízis, asszociációkutatás mind fontos eszközök a cukorbetegség kutatásában.

14.2. Az adatbányászat bölcsője: az elektronikus kereskedelem (e-commerce)

A bevezetőben szó volt arról, milyen feltételei vannak a sikeres adatbányászatnak (lásd 17.oldal). Idézzük fel ezeket a feltételeket, és nézzük meg, hogyan teljesülnek az elektronikus kereskedelemben.

sok adat: Közismert weboldalnak nagy a látogatottsága.

sok attribútum: Az on-line áruházaknál lehetőség van a vásárló fontosabb adatainak tárolására, de ezenfelül több más információt is megtudhatunk róla, pl. hogy mi iránt érdeklődik gyakran a látogató, milyen reklámokat néz meg, ...

tiszta adat: Az adatok az emberi rögzítés hibájától mentesek. A weboldal készítője határozhatja meg, milyen típusú adatok legyenek tárolva, illetve, mely mezőket kell kötelezően kitölteni.

akcióképesség: A kinyert tudás birtokában megváltoztathatjuk a weboldalt (akár a teljes designt, akár csak a linkeket), személyre szabott oldalakat készíthetünk, e-maileket küldhetünk ki, ...

befektetés megtérülése: Mivel minden elektronikusan zajlik a bevételnövekedés kiszámítása alapfeladat. Sőt még a célzott marketing hatékonyságát is könnyedén megállapíthatjuk, hiszen rögzíteni tudjuk, ha valaki egy e-mailen keresztül jutott az oldalunkra.

A fentiek ellenére az adatbányászat alkalmazása az elektronikus kereskedelemben korántsem akadálytalan [84]. A problémák forrása az, hogy az adatokat a webszerverek mentik el. A webszerver adatainak bányászata kézenfekvőnek tűnik, hiszen a webszerverek szinte mindent rögzítenek. A naplófájlokat azonban eredetileg a webszerverek debuggolására találták ki, nem pedig az adatbányászat támogatására.

A legfőbb problémák az alábbiak:

- Nem lehet egyértelműen azonosítani a felhasználót. Szemben az adatbányászattal, a webszerverek számára ez ugyanis nem fontos információ. Próbálkoznak a felhasználók cookie, IP, vagy böngésző szerinti azonosításával [32], de ezek közül egy sem nyújtja a tökéletes megoldást [15].

- A webszerverek nem tárolnak minden fontos adatot. A naplófájlokban nincs nyoma például a „berakom a kosárba”, „mennyiség megváltoztatása”, „termék törlése” műveleteknek.

- A form-ok adatai nincsenek tárolva. Pedig gondoljuk meg, hogy például a keresési form-ok éppen a vásárlások érdeklődését tükrözik.

- A naplófájlokban URL-ek szerepelnek, nem pedig az oldal tartalma. Nem mindig könnyű meghatározni, hogy adott termék melyik oldalhoz tartozik. A helyzetet tovább bonyolítja, hogy gyakran ugyanaz az információ több nyelven is elérhető.

- A dinamikus oldalak tartalmát sem lehet egyértelműen meghatározni. Melyik termék érdekelte a látogatót, ha az összes terméket a `termek.jsp` oldal mutatja? Vagy csak egy reklám volt a felbukkanó ablak? Esetleg egy „Nincs raktáron!” üzenet? Sikeres volt a keresés vagy nem hozott eredményt? Ezek a kérdések legtöbbször dinamikus oldalakhoz kötődnek és megválaszolásuk a naplófájlok alapján lehetetlen feladat.

– Az igazán nagy oldalaknak több webszerverük van, amelyek különböző helyeken helyezkednek el. Ezek mind saját naplófájllal dolgoznak, egyesítésüket nehezíti, hogy különböző időzónákban lehetnek.

A fenti problémákra megoldás nyújt, ha a vásárlásokkal kapcsolatos információk tárolását a vásárlásokat kiszolgáló program végzi, azaz az adatok tárolását az alkalmazási rétegre bízzuk. A valóságot tükröző adatok előállításának nagy ellenségei az Internetes robotok. Ezek olyan lekérdezéseket, oldalletöltéseket generálnak, amelyek nem tükröznek valóságos emberi érdeklődést. Intenzív kutatás tárgyát képezi a robotok által generált hamis adatok kiszűrése.

14.3. Adatbányász szoftverek

A továbbiakban egy rövid összefoglalót adunk a ma kapható legfontosabb adatbányász szoftverekről. A lista korántsem teljes. Ennek oka egyrészről a terjedelmi korlát, másrészről a nap mint nap változó piac.

Enterprise Miner (`http://www.sas.com/products/miner/index.html`) A SAS Institute,Inc. fejlsztette ezt a programcsomagot. A cég komoly múltra tekint vissza statisztikai elemzések terén. Az Enterprise Miner is számos statisztikai eszközt kínál fel, de már megtalálható az összes többi adatbányászati feladatra megoldás, csak úgy mint döntési fák, neurális hálózatok, regresszió, klaszterezés, sorozat-elemzés, asszociációbányászat.

Clementine (`http://www.spss.com/spssbi/clementine`) A Clementine az SPSS Inc. terméke. Integrált adatbányászati környezetet biztosít végfelhasználók és fejlesztők részére. Adatbányászati eszközök közül megtalálható a neurális hálózatok, osztályozás, sorozat-elemzés stb. A Clemetine szoftverében egyedi az az objektum-orientált interfész, amin keresztül a felhasználó saját algoritmusokat és funkciókat adhat meg.

Intelligent Miner (`http://www-3.ibm.com/software/data/iminer/fordata`) Az IBM terméke talán a legismertebb és a legelterjedtebb adatbányászati eszköz. Emellett fontos érv szól mellette: az IBM kutatóintézetében született jónéhány neves publikáció, tehát e szoftver mögött áll a legfelkészültebb kutatógárda. A programmal lehet bányászni asszociációkra, epizódokra, alkalmas osztályozási, klaszterezési feladatok ellátására, de ezenkívül lehet regressziót számolni és eltérést keresni. A fejlett adatmegjelenítés mellett képes statisztikai elemzésre és neurális hálózatokon alapuló algoritmusok futtatására. Az Intelligent Miner használatához IBM DB2 relációs adatbáziskezelő rendszernek is futnia kell.

DBMiner (`http://www.dbminer.com`) A DBMinert a Simon Fraser University által elkészített programból fejlesztette tovább a DBMiner Technology Inc.. Adatbányászati funkciók közül megtalálhatók a asszociációbányászat, karakterizáció, osztályozás, klaszterezés és jóslás. Ennek a programnak legszorosabb, legintegráltabb a kapcsolata az OLAP-pal. A szoros kapcsolat miatt itt már OLAM-ról (On-Line Analitical Mining) beszélünk. A program egy interaktív környezetet kínál a felhasználónak, aki dinamikusan váltogathat OLAP operációk és adatbányászati funkciók között.

MineSet A MineSet legnagyobb erőssége a fejlett vizualizácós képessége. Ez nem meglepő, hiszen a szoftvert a Silicon Graphics fejlesztette, amely cég mindig is a legjobbak közé tartozott a grafikában. A MineSetben megtalálható szinte az összes ismert adatbányászati funkció. További

előny, hogy a MineSet egyben fejlesztői környezetet biztosít új algoritmusok implementálásához, így ha valamely feladatra nincs kész megoldás, akkor megírhatjuk magunk, majd az eredmény megtekintéséhez használhatjuk a MineSet vizualizációs eszközeit.

A fenti öt óriásszoftver mellett felsorolás szinten szólnunk kell még az alábbi programokról: 4Thought, Alice, Darwin, Datascope, Scenario, Data Surveyor & Expert Surveyor.

14.3.1. Adatbányászati rendszerek tulajdonságai

Az előzőekben felsoroltunk néhány adatbányászati szoftvert. A felsoroltakon kívül léteznek még további szoftverek, amelyek bizonyos tekintetben akár jobbak is lehetnek a fentieknél. Ekkora választékban hogyan tudjuk megtalálni a nekünk megfelelő szoftvert, mik azok a tulajdonságok, amit mindeképpen meg kell vizsgálunk egy ilyen beruházás előtt.

Adatbányászati funkciók. Egy cég azért vásárol adatbányászati szorftvert, mert összefüggést akar kinyerni az adataiból. Már a szoftvervásárlás előtt hasznos, ha pontos elképzelése van arról, hogy milyen típusú összefüggéseket fognak keresni (asszociációs szabályok, epizódok, klaszterek stb.). A legfontosabb, hogy a szoftver funkciói között megtalálhatók legyenek az ilyen típusú összefüggések kinyerésének lehetősége.

Nem biztos, hogy a nekünk megfelelő szoftver lesz a legtöbb adatbányászati feladat megoldását támogató. Egyre több szoftver jelenik meg, amely egy adott feladatra szakosodik (pl.: weblog elemző szoftver), ugyanakkor az átfogó képeséggel rendelkezők mellett szól, hogy a jövőre is célszerű gondolni: milyen típusú összefüggéseket keresünk esetleg később.

Adattípus. A legtöbb szoftver a relációs adatbázisokban található adatokat tudja feldolgozni, de ezenkívül a sima szövegfáljt, munklapokat, ismertebb formátumú fájlokat is kezelik. Fontos tehát ellenőrizni, hogy pontosan milyen formátumú adatokon dolgozik. Ma már léteznek szoftverek, amelyek speciális adatformátumokat is kezelni tudnak, mint például földrajzi, multimédiás, web logok, DNS adatbázisok.

Adatforrás. Vannak adatbányász szoftverek, amelyeket fel kell tölteni az adatokkal mielőtt dolgozni lehet velük. Hasznosabb azonban, ha a szoftver a más adatbázisokban található adatokat is kezelni tudja. Fontos, hogy a rendszer támogassa az ODBC kapcsolatot vagy az OLE DB for ODBC-t. Ez lehetővé teszi a hozzásférést sok más relációs adatbázishoz (DB2, Informix, Microsoft SQL Server, Microsoft Access, Excel, Oracle stb.).

Adatméret, skálázhatóság. Tudnunk kell, hogy a szoftver mekkora adattal képes megbírkozni továbbá, hogy az adatbázis növelésével hogyan romlik a futási idő. Sklálázhatóság szempontjábó megkülönböztetünk *sor szerint skálázható* és *oszlop szerint skálázható* szoftvereket. Az első azt jeleti, hogy ha megduplázom a sorok számát, akkor nem nő duplájára a futási idő/memória igény. Az oszlop szerint skálázhatóság szerint a futási idő/memória igény az oszlopok számával lineárisnál nem rosszabb. Ez utóbbi feltétel teljesüléséhez kifinomultabb algoritmusokra van szükség.

Megjelenítési eszközök. A vizualizáció egy külön szakma. Az adatbányászati algoritmusok eredményeinek áttekinthető, szemléletes megjelenítése sokat segít az értelmezésben. A 3D ábrák, grafikonok, táblázatok nagyon hasznosak és sokat segítenek az adatbányászat használhatóságában és az eredmények interpretálhatóságában.

Az adatbányászat nagyon fiatal tudományág, így a szoftverek sem tekinthetnek vissza nagy múltra. A szoftverek szinte minden tekintetben különböznek egymástól. A megjelenítéssel, adatbányászati funkciókkal, terminológiával kapcsolatos egységes koncepció kialakulásáig még várnunk kell.

14.3.2. Esettanulmányok röviden

A következőkben vázolunk néhány sikeres adatbányászati projektet [105].

Szlovén médiaszokások feltárása

Ma a médiumok kezében óriási hatalom van mind politikai, mind üzleti értelemben. Az egyes újságok, tv műsorok „fogyasztóinak” megismerésével közvetlenül elérhetik az egyes cégek a célközönségeiket.

A szlovén Mediana mintegy 8000 (20 oldalas!) kérdőív adatait elemeztette adatbányászati módszerekkel. Az adatok kitűnő minőségűek voltak és rengeteg attribútumot tartalmaztak többek között az egyes személyek különböző médiumokhoz fűződő viszonyát, a személyek érdeklődési körét, életstílusát, anyagi helyzetét, demográfia adatait (lakásának, munkahelyének fekvése). Az elemzések során a következő kérdésekre keresték a választ:

– mely más újságot/magazint olvasnak még szívesen bizonyos nyomtatott médiumok olvasói,

– mi jellemző az olvasóira/hallgatóira/nézőire az egyes médiumoknak,

– milyen tulajdonságok különböztetik meg a különböző újságok olvasóit,

– az ügyfeleiket tekintve mely médiumok hasonlóak?

A kérdések megválaszolásához számos adatbányászati módszert használtak fel, csak úgy mint korreláció-elemzés, klaszterezés, döntési fák, asszociáció szabályok, Kohonen hálók. Például döntési fák segítségével próbálták megtudni, hogy jellemzően kik olvassák a ’Delo’ és a ’Slovenske Novice’ újságokat. A kinyert szabályokból két példa: „A Delo tipikus olvasója egy héten több alkalommal olvas újságot, az átlagnál magasabb az iskolai végzettsége, ismeri a különböző magazinokat, autó és sörmárkákat, szeret tv-zni, stb.” ezzel szemben a „Slovenske Novice olvasói szertenek kávézókban és bárokban időzni, kevésbé tájékozottak márkák ismeretében, mint a Delo olvasói, és általában olvassák még a Slovneski Delnicar, Jana, stb. magazinokat is.”

Klaszterezéssel profilcsoportokat hoztak létre. Kohonen háló segítségével megállapították a csoportok számát, majd a *k*-közép algoritmussal létrehoztak négy klasztert. A kapott klaszterek sajátosságait ezután döntési fák segítségével próbálták felderíteni. Az egyes csoportokat a jellemzők meghatározása után a „elhivatott fiatalok”, „inaktív idősek”, „ambiciózus emberek” és „aktív idősek” jelzőkkel illették. Például az „inaktív idősek”csoportját jellemzi, hogy nem szeretik a kihívásokat, nem érdekli őket a szórakoztatóipar, tudomány és technika, a fő örömforrásuk a család és nem szeretik a változásokat.

Az Egyesült Királyság baleseteinek elemzése

A baleseteket leíró adatbázisokból kinyert hasznos információk életeket menthetnek meg. Az okozatok, kapcsolatok feltárása olyan közúti vagy közlekedési szabályokat érintő módosításokhoz vezethet, amelyek segítségével megelőzhetők a balesetek anélkül, hogy az agyonszabályozások következtében megnehezítenék az autósok életét.

Az Egyesült Királyság adatbázisának elemzése egy nagyon sikeres adatbányászati projekt volt. Az adatbázis az 1979-1999-ig terjedő intervallum adatait, mintegy ötmillió rekordot tárolt. Minden rekordhoz tárolták a baleset körülményeit, az autó, ill. a vezető továbbá az esetleges sérülések adatait.

Az elemzéshez vizualizációs eszközöket, számos klasszikus statisztikai (pl. regresszió) és adatbányász módszert alkalmaztak. A balesetek körülményei szöveggel voltak megadva, ezért ezek feldolgozásában kiemelt szerepet kaptak a szövegbányász módszerek.

Egy érdekes és újszerű megoldás volt az földrajzi helyek klaszterezése, melynek során a hasonló baleseti dinamikával rendelkező helyek kerültek egy csoportba. Különböző időfelbontásokhoz (évi/heti/napi balesetszám alakulás) különböző klaszterezést készítettek. Észrevették például, hogy az olyan esetek amelyek az havi szinten emelkedő baleseti számmal rendelkeznek és a balesetek száma nyáron éri el a maximumot megegyeznek a közkedvelt turisztikai helyekkel. Ezeken a helyeken tehát csak a főszezonban szükséges emelni a biztonsági szintet. További fontos csoportot jellemzett az a görbe, amely napközben és hétvégén alacsony szinten volt, de a munkaidő utáni időszakban megugrott. Ez a görbe az „ipari”területekre volt jellemző.

Vegyük észre, hogy a közlekedési balesetekre nagyon jellemző a lokalitás vagy más szavakkal az ideiglenes gyakoriság. Ez azt jelenti, hogy összességében kevés baleset történik, viszont a kevés baleset nagy része ugyanabban az időben (például hóesésben, vagy munkaidő után) esik meg. Ezeket az eseteket naiv módon, gyakori mintákat kinyerő algoritmusokkal nem lehetne felderíteni, hiszen az összes baleset száma csekély.

A balesetek súlyosságának megállapítására felállított döntési fa is számos értékes összefüggéssel szolgált. Megmutatta például, hogy az 20 óra után történt motorkerékpárossal történt balesetekben a súlyos sérülések aránya jóval magasabb, mint általában.

Függelék

Függelék A

.1. tétel. *A Gyűjtőlapok és Tekintélyek során alkalmazott iteráció során $t^{(i)}$, illetve $g^{(i)}$ sorozatok konvergálnak nemnegatív értékű vektorokhoz. Tehát lássuk be, hogy amennyiben A egy tetszőleges gráf adjacencia mátrixa és $v^{(0)} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = j^t$, akkor a*

$$v^{(i)} = \frac{AA^T v^{(i-1)}}{[AA^T v^{(i-1)}]}$$

iteráció által kapott sorozat konvergál.

Megjegyzés 1: Az iterációs lépésből közvetlenül adódik, hogy $v^{(i)}$ az $(AA^T)^i j^t$ irányú egységvektor.
Megjegyzés 2: $g^{(i)}$ konvergenciájából $t^{(i)}$ konvergenciája is következik A és A^T felcserélésével.
A tétel bizonyításához szükségünk van néhány segédtételre.

.2. lemma. *Legyen $A \in \mathbb{R}(n \times n)$. Ekkor AA^T (és hasonlóan $A^T A$ is) pozitív szemidefinit szimmetrikus mátrix.*

Bizonyítás: A szimmetrikusság a mátrixszorzás szabályából közvetlenül adódik. Felhasználva a $vA = (A^T v^T)^T$ azonosságot

$$vAA^T v^T = (A^T v^T)^T (A^T v^T) = w^T w \geq 0$$

adódik, ami bizonyítja, hogy AA^T pozitív szemidefinit.

■

.3. lemma. *Ha M mátrix pozitív szemidefinit és szimmetrikus, akkor sajátértékei valósak és nemnegatívak.*

.4. tétel (Perron-Frobenius). *Ha egy mátrix aperiodikus, irreducibilis és nemnegatív elemű, akkor legnagyobb abszolútértékű sajátértékhez tartozó sajátvektor nemnegatív koordinatajú, és nincs más, ilyen abszolút értekű, sajátérték.*

.5. lemma. *M mátrix pozitív szemidefinit szimmetrikus, $\lambda_1 > \lambda_2 \geq \ldots \geq \lambda_k \geq 0$, $(k < n)$ sajátértékekkel. Ekkor tetszőleges $v \in \mathbb{R}^n$ felírható $v = \sum_{i=1}^{k} \alpha_i w^{(i)}$ alakban, ahol $||w^{(i)}|| = 1, w^{(i)} w^{(j)} = 0$ ha $i \neq j$ és $Mw^{(i)} = \lambda_i w^{(i)}$.*

Térjünk vissza az .1-ös tétel bizonyításához.

Bizonyítás:

Jelöljük AA^T mátrixot M-el. Feltehetjük, hogy M aperiodikus, hiszen m_{ii} az i-edik pontból más pontba mutató élszám négyzetének összegét adja meg ($\sum_k m_{ik}^2$), ami csak akkor lehet 0, ha i-edik pontból nem indul él. Ez a pont a konvergencia tényét nem befolyásolja, mert M minden hatványának megfelelő sora és oszlopa csupa 0 elemből fog állni, tehát jogos a feltételezés. Azt is feltehetjük, hogy M irreducibilis, mert ha nem az, akkor mátrixot irreducibilis blokkmátrixokra bonthatjuk, és a hatványozást blokkonként végezhetjük.

Tudjuk tehát, hogy M nemnegatív elemű, aperiodikus, irreducibilis, pozitív szemidefinit szimmetrikus mátrix, ami miatt minden sajtátérték nemnegatív, a legnagyobb sajátértéke egyszeres, továbbá az ehhez tartozó sajátvektor nemnegatív elemű. Legyen $v \in \mathbb{R}^n$ tetszőleges vektor. .5 alapján $v = \sum_{i=1}^k \alpha_i w^{(i)}$ és $w^{(1)}$ egyértelmű, nemnegatív elemű vektor. A $\frac{M^j v}{||M^j v||}$ kifejezés $w^{(1)}$-hez tart ha $j \to \infty$, mert

$$\frac{M^j v}{||M^j v||} = \frac{\sum_{i=1}^k \alpha_i M^j w^{(i)}}{||\sum_{i=1}^k \alpha_i M^j w^{(i)}||} = \frac{\sum_{i=1}^k \alpha_i \lambda_i^j w^{(i)}}{\sqrt{\sum_{i=1}^k (\alpha_i \lambda_i^j)^2}}$$

$$\frac{\alpha_1 \lambda_1^j w^{(1)} + \sum_{i=2}^k \alpha_i \lambda_i^j w^{(i)}}{\sqrt{(\alpha_1 \lambda_1^j)^2 + \sum_{i=1}^k (\alpha_i \lambda_i^j)^2}} \cdot \frac{\frac{1}{\lambda_1^j}}{\frac{1}{\lambda_1^j}} = \frac{\alpha_1 w^{(1)} + \sum_{i=2}^k \alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^j w^{(i)}}{\sqrt{\alpha_1^2 + \sum_{i=1}^k (\alpha_i \left(\frac{\lambda_i}{\lambda_1}\right)^j)^2}} \to w^{(1)}$$

A normálás során felhasználtuk, hogy a $w^{(i)}$ vektorok merőlegesek egymásra, és egységnyi hosszúak, a határérték meghatározásakor pedig azt, hogy λ_1 a legnagyobb sajátérték, tehát $\frac{\lambda_i}{\lambda_1} < 1, i = 2, \ldots, k$-ra. Tehát ha v nem merőleges $w^{(1)}$-re, akkor $\frac{M^j v}{||M^j v||}$ vektor $w^{(1)}$-hez konvergál. Ez azonban nem áll fenn, lévén $jw^{(1)} > 0$, mert $w^{(1)}$ nemnegatív elemű vektor.

■

ANGOL	MAGYAR
antecedent	feltételrész
approximate dependency	közelítő függőség
association rule	asszociációs szabály
authority	tekintélylap
basket	kosár
candidate	jelölt
classification	osztályozás
consequent	következményrész
clustering	klaszterezés
confidence	bizonyosság
conviction	meggyőződés
data mining	adatbányászat
dead end problem	zsákutca probléma
dense	sűrű
episode	epizód
false-positive	hamis jelölt
false-negative	hiányzó elem
frequent	gyakori
hash-tree	hash-fa
hub	gyűjtőlap
item	elem
knowledge retrieval	tudásfeltárás
kurtosis	lapultság
levelwise	szintenként haladó
lift	függetlenségi mutató
locality-sensitive hashing (LSH)	hely-érzékeny hashelés (HÉH)
market-basket problem	piaci-kosár probléma
negative border	esélyes jelölt
outlier analysis	eltérés elemzés
pattern	minta
product	termék
ranking	rangsorolás
sequence matching	sorozatillesztés
signature	lenyomat
skewness	ferdeség
sparse	ritka
spider trap problem	pókháló probléma
stripped partition	redukált partíció
support	támogatottság
threshold	küszöb
transaction	tranzakció
valid	érvényes
z-score normalization	standard normalizálás

1. táblázat. Idegen kifejezések fordítása

Tárgymutató

χ^2 eloszlás, 23
χ^2 próba, 24
11 pontos átlagos pontosság, 169

A minta nagysága, 79
AdaBoost, 168
adat
    strukturálatlan, 158
    strukturált, 158
    tanuló, 164
    teszt, 164
    validációs, 164
adatbázis
    horizontális, 63
    vertikális, 63
algoritmus
    helyesen működő, 47
    mohó, 170
    teljes, 47
anti-monoton, 42
APRIORI, 64
    módszer, 47
apriori algoritmus, 189
APRIORI-CLOSE, 50
APRIORI-HIBRID, 70
APRIORI-TID, 69
asszociációs szabály, 110
    érdekessége, 115
    érvényes, 111
    bizonyossága, 110
    egzakt, 111
    hierarchikus, 111, 112
    javítási mutató, 116
    szomszédosság alapú érdekességi mutató, 118
    támogatottsága, 110
average linkage, 153

Bayes-módszer
    naiv, 165, 170
bizottság
    osztályozóké, 167
    tagok, 167
boosting eljárások
    AdaBoost, 168

centroid kapcsolódás, 175
centroid–egyszerű kapcsolódás, 175
χ^2-statisztika, 162
complet linkage, 153
csoportosítás
    szövegeké, 173
        hierarchikus klaszterezők, 175
        jellegzetességek, 174
        k-átlag módszerek, 176

DHP, 73
dimenzió csökkentése, 161
    kategorizálásnál, 162
Direct Hashing and Pruning, 73
dokumentum
    ábrázolása, 160
    előfeldolgozása, 159
    reprezentációja, 159
        bináris, 161
        csoportosításnál, 174
dokumentum frekvencia küszöbölő, 162
dokumentumgyűjtemény
    reprezentálása, 160
dokumentumok csoportosítása, 173
dokumentumok előfeldolgozása, 159

döntési fa
szövegosztályozó, 166
Duquenne–Guigues-bázis, 114

ECLAT algoritmus, 74
egyensúlyi pont
felidézésé és pontosságé, 168
ekvivalencia-reláció, 19
elemhalmaz, 62, 86
entrópia, 22
entrópia, 175
Euklideszi-norma, 33

függetlenségvizsgálat, 24
felügyelet nélküli tanulás, 140
felidézés, 168, 179
szintenkénti, 171
felügyelet nélküli tanulás, 173
felügyelt tanulás, 164
ferdeség, 22
F-mérték, 169
csoportosításnál, 175
szintenkénti, 171
fogalomtársítás, 187
fontosság, 193
FP-fa, 76
vetített, 76
FP-growth algoritmus, 76
funkció szavak, 162
funkció szavak
elhagyása, 162
funkcionális függőség, 123
FUP algoritmus, 59

Galois-kapcsolat, 80
Galois-lezárás operátor, 81
GSP, 89
gyűjtőlap, 197
gyakorisági küszöb, 50
gyakoriság, 161

halmaz, 19
lokálisan véges, 42
rangszámozott, 42
halmazcsalád, 71
hatékonyság mérése
szövegbányászatnál általában, 162
szövegek csoportosításánál, 174
szövegosztályozás
egyszerű, 168
hierarchikus, 171
hiba
szövegosztályozásnál, 168
hierarchikus asszociációs szabály
érdekessége, 120
hierarchikus klaszterező, 175
egyesítő, 175
felosztó, 176
ierarchikus klaszterező
UPGMA, 175
hierarchikus szövegosztályozás, 170
HITEC, 171
Hoeffding-korlát, 22

információ nyereség módszer, 162
információkinyerés, 187
invariáns hasonlóság, 32
inverz dokumentum frekvencia, 161

Jaccard-koefficiens, 32
jelölt, 47
hamis, 47
jellemzők kiválasztása, 161

k-legközelebbi szomszéd gráf, 144
közelítő függőség, 123
kényszer
erősen átalakítható, 45
kanonikus reprezentáció, 70
kategóriaösvény, 171
kategóriarendszer, 163
kategorizálás *lásd* osztályozás 163
k-átlag eljárás
kettészelő, 176
kérdés-megválaszoló rendszerek, 188
kettészelő k-átlag eljárás, 176
kivonatolás, 177
csoportosítás alapú módszerek, 182
definíció, 178
hatékonyságának mérése, 179
jellemzők, 180
klasszikus módszer, 181
MEAD módszer, 183
MMR módszer, 182

mondatkiválasztással, 180
TF-IDF alapú módszer, 182
weboldalaké, 185
Klaszterezés, 139
klaszterezés *lásd* csoportosítás 173
k-NN *lásd* legközelebbi szomszédok 165
kontingencia-táblázat, 24
koszinusz-mérték, 34
kulcs, 125

lapultság, 22
látens szemantikus indexelés (LSI), 162
legközelebbi szomszédok
szövegosztályozó, 165
lexikografikus rendezés, 19
lexikon *lásd* szótár 160
lineáris kiterjesztés, 43
lineáris osztályozó, 164
LSI *lásd* látens szemantikus indexelés 162
lusta tanuló, 165

Manhattan-norma, 33
min_freq, 50
Minkowski-norma, 33
minta, 42
üres, 42
elhanyagolt, 47
gyakori, 42
gyakorisága, 50
jelölt, 47
mérete, 42
nem bővíthető, 44
ritka, 42
támogatottsága, 42
zárt, 44
mintafelismerés, 161
mintahalmaz, 42
mintatér, 42
mohó algoritmus, 170

névelem, 189
naiv Bayes-módszer, 165
hierarchikus osztályozás, 170
Naiv mintavételező algoritmus, 56
neurális hálózat, 166

oldalak rangsorolása, 192
„oszd meg és uralkodj” stratégia, 166
osztályozás
egyszerű, 163
hierarchikus, 163
szövegeké, 163
osztályozó
lineáris, 164
összegzéskészítés, 178
általános, 179
indikatív, 179
informatív, 179
kérdés-vezérelt, 179

pókháló probléma, 195
Page Rank, 193, 196
partíció, 124
partíció finomítása, 124
partíciós algoritmus, 57
PATRICIA fa, 27
perceptron, 166
pontosság, 168, 179
szintenkénti, 171
Porter-algoritmus, 189
predikátum
anti-monoton, 45
monoton, 45
prefix anti-monoton, 45
prefix monoton, 45
triviális, 45
prefix, 43
prototípusvektor, 164
pszeudo-zárt elemhalmaz, 114

részben rendezés, 19
részminta, 42
valódi, 42
rang-vektor, 193
redukált partíció, 125
Reuters-gyűjtemény, 170
Rocchio-eljárás, 164

SETM, 69
shrinkage, 170
single linkage eljárás, 153
sorozat, 20
stopwords, 162
strukturálatlan adat, 158

strukturált adat, 158
súlybeállítás
  additív, 166
  multiplikatív, 166
súlyozás
  bináris, 161
  TF, 161
  TFIDF, 161, 182
SVD *lásd* szinguláris értékfelbontás 162
SVM, 167
szófa, 25, 67
  láncolt listás implementáció, 26
  nyesett, 27
  táblázatos implementáció, 26
szabatosság, 168
szavazásos osztályozás, 167
szerkesztési távolság, 34
szerkesztési elv, 182
szeszélyes sztochasztikus szörfölő, 196
szinguláris értékfelbontás (SVD), 162, 184
szó–dokumentum mátrix, 160
szófajcímkéző, 189
szótár, 160
  mérete, 160
  méretének csökkentése, 161
    kategorizálásnál, 162
szótövező, 160, 189
szövegbányászat
  általános modellje, 159
  definíció, 158
szövegek kategorizálása, 163
szöveges információk vizualizálása, 188
szövegosztályozás, 163
  hierarchikus, 170
szövegosztályozó
  bizottság, 167
  döntési fa alapú, 166
  HITEC, 171
  legközelebbi szomszédokon alapuló, 165
  naiv Bayes-módszer, 165, 170
  neurális hálózat alapú, 166
  Rocchio-eljárás, 164
  SVM, 167
  szavazásos, 167
sztochasztikus szörfölő, 194
szuperkulcs, 125

támogatottsági függvény, 42
támogatottsági küszöb, 42
TANE, 124
tanulás
  felügyelet nélküli, 173
  felügyelt, 164
tanulási ráta, 166
tanulóhalmaz, 164
taxonómia, 112
taxonómia, 163, 170, 173
tekintélylapok, 197
teljes rendezés, 19
témakövetés, 187
teszthalmaz, 164
tesztkorpuszok
  szövegszo2vegklaszterezeshez, 177
  szövegosztályozáshoz, 190
token, 160

újraparametrizálás, 162
univerzálisan népszerű lapok, 198
UPGMA módszer, 175

validációs halmaz, 164
variáns hasonlóság, 32
vektortér-modell, 160

Ward módszer, 153
Webes adatbányászat, 192
Winnow, 166
  kiegyensúlyozott, 166

zárt elemhalmaz, 81
zsákutca probléma, 195

Irodalomjegyzék

[1] L. Aas – L. Eikvil: Text categorisation: A survey. NR 941. Raport, 1999, Norwegian Computing Center.

[2] Pieter Adriaans – Dolf Zantinge: *Adatbányászat*. Budapest, 2002, Panem Kiadó.

[3] Ramesh C. Agarwal – Charu C. Aggarwal – V. V. V. Prasad: A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing*, 61. évf. (2001) 3. sz. URL `http://citeseer.nj.nec.com/agarwal99tree.html`.

[4] R. Agrawal – R. Srikant: Fast algorithms for mining association rules in large databases. In *Proc. of VLDB 94, the 20th Int. Conf. on Very Large Data Bases* (konferenciaanyag). Santiago de Chile, Chile, 1994, 487–499. p.

[5] Rakesh Agrawal – Tomasz Imielinski – Arun N. Swami: Mining association rules between sets of items in large databases. In Peter Buneman – Sushil Jajodia (szerk.): *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (konferenciaanyag). Washington, D.C., 1993. 26-28, 207–216. p.
URL `http://citeseer.nj.nec.com/agrawal93mining.html`.

[6] Rakesh Agrawal – Heikki Mannila – Ramakrishnan Srikant – Hannu Toivonen – A. Inkeri Verkamo: Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining* (konferenciaanyag). 1996, 307–328. p.

[7] Rakesh Agrawal – Ramakrishnan Srikant: Fast algorithms for mining association rules. In Jorge B. Bocca – Matthias Jarke – Carlo Zaniolo (szerk.): *Proceedings of the 20th International Conference Very Large Data Bases, VLDB* (konferenciaanyag). 1994. 12-15, Morgan Kaufmann, 487–499. p. ISBN 1-55860-153-8.
URL `http://citeseer.nj.nec.com/agrawal94fast.html`.

[8] Rakesh Agrawal – Ramakrishnan Srikant: Mining sequential patterns. In Philip S. Yu – Arbee L. P. Chen (szerk.): *Proceedings of the 11th International Conference on Data Engineering, ICDE* (konferenciaanyag). 1995. 6-10, IEEE Computer Society, 3–14. p. ISBN 0-8186-6910-1. URL `http://citeseer.nj.nec.com/agrawal95mining.html`.

[9] Rényi Alfréd: *Valószínűségszámítás*. 1968, Tankönyvkiadó.

[10] Brian Amento – Loren G. Terveen – William C. Hill: Does „authority” mean quality? predicting expert quality ratings of web documents. In *Research and Development in Information Retrieval* (konferenciaanyag). 2000, 296–303. p.
URL `http://citeseer.nj.nec.com/417258.html`.

[11] Amihood Amir – Ronen Feldman – Reuven Kashi: A new and versatile method for association generation. In *Principles of Data Mining and Knowledge Discovery* (konferenciaanyag). 1997, 221–231. p. URL `http://citeseer.nj.nec.com/amir97new.html`.

[12] C. Apte – F. J. Damerau – S. M. Weiss: Automated learning of decision rules for text categorization. *ACM Trans. Information Systems*, 12. évf. (1994. July) 3. sz.

[13] Necip Fazil Ayan – Abdullah Uz Tansel – M. Erol Arkun: An efficient algorithm to update large itemsets with early pruning. In *Knowledge Discovery and Data Mining* (konferenciaanyag). 1999, 287–291. p. URL `http://citeseer.nj.nec.com/ayan99efficient.html`.

[14] R. Bayardo – R. Agrawal – D. Gunopulos:. Constraint-based rule mining in large, dense databases, 1999. URL `http://citeseer.nj.nec.com/bayardo99constraintbased.html`.

[15] B. Berendt – B. Mobasher – M. Spiliopoulou – J. Wiltshire:. Measuring the accuracy of sessionizers for web usage analysis, 2001.
URL `http://citeseer.nj.nec.com/berendt01measuring.html`.

[16] M. W. Berry – S. T. Dumais – G. W. O'Brien: Using linear algebra for intelligent information retrieval. *SIAM Review*, 37. évf. (1995) 4. sz.

[17] Krishna Bharat – Monika Rauch Henzinger: Improved algorithms for topic distillation in a hyperlinked environment. In *Research and Development in Information Retrieval* (konferenciaanyag). 1998, 104–111. p.
URL `http://citeseer.nj.nec.com/bharat98improved.html`.

[18] R. Blumberg – S. Arte: The problem with unstructured data. *DM Review*, 2003. February. `http://www.dmreview.com/editorial/dmreview/print_action.cfm?articleId=6%287`.

[19] Ferenc Bodon: A fast apriori implementation. In Bart Goethals – Mohammed J. Zaki (szerk.): *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, CEUR Workshop Proceedings konferenciasorozat, 90. köt. Melbourne, Florida, USA, 2003. November 19..

[20] Richard J. Bolton – David J. Hand: Significance tests for patterns in continuous data. In *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDE)* (konferenciaanyag). 2001.

[21] Christian Borgelt – Rudolf Kruse: Induction of association rules: Apriori implementation. In *Proceedings of the 15th Conference on Computational Statistics (Compstat 2002, Berlin, Germany)* (konferenciaanyag). Heidelberg, Germany, 2002, Physika Verlag.

[22] L. Breiman: Bagging predictors. *Machine Learning*, 24. évf. (1996).

[23] Sergey Brin – Rajeev Motwani – Jeffrey D. Ullman – Shalom Tsur: Dynamic itemset counting and implication rules for market basket data. *SIGMOD Record (ACM Special Interest Group on Management of Data), 26(2):255*, 1997.

[24] Sergey Brin – Lawrence Page: The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30. évf. (1998) 1–7. sz.
URL `http://citeseer.nj.nec.com/brin98anatomy.html`.

[25] Richárd Bugnics: *Bevezetés az ökonometriába előadásvázlatok*. 1999, BKÁE.

[26] Douglas Burdick – Manuel Calimlim – Johannes Gehrke: Mafia: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th International Conference on Data Engineering* (konferenciaanyag). Heidelberg, Germany, 2001, IEEE Computer Society, 443–452. p. ISBN 0-7695-1001-9.

[27] S. Chakrabarti – B. Dom – R. Agrawal – P. Raghavan: Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *The VLDB Journal*, 7. évf. (1998) 3. sz.

[28] Soumen Chakrabarti – Byron Dom – Prabhakar Raghavan – Sridhar Rajagopalan – David Gibson – Jon Kleinberg: Automatic resource compilation by analyzing hyperlink structure and associated text. *Computer Networks and ISDN Systems*, 30. évf. (1998) 1–7. sz.
URL `http://citeseer.nj.nec.com/chakrabarti98automatic.html`.

[29] Pete Chapman – Julian Clinton – Randy Kerber – Thomas Khabaza Thomas Reinartz – Colin Shearer – Rüdiger Wirth: Cross industry standard process for data mining (crisp-dm) – step by step data mining guide. Jelentés, 1999.

[30] David Wai-Lok Cheung – Jiawei Han – Vincent Ng – C. Y. Wong: Maintenance of discovered association rules in large databases: An incremental updating technique. In *ICDE* (konferenciaanyag). 1996, 106–114. p.
URL `http://citeseer.nj.nec.com/cheung96maintenance.html`.

[31] David Wai-Lok Cheung – Sau Dan Lee – Ben Kao: A general incremental technique for maintaining discovered association rules. In *Database Systems for Advanced Applications* (konferenciaanyag). 1997, 185–194. p.
URL `http://citeseer.nj.nec.com/cheung97general.html`.

[32] Robert Cooley – Bamshad Mobasher – Jaideep Srivastava: Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems*, 1. évf. (1999) 1. sz.
URL `http://citeseer.nj.nec.com/cooley99data.html`.

[33] Thomas M. Cover – Joy A. Thomas: *Elements of Information Theory*. Wiley Series in Telecommunications sorozat. 1991, John Wiley & Sons, Inc.

[34] I. Dagan – Y. Karov – D. Roth: Mistake-driven learning in text categorization. In Claire Cardie – Ralph Weischedel (szerk.): *Proc. of EMNLP-97, 2nd Conference on Empirical Methods in Natural Language Processing* (konferenciaanyag). Providence, RI, 1997, Association for Computational Linguistics, 55–63. p.

[35] S. D'Alessio – K. Murray – R. Schiaffino – A. Kershenbaum: The effect of using hierarchical classifiers in text categorization. In *Proc. of 6th Int. Conf. Recherche d'Information Assistee par Ordinateur (RIAO-00)* (konferenciaanyag). Paris, France, 2000, 302–313. p. `http://citeseer.ist.psu.edu/410559.html`; retrieved on 2005.08.26.

[36] R. de la Briandais: File searching using variable-length keys. In *Western Joint Computer Conference* (konferenciaanyag). 1959. March, 295–298. p.

[37] S. Deerwester – S. T. Dumais – G. W. Furnas – T. K. Landauer – R. Harshman: Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41. évf. (1990) 6. sz.

[38] P. Domingos – M. J. Pazzani: On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning*, 29. évf. (1997) 2–3. sz.

[39] Guozhu Dong – Jinyan Li: Interestingness of discovered association rules in terms of neighborhood-based unexpectedness. In Xindong Wu – Kotagiri Ramamohanarao – Kevin B. Korb (szerk.): *Research and Development in Knowledge Discovery and Data Mining, Proceedings of the 2nd Pacific-Asia Conference Knowledge Discovery and Data Mining, PAKDD* (konferenciaanyag), 1394. köt. 1998. 15-17, Springer, 72–86. p.
URL `http://citeseer.nj.nec.com/dong98interestingness.html`.

[40] S. T. Dumais: Improving the retrieval information from external sources. *Behaviour Research Methods, Instruments and Computers*, 23. évf. (1991) 2. sz.

[41] S. T. Dumais – J. Platt – D. Heckerman – M. Sahami: Inductive learning algorithms and representations for text categorization. In *Proc. of 7th ACM Int. Conf. on Information and Knowledge Management (CIKM-98)* (konferenciaanyag). Bethesda, MD, 1998, 148–155. p.

[42] Herb Edelstein: Mining large databases – a case study. Jelentés, 1999, Two Crows Corporation.

[43] M. Ester – H.-P. Kriegel – X. Xu.: A database interface for clustering in large spatial databases. In *Proceedings of the Knowledge Discovery and Data Mining Conference, Montreal, Canada* (konferenciaanyag). 1995, 94–99. p.

[44] Martin Ester – Hans-Peter Kriegel – Jorg Sander – Xiaowei Xu: A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis – Jiawei Han – Usama Fayyad (szerk.): *Second International Conference on Knowledge Discovery and Data Mining* (konferenciaanyag). Portland, Oregon, 1996, AAAI Press, 226–231. p.
URL `http://citeseer.nj.nec.com/chu02incremental.html`.

[45] C. J. Fall – A. Törcsvári – P. Fievét – G. Karetka:. Additional readme information for WIPO-de autocategorization data set, 2003. March. `http://www.wipo.int/ibis/datasets/wipo-de-readme.html`.

[46] C. J. Fall – A. Törcsvári – G. Karetka:. Readme information for WIPO-alpha autocategorization training set, 2002. December. `http://www.wipo.int/ibis/datasets/wipo-alpha-readme.html`.

[47] C. J. Fall – A. Törcsvári – K. Benzineb – G. Karetka: Automated categorization in the international patent classification. *ACM SIGIR Forum archive*, 37. évf. (2003. Spring) 1. sz.

[48] W. Fan – L. Wallace – S. Rich – Z. Zhang: Tapping into the power of text mining. *Communications of the ACM*, (in press). évf. (2005). `http://filebox.vt.edu/users/wfan/paper/text_mining_final_preprint.pdf`.

[49] Usama M. Fayyad – Gregory Piatetsky-Shapiro – Padhraic Smyth: From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*. 1996, AAAI Press/The MIT Pres, 1–34. p.

[50] William Feller: *Bevezetés a Valószínűségszámításba és Alkalmazásaiba*. 1978, Műszaki Könyvkiadó.

[51] Bodon Ferenc: Hash-fák és szófák az adatbányászatban. *Alkalmazott Matematikai Lapok*, 21. évf. (2003).

[52] E. W. Forgy: Cluster analysis of multivariate data: Efficiency versus interpretability of classifications. *Biometric Soc. Meetings, Riverside, California*, 21. évf. (1965).

[53] Scott Fortin – Ling Liu: An object-oriented approach to multi-level association rule mining. In *CIKM* (konferenciaanyag). 1996, 65–72. p.

[54] Edward Fredkin: Trie memory. *Communications of the ACM*, 3. évf. (1960) 9. sz. ISSN 0001-0782.

[55] Y. Fu:. Discovery of multiple-level rules from large databases, 1996. URL `http://citeseer.nj.nec.com/fu96discovery.html`.

[56] Iván Futó (szerk.): *Mesterséges Intelligencia*. Budapest, 1999, Aula Kiadó.

[57] T. Gedeon – L. T. Kóczy: A model of intelligent information retrieval using fuzzy tolerance relations based on hierarchical co-occurrence of words. In F. Crestani – G. Pasi (szerk.): *Soft Computing in Information Retrieval: Techniques and Applications*. Studies in Fuzziness and Soft Computing sorozat, 50. köt. Heidelberg, Germany, 2000, Physica-Verlag, 48–74. p.

[58] Bart Goethals – Mohammed J. Zaki: Advances in frequent itemset mining implementations: Introduction to fimi03. In Bart Goethals – Mohammed J. Zaki (szerk.): *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, CEUR Workshop Proceedings konferenciasorozat, 90. köt. Melbourne, Florida, USA, 2003. November 19..

[59] M. D. Gordon – R. Lindsay – W. Fan: Literature-based discovery on the www. *ACM Transactions on Internet Technology (TOIT)*, 2. évf. (2002) 4. sz.

[60] Gosta Grahne – Jianfei Zhu: Efficiently using prefix-trees in mining frequent itemsets. In Bart Goethals – Mohammed J. Zaki (szerk.): *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations (FIMI'03)*, CEUR Workshop Proceedings konferenciasorozat, 90. köt. Melbourne, Florida, USA, 2003. November 19..

[61] Sudipto Guha – Rajeev Rastogi – Kyuseok Shim: CURE: an efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on Management of Data* (konferenciaanyag). 1998. June, 73–84. p.
URL `http://citeseer.nj.nec.com/article/guha98cure.html`.

[62] J. Han – Y. Fu: Discovery of multiple-level association rules from large databases. *Proceedings of the 21st International Conference on Very Large Databases (VLDB), Zurich, Switzerland*, 1995.

[63] Jiawei Han – Micheline Kamber: *Data mining: concepts and techniques*. 2001, Morgan Kaufmann Publisher.

[64] Jiawei Han – Jian Pei – Yiwen Yin: Mining frequent patterns without candidate generation. In Weidong Chen – Jeffrey Naughton – Philip A. Bernstein (szerk.): *2000 ACM SIGMOD International Conference on Management of Data* (konferenciaanyag). 2000. 05, ACM Press, 1–12. p. ISBN 1-58113-218-2. URL `http://citeseer.nj.nec.com/han99mining.html`.

[65] K. Hatonen – Mika Klemettinen – Heikki Mannila – P. Ronkainen – Hannu Toivonen: Knowledge discovery from telecommunication network alarm databases. In Stanley Y. W. Su (szerk.): *Proceedings of the twelfth International Conference on Data Engineering, February 26–March 1, 1996, New Orleans, Louisiana* (konferenciaanyag). 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1996, IEEE Computer Society Press, 115–122. p.
URL `http://citeseer.nj.nec.com/hatonen96knowledge.html`.

[66] Maurice Houtsma – Arun Swami:. Set-oriented mining of association rules, 1993.

[67] Yka Huhtala – Juha Kinen – Pasi Porkka – Hannu Toivonen: Efficient discovery of functional and approximate dependencies using partitions. In *ICDE* (konferenciaanyag). 1998, 392–401. p. URL `http://citeseer.nj.nec.com/huhtala97efficient.html`.

[68] Ykä Huhtala – Juha Kärkkäinen – Pasi Porkka – Hannu Toivonen: TANE: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42. évf. (1999) 2. sz. URL `http://citeseer.nj.nec.com/huhtala99tane.html`.

[69] D. A. Hull: Improving text retrieval for the routing problem using latent semantic indexing. In *Proc. of SIGIR-94, 17th ACM Int. Conf. on Research and Development in Information Retrieval* (konferenciaanyag). Dublin, Ireland, 1994, 282–289. p.

[70] Akihiro Inokuchi – Takashi Washio – Hiroshi Motoda: An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery* (konferenciaanyag). 2000, Springer-Verlag, 13–23. p. ISBN 3-540-41066-X.

[71] Akihiro Inokuchi – Takashi Washio – Nishimura Yoshio – Hiroshi Motoda: A fast algorithm for mining frequent connected graphs,. Jelentés, 2002, IBM research, Tokyo Research Laboratory.

[72] Korcsmáros István:. Szövegbányászat (text mining) — új fogalom az üzleti intelligencia témakörében. `http://www.controllingportal.hu/index.php?doc=tk_t&t=16&d=75`, 2003.

[73] Fazekas István: *Bevezetés a matematikai statisztikába*. 2000, Debreceni Egyetem Kossuth Egyetemi Kiadója.

[74] R. C. Jancey: Multidimensional group analysis. *Austral. J. Botany*, 14. évf. (1966).

[75] T. Joachims: A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *Proc. of ICML-97, 14th Int. Conf. on Machine Learning* (konferenciaanyag). Nashville, TN, USA, 1997, 143–151. p.

[76] T. Joachims: Text categorization with support vector machines: Learning with many relevant features. Technical Report, Dortmund, Germany, 1997, University of Dortmund, Dept. of Informatics.

[77] Richard A. Johnson – Dean W. Wichern: *Applied Multivariate Statistical Analysis*. Fifth. kiad. Upper Saddle River, NJ, 2002, Prentice-Hall.

[78] Ravi Kannan – Santosh Vempala – Adrian Vetta: On clusterings: Good, bad and spectral. In *Proceedings of the 41th Annual Symposium on Fundations of Computer Science* (konferenciaanyag). 2000. URL `http://citeseer.nj.nec.com/495691.html`.

[79] O. Kariv – S.L.Hakimi: An algorithmic approach to network location problems, part ii: p-medians. *SIAM J. Appl. Math.*, 37. évf. (1979).

[80] L. Kaufman – P.J. Rousseeuw: *Finding Groups in Data: an Introduction to Cluster Analysis*. 1990, John Wiley & Sons.

[81] Jon Kleinberg: An impossibility theorem for clustering. *Advances in Neural Information Processing Systems (NIPS) 15*, 2002. URL `http://citeseer.nj.nec.com/561287.html`.

[82] Jon M. Kleinberg: Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46. évf. (1999) 5. sz.
URL `http://citeseer.nj.nec.com/kleinberg97authoritative.html`.

[83] Mika Klemettinen:. A knowledge discovery methodology for telecommunication network alarm databases, 1999.
URL `http://citeseer.nj.nec.com/klemettinen99knowledge.html`.

[84] Ron Kohavi: Mining e-commerce data: The good, the bad, and the ugly. In Foster Provost – Ramakrishnan Srikant (szerk.): *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (konferenciaanyag). 2001, 8–13. p.
URL `http://citeseer.nj.nec.com/kohavi01mining.html`.

[85] D. Koller – M. Sahami: Hierarchically classifying documents using a very few words. In *Proc. of ICML-97, 14th Int. Conf. on Machine Learning* (konferenciaanyag). Nashville, TN, 1997, 170–178. p.

[86] Michihiro Kuramochi – George Karypis: Frequent subgraph discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining* (konferenciaanyag). 2001, IEEE Computer Society, 313–320. p. ISBN 0-7695-1119-8.

[87] Rónyai Lajos – Ivanyos Gábor – Szabó Réka: *Algoritmusok*. 1998, Typotex Kiadó.

[88] Nada Lavrac – Dragan Gamberger – Hendrik Blockeel – Ljupco Todorovski (szerk.). *ExAnte: Anticipated Data Reduction in Constrained Pattern Mining*, Lecture Notes in Computer Science konferenciasorozat, 2838. köt. Springer, 2003. ISBN 3-540-20085-1.

[89] Wenke Lee – Salvatore Stolfo: Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium* (konferenciaanyag). San Antonio, TX, 1998. URL `http://citeseer.nj.nec.com/article/lee00data.html`.

[90] Wenke Lee – Salvatore J. Stolfo: A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and System Security*, 3. évf. (2000) 4. sz. URL `http://citeseer.nj.nec.com/article/lee00framework.html`.

[91] Wenke Lee – Salvatore J. Stolfo – Kui W. Mok: A data mining framework for building intrusion detection models. In *IEEE Symposium on Security and Privacy* (konferenciaanyag). 1999, 120–132. p. URL `http://citeseer.nj.nec.com/article/lee99data.html`.

[92] R. Lempel – S. Moran: The stochastic approach for link-structure analysis (SALSA) and the TKC effect. In *WWW9* (konferenciaanyag). 2000. URL `http://citeseer.nj.nec.com/346353.html`.

[93] D. D. Lewis: An evaluation of phrasal and clustered representations on a text categorization task. In *Proc. of SIGIR-92, 15th ACM Int. Conf. on Research and Development in Information Retrieval* (konferenciaanyag). Copenhagen, Denmark, 1992, 37–50. p.

[94] D. D. Lewis: Naive (Bayes) at forty: The independence assumption in information retrieval. In *Proc. of ECML-98, 10th European Conference on Machine Learning* (konferenciaanyag). Chemnitz, Germany, 1998, 4–15. p.

[95] Y. H. Li – A. K. Jain: Classification of text documents. *Comput. J.*, 41. évf. (1998) 8. sz.

[96] Heikki Mannila – Hannu Toivonen: Discovering generalized episodes using minimal occurrences. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD'96)* (konferenciaanyag). 1996. August, AAAI Press, 146–151. p. URL `http://citeseer.nj.nec.com/mannila96discovering.html`.

[97] Heikki Mannila – Hannu Toivonen – A. Inkeri Verkamo: Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)* (konferenciaanyag). 1995. August, AAAI Press, 210–215. p.

[98] Heikki Mannila – Hannu Toivonen – A. Inkeri Verkamo: Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1. évf. (1997) 3. sz. ISSN 1384-5810. URL `http://citeseer.nj.nec.com/mannila97discovery.html`.

[99] Heikki Mannila – Hannu Toivonen – A. Inkeri Verkamo: Efficient algorithms for discovering association rules. In Usama M. Fayyad – Ramasamy Uthurusamy (szerk.): *AAAI Workshop on Knowledge Discovery in Databases(KDD-94)* (konferenciaanyag). Seattle, Washington, 1994, AAAI Press, 181–192. p. URL `http://citeseer.nj.nec.com/mannila94efficient.html`.

[100] A. McCallum – R. Rosenfeld – T. Mitchell – A. Ng: Improving text classification by shrinkage in a hierarchy of classes. In *Proc. of ICML-98, 15th Int. Conf. on Machine Learning* (konferenciaanyag). Madison, US, 1998, 359–367. p. `http://citeseer.ist.psu.edu/mccallum98improving.html`.

[101] Brendan D. McKay: Practical graph isomorphism. *Congressus Numerantium*, 30. évf. (1981). URL `http://cs.anu.edu.au/people/bdm/nauty/`.

[102] N. Megiddo – K.Supowitz: On the complexity of some common geometric location problems. *SIAM J. Comput.*, 1984.

[103] Jesus Mena: *Data Mining und E-Commerce*. Düsseldorf, 2000, Symposion Publishing. URL `http://www.symposion.de/datamining`.

[104] T. M. Mitchell: *Machine Learning*. New York, NY, 1996, McGraw Hill.

[105] Dunja Mladenic – NADA Lavrac – Marko Bohanec – Steve Moyle: *Data Mining and Decision Support: Integration and Collaboration*. 2003, Kluwer Academic Publishers.

[106] Andreas Mueller: Fast sequential and parallel algorithms for association rule mining: A comparison. CS-TR-3515. Jelentés, College Park, MD, 1995, Departure of Computer Science, University of Maryland. URL `http://citeseer.nj.nec.com/mueller95fast.html`.

[107] Raymond T. Ng – Jiawei Han: Efficient and effective clustering methods for spatial data mining. In Jorge B. Bocca – Matthias Jarke – Carlo Zaniolo (szerk.): *Proceedings of the 20th International Conference Very Large Data Bases, VLDB* (konferenciaanyag). 1994. 12-15, Morgan Kaufmann, 144–155. p. ISBN 1-55860-153-8.
URL `http://citeseer.nj.nec.com/571734.html`.

[108] Edward Omiecinski – Ashoka Savasere: Efficient mining of association rules in large dynamic databases. In *British National Conference on Databases* (konferenciaanyag). 1998, 49–63. p.

[109] Banu Ozden – Sridhar Ramaswamy – Abraham Silberschatz: Cyclic association rules. In *ICDE* (konferenciaanyag). 1998, 412–421. p.
URL `http://citeseer.nj.nec.com/ozden98cyclic.html`.

[110] Lawrence Page – Sergey Brin – Rajeev Motwani – Terry Winograd: The pagerank citation ranking: Bringing order to the web. Jelentés, 1998, Stanford Digital Library Technologies Project. URL `http://citeseer.nj.nec.com/page98pagerank.html`.

[111] Jong Soo Park – Ming-Syan Chen – Philip S. Yu: An effective hash based algorithm for mining association rules. In Michael J. Carey – Donovan A. Schneider (szerk.): *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data* (konferenciaanyag). San Jose, California, 1995. 22-25, 175–186. p.
URL `http://citeseer.nj.nec.com/park95effective.html`.

[112] N. Pasquier – Y. Bastide – R. Taouil – L. Lakhal: Pruning closed itemset lattices for association rules. In *Proceedings of the BDA French Conference on Advanced Databases* (konferenciaanyag). 1998. October. URL `http://citeseer.nj.nec.com/pasquier98pruning.html`.

[113] N. Pasquier – Y. Bastide – R. Taouil – L. Lakhal: Efficient mining of association rules using closed itemset lattices. In *Journal of Information systems* (konferenciaanyag). 1999, 25–46. p.

[114] Nicolas Pasquier – Yves Bastide – Rafik Taouil – Lotfi Lakhal: Discovering frequent closed itemsets for association rules. In *ICDT* (konferenciaanyag). 1999, 398–416. p.
URL `http://citeseer.nj.nec.com/pasquier99discovering.html`.

[115] Jian Pei – Jiawei Han – Laks V. S. Lakshmanan: Mining frequent item sets with convertible constraints. In *ICDE* (konferenciaanyag). 2001, 433–442. p.
URL `http://citeseer.ist.psu.edu/383962.html`.

[116] Jian Pei – Jiawei Han – Runying Mao: CLOSET: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery* (konferenciaanyag). 2000, 21–30. p.
URL `http://citeseer.nj.nec.com/pei00closet.html`.

[117] G. Piatetsky-Shapiro: Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro – W. Frawley (szerk.): *Knowledge Discovery in Databases* (konferenciaanyag). 1991, AAAI/MIT Press, Menlo Park, CA, 229–248. p.

[118] Wim Pijls – Jan C. Bioch: Mining frequent itemsets in memory-resident databases. In *Proceedings of the Eleventh Belgium /Netherlands Articial Intelligence Conference (BNAIC 99)* (konferenciaanyag). 1999, 75–82. p.
URL `http://citeseer.nj.nec.com/pijls99mining.html`.

[119] Jim Porter: Disk/trend report. In *Proceedings of the 100th Anniversary Conference on Magnetic Recording and Information Storage*. Santa Clara Univerity, 1998.

[120] M. F. Porter: An algorithm for suffix stripping. *Program*, 14. évf. (1980. July) 3. sz.

[121] Pál Rózsa: *Lineáris algebra és alkalmazásai*. 1991, Tankönyvkiadó, Budapest.

[122] S. Sahni – T. Gonzales: P-complete approxiamtion problems. *JACM*, 23. évf. (1976).

[123] G. Salton – C. Buckley: Term weighting approaches in autmatic text retrievel. *Information Processing and Management*, 24. évf. (1998) 5. sz.

[124] G. Salton – M. J. McGill: *An Introduction to Modern Information Retrieval*. 1983, McGraw-Hill.

[125] Nandlal L. Sarda – N. V. Srinivas: An adaptive algorithm for incremental mining of association rules. In *DEXA Workshop* (konferenciaanyag). 1998, 240–245. p.

[126] Ashoka Savasere – Edward Omiecinski – Shamkant B. Navathe: An efficient algorithm for mining association rules in large databases. In *The VLDB Journal* (konferenciaanyag). 1995, 432–444. p. URL `http://citeseer.nj.nec.com/sarasere95efficient.html`.

[127] R. E. Schapire – Y. Singer: BoosTexter: a boosting-based system for text categorization. *Machine Learning*, 39. évf. (2000) 2/3. sz.

[128] R. E. Schapire – Y. Singer – A. Singhal: Boosting and Rocchio applied to text filtering. In *Proc. of SIGIR-98, 21st ACM International Conference on Research and Development in Information Retrieval* (konferenciaanyag). Melbourne, Australia, 1998, 215–223. p.

[129] Matthew G. Schultz – Eleazar Eskin – Salvatore J. Stolfo:. Mef: Malicious email filter - a unix mail filter that detects malicious windows executables.
URL `http://citeseer.nj.nec.com/417909.html`.

[130] Matthew G. Schultz – Eleazar Eskin – Erez Zadok – Salvatore J. Stolfo:. Data mining methods for detection of new malicious executables.
URL `http://citeseer.nj.nec.com/417492.html`.

[131] H. Schütze – D. A. Hull – J. O. Pedersen: A comparison of classifiers and document representations for the routing problem. In *Proc. of SIGIR-95, 18th ACM Int. Conf. on Research and Development in Information Retrieval* (konferenciaanyag). Seattle, WA, 1995, 229–237. p.

[132] F. Sebastiani: Machine learning in automated text categorization. *ACM Computing Surveys*, 34. évf. (2002. March) 1. sz.

[133] F. Sebastiani – A. Sperduti – N. Valdambrini: An improved boosting algorithm and its application to automated text categorization. In *Proc. of CIKM-00, 9th ACM Int. Conf. on Information and Knowledge Management* (konferenciaanyag). McLean, VA, 2000, 78–85. p.

[134] Dennis G. Severance: Identifier search mechanisms: A survey and generalized model. *ACM Comput. Surv.*, 6. évf. (1974) 3. sz. ISSN 0360-0300.

[135] Ron Shamir – Dekel Tsur: Faster subtree isomorphism. *Journal of Algorithms*, 33. évf. (1999) 2. sz. ISSN 0196-6774.

[136] Li Shen – Hong Shen: Mining flexible multiple-level association rules in all concept hierarchies (extended abstract). In *Database and Expert Systems Applications* (konferenciaanyag). 1998, 786–795. p.

[137] Abraham Silberschatz – Alexander Tuzhilin: On subjective measures of interestingness in knowledge discovery. In *Knowledge Discovery and Data Mining* (konferenciaanyag). 1995, 275–281. p. URL `http://citeseer.nj.nec.com/silberschatz95subjective.html`.

[138] Spencer: The probabilistic method. In *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)* (konferenciaanyag). 1992.

[139] Ramakrishnan Srikant – Rakesh Agrawal: Mining generalized association rules. *Proceedings of the 21st International Conference on Very Large Databases (VLDB), Zurich, Switzerland*, 1995.

[140] Ramakrishnan Srikant – Rakesh Agrawal: Mining sequential patterns: Generalizations and performance improvements. In Peter M. G. Apers – Mokrane Bouzeghoub – Georges Gardarin (szerk.): *Proceedings of 5th International Conference Extending Database Technology, EDBT* (konferenciaanyag), 1057. köt. 1996. 25-29, Springer-Verlag, 3–17. p. ISBN 3-540-61057-X.
URL `http://citeseer.nj.nec.com/article/srikant96mining.html`.

[141] D. R. Swanson: Two medical literatures that are logically but not bibliographically connected. *JASIS*, 38. évf. (1987) 4. sz.

[142] Pang-Ning Tan – Vipin Kumar – Jaideep Srivastava: Selecting the right interestingness measure for association patterns. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (konferenciaanyag). New York, NY, USA, 2002, ACM Press, 32–41. p. ISBN 1-58113-567-X.

[143] Lyn C. Thomas: A survey of credit and behavioural scoring; forecasting financial risk of lending to consumers. *International Journal of Forecasting 16*, 2000.

[144] Lyn C. Thomas: A survey of credit and behavioural scoring; forecasting financial risk of lending to consumers. *International Journal of Forecasting*, 16. évf. (2000).

[145] Shiby Thomas – Sreenath Bodagala – Khaled Alsabti – Sanjay Ranka: An efficient algorithm for the incremental updation of association rules in large databases. In *Knowledge Discovery and Data Mining* (konferenciaanyag). 1997, 263–266. p.
URL `http://citeseer.nj.nec.com/thomas97efficient.html`.

[146] Shiby Thomas – Sunita Sarawagi: Mining generalized association rules and sequential patterns using SQL queries. In *Knowledge Discovery and Data Mining* (konferenciaanyag). 1998, 344–348. p. URL `http://citeseer.nj.nec.com/thomas98mining.html`.

[147] D. Tikk – Gy. Biró: Experiments with multilabel text classifier on the Reuters collection. In *Int. Conf. on Computational Cybernetics (ICCC03)* (konferenciaanyag). Siófok, Hungary, 2003, 33–38. p.

[148] D. Tikk – Gy. Biró – J. D. Yang: A hierarchical text categorization approach and its application to FRT expansion. *Australian Journal of Intelligent Information Processing Systems*, 8. évf. (2004) 3. sz.

[149] D. Tikk – Gy. Biró – J. D. Yang: Experiments with a hierarchical text categorization method on WIPO patent collections. In N. O. Attok-Okine – B. M. Ayyub (szerk.): *Applied Research in Uncertainty Modelling and Analysis*. International Series in Intelligent Technologies sorozat, 20. köt. 2005, Springer, 283–302. p.

[150] Hannu Toivonen: Sampling large databases for association rules. In *The VLDB Journal* (konferenciaanyag). 1996, 134–145. p.
URL `http://citeseer.nj.nec.com/toivonen96sampling.html`.

[151] K. Tumer – J. Ghosh: Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8. évf. (1996) 3–4. sz.

[152] J. R. Ullmann: An algorithm for subgraph isomorphism. *J. ACM*, 23. évf. (1976) 1. sz. ISSN 0004-5411.

[153] C. J. van Rijsbergen: *Information Retrieval*. 2nd. kiad. London, 1979, Butterworths. `http://www.dcs.gla.ac.uk/Keith`.

[154] Jianyong Wang – Jiawei Han – Jian Pei: Closet+: Searching for the best strategies for mining frequent closed itemsets. In *In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'03)* (konferenciaanyag). Washington, DC, USA, 2003. URL `http://citeseer.nj.nec.com/wang03closet.html`.

[155] S. M. Weiss – C. Apte – F. J. Damerau – D. E. Johnson – F. J. Oles – T. Goetz – T. Hampp: Maximizing text-mining performance. *IEEE Intelligent Systems*, 14. évf. (1999. July/August) 4. sz.

[156] W. Wibovo – H. E. Williams: Simple and accurate feature selection for hierarchical categorisation. In *Proc. of the 2002 ACM symposium on Document engineering* (konferenciaanyag). McLean, Virginia, USA, 2002, 111–118. p.

[157] E. D. Wiener – J. O. Pedersen – A. S. Weigend: A neural network approach to topic spotting. In *Proc. of the SDAIR-95, 4th Annual Symposium on Document Analysis and Information Retrieval* (konferenciaanyag). Las Vegas, NV, 1995, 317–332. p.

[158] Y. Yang: An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1. évf. (1999) 1–2. sz.

[159] Y. Yang – X. Liu: A re-examination of text categorization methods. In *Proc. of SIGIR-99, 22nd ACM Int. Conf. on Research and Development in Information Retrieval* (konferenciaanyag). Berkeley, CA, 1999, 42–49. p.

[160] Y. Yang – J. P. Pedersen: Feature selection in statistical learning of text categorization. In *Proc. of the 14th Int. Conf. on Machine Learning* (konferenciaanyag). 1997, 412–420. p.

[161] S. B. Yao: Tree structures construction using key densities. In *Proceedings of the 1975 annual conference* (konferenciaanyag). 1975, ACM Press, 337–342. p.

[162] Mohammed J. Zaki: Efficiently mining frequent trees in a forest. Jelentés, Troy, NY, 12180, 2001. July, Computer Science Department, Rensselaer Polytechnic Institute.

[163] Mohammed J. Zaki: Efficiently mining frequent trees in a forest. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining* (konferenciaanyag). 2002, ACM Press, 71–80. p. ISBN 1-58113-567-X.

[164] Mohammed J. Zaki – Karam Gouda: Fast vertical mining using diffsets. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (konferenciaanyag). 2003, ACM Press, 326–335. p. ISBN 1-58113-737-0.

[165] Mohammed Javeed Zaki: Sequence mining in categorical domains: Incorporating constraints. In *CIKM* (konferenciaanyag). 2000, 422–429. p.
URL `http://citeseer.nj.nec.com/zaki00sequence.html`.

[166] Mohammed Javeed Zaki – Mitsunori Ogihara: Theoretical foundations of association rules. In *Proceedings of third SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD'98)* (konferenciaanyag). Seattle, Washington, 1998.
URL `http://citeseer.nj.nec.com/zaki98theoretical.html`.

[167] Mohammed Javeed Zaki – Srinivasan Parthasarathy – Mitsunori Ogihara – Wei Li: New algorithms for fast discovery of association rules. In David Heckerman – Heikki Mannila – Daryl Pregibon – Ramasamy Uthurusamy – Menlo Park (szerk.): *Proceedings of the third International Conference on Knowledge Discovery and Data Mining* (konferenciaanyag). 1997. 12-15, AAAI Press, 283–296. p. ISBN 1-57735-027-8.
URL `http://http://citeseer.nj.nec.com/30063.html`.