Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Diplomski rad br. 45

Primjena metoda umjetne inteligencije na povećanje sigurnosti uloga za pristup bazama podataka

Marko Pletikosa

Zagreb, lipanj 2010.

Sveučilište u Zagrebu
Fakultet elektrotehnike i računarstva

Diplomski rad br. 45

Primjena metoda umjetne inteligencije na povećanje sigurnosti uloga za pristup bazama podataka

Marko Pletikosa

Zagreb, lipanj 2010.

Diplomski zadatak br. 45

Primjena metoda umjetne inteligencije na povećanje sigurnosti uloga za pristup bazama podataka

Zahvala

Mojim roditeljima...

Hvala Ivanu Kokanu na pruženoj pomoći prilikom rješavanja početničkih nedoumica u pisanju rada u LaTeX-u te Mariu Lučiću na pomoći s učinkovitom implementacijom generiranja svih podskupova danog skupa.

Sadržaj

Poglavlje 1

Uvod

Tijekom XX. stoljeća, razvojem informacijskih tehnologija, a posebice razvojem interneta, informacije su sve dostupnije. Prema istraživanjima [8], u 2000.g. u svijetu je pohranjen 1.5 exabajt podataka, dok se u 2003.g. taj podatak popeo na 3.5 exabajta. Prema [6], rast i dostupnost magnetskih i optičkih spremišta podataka poput tvrdih diskova dostiže brzinu rasta količine generiranih informacija. Samim time, uskoro ćemo moći pohraniti svaku generiranu informaciju.

Postavlja se pitanje što možemo učiniti s tim informacijama, kako ih najbolje iskoristiti i kako otkriti pravilnosti unutar informacija koje su nam od interesa. Strojno učenje kao jedno od grana umjetne inteligencije se bavi načinima ekstrakcije ovakvih pravilnosti i dodatnog znanja iz samog skupa informacija.

Jedno od najpopularnijih rješenja za pohranu informacija su baze podataka. S obzirom na takav rast pohranjenih informacija, nameće se pitanje dostupnosti i upravljanjem pristupu tim podatcima, tj. o sigurnosti baza podataka. Uloge za pristup bazi (*eng. database roles*) jedan su od glavnih aspekata sigurnosti baze.

Cilj ovog rada je ponuditi metodu za poboljšanje sigurnost pristupa bazama podataka primjenjujući tehnike strojnog učenja kako bi otkrili pravilnosti i sigurnosne prijetnje unutar opisa uloga za pristup bazi podataka. Pri tome se posebice misli na primjenu algoritama za dubinsku analizu podataka, učenje asocijacija i otkrivanje učestalih uzoraka.

U poglavlju 2 je dana šira slika problema koji se rješavaju metodama strojnog učenja te je dana formalna definicija problema učenja skupova učestalih predmeta. U poglavlju 3 je dan pregled dvaju algoritma za učenje skupova učestalih predmeta: Iscrpno pretraživanje i Apriori, zatim u poglavlju 4 je predstavljena struktura podataka 4.1.2 FP-stablo, njena svojstva te na koji način se koristi u Algoritmu 4.2.4 rasta učestalih skupova predmeta. Nakon toga, u poglavlju 5 slijedi opis programskog ostvarenja

koje je sastavni dio ovog rada. U poglavlju 6 je dan pregled ispitivanja učinkovitosti ostvarenog algortma na sintetičkim podatcima koji se koriste za ispitivanje ovakvih algoritama. Nakon toga, u poglavlju 7 je dan zaključak, a zatim i sažetak rada.

Poglavlje 2

Definicija problema učenja učestalih skupova podataka

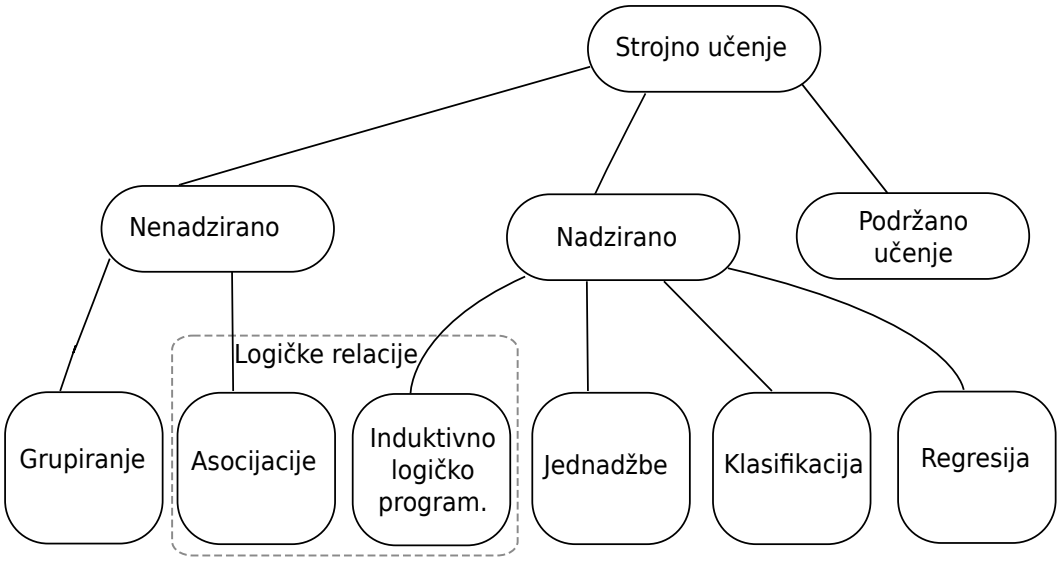
U ovom poglavlju se formalno i na primjeru definirama problem učenja učestalih skupova podataka. Potrebno je istaknuti da se u ovom radu pojmovi učenja učestalih skupova podataka i učenja učestalih skupova predmeta smatraju sinonimima. Prvo je dan kratak pregled strojnog učenja, a zatim i uvod u problematiku učenja asocijacija.

2.1 Strojno učenje

Strojno učenje (*eng. machine learning*) je područje umjetne inteligencije koje se fokusira na dizajn i implementaciju algoritama koji poboljšavaju svoje performanse kroz iskustvo [4]. Ti algoritmi često koriste heurističke, statističke i empirijske metode kako bi uspjeli reducirati prostor pretraživanja i riješili probleme koje klasični algoritmi ne uspijevaju.

2.1.1 Pregled metoda stojnog učenja

U ovom pododjeljku dan je kratak pregled područja strojnog učenja u kojem su detaljniji opisi izostavljeni. Za potpuniji pregled strojnog učenja, čitatelju se preporuča [1]. Razlikujemo metode strojnog učenja s obzirom na način uporabe dobivenog znanja: klasifikacija, regresija, grupiranje, učenje asocijacija, relacija i (diferencijalnih) jednadžbi. Na slici 2.1 je strukturno prikazana raščlamba strojnog učenja.



Slika 2.1. *Raščlamba Strojnog učenja*

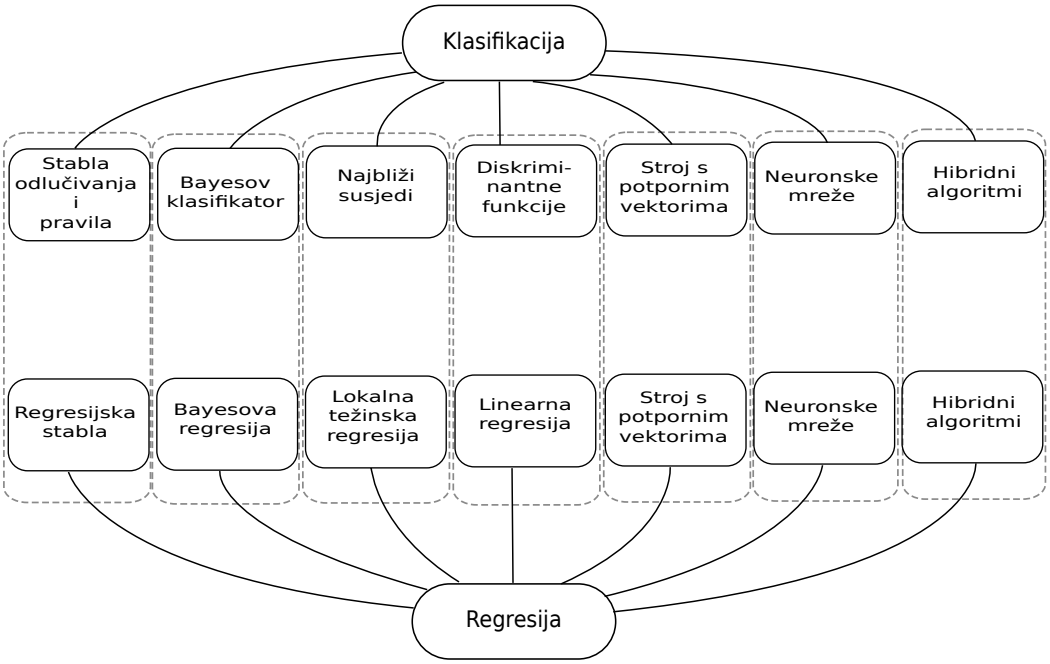
Klasifikacija

Metode strojnog učenja najčešće se koriste za probleme klasifikacije. Predpostavimo da imamo objekt, opisan s mnogo atributa. Svakom objektu se može dodijeliti točno jedna klasa iz konačnog skupa mogućih klasa. Atributi su nezavisno promatrane varijable, diskretne ili kontinuriane. Klasa je zavisna nepromatrana varijabla i njenu vrijednost određujemo iz vrijednosti odgovarajućih nezavisnih varijabli. Zadatak *klasifikatora* je odrediti koju klasu treba dodijeliti promatranom objektu.

Tipičan klasifikacijski zadatak je medicinska dijagnostika: Pacijent je opisan skupom kontinuiranih (dob, visina, težina, tjelesna temperatura, krvni tlak) i diskrenim (spol, lokacija boli, broj otkucaja srca u minuti) atributima. Zadatak klasifikatora je odrediti dijagnozu - označiti pacijenta s jednom od mogućih medicinskih dijagnoza (zdrav, prehlada, upala pluća...).

Kako bi odredio klasu, klasifikator mora moći stanju opisati diskretnu funkciju, **preslikavanje iz prostora atributa u prostor klasa**. Tip ove funkcije se može unaprijed zadati, ali se može i učiti na temelju podataka. Podatci se sastoje od primjera za učenje koji opisuju prijašnje riješene primjere danog problema.

Različiti klasifikatori reprezentiraju funkciju preslikavanja na različite načine. Najčešći klasifikatori su *stabla odlučivanja*, *pravila odlučivanja*, *naivni Bayesov klasifikatori*, *Bayesove mreže vjerovanja*, *klasifikatori najbližih susjeda*, *linearne diskriminantne funkcije*, *logička regresija*, *strojevi s potpornim vektorima* i *umjetne neuronske mreže*. Na slici 2.2 možemo vidjeti pregled metoda vezanih uz klasifikaciju i regresiju.



Slika 2.2. *Klasifikacija i regresija*

Regresija

Kao i u klasifikacijskim problemima, i u regresiji imamo skup objekata za učenje, opisanih s više atributa (svojstava). Atributi su nezavisne promatrane varijable (diskretne ili kontinuirane). Zavisna varijabla (regresija) je kontinuirana i njena vrijednost je određena funkcijom nezavisnih varijabli. Zadatak *regresijskog predviđanja* je odrediti vrijednost zavisne, nepromatrane kontinuirane varijable za promatrani objekt.

Slično kao klasifikatori, cilj regresije je implementirati kontinuiranu funkciju preslikavanja iz prostora atributa u prostor predviđenih vrijednosti. Oblik (tip) ove funkcije može biti dan unaprijed ili naučen iz već riješenih primjera za učenje. Zadatak algoritma za učenje je odrediti kontinuiranu funkciju učeći na skupu podataka za učenje. Kasnije ovom funkcijom predviđamo nove vrijednosti za nove, prethodno neviđene objekte.

Algoritmi regresije se razlikuju s obzirom na način prezentacije regresijske funkcije. Česta je uporaba *linearne regresije, regresijskih stabala, lokalne težinske regresije, strojeva s potpornim vektorima za regresiju i višeslojnih neuronskih mreža za regresiju.* Pregled ovih metoda je dan na slici 2.2.

Logičke relacije

Logičke relacije možemo promatrati kao generalizirane diskretne funkcije. Za razliku od klasifikacije, nemamo jedinstvenu ovisnu diskretnu varijablu (klasu), već sve varijable (atribute) tretiramo ekvivalentno. Ponekad, neke varijable imaju nepoznate vrijednosti i mi te vrijednosti želimo predvidjeti. U drugim slučajevima, poznajemo vrijednosti svih varijabli, a zadatak je provjeriti da li relacija vrijedi za dani skup vrijednosti. Prema ekspresivnoj moći korištenog jezika za opisivanje skupa za učenjem razlikujemo *učenje asocijacija* i *induktivno logičko programiranje.* Više o učenju asocijacija se može pronaći u slijedećim poglavljima.

Jednadžbe

Pri modeliranju raznih realnih problema, često se susrećemo s mjerenjima različitih, povezanih i međusobno zavisnih procesa. Učenjem sustava jednadžbi želimo eksplicitno opisati ove međuovisnosti. Tada možemo taj sustav iskoristiti za simulaciju i predviđanje.

Struktura jednadžbi može biti unaprijed dana iz pozadinskog znanja problema ili algoritam za učenje može sam odrediti jednadžbe kao i njihove odgovarajuće koeficijente. Za kontinuirano promjenjive sustave, potreban nam je sustav parcijalnih diferencijalnih jednadžbi. Ove metode se često koriste u meteorologiji, ekologiji (modeliranje bioloških sustava) i inženjerstvu, za modeliranje dinamičnih procesa.

Grupiranje (engl. *Clustering*)

Nenadzirano učenje se razlikuje od klasifikacije i regresije prema tome što je dan samo opis primjera s atributima, bez nadzirane (ciljne) varijable. Grupiranje je najpopularnija metoda *nenadziranog učenja.* Zadatak algoritma za učenje je odrediti koherentne podskupove (klastere) primjera za učenje. Metoda se često koristi u prirodnim znanostima, analizi procesa te psihološkim i sociološkim istraživanjima. Kako bi grupiranje bilo uspješno, najvažnije je odrediti mjeru sličnosti pojedinih primjera, na temelju znanja o problemu.

Podržano učenje

Podržano učenje se bavi problemom učenja autonomnog agenta koji djeluje i percipira svoje okruženja kako bi odabrao optimalne akcije koji omogućuju ostvarenje njegovog cilja. Agent prima (percipira) informacije o trenutnom stanju svijeta te obavlja akcije kako bi ga promijenio. Nakon svake akcije, agent dobiva nagradu (pozitivnu

ili negativnu). Nagrada nije nužno rezultat samo posljednje akcije, već se razmatra n akcija unatrag. Pošto je nagrada odgođena, problem učenja je težak i ponekad spor. Agent se ponekad modelira tako da koristi samo poznato znanje o svijetu, a ponekad i da eksperimentira.

Najčešće korištena metoda je tzv. *Q-učenje*. Ovom metodom agent iterativno uči evaluacijsku funkciju stanja i akcija. Može se primjeniti čak i kad agent nema prethodno znanje kako njegove akcije utječu na svijet.

2.2 Opis problema na primjeru

Kako bismo mogli dati opis problematike pojedinih algoritama za učenje asocijacija, (učenje uzoraka, učenje čestih skupova predmeta) potrebno je definirati osnovne pojmove koji se koriste. Pojmovi se definiraju na primjeru koji je jedan od najčešće rješavanih ovom metodom, a to je analiza potrošačke košarice. Riječ je o analizi računa (transakcija) ostvarenih na promatranom mjestu, npr. trgovini. Osnovni pojam kojeg trebamo definirati je predmet. Predmet je najosnovnija jedinica s kojom radimo, a u ovom primjeru to je artikal. Nadalje, treba definirati pojam transakcije koju definiramo kao je skup predmeta. *Transakcijska baza podataka - TDB* je skup transakcija koje promatramo. U našem primjeru, to su računi izdani na blagajni. Primjer transakcija možemo vidjeti na slici 2.3.

Slika 2.3. *Primjer transakcija*

Cilj analize potrošačke košarice je otkriti nama zanimljive asocijacije. Asocijacije su implikacije tipa:

$$X \Longrightarrow Y. \tag{2.1}$$

Rezultat dubinske analize je asocijacija, npr. u 40% slučajava, kad kupac kupi čips, on kupi i sok. Ovakvi podatci omogućavaju osmišljavanje marketinških strategija. Vlasnici trgovine mogu uvesti akcijsku prodaju čipsa, računajući pritom na profit ostvaren povećanom prodajom soka. Također, jedna od mogućnosti je grupiranje artikala tako da oni budu blizu (tako da podsjete kupca da je čips jako dobar uz sok), ili daleko (da se na neki način prisili kupca da prođe cijelu trgovinu od soka do čipsa te usputno kupi još artikala koje nije namjeravao kupiti).

2.2.1 Potpora

Potpora skupa predmeta **s** (*eng. support*) je broj pojavljivanja tog skupa predmeta u bazi transakcija. Potpora asocijacijskog pravila je jednaka potpori skupa antecedensa i konsekvensa 2.1.

$$s = \sigma(predmetiX \cup predmetiY) \tag{2.2}$$

2.2.2 Povjerenje

Povjerenje asocijacijskog pravila **c** (*eng. confidence*) definiramo kao udio pojavljivanja (skupa) predmeta Y u transakcijama koje sadrže X implikacije 2.1.

$$c = \frac{\sigma(predmetiX \cup predmetiY)}{\sigma(predmetiX)} \tag{2.3}$$

2.3 Formalna definicija

Problem učenja učestalih skupova podataka uveli su R. Agrawal i suradnici u radu [9].

Uzmimo $I = \{i_1, \ldots, i_n\}$ kao skup *predmeta*. *Skup predmeta* $X \subseteq I$ je (pod)skup predmeta. Skup predmeta koji sadrži k predmeta nazivamo *k-predmetni skup* ili *k-člani skup*.

Transakcija $T = (tid, X)$ je uređeni par gdje *tid* označava jedinstveni identifikator transakcije, a X je skup predmeta. Kažemo da *transakcija* T *sadrži* skup predmeta Y ako $Y \subseteq X$.

Transakcijska baza podataka, TDB je skup transakcija koje definiraju problem. *Potpora* danog skupa predmeta X u transakcijskoj bazi, označena s $sup_{TDB}(X)$ ili $sup(X)$ je broj transakcija u *TDB* koji sadrže X, tj. :

$$sup(X) = |\{(tid, Y)|((tid, Y) \in TDB) \wedge (X \subseteq Y)\}| \tag{2.4}$$

Definicija problema: Uz korisnički zadanu *minimalnu potporu* min_sup, X je *skup učestalih predmeta* ili *učestali uzorak (kraće uzorak)*, ako $sup(X) \geq min\_sup$.

Problem pronalaska čestih skupova predmeta svodi se na pronalazak potpunog skupa čestih uzoraka u transakcijskoj bazi TDB uzimajući pritom u obzir zadani prag min_sup.

Asocijacijska pravila se mogu izvesti iz skupa čestih uzoraka. *Asocijacijsko pravilo* je implikacija u obliku 2.1, gdje su X i Y uzorci takvi da $X \cap Y = \emptyset$. Pravilo $X \Longrightarrow Y$ ima *potporu* s, ako $sup_{TDB}(X \cup Y) = s$. Pravilo $X \Longrightarrow Y$ vrijedi u transakcijskoj bazi TDB s *pouzdanošću (povjerenjem)* $c = \frac{sup(X \cup Y)}{sup(X)}$.

Uz danu transakcijsku bazu TDB, pragove potpore min_sup i povjerenja min_conf, problem pronalaska asocijacijskih pravila je pronaći potpun skup asocijacijskih pravila koja imaju potporu i povjerenje iznad korisnički definiranih pragova.

Problem se može različiti na dva dijela:

1. Pronalazak čestih uzoraka u TDB, uzimajući u obzir definirane pragove te
2. generiranje asocijacijskih pravila iz pronađenih uzoraka.

Kao što će kasnije biti pokazano u radu, ponalazak učestalih skupova predmeta primjenu pronalazi ne samo u učenju asocijacijskih pravila, već i kao temelj u drugim metodama za dubinsku analizu podataka, kao što je učenje ulančanih uzoraka, višerazinskim asocijativnim pravilima i sl.

Poglavlje 3

Algoritmi za učenje asocijacija

U ovom poglavlju dan je pregled razvoja algoritama za učenje asocijacija, od najosnovnijih, do trenutno najuspješnijih inačica za savladavanje ovog problema.

3.1 Iscrpno pretraživanje

Iscrpno pretraživanje (*eng. brute force*) se uvijek odnosi na algoritme koji pokušavaju generirati sva rješenja i onda odrediti koje je najbolje od njih. Često je problem u tome što je to iznimno računalno zahtjevno i neostvarivo sa danim ograničenjima brzine računala i razumnosti vremena rješavanja.

Iscrpno pretraživanje primjenjeno na rješavanje problema učenja učestalih skupova predmeta moglo bi se opisati kao:

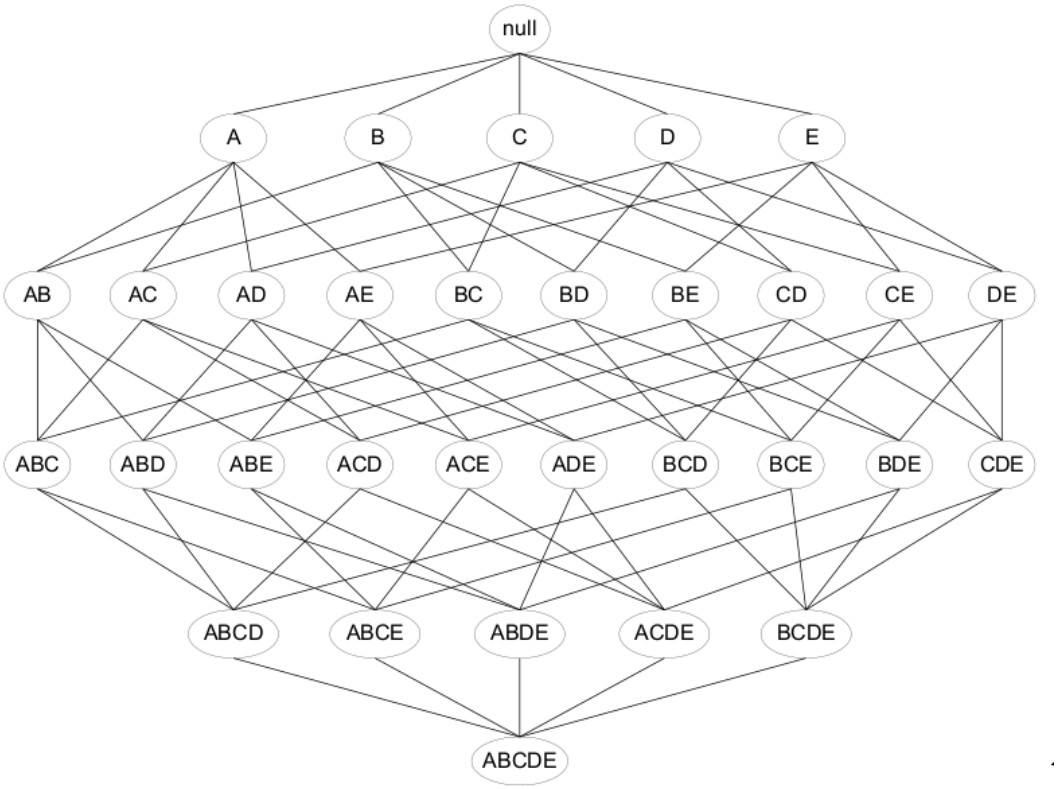
Algoritam 1: Iscrpno pretraživanje

1. izlistaj sve moguće skupove predmeta,
2. izračunaj potporu i povjerenje za svaku od generiranih skupova predmeta,
3. ukloni one skupove koji ne zadovoljavaju dana ograničenja minimalne potpore i minimalnog povjerenja
4. generiraj sve asocijacije za svaki od skupova predmeta.

Problem je u koraku 1. i 4. jer nailazimo na problem kombinatorne eksplozije. Ukoliko imamo **d** različitih predmeta, tad imamo

$$M = 2^d \tag{3.1}$$

mogućih skupova zadanih predmeta (kandidata). Primjer za pet premeta se nalazi na slici 3.1.



Slika 3.1. *Primjer generiranih skupova podataka za skup od četiri podatka*

Uz danih **d** različitih pojedinačnih predmeta, postoji 2^d različitih skupova predmeta te

$$\sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right] = 3^d - 2^{d+1} + 1 \tag{3.2}$$

različitih asocijacijskih pravila [5]. Primjerice, za $d = 6$, postoje 602 različita pravila.

Vidimo da je potrebno osmisliti strategiju smanjenja broja kandidata, a po mogućnosti i smanjenja broja transakcija. Implementacija ovih heurističkih metoda dovodi nas do novog, *Apriori* algoritma.

3.2 Apriori algoritam

Apriori algoritam predstavlja poboljšanje iscrpnog pretraživanja jer pomoću antimonogonog *apriori* svojstva reducira prostor pretraživanja. Svojstvo je prvi put pred-

stavljeno u [9].

3.2.1 *Apriori* heuristika

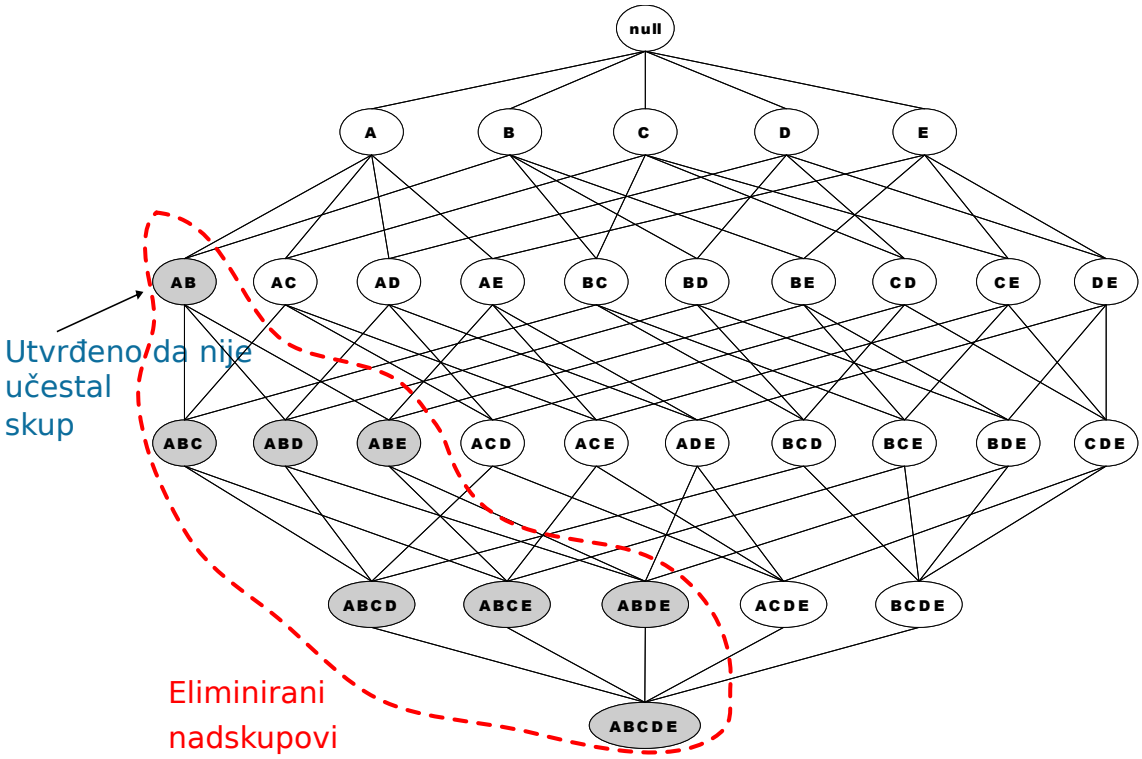
Teorem 1. **(*Apriori*)** *Bilo koji nadskup nedovoljno čestog skupa predmeta ne može biti čest. Drugim riječima, svaki podskup čestog skupa mora biti čest,* [7] tj. vrijedi formula 3.3:

$$\forall X, Y : (X \subseteq Y) \Longrightarrow s(X) \geq s(Y). \tag{3.3}$$

Dokaz. Da bismo dokazali teorem, potrebno je pokazadi da $sup(X) \leq sup(Y)$ ako $X \supseteq Y$.

Uz dane transakcijsku bazu TDB, uzmimo X i Y kao dva skupa predmeta t.d. $X \supseteq Y$. Za svaku transakciju T, koja sadrži skup X, ona ujedno sadrži i Y, koji je podskup od X. Samim time, $sup(X) \leq sup(Y)$. ■

Apriori heuristika dramatično reducira prostor pretraživanja. Primjer reduciranja prostora pretraživanja dan je na slici 3.2 [5]



Slika 3.2. *Primjer redukcije pretraživanja Apriori heuristikom*

Na temelju ove heuristike, dizajniran je brzi algoritam za učenje čestih uzoraka predmeta, nazvan *Apriori*.

Pogledajmo rad algoritma na primjeru transakcija sa slike 2.3. Uzmimo da je zadana

minimalna potpora $min\_sup = 4$

1. Uzmimo $k = 1$. Algoritam prolazi kroz TDB i broji pojavljivanja svakog od predmeta u bazi. Nakon prolaska kroz TDB, uklanjanju se predmeti čija je potpora manja od min_sup. Time je dobiven L_1, potpun skup 1-članih skupova. U ovom primjeru, to su *{Kruh, Kikiriki, Mlijeko, Voće, Pekmez, Sok, Čips}*. Izbačeni su elementi *{Odrezak, Sir, Jogurt}*.

2. $k = 2$. Skup 2-članih skupova kandidata, označenih s C_2 dobijemo iz L_1. U ovom koraku koristimo *Apriori heuristiku* kako bismo eliminirali nepotrebne elemente. Samo oni kandidati kojima su **svi** podskupovi česti, mogu biti potencijalno česti. Predmet $xy \in C_2$ *akko* $x, y \in L_1$. Prema tome, C_2 = *{KruhKikiriki, KruhMlijeko, . . . , KruhČips, . . . , SirJogurt }*. Ovdje 2-člani skup {Sir, Jogurt } pišemo kao *SirJogurt*. Postoji $\binom{7}{2} = 21$ elemenat u C_2. Oni formiranju skup kandidata učestalih 2-članih skupova.

3. Slijedećim prolazom kroz TDB se utvrđuje koji od kandidata iz C_2 ima potporu veću (ili jednaku) od min_sup. Elementi koji zadovoljavaju taj uvjet postaju elementi L_2. U ovom primjeru, L_2 = *{KruhPekmez, KikirikiVoće, MlijekoVoće, MlijekoPekmez, MlijekoSok, VoćeSok, PekmezSok, SokČips}*.

4. $k = 3$. Iz L_2 se gradi C_3 kombinacijama elemenata, imajući na umu *Apriori heuristiku*. Da bi element *KruhPekmezVoće* bio član C_3, nužno je da su *{KruhPekmez, KruhVoće, PekmezVoće }* $\subseteq L_2$. Vidimo da *{KruhVoće, PekmezVoće}* $\notin L_2$ pa samim time *(KruhPekmezVoće)* $\notin C_3$. Daljnjim pretraživanjem vidimo da je C_3 = *{MlijekoVoćeSok}*.

5. Dodatnim prolazom kroz TDB se utvrđuje potpora $\forall x \in C_3$ te se uklanjaju oni koji ne zadovoljavaju min_sup. U ovom slučaju, $L_3 = C_3$.

6. $k = 4$. Pošto $|L_3| = 1$, algoritam završava i utvđen je potpun skup svih učestalih skupova predmeta: $\bigcup_{i=1...k} L_i$.

3.2.2 *Apriori* algoritam

Algoritam 2. ***Apriori***

Ulaz: baza transakcija TDB i prag potpore min_sup

Izlaz: potpuni skup čestih uzoraka u TDB koji zadovoljavaju min_sup

Pseudokod:

1. prođi kroz TDB i odredi L_1 skup 1-članih učestalih skupova predmeta

2. za $(k = 2; L_{k-1} \neq \emptyset; k++)$ radi:

   (a) generiraj C_k, skup k-članih kandidata. k-člani skup x je član C_k akko svaki (k-1)-člani podskup od x se nalazi u L_{k-1};

   (b) ako je $C_k = \emptyset$ idi na korak 3.;

   (c) prođi kroz TDB i utvrdi potporu svakog skupa predmeta iz C_k;

   (d) $L_k = \{X | (X \in C_k) \wedge (sup(X) \geq min\_sup)\}$;

3. vrati $\cup_{i=1,\ldots,k} L_i$

3.2.3 Generiranje pravila

Kako bismo generirali pravila, za svaki skup čestih predmeta l moramo pronaći sve neprazne podskupove od l. Za svaki podskup a, dobivamo pravilo $a \Longrightarrow (l - a)$, ako je zadovoljeno minimalno povjerenje $(s(l)/s(a) \geq min\_conf)$ [10]. Vidimo da za k-člani skup učestalih predmeta postoji $2^k - 2$ mogućih asocijacijskih pravila (isključujući slučajeve kad je jedna strana prazan skup ili skup jednak skupu čestih predmeta) [5]. Cilj nam je generirati pravila sa što većim povjerenjem. Povjerenje nema anti-monotono svojstvo (*Apriori*) kao potpora. Primjerice, $c(ABC \longrightarrow D)$ može biti veće ili manje od $c(AB \longrightarrow D)$.

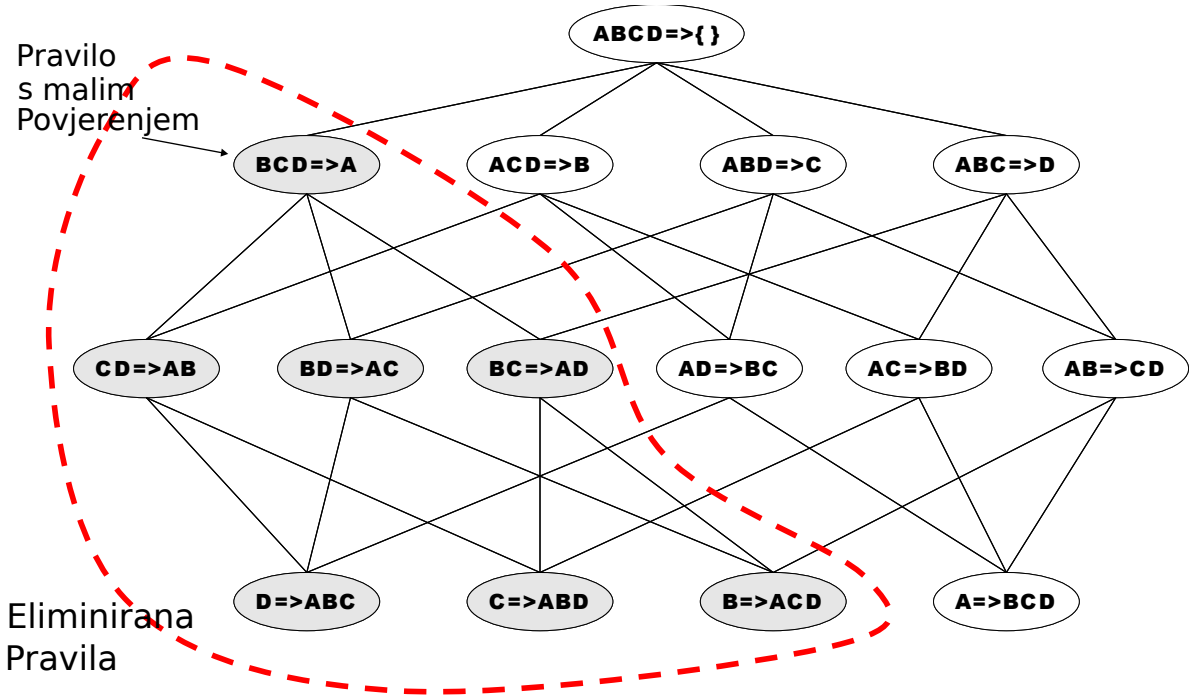
Međutim, povjerenje pravila generiranih iz istog skupa čestih predmeta ima to svojstvo:

Uzmimo da je česti skup

$L = A, B, C, D$, tada:

$c(ABC \longrightarrow D) \geq c(AB \longrightarrow CD) \geq c(A \longrightarrow BCD)$

Ovo su temeljna razmatranja koja pridonose učinkovitosti algoritma generiranja pravila. Na slici 3.3 [5] vidimo primjer redukcije pretrazivanja prostora pravila kad utvrdimo da jedno pravilo nema dovoljnu razinu potpore. Vidimo da je situacija analogna pretraživanju skupa čestih predmeta koristeći *Apriori* heuristiku. Više o generiranju pravila se može pronaći u literaturi ([11])



Slika 3.3. *Primjer redukcije pretraživanja prostora pravila*

3.2.4 Analiza učinkovitosti *Apriori* algoritma

U danom primjeru sa slike 2.3, postoji 10 različitih predmeta te $\binom{10}{2} = 45$ dvočlanih podskupova tog skupa. Koristeći heuristiku, *Apriori* je generirao samo 8 elemenata u skupu L_2. Time je ostvarena ušteda od 82.2%. Kako se povećava broj pojedinačnih elemenata u TDB, tako raste i broj mogućih dvočlanih podskupova pa je i učinkovitost podrezivanja ove heuristike povećana.

Nedostatci

Iako ovako definiran algoritam podrezuje veći broj kandidata, i dalje je iznimno računski zahtjevno upravljati s velikim brojem kandidata u velikim transakcijskim bazama. Uzmimo na primjer bazu s milijun pojedinačnih predmeta, od kojih je samo 1% čestih. Prije podrezivanja, potrebno je generirati i odrediti potporu za svaki od $\approx 10^7$ 2-članih kandidata.

Pri svakom koraku, potrebno je proći kroz cijelu bazu transakcija kako bismo utvrdili potporu svakog od kandidata za česte uzorke. Prolasci kroz poveće TDB su jako skupi i troše mnogo vremena. Kako bi *Apriori* utvrdio k-člani učestali skup, potrebno je skenirati cijelu bazu k puta.

Mogućnosti poboljšanja

Za opisani *Apriori* algoritam postoje različite mogućnosti poboljšanja. U nedostatcima tražimo mogućnosti poboljšanja:

1. Prolazak kroz čitavu bazu transakcija je vremenski zahtjevan posao. Možda možemo smanjiti broj prolazaka kroz bazu.
2. *Apriori* generira velik broj kandidata prije podrezivanja. Razvijeni su načini kako smanjiti broj generiranih kandidata kako bi se proces ubrzao.
3. Određivanje potpore za svakog od kandidata je prevladavajuća i zahtjevna operacija. Predložene su metode ubrzanja korištenjem sažetka (*eng. hash*) i dinamičko brojanje skupova predmeta.

Više o ovim metodama može se pronaći u literaturi [7], [9].

Poglavlje 4

Algoritam rasta učestalih skupova predmeta

U prethodnom poglavlju smo pogledali *Apriori* algoritam i utvrdili da su generiranje velikog broja kandidata i skeniranja baze transakcija pri svakom koraku osnovna *uska grla* tog algoritma. Često se ovaj nedostatak naziva *generiranje-i-testiranje Apriori algoritma.*

Može li se izbjeći *generiraj-i-testiraj* paradigma? Algoritam *Rasta učestalih skupova predmeta* (*eng. FP-growth algorithm*) predstavlja rješenje tog problema. U ovom poglavlju predstavljena je osnovna struktura korištenu u radu algoritma - *Stablo učestalih uzoraka* (engl. *Frequent-pattern tree - FP-tree*) koje je u daljnjem tekstu referencirano kao *FP-stablo* ili engleskom srkaćenicom *FP-tree*. Nakon toga predstavljen je sam algoritam i njegov rad na primjeru.

4.1 FP-stablo: dizajn i konstrukcija

Informacije iz transakcijske baze podataka *TDB* su ključne za uspješno učenje učestalih uzoraka. Od velike koristi bi bila konstrukcija kompaktne strukture podataka kako bismo mogli velike *TDB* pohraniti u radnoj memoriji računala. Bitna osobina ovakve strukture je potpunost podataka potrebnih za učenje uzoraka čime bi potpuno eliminirali potrebu sa prolazak kroz *TDB* prilikom *k-tog* koraka.

4.1.1 Primjer izgradnje stabla

Kako bismo što jasnije prezentirali proces izgradnje i analize stabla, pogledajmo ga na jednostavnom primjeru.

Primjer 4.1 Uzmimo transakcijsku bazu *TDB* danu u tablici 4.1 [7]. Prvi stupac je identifikator transakcije, u drugom stupcu su navedeni predmeti, dok su u trećem poredani učestali 1-člani skupovi predmeta. Zadani prag potpore je $min\_sup = 3$

Tablica 4.1. *Primjer transakcijske baze*

TID	Kupljeni predmeti	Sortirani česti predmeti
1	f, a, c, d, g, i, m, p	f, c, a, m, p
2	a, b, c, f, l, m, o	f, c, a, b, m
3	b, f, h, j, o	f, b
4	b, c, k, s, p	c, b, p
5	a, f, c, e, l, p, m, n	f, c, a, m, p

Promotrimo prvo koja količina podataka nam je potrebna iz *TDB*:

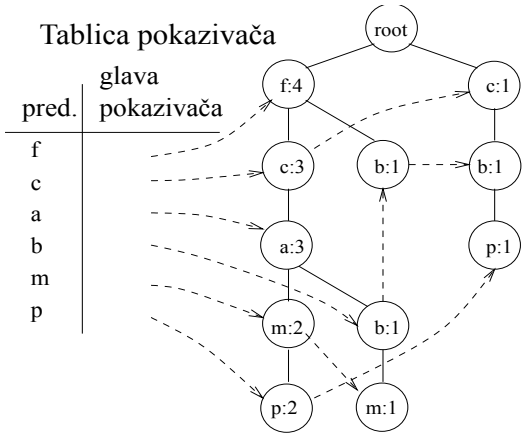
1. Budući da samo predmeti koji se dovoljno puta pojavljuju u transakcijama, imaju utjecaj u učenju učestalih uzoraka, nužno je utvrditi koji su to predmeti jednim prolazkom kroz *TDB* te odrediti potporu svakog od njih.

2. Ukoliko uspijemo prikupiti i sažeti samo bitne podatke potrebne za učenje, možda uspijemo cijelu *TDB* spremiti u glavnu memoriju.

3. Ako više transakcija dijeli isti (pod)skup predmeta, možda je moguće spojiti te skupove zajedno s pripadajućim brojem ponavljanja svakog od predmeta iz tih skupova. Ako se držimo fiksnog redoslijeda, lako kasnije možemo razlučiti ove transakcije.

4. Ako dvije, ili više transakcija dijele isti početni podskup predmeta, prethodno poredanih po nekom kriteriju, moguće je dobiti prefiksnu strukturu dokle god pazimo na pravilno brojanje broja pojavljivanja elemenata. Ako predmete poredamo po silaznom broju pojavljivanja, intuitivno očekujemo da će više transakcija dijeliti isti prefiks.

Izgradnja stabla na temelju *TDB* iz primjera 4.1



Slika 4.1. *Stablo izgrađeno na temenju baze transakcija iz tablice 4.1*

Uzevši ta razmatranja, možemo izgraditi stablo prikazano na slici 4.1 na slijedeći način:

1. Prvim prolaskom kroz *TDB* utvrdimo učestale predmete i njihovu potporu. Time dobijamo *listu*: $\langle(f:4),(c:4),(a:3),(b:3),(m:3),(p:3)\rangle$. Primjetimo da su predmeti sortirani prvo prema broju pojavljivanja (broj nakon ":" u listi), a zatim abecedno. Ovaj redoslijed je jako bitan jer će se on poštivati i tijekom izgradnje samog FP-stabla. Kako bi omogućili lakše praćenje, lista sortiranih predmeta se nalazi u najdesnijem stupcu tablice 4.1.

2. Nakon toga, stvaramo korijenski (*eng. root*) čvor stabla kojeg označavamo s *null*. Slijedno prolazimo kroz *TDB* te unosimo transakciju po transakciju u stablo. Ako transakcija ne sadrži nijedan od prethodno utvrđenih čestih predmeta, ona se preskače.

   (a) Učitavanje prve transakcije rezultira s prvom granom u stablu: $\langle(f:1),(c:1),(a:1),(m:1),(p:1)\rangle$. Predmeti su sortirani kao u *listi*.

   (b) Druga transakcija: $\langle f,c,a,b,m\rangle$ dijeli prefiks s prvom: $\langle f,c,a\rangle$. Elementima koji su u zajedničkom prefiksu samo se inkrementira potpora za 1 te se stvara čvor $(b:1)$, kao dijete čvora $(a:2)$. Na $(b:1)$ nadovezujemo $(m:1)$. Time je završen unos druge transakcije.

   (c) Za treću transakciju $\langle f,b\rangle$ vidimo da već postoji prefiks $(f:2)$ te se spuštamo na donju razinu stabla, inkrementirajući potporu za taj čvor koji sad glasi $(f:3)$. Na njega nadovezujemo novi čvor, $(b:1)$

   (d) Četvrta transakcija nema dijeljeni prefiks s postojećima u stablu te se stvara nova grana: $\langle(c:1),(b:1),(p:1)\rangle$

(e) Posljednja transakcija $\langle f, c, a, m, p \rangle$ je identična prvoj pa ne rezultira novim čvorovima, već samo povećanjem potpore svakog od njih.

Kako bi povećali brzinu obilaska stabla, posebna tablica pokazivača, kao na slici 4.1 se izgrađuje tako da svaki od predmeta ima pokazivač na prvo pojavljivanje tog predmeta u samom stablu. Dodatno, svaki čvor stabla ima po jedan pokazivač na slijedeći čvor u stablu koji ima isti predmet (crtkane strelice na slici 4.1).

4.1.2 FP-stablo - Definicija i algoritam izgradnje

Definicija 4.1 FP-stabla

Stablo učestalih uzoraka predmeta (engl. *Frequent-pattern tree - FP-tree*) je stablo sa slijedećim svojstvima:

1. Sastoji se od posebnog korijenskog (engl. *root*) čvora, skupa *predmetno-prefiksnih podstabala* kao djece korijenskog čvora te tablice s pokazivačima na prvo pojavljivanje učestalih predmeta u stablu.
2. Svaki čvor u svakom od *predmetno-prefksnih stabala* se sastoji od slijedećih elemenata:
   (a) Imena predmeta;
   (b) Broja pojavljivanja tog predmeta u tom podstablu, koje odgovara broju putanja koje dolaze do promatranog čvora, pokazivača na roditelja te pokazivača na slijedeći čvor stabla koji sadrži isti predmet (ili null, ako takav ne postoji).
   (c) Tablica pokazivača se sastoji od dva polja: imena predmeta i (glavu) pokazivača na prvi čvor u stablu koji sadrži taj predmet.

Algoritam 3: Izgradnja FP-stabla

Ulaz: Transakcijska baza *TDB* i prag potpore *min_sup*

Izlaz: Stablo učestalih predmeta - *FP-stablo*.

Pseudokod:

1. Prođi kroz *TDB* jednom, utvrdi *F*, skup učestalih predmeta i utvrdi potporu za svakog od njih. Sortiraj *F* prema silaznom broju pojavljivanja i to spremi kao *F-listu*, *listu* čestih predmeta.

2. Stvori korijenski čvor *FP-stabla*, T, i označi ga kao *null*. Za svaku transakciju t iz TDB radi slijedeće:

   Sortiraj elemente prema redosljedu u *F-listi*. Elemente kojima je potpora manja od praga, odbaci. Nazovimo tu listu sortiranih elemenata transakcije P. Pozovi funkciju $unesi\_transakciju(P, T)$.

   Funkcija $unesi\_transakciju(P, T)$ radi slijedeće:

   (a) Iz liste P ukloni prvi (najveća potpora) element i nazovi ga p.

   (b) Ako T ima dijete N za koje vrijedi $N.predmet = p.predmet$, povećaj $N.brPojavljivanja$ za 1. Inače stvori novi čvor N s brojem pojavljivanja postavljenim na 1 i dodaj N kao dijete čvoru T. Listaj tablicu pokazivača za svoj predmet dok ne dođeš do čvora kojem je pokazivač na slijedeći *null* pokazivač. Njega postavi da pokazuje na N.

   (c) Ako $p \neq \emptyset$ pozovi $unesi\_transakciju(P, N)$ rekurzivno.

Konstrukcija stabla zahtjeva samo dva prolaza kroz TDB: Prvi za određivanje čestih predmeta i drugi za unos pojedinih transakcija u strukturu. Složenost unošenja transakcije linearno ovisi o njenoj duljini.

4.1.3 Svojstva FP-stabla

U ovom odjeljku dana su neka od najvažnijih svojstava ove strukture podataka uz kraća objašnjenja, ne ulazeći u dokaze koje znatiželjni čitatelj može pronaći u [7].

1. *FP-tree* sadrži sve informacije potrebne za učenje čestih uzoraka predmeta. Temeljeno na *Apriori* principu, samo česti predmeti su potrebni za učenje čestih uzoraka pa tako i struktura koja sadrži sve česte predmete i njihove potpore sadrži dovoljno informacija.

2. Uz danu transakcijsku bazu TDB i prag potpore min_sup, iz *FP-tree-a* koje odgovara toj bazi se može dobiti potpora svakog od predmeta. Koristeći tablicu s pokazivačima, za svaki od predmeta možemo slijediti putanju kroz stablo, obilazeći samo čvorove koji sadrže dani predmet. Sumiranjem svih pojedinačnih potpora dolazimo do globalne potpore tog predmeta.

3. Za zadanu TDB i prag potpore min_sup, broj čvorova u stablu je ograničen s gornje strane s $\sum_{t \in TDB} |freq(t)|+1$, gdje je $freq(t)$ projekcija čestih elemenata te transakcije, tj. $freq(t) = t \cap F$. Isto tako, najveća dubina stabla je ograničena s $max_{t \in TDB}\{|freq(t)|\}$. Broj čvorova je maksimalan kad nijedna transakcija nema

zajednički prefiks, tj. broj pojavljivanja svakog od čvorova je 1. Dodatni čvor je korijenski (+ 1). Dubina je ograničena s najvećim brojem čestih predmeta unutar jedne transakcije.

Ovako definirano stablo je iznimno kompaktna struktura, ponajviše zahvaljujući prethodnom silaznom sortiranju elemenata prema potpori. Ipak, valja napomenuti da, u posebnim slučajevima, postoje bolja rješenja od ovog. Međutim, ona se ne pojavljuju često u praksi te, zasada, nije otkrivena općenito bolja opcija.

4.2 Korištenje FP-stabla za učenje čestih uzoraka predmeta

U prethodnom odjeljku opisan je dizajn i implementaciju kompaktne strukture korištene za učenje čestih uzoraka predmeta. Ta struktura eliminira potrebu za brojanjem potpore (prolaskom kroz cijelu *TDB*) koja je jedan od nedostataka *Apriori* algoritma. Preostaje eliminirati problem kombinatorne eksplozije prilikom generacije kandidata. U ovom odjeljku dan je opis algoritma nazvanog *Algoritam rasta učestalih skupova predmeta* koji iskorištava sve prednosti spomenutog *FP-stabla* te omogućava učinkovito učenje potpunog skupa čestih skupova predmeta.

4.2.1 Principi rasta čestih uzoraka predmeta

Prije nego prijeđemo na opis rada algoritma na primjeru, valja istaknuti bitno svojstvo:

Svojstvo poveznica čvorova stabla (engl. *Node-link property*):

Prateći tablicu pokazivača i poveznice među čvorovima s istim predmetom a_i u stablu, moguće je pronaći sve uzorke koji sadrže predmet a_i.

Svojstvo proizilazi direktno iz konstrukcije stabla, a demonstrira punu učinkovitost tablice pokazivača kao dodatnog pomagala u obilasku stabla.

Primjer 4.2 Proces učenja čestih uzoraka za podatke dane u tablici 4.1.

Prema *listi* čestih predmeta, *f-c-a-b-m-p*, svi česti uzorci se mogu podijeliti u šest podskupova, bez preklapanja:

1. uzorci koji sadrže p,
2. uzorci koji sadrže m, ali ne p

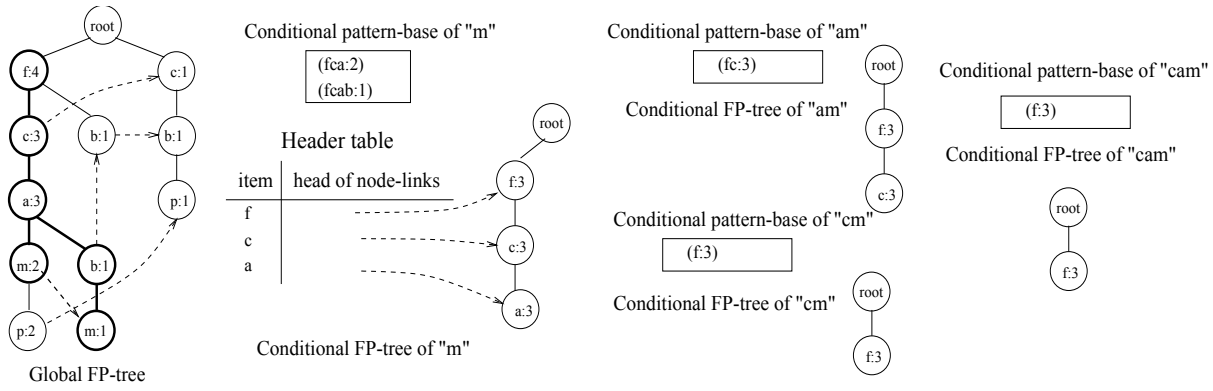
3. uzorci koji sadrže b, ali ne m i p
4. uzorci koji sadrže a, ali ne b, m i p
5. uzorci koji sadrže c, ali ne a, b, m i p
6. uzorci koji sadrže f, ali ne c, a, b, m i p

Ovdje primjenjujemo podijeli-pa-vladaj (engl. *Devide-and-conquer*) strategiju: svaki od ovih podskupova zasebno rješavamo. Slijedi proces učenja na svakom od ovih podskupova:

1. Prvo se promatraju uzorci koji sadrže predmet p. Očiti uzorak je $(p : 3)$.

   Kako bi algoritam pronašao uzorke koji sadrže p, mora pristupiti svim projekcijama predmeta koje sadrže p. Temeljeno na svojstvu *Poveznica čvorova stabla*, može se pristupiti svim tim projekcijama prateći putanju pokazivača iz tablice pokazivača za predmet p te prateći pokazivače svakog čvora na slijedeći čvor s isim predmetom.

   Prateći te pokazivače, može se utvrditi da p ima dvije putanje u *FP-stablu:* $\langle(f : 4, c : 3, a : 3, m : 2, p : 2)\rangle$ i $\langle(c : 1, b : 1, p : 1)\rangle$. Prva putanja ukazuje da postoje dvije transakcije koje sadrže $\{f, c, a, m, p\}$. Isto tako, utvrđuje se da se $\langle(f, c, a)\rangle$ pojavljuje tri, dok se $\langle f\rangle$ pojavljuje čak četiri puta. Međutim, oni se pojavljuju samo dva puta *zajedno* s p. Dakle, kako bi utvrdilo uzorke koji se pojavljuju skupa s p, samo se *p-ova prefiksna putanja* $\langle(f : 2, c : 2, a : 2, m : 2)\rangle$, ili kraće $\langle(fcam : 2)\rangle$ broji. Slično tome, za drugu putanju dobijemo *prefiksnu putanju* $\langle cb : 1\rangle$. Prema tome, *p-ove prefiksne putanje:* $\{(fcam : 2), (cb : 1)\}$ formiraju podbazu predmeta za p, koja se naziva *uvjetna baza uzoraka* (tj. podbaza čestih uzoraka pod uvjetom postojanja predmeta p). Konstrukcija *FP-stabla* za ovu uvjetnu bazu (ovakvo stablo nazivamo *p-ovo uvjetno FP-stablo*) vodi do samo jedne grane: $(c : 3)$. Ostale grane se eliminiraju zbog nedovoljne potpore ($min\_sup = 3$!) Prema tome, utvrđen je jedan česti uzorak $(cp : 3)$ i pretraživanje za p završava.



Slika 4.2. *Učenje uzoraka za stablo* FP-tree | m

2. Predstoji traženje čestih uzoraka koji sadrže m, ali ne p. Odmah se identificira $(m : 3)$. Prateći *m-ove* pokazivače, dvije putanje u *FP-stablu* se pronalaze: $\langle(f : 4, c : 3, a : 3, m : 2)\rangle$ i $\langle(f : 4, c : 3, a : 3, b : 1, m : 1)\rangle$. Ovdje valja istaknuti da se i p pojavljuje u ovim putanjama, ali nema potrebe za uključivanjem p u pretragu, jer su već identificirani česti uzorci koji sadrže p. Analogno analizi iz prethodnog koraka, *m-ovo* pripadajuće *FP-stablo*, $\langle(f : 3, c : 3, a : 3)\rangle$, sadrži jednu putanju, kako je prikazano na slici 4.2 preuzetoj iz [7]. Ovo stablo se dalje rekurzivno analizira pozivima $mine(\langle(f : 3, c : 3, a : 3)\rangle \mid m)$. Slika 4.2 pokazuje da prethodni poziv funkcije *mine* uključuje tri predmeta (a), (b), (c) u sljedu (*listi*). Prvi pronalazi uzorak (am:3), uz uvjetnu bazu {(fc:3)}, i zatim poziva poziv $mine(\langle(f : 3, c : 3)\rangle \mid am)$; drugi pronalazi (cm:3) uz uvjetnu bazu {(f:3)}, zatim poziv $mine(\langle(f : 3)\rangle \mid cm)$; dok treći pronalazi (fm:3). Daljnji poziv $mine(\langle(f : 3, c : 3)\rangle \mid am)$ pronalazi (cam:3), (fam:3), uz uvjetnu bazu {(f:3)}, slijedi poziv $mine(\langle(f : 3\rangle \mid cam)$ koji vraća najdulji uzorak (fcam:3). Analogno njemu, poziv $mine(\langle(f : 3)\rangle \mid cm)$ vraća samo (fcm:3). Prema tome, skup čestih uzoraka koji sadrže m je {(m:3), (am:3), (cm:3), (fm:3), (cam:3), (fam:3), (fcam:3), (fcm:3)}. Ovaj primjer pokazuje da ***stablo sa samo jednim putem (lista) se može razriješiti listanjem svih kombinacija elemenata u stablu (listi)***.

3. Slično određujemo uzorke koji sadrže b, ali ne m i p. Čvor b vraća (b:3) i tri putanje: $\langle f : 4, c : 3, a : 3, b : 1\rangle$, $\langle f : 4, b : 1\rangle$ i $\langle c : 1b : 1\rangle$. Pošto je potpora svakog od predmeta u ovim putanjama manja od praga, učenje završava i pronađeno je samo (b:3).

4. Za sve uzorke koji sadrže a, ali ne b, m i p, čvor a vraća {(a:3)} i jednu uvjetnu bazu {(fc:3)}, koja u obliku liste. Tako, imajući na umu prethodni zaključak, generiraju se njihove kombinacije s a dobivamo: {(fa:3), (ca:3), (fca:3)}.

5. Slijede uzorci koji sadrže c, ali ne a, b, m i p. Od čvora c se dobiva (c:4) i uvjetna baza {(f:3)} te na posljetku i (fc:3).

6. Zadnji od podskupova, uzorci koji sadrže samo f, rezultiraju s (f:4).

Tablica 4.2. *Rezultati učenja za primjer iz tablice 4.1*

Predmet	Uvjetna baza predmeta	Uvjetno *FP-stablo*
p	*{(fcam:2), (cb:1) }*	*{(c:3)|p}*
m	*{(fca:2), (fcab:1) }*	*{(f:3, c:3, a:3)|m}*
b	*{(fca:1), (f:1), (c:1) }*	∅
a	*{(fc:3)}*	*{(f:3, c:3)|a}*
c	*{(f:3)}*	*{(f:3)}|c*
f	∅	∅

Rezultat učenja dan je u tablici 4.2.

4.2.2 Dodatna svojstva

Svojstvo prefiksne putanje (engl. *Prefix path property*)

Kako bi izračunali česte uzorke koji sadrže sufiks a_i, samo prefiksne podputanje čvorova označenih s a_i u FP-stablu trebaju biti uračunate, i potpora svakog čvora na toj prefiksnoj putani treba biti jednaka potpori odgovoarajućeg čvora a_i na toj putanji.

Dokaz. Uzmimo da su čvorovi na putanji P označeni kao $a_1, \ldots, a_n$ na takav način da je a_1 korijen prefiksnog podstabla, a a_n je list. Promatrani čvor označavamo s $a_i (1 \leq i \leq n)$. Prema algoritmu izgradnje stabla, svaki prefiksni čvor $a_k, (1 \leq k \leq i)$, prefiksna podputanja čvora a_i u P se pojavljuje, zajedno s a_k točno $a_i.brojPojavljivanja$. Prema tome, svaki prefiksni čvor bi trebao imati svoj brojač postavljen tako da bude jednak brojaču čvora a_i. Također vrijedi da se svaki čvor $a_j, (i < j \leq n)$ pojavljuje u transakcijama zajedno s a_i. Međutim, on nam nije bitan u trenutnom razmatranju jer je već obrađen u prethodnim razmatranjima (čvor a_j je dublje u stablu, a algoritam kreće od listova prema korijenu - po tzv. *bottom-up* principu) i njegovi česti uzorci su već poznati (koji eventualno uključuju i a_i). ■

Uzmimo za primjer čvor m iz primjera iz tablice 4.1. m se nalazi na putanji $\langle f : 4, c : 3, a : 3, m : 2, p : 2\rangle$. Kako bi izračunao česte uzorke uz m na ovoj putanji, samo prefiksna podputanja od m se treba uzeti u obzir, tj. $\langle f : 4, c : 3, a : 3\rangle$. Broj pojavljivanja svakog od ovih predmeta u uvjetnom stablu treba postaviti na broj pojavljivanja promatranog čvora, m. Prema tome, prefiksna putanja u ovom slučaju treba izgledati: $\langle f : 2, c : 2, a : 2\rangle$.

Prema ovom svojstvu, prefiksna podputanja čvora a_i u putanji P može se kopirati i transformirati u novu prefiksnu putanju s podešenim brojevima pojavljivanja elemenata tako da odgovaraju broju pojavljivanja promatranog čvora. Ovu operaciju

nazivamo *transformacija prefiksne putanje* čvora a_i na putanji P.

Skup transformiranih prefiksnih putanja čvora a_i čine bazu putanja koji se pojavljivaju skupa s a_i. Takvu bazu putanja nazivamo *uvjetnom bazom putanja čvora* a_i, i označavamo kao "$baza\_putanja|a_i$". Svi česti uzorci povezani s tim čvorom se mogu pronaći učeći na *FP-stablu* stvorenom na temelju tih putanja. Takvo stablo nazivamo a_i*-ovo uvjetno stablo*, koje označavamo s "$FP - stablo|a_i$". Proces se dalje nastavlja rekurzivno.

Rast fragmenata (engl. *Fragment growth*)

Uzmimo da je α skup predmeta u bazi, da je B α-ina uvjetna baza i da je β skup predmeta u B. Tad je potpora $\alpha \cup \beta$ u bazi jednaka potpori β u B.

Dokaz. Prema definiciji uvjetne baze predmeta, svaka (pod)transakcija u B se pojavljuje pod uvjetom pojavljivanja α u originalnoj bazi. Ako se skup predmeta β pojavljuje u bazi ψ puta, onda se pojavljuje i ψ puta u bazi. Štoviše, budući da se svi takvi predmeti iz uvjetne baze α-e, $\alpha \cup \beta$ se pojavljuje točno ψ puta u bazi (TDB). ■

Rast uzoraka (engl. *Pattern growth*)

Uzmimo da je α čest uzorak u TDB, da je B α-ina uvjetna baza i da je β skup predmeta u B. Tad je $\alpha \cup \beta$ čest skup predmeta u TDB akko je β čest u B.

Dokaz.

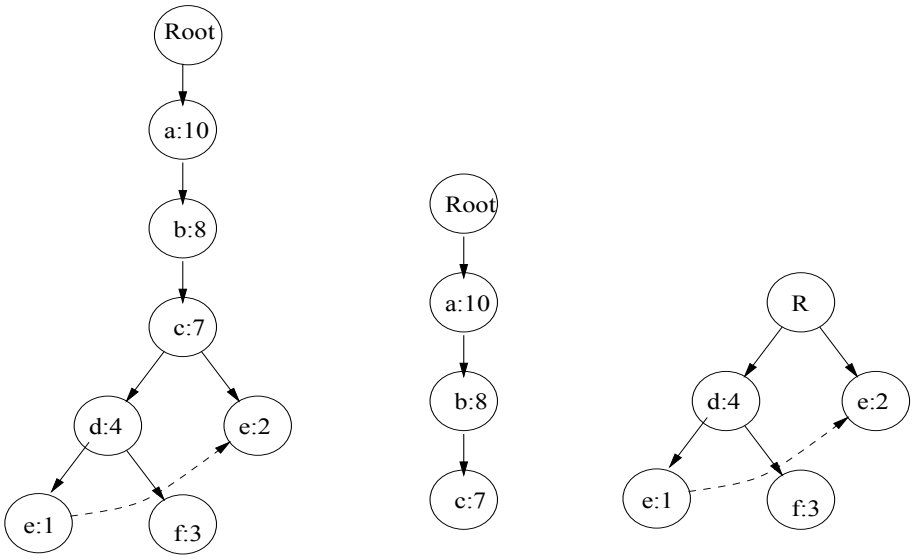
($\Longrightarrow$) Ako je β čest u B, tj. β se u B pojavljuje barem ξ ($min\_sup = \xi$) puta. Pošto je B α-ina uvjetna baza uzoraka, svaka transakcija u B se pojavljuje pod uvjetom postojanja α. To jest, β se pojavljuje u *TDB* barem ξ puta. Prema tome, $\alpha \cup \beta$ je čest skup u *TDB*.

($\Longleftarrow$) Pretpostavimo da β nije čest skup u B, tj. da se pojavljuje manje od ξ puta. Pošto je B uvjetna α-ina baza, svi skupovi predmeta koji sadrže β i pojavljuju se zajedno s α su u B. Prema tome, $\alpha \cup \beta$ nije čest skup u *TDB*. ■

4.2.3 Rast čestih uzoraka s FP-stablom s jedinstvenim prefiksom

Prethodno opisana metoda primjenjiva je na sve oblike *FP-stabla*. Međutim, dodatnu optimizaciju je moguće ostvariti na posebnoj vrsti *FP-stabla*, *FP-stablu s jedinstvenom prefiksnom putanjom*. Ta optimizacija omogućuje brzo učenje dugih uzoraka.

Stablo s jedinstvenom prefiksnom putanjom je stablo koje se sastoji od samo jedne prefiksne putanje, ili prefiksne putanje koja ide do prvog granajućeg čvora, čvora koje ima više od jednog djeteta. Pogledajmo primjer.



(a) Stablo s prefiks. putanjom (b) Prefiksna putanja P (c) Dio s više putanja Q

Slika 4.3. *Učenje na stablu s jedinstvenim prefiksom*

Primjer 4.3 Na slici 4.3(a) vidimo *FP-stablo* koje možemo raščlaniti na dva podstabla. Prvo stablo je prikazano na 4.3(b), i predstavlja *stablo jedinstvene prefiksne putanje* $\langle(a:10) \rightarrow (b:8) \rightarrow (c:7)\rangle$ koje se proteže od korijena do granajućeg čvora $(c:7)$. Iako je moguće odrediti česte uzorke na ovom stablu uobičajenim rekurzivnim pozivima, bolja strategija je upravo prikazana raščlamba stabla (a) na stablo s prefiksnom putanjom (koje izgleda kao *lista*) te na dio s višestrukim putanjama, prikazano na 4.3(c) s korijenom označenim s R kao posebnim tipom čvora - tzv. pseudo-korijenom. Proces učenja na ovim stablima možemo izvršiti nezavisno jedan o drugom pa naknadno kombinirati njihove rezultate.

Pogledajmo proces učenja na ova dva stabla. Stablo s jedinstvenom prefiksnom putanjom $P = \langle(a:10) \rightarrow (b:8) \rightarrow (c:7)\rangle$ razrješavamo jednostavnim *prebrojavanjem svih kombinacija čvorova uzduž te putanje. Potporu* takvih kombinacija *postavljamo* tako da bude *jednaka minimalnoj potpori čvorova koji čine tu kombinaciju.* Prema tome, dobiveni česti uzorci nad putanjom P su: *česti_ uzorci(P)* = {(a:10), (b:8), (c:7), (ab:8), (ac:7), (bc:7), (abc:7) }.

Nazovimo stablo s višestrukim putanjama Q. Postupak učenja nad Q je kako slijedi: R smatramo kao standardni *null* korijen, tada Q tretiramo kao obično *FP-stablo* pa proces učenja vraća slijedeći rezultat: *česti_ uzorci(Q)* = {(d:4), (e:3), (f:3), (df:3)}. Nadalje, za svaki skup predmeta iz Q, R možemo promatrati kao uvjetnu bazu čestih uzoraka te svaki skup predmeta u Q, u kombinaciji sa svakim od čestih uzoraka do-

bivenih iz R, mogu činiti novi česti uzorak. Primjerice, za $(d:4) \in$ *česti_uzorci(Q)*, P možemo promatrati kao uvjetnu bazu uzoraka tako da (a:10) $\in$ *česti_uzorci(P)* s (d:4) stvara novi uzorak (ad:4). Prema tome, za (d:4), skup čestih uzoraka dobivamo tako da (d:4) $\times$ *česti_uzorci*(P) = {(ad:4), (bd:4), (cd:4), (abd:4), (acd:4), (bcd:4), (abcd:4)}. Ovdje je korištena standardna oznaka za kartezijev produkt dvaju skupova $X \times Y$. Primjetimo da se potpora uvijek postavlja na onu koja odgovara članu s minimalnom potporom u kombinaciji. Shodno tome, rezultat učenja čestih uzoraka nad stablom 4.3(c) dobijemo kao kartezijev produkt čestih uzoraka dobivenih iz originalnog dijela stabla s jedinstvenom prefiksnom putanjom, s čestim uzorcima dobivenim iz stabla s višestrukim prefiksnim putanjama, tj: *česti_uzorci(P)* $\times$ *česti_uzorci(Q)*.

Kao zaključak razmatranja ovog primjera, slijedi da česte uzorke stabla, koje se može rasčlaniti na dio s jedinstvenom prefiksnom putanjom i dio s višestrukim prefiksnim putanjama, možemo dobiti iz tri disjunktna skupa:

1. *česti_uzorci(P)*, skup čestih uzoraka generiranih iz prefiksno-jedinstvenog dijela stabla P;
2. *česti_uzorci(Q)*, skup čestih uzoraka dobivenih iz stabla s višestrukim prefiksnim putanjama i
3. *česti_uzorci(P)* $\times$ *česti_uzorci(Q)*, skup uzoraka koji uključuje oba dijela ■

Time smo na primjeru prikazali dodatna svojstva koja valja istaknuti:

Učenje uzoraka na FP-stablu s jedinstvenom putanjom

Pretpostavimo da se FP-stablo T sastoji od jedinstvene putanje $P = \langle root \to a_1 : s_1 \to a_2 : s_2 \to \ldots \to a_k : s_k \rangle$. Skup predmeta $X = a_{i_1} \ldots a_{i_j} (1 \leq i_1 < \ldots < i_j \leq k)$ je čest uzorak i potpora skupa X je jednaka potpori čvora a_{i_j} u stablu T.

Dokaz. Prema konstrukciji *FP-stabla*, $a_1, \ldots, a_i, \ldots, a_k$ su česti predmeti. Budući da u stablu T postoji samo jedna putanja, potpora opada s indeksom i, tj. vrijedi: $sup(a_1 \ldots a_k) = sup(a_k) = s_k \geq min\_sup$. Prema *Apriori* principu, vrijedi tvrdnja da je X čest uzorak (skup) predmeta.

Za svaki skup predmeta X, definiran kako je i izrečen u ovom svojstvu, svaka transakcija koja sadrži X mora odgovarati podputanji od korijena do čvora $a_l (l \geq i_j)$ i prema tome, uvećava brojač potpore za čvor a_l za 1 prilikom konstrukcije. To znači da potpora čvora a_{i_j} nije veća od $sup(X)$. S druge strane, prema *Apriori* principu, potpora čvora a_{i_j} ne može biti veća od potpore skupa X. Slijedi da je $sup(X) = sup(a_{i_j})$. ■

Generiranje čestih uzoraka iz FP-stabla koje sadrži jedinstvenu prefiksnu putanju

Pretpostavimo da se FP-stablo T, slično onom na slici 4.3(a) sastoji od (1) jedinstvene prefiksne putanje P, slične onoj na slici 4.3(b) i (2) dijela s višestrukim putanjama, Q, koje možemo promatrati kao neovisno FP-stablo s pseudo-korijenom R, slično stablu Q na slici 4.3.

Potpuni skup čestih predmeta možemo dobiti na slijedeći način:

1. *Skup čestih predmeta dobivenih iz P dobijemo prebrojavanjem svih kombinacija predmeta duž putanje P, s potporom svakog od tih skupova postavljenom na vrijednost potpore čvora s najmanjom potporom unutar skupa.*
2. *Skup čestih predmeta dobivenih iz Q, uzimajući R kao "null" korijen.*
3. *Skup čestih predmeta dobivenih kombinacijom P i Q dobivenih kartezijevim produktom ta dva skupa, uz postavljanje potpore na manju vrijednost od dva kombinirana skupa.*

Dokaz ove tvrdnje znatiželjni čitatelj može pronaći u [7].

4.2.4 Algoritam rastućih skupova uzoraka

Navedena razmatranja dovode do konstrukcije novog algoritma za učenje učestalih skupova predmeta - Algoritam rastućih skupova uzoraka.

Algoritam 4 (Algoritam rastućih skupova predmeta) Učenje čestih skupova predmeta koristeći FP-stabla rastom dijelova uzoraka

Ulaz: Baza transakcija *TDB*, prezentirana *FP-stablom*, sagrađenim prema Algoritmu 3 te prag potpore ξ.

Izlaz: Potpuni skup čestih uzoraka

Poziv funkcije: *FP-growth (FP-tree, null).*

Funkcija *FP-growth (T, α)* {

(a) **Ako** T sadrži jedinstvenu prefiksnu putanju,

(b) **Tada** {

(c) uzmimo da je P dio stabla s jedinstvenom putanjom stabla T

(d) uzmimo da je Q dio s tabla s višestrukim putanjama, s korijenskim čvorom *null*

(e) **za svaku** kombinaciju β čvorova s putanje P **radi**

(f) **generiraj** uzorak $\beta \cup \alpha$ s *potpora= min potpora čvorova* β

(g) uzmimo da je *česti_ uzorci(P)* tako generiran skup čestih predmeta }

(h) **inače** uzmimo Q *stablo*

(i) **za svaki** predmet $a_i \in Q$ **radi** {

(j) **generiraj** uzorak $\beta = a_i \cup \alpha$ s *potpora* $= a_i.potpora$;

(k) **konstruiraj** β-inu uvjetnu bazu uzoraka i β-ino uvjetno stablo $Stablo_\beta$;

(l) **ako** $Stablo_\beta \neq \emptyset$

(m) **tada** pozovi *FP-growth (*$Stablo_\beta$*,* β*)*;

(n) uzmimo da je *česti_ uzorci(Q)* tako generiran skup čestih predmeta }

(o) **vrati** *česti_ uzorci(P)* $\cup$ *česti_ uzorci(Q)* $\cup$ *(*česti_uzorci(P) $\times$ *česti_ uzorci(Q)))*

Analiza. Prema svojstvima navedenim u ovom poglavlju, algoritam raščlanjuje stablo na dio s jedinstvenom prefiksnom putanjom i dio s višestrukom prefiksnom putanjom. Algoritam je potpun jer pronalazi potpun skup čestih skupova predmeta.

Obratimo pažnju na učinkovitost algoritma. Proces učenja ovim algoritmom prolazi kroz *TDB* jednom i generira malu bazu čestih predmeta B_{ai}. Za svaki od čestih predmeta konstruira se *FP-stablo*$|a_i$ prema opisanom algoritmu. Često je i početno stablo manje od baze, dok je svako od uvjetnih stabala $FP - stablo|a_i$ znatno manje od početnog. Isto tako su i uvjetne baze čestih predmeta manje od originalne baze čestih predmeta (uzoraka). Prema tome, svaki silazak u rekurziju radi sa sve manjim uvjetnim bazama i stablima. Proces učenja se sastoji od operacija podešavanja brojača potpore čvorova i spajanja čestih skupova. Te operacije su puno manje zahtjevne od generiranja i testiranja za veliki skup kandidata.

Vidimo da se algoritam temelji na principu *podijeli-pa-vladaj* i da je faktor smanjenja prilično velik. Ako je faktor smanjenja za konstrukciju osnovnog *FP-stabla* iz *TDB* između 20 i 100, za očekivati je faktor smanjenja od još nekoliko stotina puta pri konstrukciji svakog od uvjetnih *FP-stabala* iz već prilično male baze uzoraka.

Primjetimo da, čak i u slučaju da baza generira mnogo čestih uzoraka, veličina *FP-stabla* je obično vrlo mala. Na primjer, za česti skup predmeta duljine 100, $a_1 \ldots a_{100}$,

FP-stablo ima samo jednu putanju duljine 100, i to $\langle a_1 \to \ldots \to a_{100} \rangle$. Algoritam će i dalje generirati oko 10^{30} čestih uzoraka. Međutim, ako se stablo sastoji od samo jedne prefiksne putanje, nije potrebno konstruirati uvjetna stabla, već samo prebrojati česte uzorke.

Poglavlje 5

Programsko ostvarenje Algoritma rasta učestalih skupova predmeta

U sklopu rada je ostvaren Algoritam rasta učestalih skupova predmeta (FP-growth) algoritam u programskom jeziku Java. Programski jezik je odabran prvenstveno imajući na umu rasprostranjenost i platformsku neovisnost. Java, kao objektno orijentiran jezik zadovoljava kriterije platformske neovisnosti te je široko rasprostranjen i korišten na različitim operacijskim sustavima poput Unix-a, Linux-a, MAC OS-a i Windowsa. Programski kod je pisan u orkuženju *eclipse* koje je besplatno dostupno na www.eclipse.org. Programsko ostvarenje je napravljeno uz pomoć predložaka tako da je proizvoljan tip (razred) predmeta koji čine učestale skupove. Ostvarenje se sastoji od više razreda (engl. *class*).

5.1 Karakteristike ostvarenja

U slijedećim odlomcima su dane karakteristike i opisi svakog od spomenutih razreda.

5.1.1 Razred *Preprocessor*

Razred *Preprocessor* je izvršni program koji kao ulazne parametre prilikom pokretanja prima prag potpore *min_sup*, datoteku s transakcijama, ime datoteke u koju sprema tablicu preimenovanja i ime datoteke u koju sprema procesirane transakcije.

Kako bi se ostvarila dodatna optimizacija, implementirana je dodatna heuristična metoda. Naime, razred *Preprocessor* prolazi kroz originalnu transakcijsku bazu jednom, broji potporu svakog od predmeta te uklanja one kojima je potpora manja od danog praga. Nakon toga, preimenuje elemente tako da predmet s najvećom učestalošću

preimenuje u 1, drugi najčešći u 2 itd. Ovim načinom ostvaruje se veća kompaktnost rezultirajućeg FP-stabla jer očekujemo da će više čvorova stabla biti na manjim dubinama. Ova heuristika opisana je u literaturi [3]. Nakon preimenovanja, originalni broj predmeta (njegovo ime), novi broj (ime) i potpora se zapisuju u novu datoteku - tablicu preimenovanja predanu kao ulazni argument. Ulazne datoteke su nizovi brojeva odvojenih razmacima.

5.1.2 Razred *ItemSet*

ItemSet (prevedeno na hrvatski - skup predmeta) je jednostavan razred koji nasljeđuje *LinkedHashSet*<*T*>. Vidimo da je razred ostvaren pomoću predlošla (engl. *generics*); parametiziran je s argumentom *T* koji predstavlja tip podatka predmeta. Najčešće je korišten *Short*, kako bi se osigurala učinkovitost: vremenska i prostorna. Pošto razred nasljeđuje *LinkedHashSet*, tako nasljeđuje sve njegove metode. To znači da možemo Java virtualnom stroju (engl. *Java Virtual Machine - JVM*) prepustiti brigu o veličini spremnika u kojeg spremamo predmete razreda *T*. Ta memorija se dinamički zauzima i otpušta. Uz kolekciju predmeta, dodano je polje koje označava vrijednost potpore promatranog skupa predmeta.

5.1.3 Razred *PowerSet*

Razred *PowerSet* je također ostvaren pomoću predložaka - proizvoljnih skupova. Predstavlja implementaciju generiranja svih podskupova danog skupa. Pošto radi s predlošcima, nebitno je koji tip skupa mu se predaje kao argument. Korišten je prilikom generiranja svih kombinacija objekta razreda *ItemSet*<*T*>, nakon utvrđivanja jedinstvene prefiksne putanje unutar stabla razreda *FPTree*<*T*>.

5.1.4 Razred *FPTreeNode*

Ovaj razred predstavlja implementaciju samog čvora unutar FP-stabla. Također radi s predlošcima. Možemo ga promatrati kao jednostavnu strukturu koja sadrži objekt s imenom predmeta (sam tip ili razred objekta se predaje kao predložak), pokazivač na roditelja u stablu, pokazivače na djecu te brojač pojavljivanja tog čvora (koliko različitih putanja od korijena prolazi tim čvorom).

5.1.5 Razred *FPTree*

Razred *FPTree* predstavlja implementaciju FP-stabla i kao takav je, uz razred *FPGrowth*, okosnica cijelog ostvarenja. Sastoji se od polja korijenskog čvora, tablice pokazivača na prva pojavljivanja svakog od predmeta u stablu i tablice potpore za svaki od predmeta, kako je opisano u 4.1.2.

Glavne metode su:

1. `public void insertTransaction(List<T> transactionItems, FPTreeNode<T> parent);`

   Metoda rekurzivno umeće listu predmeta iz liste *transactionItems* u stablo. Početno se čvor *parent* postavlja na korijenski. Međutim, on se mijenja tako da predstavlja roditeljski čvor pri svakom sljedećem pozivu. Ukoliko se stvara novi čvor za neki od predmeta, onda se ažurira i tablica pokazivača na čvorove s tim predmetom. Inače se samo uvećava potpora promatranog čvora.

2. `public void pruneUnfrequentItems(short minSupport);`

   Ova metoda iz stabla uklanja čvorove koji imaju potporu manju od praga danim s *minSupport* te preslaže pokazivače kako bi očuvali konzistentnost stabla.

3. `public FPTree<T> getConditionalFPTree(T forItem, short minSupp);`

   Metoda izračunava uvjetnu bazu transakcija te konstruira uvjetno stablo. Transakcije i stablo nazivamo uvjetnima jer je njihovo pojavljivanje uvjetovano pojavljivanjem predmeta *forItem*. Doatni parametar *minSupp* predstavlja prag potpore za svaki od predmeta, kako bi se odmah uklonili oni koji nisu česti.

4. `public List<FPTree<T>> separateSingleMulti();`

   separateSingleMulti metoda rastavlja eventualni dio stabla s jedinstvenom prefiksnom putanjom i onaj s višestrukom. Ovo rastavljanje se obavlja s ciljem optimizacije jer se stablo s jedinstvenom prefiksnom putanjom brže razjršava.

5.1.6 Razred *FPGrowth*

Razred *FPGrowth* predstavlja implementaciju algoritma učenja učestalih skupova predmeta rastom uzoraka. Algoritam je opisan u 4.2.4, a glavni dio implementacije se odvija u funkciji

```
public Set<ItemSet<T>> mine(FPTree<T> tree, ItemSet<T> minedSet);
```

Metoda rekurzivno pokušava razdvojiti FP-stablo *tree* u dio s prefiksnom putanjom i dio s višestrukim prefiksnim putanjama. Ukoliko postoji dio s jedinstvenom prefiksnom putanjom, onda skupove dobijamo korištenjem razreda *PowerSet*, tj. generiranjem svih kombinacija predmeta na toj putanji. Pritom potporu dobivenog skupa uvijek izjednačavamo s najmanjom potporom elemenata koji čine taj skup. Prilikom učenja stabla s višestrukim putanjama, grade se uvjetna stabla za promatrane predmete. Ta stabla predajemo kao argument *tree* u sljedećim rekurzivnim pozivima, a (skup) predmeta za koji smo izgradili uvjetno stablo (zajedno s potporom tog skupa) predajemo kao argument *minedSet*. Ukupan rezultat se dobija kao skup rezultata dobivenih iz stabla s jedinstvenom putanjom dodanih skupu dobivenim iz stabla koje nema jedinstvenu putanju te na kraju dodamo kartezijev produkt ta dva skupa.

5.2 Primjer pokretanja

Nakon raspakiranja programske implementacije u direktorij po izboru i pozicioniranja u njega pomoću terminala ili komandne linije, program se prevodi pomoću Java prevodioca naredbom:

```
javac -cp bin -sourcepath src -d bin/ src/hr/fer/zemris/diplomski45/data/*
```

Nakon toga, potrebno je pokrenuti razred *Preprocessor* naredbom:

```
java -cp bin/ hr.fer.zemris.diplomski45.data.Preprocessoor a b c d
```

Argumente *a, b, c,* i *d* treba zamijeniti redom s pragom potpore, putanjom do datoteke s ulaznim podatcima, imenom datoteke u koju će se spremiti tablica preimenovanja elemenata te imenom datoteke u koju će se spremiti preprocesirana ulazna baza podataka (sadrži preimenovane predmete koji zadovoljavaju dani prag potpore). Primjer pokretanja je (sve u jednoj liniji):

```
java -cp bin/ hr.fer.zemris.diplomski45.data.Preprocessoor 2000
datasets/T40I10D100K.dat RenameTable.txt Preprocessed.txt
```

Primjer izlaza ovog programa:

```
Reading input data
Pruning...
Sorting...
Renaming...
Transforming...
Done!
```

Nakod predprocesiranja ulazne baze transakcija, potrebno je pokrenuti razred *Tester*:

```
java -cp bin/ hr.fer.zemris.diplomski45.data.Tester a b c d
```

Gdje su *a, b, c, d* redom prag potpore (identičan onom kojeg smo prethodno predali razredu *Preprocessor*), ime datoteke s tablicom preimenovanja predmeta, imenom datoteke s preprocesiranim ulaznim podatcima i imenom datoteke u koju se spremaju rezultati. Prema tome, primjer pokretanja može biti:

```
java -cp bin/ hr.fer.zemris.diplomski45.data.Tester 2500 RenameTable.txt
Preprocessed.txt FrequentItemsets.txt
```

Primjer izlaza ovog programa je:

```
Support is set to: 2500
Reading rename table...
Reading input database - building the Tree...
Finished reading TDB and constructing the Tree in 10.38 s
Calculating support...
Finished calculating support in 0.018 s
Starting mining procedure...
Finished mining in 35.448 s
Found 107 frequent patterns with support above 2500
Writing results to FrequentItemsets.txt
```

Poglavlje 6

Rezultati

U ovom poglavlju dan je pregled ispitivanja implementiranog algoritma. Ispitivanja su izvršena sa sintetičkim podatcima preuzetima s repozitorija podataka za metode strojnog učenja. Dana su vremena izvođenja za pojedine skupove ulaznih podataka s obzirom na pragove potpore. Uz to, izvršena je analiza zauzeća radne memorije u ovisnosti o pragovima potpore.

6.1 Okruženje

Programski kod je pisan u programskom okruželju eclipse (www.eclipse.org). Ispitivanja su izvršena na računalu s Intel Core2Duo^{TM}procesorom takta 2.27GHz, s 2GB radne memorije. Za operacijski sustav je korišten Ubuntu 10.04.LTS (www.ubuntu.com), s jezgrom operacijskog sustava verzije 2.6.32-22-generic.

6.2 Ulazni podatci

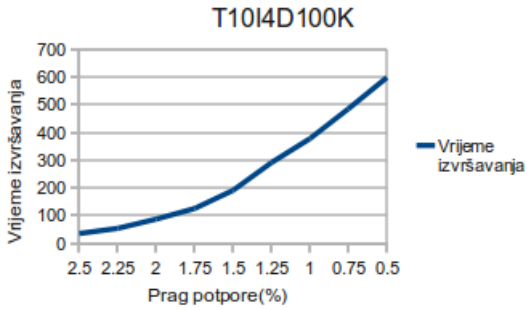
Za ispitivanje su korišteni standardni skupovi ulaznih podataka, korišteni i u drugim ispitivanjima ove vrste. Podatci su preuzeti s repozitorija http://fimi.cs.helsinki.fi/data/. Konkretno, riječ je o sintetički generiranim ulaznim podatcima.

Iz samog imena možemo saznati opis ulaznih podataka. Uzmimo za primjer datoteku T10I4D100K.dat. Iz imena redom isčitavamo da je to datoteka s prosječnom veličinom transakcija 10, prosječna veličina maksimalnog skupa čestih predmeta 4, da se u datoteci nalazi 1000 različitih predmeta ukupno složenih u 100 000 transakcija. Ova datoeka je veličine 3.8 MB. Česti skupovi predmeta su kratki i nema ih puno, u usporedbi s drugim datotekama. Slično tome, za datoteku T40I10D100K.dat vidimo da je prosječna veličina transakcije 40, prosječna veličina maksimalnog skupa čestih pred-

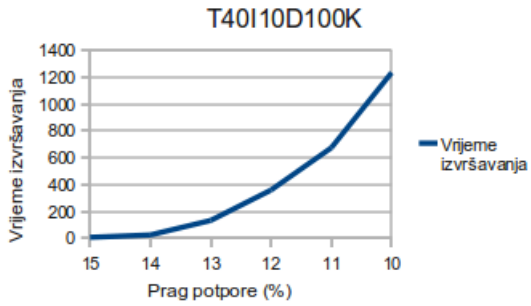
meta 10, da se sastoji od 1000 različitih predmeta, u 100 000 transakcija. Datoteka je velika 14.8 MB.

6.3 Analiza skalabilnosti

Rezultate vremena izvođenja, koji ne uključuju vrijeme predprocesiranja, već samo vrijeme traženja čestih uzoraka, možemo vidjeti na slikama 6.1 i 6.2. Rezultati su komparativno lošiji od onih postignutih s najoptimiranijim algoritmima prema [2] ili [12], ali za potrebe analize uloga za pristup bazama podataka su dovoljno učinkoviti.



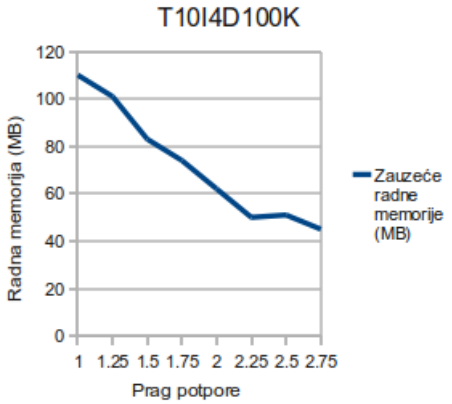
Slika 6.1. *Ovisnost vremena izvođenja o pragu potpore - T10I4D100K*



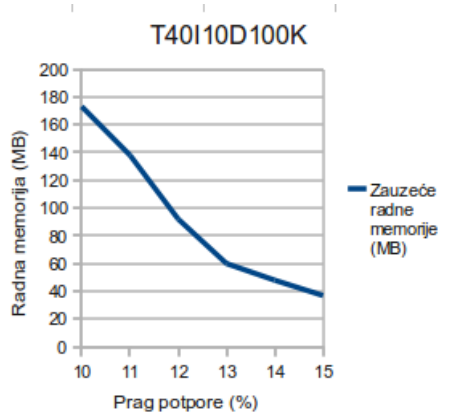
Slika 6.2. *Ovisnost vremena izvođenja o pragu potpore - T40I10D100K*

Vidimo da vrijeme izvršavanja eksponencijalno raste o smanjenju praga potpore.

Na slici 6.3 vidimo ovisnost zauzeća radne memorije o pragu potpore za ulazni skup T10I4D100K, dok na slici 6.4 vidimo isto za skup T40I10D100K. O zauzeću memorije valja napomenuti da je veličina mjerena nakon izvršenja dretve koja se brine o potrošnji memorije (engl. *garbage collector*). Maksimalna (engl. *peak*) potrošnja memorije je nešto veća. Prilikom pokretanja programa, naredbama za *java virtualnu mašinu* (-Xms1024m -Xmx1024m) se omogućilo programima zauzeće do 1GB radne memorije.

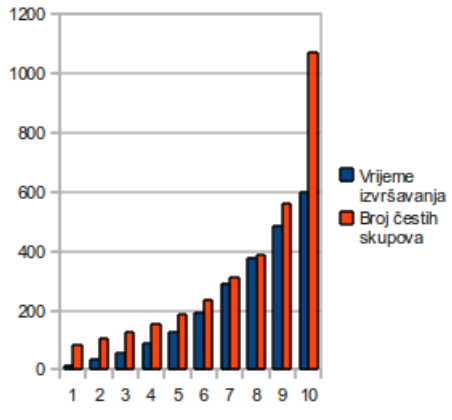


Slika 6.3. *Ovisnost zauzeća radne memorije o pragu potpore - T10I4D100K*

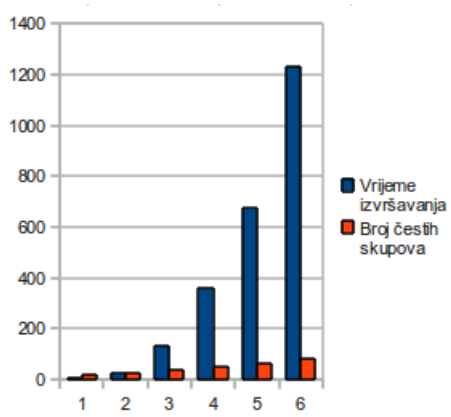


Slika 6.4. *Ovisnost zauzeća radne memorije o pragu potpore - T40I10D100K*

Zauzeće memorije je mjereno pomoću alata *JConsole* verzije 1.6.0_20-b02, nakon pozivanja dretve koja oslobađa nepotrebno zauzetu memoriju.



Slika 6.5. *Prikaz vremena izvršavanja i broja pronađenih čestih skupova - T10I4D100K*



Slika 6.6. *Prikaz vremena izvršavanja i broja pronađenih čestih skupova - T40I10D100K*

Na slikama 6.5 i 6.6 vidimo razliku između ova dva ispitna skupa podataka. Za prvi skup vidimo da je potrebno manje vremena za utvrđivanje svakog od čestih skupova predmeta, dok je taj posao zahtjevniji u skupu s više predmeta po transakciji.

6.4 Analiza uloga za pristup bazi podataka

U sklopu rada je obavljena analiza manje skupine od 17 uloga za pristupu DWH (engl. *data warehousing*) bazi podataka. Uloge u ovom sustavu same po sebi nemaju definirana prava pristupa poput čitanja, pisanja i brisanja, već se prati popis objekata na kojima ima pravo pristupa. Izvršena je analiza postojećeg sustava na temelju koje su uloge reorganizirane i njihov broj je smanjen na 14. Utvrđeno je da je jedna uloga podskup druge (riječ je o hijerarhijskim ulogama za unošenje novih korisnika). Daljnje smanjene je, između ostalog, omogućeno utvrđivanjem velikog stupnja potpore od 48% za peteročlani skup objekata koji sačinjava veći postotni udio u jednom dijelu uloga. Taj podskup je proglašen novom ulogom, a ostale su promijenjene.

Ovo je primjer u kojem je jednostavno preslikavanje iz opisa uloge u transakcije opisane u primjerima sa sintetičkim ulaznim podatcima - jednostavno zamijenjujemo imena objekata s rednim brojevima. U slučaju definiranja npr. tablica i njihovih prava pristupa, predloženo je kodiranje na način da viši bitovi označavaju tablicu dok niža 3 bita (po potrebi i više) definiraju prava pristupa poput: čitanje, pisanje, čitanje i pisanje, davanje prava pristupa drugim korisnicima itd. Ukoliko se koristi ovakav način kodiranja, onda je potrebna dodatna obrada izlaznih podataka ovog sustava.

6.5 Budući rad

Predloženi sustav ostvaruje performanse koje su dovoljne za analizu uloga za pristup bazama podataka. Međutim, sustav je primjenjiv u analizi potrošačkih košarica koje zahtjevaju veće ulazne skupove podataka. Mjesta za poboljšanje performansi ima, te bi izvorni kod trebalo revidirati kako bi učinkovitije radio s većim ulaznim skupovima. Uz to, predloženi sustav predstavlja okosnicu sustava za analizu uloga pristupa bazama podataka. U tom sustavu potrebno je omogućiti lakši rad za krajnjeg korisnika implementacijom grafičkog sučelja (engl. *GUI - Graphical User Interface*). Tu vidimo kompromise korištenjem programskog jezika Java: Možda konačna ispitivanja brzine revidiranog izvornog koda neće biti u rangu najboljih trenutno dostupnih, ali je izrada uslužnog programa jednostavnija i isplativija u Javi te nadomješćuje moguće nedostatke u brzini izvođenja.

Poglavlje 7

Zaključak

U ovom radu dan je opis i ostvarenje algoritma za analizu supova čestih elemenata. Primjene ove metode pronalazimo u učenju asocijacijskih pravila, učenju sekvencijskih pravila, višerazinskih asocijacijskih pravila i sl.

U ispitivanjima smo utvrdili da za skupove npr. 100 000 transakcija s prosječnom duljinom transakcije od 10 elemenata i potporom od 0.5% izvođenje traje oko 10 minuta.

Primjenom ovog modela na uloge za pristup bazama podataka, možemo otkriti pravilnosti unutar samih uloga. Čest je slučaj da se sustavi temeljeni na bazama podataka nadograđuju i međusobno spajaju u složenije. Tad je potrebno napraviti reviziju postojećih uloga kako bi se osigurala konzistentnost u dodjeljivanja pristupa podatcima i osigurala zadovoljavajuća razina sigurnosti. Ovaj model omogućuje analizu postojećih uloga i na temelju nje se može napraviti revizija.

Bibliografija

[1] *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing Chichester, UK, 2007.

[2] A. B Abdullah A. M. Said, D. Dominic: *A Comparative Study of FP-growth Variations*. International Journal of Computer Science and Network Security, 2009.

[3] LUCS KDD research group: *The LUCS-KDD implementation of the FP-Growth algorithm*. http://www.csc.liv.ac.uk/~frans/KDD/Software/FPgrowth/fpGrowth.html, May 2010.

[4] Wikimedia Fundation inc. http://en.wikipedia.org/wiki/Machine_learning, May 2010.

[5] Pier Luca Lanzi: *Association rule mining*. PDF lectures, Politechnico di Milano.

[6] Michael Lesk: *How Much Information Is There In the World?* http://www.lesk.com/mlesk/ksg97/ksg.html, May 2010.

[7] Jian Pei: *Pattern-growth methods for frequent pattern mining*. Disertacija, Computing Science, Simon Fraser University, 2002.

[8] Hal R. Varian Peter Lyman: *How Much Information?* http://www2.sims.berkeley.edu/research/projects/how-much-info-2003/, May 2010.

[9] T. Imielinski R. Agrawal i A. Swami: *Mining association rules between sets of items in large databases*. U *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data (SIGMOD'93)*, 1993.

[10] Ramakrishnan Srikant Rakesh Agrawal: *Fast algorithms for mining association rules*. U *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, 1994.

[11] Ramakrishnan Srikant Hannu Toivonen A. Inkeri Verkamo Rakesh Agrawal, Heikki Mannila: *Fast discovery of association rules*. U *Advances in knowledge discovery and data mining*, stranice 307 – 328, 1996, ISBN 0-262-56097-6.

[12] Walter A. Kosters Wim Pijls: *How to find frequent patterns?* Tehnički izvještaj, Econometric institute, 2005.

Sažetak

Primjena metoda umjetne inteligencije na povećanje sigurnosti uloga za pristup bazama podataka

S obzirom na brzinu rasta informacija i količinu pohrane istih, korisno je analizirati postojeće podatke u potrazi za pravilnostima. U analizi potrošačkih košarica se pokušava utvrditi koje grupacije artikala kupci često kupuju zajedno. Pri toj analizi se koriste algoritmi za učenje učestalih skupova. U ovom radu dan je pregled nekih od algoritama korištenih za rješavanje ovog, i sličnih problema. Ponuđeno je programsko ostvarenje napisano u programskom jeziku Java, temeljeno na Algoritmu rasta učestalih skupova predmeta. Obavljena je analiza ispitivanja zahtjevnosti izvođenja na sintetičkim ulaznim podatcima. Predložena je uporaba u unapređenju sustava uloga za pristup bazama podataka. Budući da je su pravilno uređene uloge jedan od osnovih preduvjeta za sigurnost baze i podataka spremljenih u njoj, analiza ostvarena ovim sustavom može doprinjeti povećanju sigurnosti.

Ključne riječi

Strojno učenje, Učenje asocijacijskih pravila, Učenje skupova čestih predmeta, Analiza uloga za pristup bazama podataka, Algoritam rastućih skupova uzoraka, Apriori algoritam

Summary

Database roles security enhancement using Artificial intelligence methods.

Considering the speed growth of generated information and the growth of total data stored, it is useful to analize data in search of patterns (regullarities). In basket case analysis the goal is to determine wich products do costumers buy together. Algorithms for mining frequent itemsets are used for this purpose. This paper offers a solution to this problem based on Frequent Pattern Growth Algorithm - FP-Growth. Implementation, written in Java programming language is presented, for wich the run time analysis is given. Testing has been run on sintetic data frequently used as benchmark data for machine learning tools. This implementation is offered as a tool for database roles analysis. Database security relies, in great part, on properly formed roles. Therefore database roles analysis using this tool can contribute to whole database security.

Keywords

Machine learning, Association rule mining, Frequent itemset mining, Database roles analisis, Frequent Pattern Growth Algorithm, FP-Growth, Apriori Algorithm