SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5926

ANALIZA POTROŠAČKE KOŠARICE KORIŠTENJEM ALGORITMA FP-GROWTH

Borna Popović

Zagreb, lipanj 2019.

Zahvaljujem se svom mentoru, doc. dr. sc. Alanu Joviću, koji mi je pomogao pri izradi ovog završnog rada svojim savjetima i dodatnom literaturom te imao strpljenja i vremena odgovarati na moje upite.

Mentor: doc. dr. sc. Alan Jović
Broj stranica: 27

SADRŽAJ

1. UVOD

Dubinska analiza podataka (eng. *Data mining*) je sortiranje, organiziranje ili grupiranje velikog broja podataka i izvlačenje relevantnih informacija [13]. Jedan od primjera dubinske analize podataka je analiza potrošačke košarice. To je primjer analitičke tehnike kojom se koriste trgovci kako bi bolje razumjeli ponašanje kupaca pri kupnji. Koristi se kako bi se pronašli predmeti koji se često pojavljuju zajedno u transakcijama. Naprimjer, analizom podataka s računa iz neke trgovine trgovac može utvrditi koji artikli se često prodaju zajedno. To im može omogućiti da odrede raspodjelu artikala po trgovini, postavljanje specifičnih popusta na pojedine artikle iz skupa artikala koji se često prodaju zajedno, davanje raznih kupona kojima će privući ljude i navesti ih da potroše više nego što su početno planirali, itd. Dubinska analiza podataka također se može koristiti i u telekomunikacijama da se odredi koje pakete i ponude ljudi često kupuju i koriste zajedno što omogućuje trgovcu da usmjeri marketing prema klijentima za koje je vjerojatnije da će koristiti te pakete i pogodnosti davajući im razne ponude. Kartične kuće analiziraju kupnje svojih korisnika kako bi im izgradili vlastite profile pomoću kojih mogu uočiti pokušaje prijevare i mogu blokirati karticu u slučaju krađe ako primjete nesvakidašnje transakcije.

Za generiranje čestih podskupova u skupovima podataka koriste se razni algoritmi, a neki od poznatijih su algoritmi *Apriori* i *FP-Growth.* Algoritam *Apriori* generira jedinične skupove predmeta, zatim parove pa trojke, itd. U svakom koraku izbacuje elemente koji ne zadovoljavaju minimalni broj ponavljanja u skupu podataka. Algoritam *FP-Growth* koristi strukturu podataka *FP-Tree* u kojoj svaki čvor predstavlja neki artikl i ima svoj broj ponavljanja, a svaka grana predstavlja različitu asocijaciju. Najveća prednost algoritma *FP-Growth* je to što on mora pročitati datoteku samo dvaput, dok *Aprio*ri to mora obaviti prilikom svake iteracije.

U ovom završnom radu posvetit ću se obradi i implementaciji algoritma *FP-Growth* te primjeni jednostavnog postupka za generiranje asocijativnih pravila na temelju dobivenih čestih podskupova artikala.

2. ALGORITAM FP-GROWTH

2.1. Konstrukcija strukture podataka *FP-Tree*

FP-Tree je sažeti prikaz ulaznih podataka. Stablo je konstruirano čitanjem jedne po jedne transakcije i zabilježavanjem svake transakcije u granu stabla. Transakcije često imaju neke zajedničke artikle pa se njihovi putovi u stablu preklapaju i time se ostvaruje sažimanje velike količine podataka. Nakon toga, česte podskupove generiramo direktno iz konstruiranog stabla [2].

U nastavku je prikazan detaljan postupak konstrukcije *FP-Tree*-a za zadani skup podataka:

TID	Artikli u transakciji
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

Tablica 1. Skup transakcija [2]

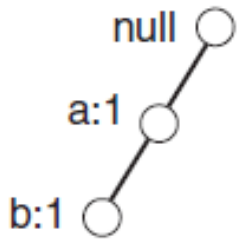
Pri prvom prolasku kroz skup transakcija sumira se pojavljivanje svakog artikla i broj pojavljivanja uspoređuje se s minimalnim pragom podrške (*support threshhold*). Svaki artikl koji ne zadovoljava zadani prag se izbacuje, a ostali se sortiraju silazno prema broju pojavljivanja, što je prikazano u tablici 2 [2].

Artikl	Broj ponavljanja
{a}	8
{b}	7
{c}	6
{d}	5
{e}	3

Tablica 2. Broj pojavljivanja pojedinih artikala iz skupa

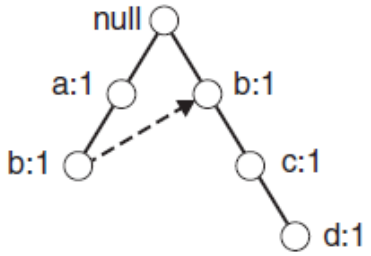
Iz tablice 2 vidimo da se artikl *a* najčešće pojavljuje, a iza njega slijede redom artikli *b*, *c*, *d* i na kraju *e*.

U idućem koraku algoritam drugi put prolazi kroz skup podataka iz datoteke kako bi konstruirao stablo (*FP-Tree*). Algoritam čita transakciju po transakciju, dodaje artikle u stablo i pridružuje im odgovarajuću vrijednost. U svakoj transakciji algoritam sortira pojedine artikle iz transakcije po broju pojavljivanja (eng. *supp. count*) u skupu podataka i tim redoslijedom ih umeće u stablo [2].



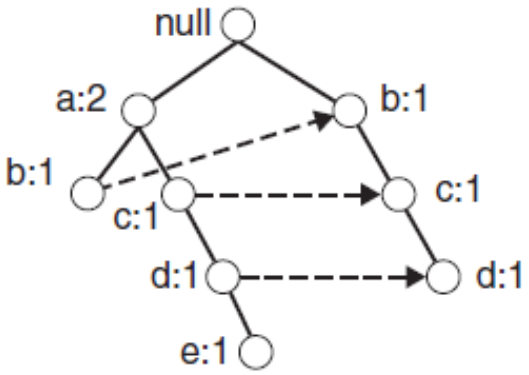
Slika 1. Nakon čitanja TID=1 [2]

Slika 1 prikazuje čvorove koji su konstruirani nakon čitanja prve transakcije, {*a*, *b*}, iz datoteke. Konstruirani su čvorovi *a* i *b* s brojem pojavljivanja koji je zasad jedan.



Slika 2. Nakon čitanja TID=2 [2]

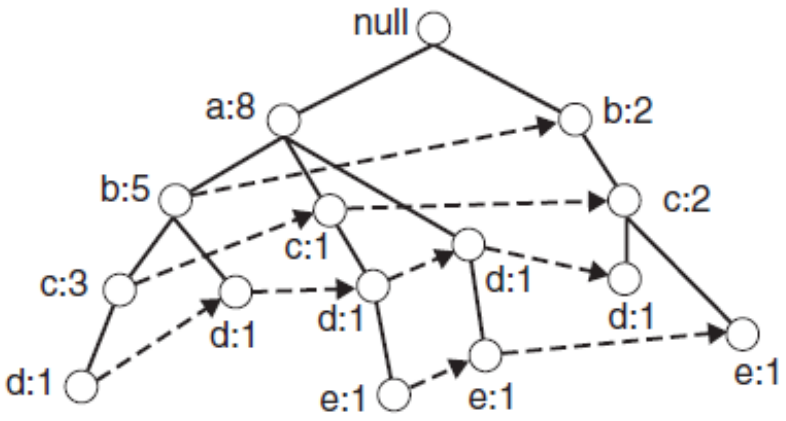
Nakon čitanja druge transakcije, {*b*, *c*, *d*}, konstruirani su novi čvorovi, slika 2. Možemo uočiti da se u obje transakcije pojavljuje artikl *b,* no njihovi putovi se ne podudaraju jer imaju različite prefikse (artikl sa najvećim brojem ponavljanja) [2].



Slika 3. Nakon čitanja TID=3 [2]

Čitanjem treće transakcije, {*a*, *c*, *d*, *e*}, dodani su čvorovi kao što je prikazano na slici 3. U ovom slučaju možemo primjetiti da transakcija dijeli zajednički prefiks s prvom transakcijom što dovodi do preklapanja putova tih dviju transakcija [2]. Zbog preklapanja se zajednički čvor *a* transakcija uvećava za jedan, dok su čvorovi *c*, *d* i *e* konstruirani s brojem ponavljanja jednakim jedan. U slučaju da je treća transakcija imala i artikl *b,* došlo bi do još jednog preklapanja te bi se broj ponavljanja čvora *b* također uvećao za jedan, a ostali čvorovi bi se nastavili na čvor *b*.

Na ovaj način obrađuje svaku transakciju koju pročitamo iz datoteke i na kraju dobivamo konačno stablo prikazano na slici 4.



Slika 4. Nakon čitanja TID=10 [2]

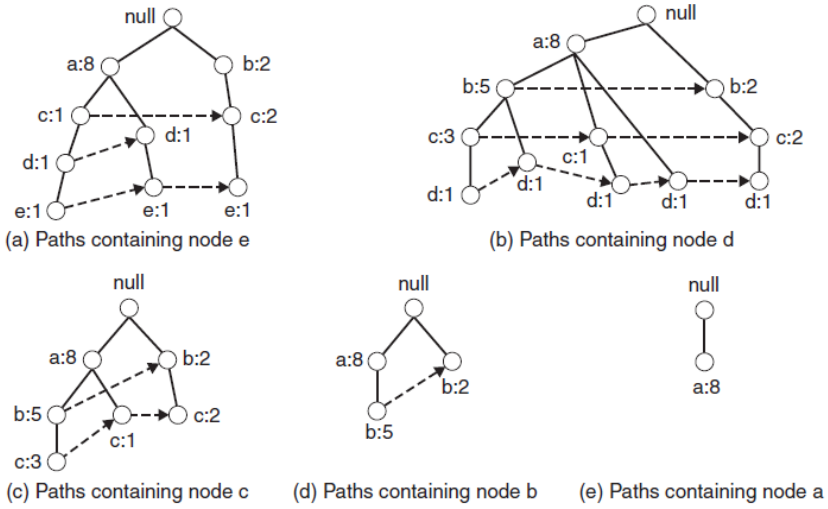
Veličina stabla je obično manja u usporedbi s datotekom iz koje čitamo podatke jer se u dosta transakcija pojavljuju zajednički artikli [2]. Međutim, prostor za fizičku pohranu stabla je veći jer moramo spremati dodatne informacije kao što su broj ponavljanja pojedinog artikla i pokazivači između čvorova.

Pokazivači povezuju čvorove koji predstavljaju iste artikle i time omogućuju brzo kretanje po stablu, odnosno brz pristup pojedinim artiklima u stablu. Oni također imaju utjecaj kod generiranja čestih podskupova artikala [2].

2.2 Generiranje čestih podskupova

Česti podskupovi generiraju se iz stabla pretraživanjem od dna prema vrhu. U navedenom primjeru algoritam prvo traži podskupove u kojima se pojavljuje artikl *e*, zatim *d*, *c*, *b* i na kraju *a*, tj. po broju ponavljanja iz tablice 2, krenuvši od najmanjeg. Budući da su sve transakcije zabilježene u stablu, možemo lako pronaći sve moguće putove koji završavaju sa pojedinim artiklom.
Proučavanjem putova koji sadrže artikl *e* pronalazimo sve putove koji završavaju

s tim artiklom. Pristup putovima je brz i efektivan zbog pokazivača povezanih sa čvorom *e*. Isti postupak primjenimo za sve ostale artikle u navedenom redoslijedu. Izvedeni putovi prikazani su na slici 5.



Slika 5. Raspodjela problema na podprobleme pri generiranju čestih podskupova [2]

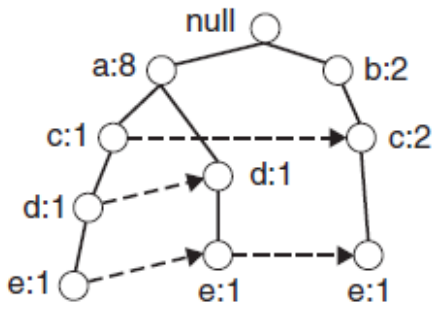
Ovime smo inicijalni problem raspodijelili na više manjih podproblema. Nakon toga rješavamo svaki pojedini podproblem i na kraju dobivamo česte podskupove za odgovarajuće sufikse. Sufiksi su poredani prema broju ponavljanja pojedinog artikla od najmanjeg prema najvećem. Tablica 3 prikazuje ove podatke.

Sufiks	Česti podskupovi
e	{e}, {d, e}, {a, d, e}, {c, e}, {a, e}
d	{d}, {c, d}, {b, c, d}, {a, c, d}, {b, d}, {a, b, d}, {a,d}
c	{c}, {b, c}, {a, b, c}, {a, c}
b	{b}, {a, b}
a	{a}

Tablica 3. Generirani podskupovi [2]

Za pronalaženje čestih podskupova za pojedine sufikse, algoritam koristi strategiju podijeli pa vladaj, tj. problem se raspodijeli na više manjih podproblema [2]. Konkretan postupak pokazat ćemo za sufiks *e.* Znači cilj nam je pronaći sve česte podskupove koji završavaju na *e.*

Prvo što moramo napraviti jest pronaći sve putove u stablu koji sadrže čvor *e*. Ti putovi nazivaju se prefiksni putovi. Slika 6 ih prikazuje.



Slika 6. Prefiksni putevi koji sadrže čvor *e* [2]

Iz prefiksnih putova vidimo da je broj ponavljanja čvora *e* jednak 3. Ako za minimalni prag podrške uzmemo broj 2, podskup {*e*} možemo deklarirati kao česti podskup [2].

2.2.1 Konstrukcija uvjetnog FP-stabla

Kako je {e} česti podskup, algoritam mora podijeliti taj problem na podprobleme koji su pronalazak čestih podskupova koji završavaju na *de*, *ce*, *be* i *ae*. Prvo što treba napraviti je pretvoriti prefiksne puteve u uvjetno FP-stablo (eng. *conditional FP-tree*) [2]. Uvjetno FP-stablo nam omogućuje pronalazak čestih podskupova za pojedine sufikse.

Kako gledamo samo transakcije koje sadrže artikl *e,* moramo ažurirati naše prefiksne putove i izbaciti transakcije koje ne sadrže artikl *e.* Naprimjer, na slici 6

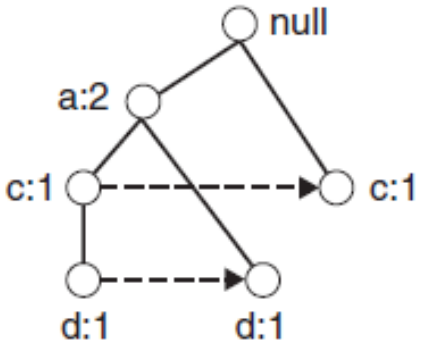
u desnoj grani imamo put: **null** ⟶ **b:2** ⟶ **c:2** ⟶ **e:1** [2]. Taj put također sadrži transakciju {*b*, *c*} koju moramo izbaciti jer ne sadrži artikl *e*. Prema tome prefiksni put mora biti prilagođen tako da umanjimo vrijednosti čvora *c* i *b* za 1.

Nakon što smo uklonili sve transakcije i ažurirali vrijednosti čvorova, moramo još ukloniti čvorove *e* iz prefiksnog stabla.

Čvorove koji sadrže artikl *e* možemo ukloniti zato što su sve vrijednosti ažurirane tako da se odnose samo na transakcije koje sadrže navedeni artikl. Podproblemi koje moramo riješiti također više ne trebaju informaciju o čvoru *e*.

Nakon ažuriranja vrijednosti postoji mogućnost da neki artikli više nisu česti, tj. da je njihov broj pojavljivanja manji od minimalnog praga podrške. Naprimjer, čvor *b* pojavljuje se samo jednom što znači da postoji samo jedna transakcija koja sadrži čvor *b* i *e*. Kako smo postavili minimalni prag podrške na 2, možemo izbaciti čvor *b* i ignorirati ga u daljnjoj analizi vezanoj za artikl *e*.

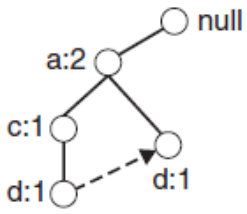
Konačno uvjetno FP-stablo za artikl *e* prikazano je na slici 7. Možemo primjetiti da stablo izgleda drugačije nego originalno jer smo ažurirali vrijednosti, uklonili čvorove *e* i izbacili čvorove koji ne zadovoljavaju minimalni prag podrške [2].



Slika 7. Uvjetno FP-stablo za *e* [2]

2.2.2 Generiranje čestih podskupova pomoću uvjetnog FP-stabla

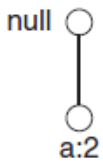
Algoritam FP-growth koristi uvjetno FP-stablo za artikl kako bi riješio podprobleme pronalaska čestih podskupova artikala. U našem slučaju za artikl *e* tražimo česte podskupove koji završavaju na *de*, *ce* i *ae*. Da bi pronašli česte podskupove koji završavaju na *de*, moramo iz uvjetnog FP-stabla za artikl *e* odabrati sve prefiksne putove koji sadrže *d*, a to je prikazano na slici 8 [2] .



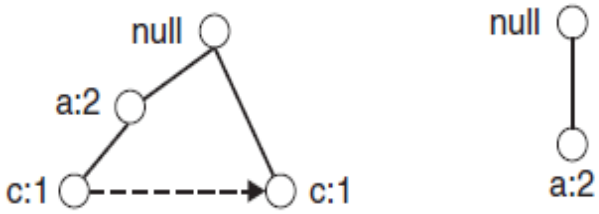
Slika 8. Prefiksni putovi koji završavaju na *de* [2]

Dodavajući broj pojavljivanja povezan sa čvorom *d,* dobivamo broj pojavljivanja podskupa {d, e}. Budući da je u našem slučaju minimalni prag podrške 2, podskup je deklariran kao česti.

Nakon toga algoritam konstruira uvjetno FP-stablo za *de* na isti način kao što je prikazano u prethodnom poglavlju. Nakon ažuriranja vrijednosti čvorova i izbacivanja čvorova sa manjim brojem pojavljivanja od minimalnog praga dobivamo stablo prikazano na slici 9 [2]. U ovom slučaju stablo sadrži samo artikl *a*, čija je vrijednost jednaka minimalnom pragu, pa algoritam dohvaća česti podskup {*a*, *d*, *e*} i nastavlja na idući podproblem.



Slika 9. Uvjetno FP-stablo za *de* [2]



Slika 10. Prefiksni putevi koji završavaju na *ce*, *ae* [2]

Ostaje nam za pronaći česte podskupove koji završavaju na *ce* i *ae*. Nakon obrade podproblema za *ce*, podskup {c, e} je jedini česti podskup. U podproblemu *ae* pronašli smo također samo jedan česti podskup, a to je {*a*, *e*}. Uvjetna FP-stabla oba podproblema prikazana su na slici 10 [2].

Konačno za artikl *e* dobili smo podskupove {*e*}, {*d*, *e*}, {*a*, *d*, *e*}, {*c*, *e*}, {*a*, *e*} kao što je prikazano u tablici 3. Isti postupak primjenjujemo i na sve ostale sufikse iz tablice kako bi odredili sve česte podskupove za zadani skup transakcija.

3. ASOCIJATIVNA PRAVILA

Asocijativno pravilo je implikacija izraza gdje su *X* i *Y* nepovezani skupovi. Snaga asocijativnog pravila mjerena je na temelju podrške (eng. *support*) i uvjerenosti (eng. *confidence*). Podrška određuje koliko je često pravilo prihvatljivo za dani skup podataka, dok uvjerenost određuje koliko se često artikli iz *Y* skupa pojavljuju u transakcijama koje sadrži *X* skup [2]. Formalne definicije prikazane su ispod navedenim formulama.

$$\text{Support, } s(X \longrightarrow Y) \quad = \quad \frac{\sigma(X \cup Y)}{N}$$

Formula 1. [2]

$$\text{Confidence, } c(X \longrightarrow Y) \quad = \quad \frac{\sigma(X \cup Y)}{\sigma(X)}$$

Formula 2. [2]

Razmotrimo jedno jednostavno pravilo, {Špinat, Pomfrit} ⟶{Sladoled}. Ako nam je broj pojavljivanja (eng. *support count*) za {Špinat, Pomrit, Sladoled} bio 5 i ukupan broj transakcija 15, time je prema formuli podrška (eng. *support*) jednaka 5/15 = 0,33. Uvjerenost (eng. *confidence*) pravila računamo tako da podijelimo broj pojavljivanja podskupa {Špinat, Pomrit, Sladoled} s brojem pojavljivanja {Špinat, Pomfrit} koji je u ovom slučaju 7. Prema tome uvjerenost pravila je 5/7 = 0,71.

Podrška je vrlo važna mjera zato što pravilo s jako niskom vrijednošću podrške može se jednostavno pojaviti slučajno. Pravilo niske podrške također uglavnom nije interesantno iz poslovne perspektive zato što se poslodavcima ne isplati promovirati artikle koje kupci jako rijetko kupuju zajedno. Zbog toga nam podrška služi kako bi eliminirali pravila koja nam nisu interesantna.

Uvjerenost mjeri pouzdanost zaključka izvedenog iz pravila. Za dano pravilo, $X \rightarrow Y$, veća vrijednost uvjerenosti znači da je veća vjerojatnost pojavljivanja *Y* u transakciji koja sadrži *X* [2].

Definicija 1. Pronalaženje asocijativnih pravila: Za zadani skup transakcija *T*, pronađi sva pravila koja imaju mjeru podrške veću ili jednaku minimalnom pragu podrške (eng. *support* ≥ *minsupport*) i uvjerenost veću ili jednaku od zadane minimalne (eng. *confidence* ≥ *minconf*) [2].

Ovakav pristup pronalaska asocijativnih pravila je pretjerano skup zato što broj mogućih pravila koje možemo pronaći u skupu podataka raste eksponencijalno. Zbog toga najčešći i najučinkovitiji pristup je rastaviti problem pronalaska asocijativnih pravila na pronalazak čestih podskupova artikala i generiranje pravila. Za pronalazak čestih podskupova zadamo određeni minimalni prag podrške (eng. *minsup threshold*) i iskoristimo neki od algoritama za generiranje čestih podskupova. U ovom završnom radu koristi se ranije opisani algoritam *FP-growth*. Nakon toga iz pronađenih čestih podskupova generiramo pravila

koja uobičajeno imaju visoku vrijednost uvjerenosti (eng*. high-confidence*). Ta pravila nazivamo jaka asocijativna pravila.

4. IMPLEMENTACIJA

Implementacija započinje sa skupom podataka koje korisnik predaje algoritmu. Ti podaci prije toga moraju biti očišćeni i pravilno oblikovani kako bi ih algoritam mogao obrađivati. Nakon toga trebamo implementirati algoritam FP-growth kojem predaje skup transakcija i koji iz danog skupa generira najčešće podskupove za zadanu minimalnu podršku (eng. *minsup*) . Te podskupove predajemo funkciji za generiranje asocijativnih pravila koja onda generira jaka asocijativna pravila na temelju zadane minimalne uvjerenosti (eng. *minconf*). Implementacija je napisana u programskom jeziku *python*.

4.1 Implementacija FP-growth algoritma

Kao što je prije navedeno algoritam FP-growth za zadani skup transakcija konstruira FP-stablo. Za to su nam potrebna dva razreda, čvor stabla i stablo. Svaki čvor mora imati svoj naziv koji predstavlja artikl o kojem je riječ, svoju vrijednost ponavljanja, čvor koji predstavlja njegovog roditelja u stablu, listu čvorova djece i pokazivač na sebi sličan čvor koji omogućuje efektivniju pretragu po stablu. Razred također ima metodu za ispis stabla. Razredu stablo predajemo minimalni prag podrške (eng*. minsup threshold*), skup transakcija u obliku liste i inicijalne vrijednosti korijena stabla. Na temelju tih podataka razred u svojoj inicijalizaciji prvo prebraja broj ponavljanja svakog pojedinog artikla te ih sprema u rječnik čestih artikala u kojem ključ predstavlja ime pojedinog artikla, a vrijednost broj njegovih ponavljanja u svim transakcijama. Sve artikle koji ne zadovolje minimalni prag podrške izbacuje se iz rječnika. Taj proces obavljaju dvije statičke metode: *pronađi_ponavljanja* i *izbaci_manjeOd_support*, slika 11*.*

```
@staticmethod
def pronadi_ponavljanja(itemsets,min_support):
    itemsDic ={}

    for items in itemsets:
        for item in items:
            if item in itemsDic:
                itemsDic[item] +=1
            else:
                itemsDic[item] = 1

    Stablo.izbaci_manjeOd_support(itemsDic,min_support)


    return itemsDic

@staticmethod
def izbaci_manjeOd_support(itemsDic,min_support):
    for name in list(itemsDic.keys()):
        if itemsDic[name] < min_support:
            del itemsDic[name]
    return itemsDic
```

Slika 11. Funckije za pronalazak broja pojavljivanja pojedinih artikala

Također moramo inicijalizirati tablicu *header* koja će nam omogućiti brzi pristup svim elementima zadanog tipa. Nakon toga konstruirat ćemo stablo. U zadanom skupu transakcija sortirat ćemo artikle u svakoj transakciji prema njihovom broju pojavljivanja u skupu podataka. To ćemo obaviti u funkciji *izgradi_stablo* prikazanoj na slici 12 koja poziva funkciju *dodaj_u_stablo* koja obrađuje svaku transakciju iz liste sortiranih transakcija i umeće ju u stablo te samu sebe ponovno poziva dok nije obradila sve transakcije.

```
def izgradi_stablo(self,itemsets,k_ime,k_broj,cesti_itemi,header_table):
    korijen = Cvor(k_ime,k_broj,None)

    for itemset in itemsets:

        sorted_items=[x for x in itemset if x in cesti_itemi]
        sorted_items.sort(key = lambda x: cesti_itemi[x],reverse=True)

        if sorted_items:
            self.dodaj_u_stablo(sorted_items,korijen,header_table)

    return korijen
```

Slika 12. Funkcija za konstruiranje FP-stabla

Nad tako definiranim primjerkom razreda stablo pozivamo funkciju *pronađi_obrasce* koja vraća riječnik u kojem su pohranjeni svi česti podskupovi za zadani minimalni prag podrške. Funkcija prvo provjerava ima li stablo samo jednu granu, a to će biti slučaj kasnije jer primjenjujemo strategiju podijeli pa

vladaj pa problem rastavljamo na puno manjih podproblema iz kojih onda možemo generirati obrasce. Ako smo u nekom od generiranih uvjetnih FP-stabala, sufiks čije smo uvjetno stablo konstruirali također predstavlja obrazac pa i njega dodajemo u rječnik obrazaca. U suprotnom poziva se metoda *pretraži_podstablo* koja generira uvjetno FP-stablo i u njemu pronalazi česte podskupove, odnosno obrasce. Prvo se česti artikli sortiraju prema broju pojavljivanja uzlazno i tim redoslijedom generiramo uvjetna podstabla i obrasce. Za svaki česti artikl (sufiks) provjeravamo grane u kojemu se on nalazi i pratimo put od tog artikla do korijena te ga pohranjujemo u pomoćnu listu. Pomoću te liste i zadanog minimalnog praga podrške stvaramo uvjetno FP-stablo kojemu kao korijen postavljamo sufiks. Nad tim stablom onda pozivamo funkciju *pronađi_obrasce* koja će vratiti rječnik generiranih obrazaca tog podstabla. Nakon toga pozivamo metodu *unesi_nove_obrasce* koja će ažurirati rječnik svih obrazaca tako da doda nove obrasce u rječnik. Ovaj postupak ponavljat će se dok se ne pretraži čitavo stablo, a njegova implementacija prikazana je na slici 13.

```
def pretrazi_podstablo(self,min_support):
    obrasci = {}

    redoslijed_itema = sorted(self.cesti_itemi.keys(), key= lambda x: self.cesti_itemi[x])

    for item in redoslijed_itema:

        lista_pojavljivanja=[]
        conditional_fp_tree_itemsets=[]
        cvor = self.header_table[item]

        while cvor is not None:
            lista_pojavljivanja.append(cvor)
            cvor = cvor.link


        for it in lista_pojavljivanja:
            put_do_korijena=[]
            broj_pojavljivanja = it.broj
            roditelj_itema = it.roditelj

            while roditelj_itema.roditelj is not None:
                put_do_korijena.append(roditelj_itema.ime)
                roditelj_itema=roditelj_itema.roditelj

            for y in range(broj_pojavljivanja):
                conditional_fp_tree_itemsets.append(put_do_korijena)

        conditional_fp_tree = Stablo(conditional_fp_tree_itemsets,min_support,item,self.cesti_itemi[item])


        conditional_fp_tree_obrasci = conditional_fp_tree.pronadi_obrasce(min_support)
        self.unesi_nove_obrasce(obrasci,conditional_fp_tree_obrasci)

    return obrasci
```

Slika 13. Pretraživanje podstabla

4.2 Implementacija asocijativnih pravila

Generiranje asocijativnih pravila implementirano je u funkciji *generiraj_asocijativna_pravila* kojoj predajemo generirane česte obrasce pomoću algoritma FP-*growth* i minimalnu uvjerenost (eng. *minimal confidence*). Funkcija za svaki česti podskup generira sve njegove neprazne podskupove. Za svaki generirani neprazni podskup generiramo antecedent koji je zapravo razlika nepraznog podskupa i obrasca kojeg trenutno obrađujemo. Također generiramo konsekvens koji je zapravo razlika obrasca i antecedenta. Konkretni primjer generiranih konsekvensa i antecedensa za česti podskup {voda, mlijeko, kruh} napisan je u tablici 4. Nakon toga provjeravamo je li antecedent jedan od obrazaca. Ako jest, računamo uvjerenost (eng. *confidence*) tako da podijelimo broj ponavljanja (eng. *support count*) antecedenta sa brojem ponavljanja promatranog obrasca. Tu vrijednost onda uspoređujemo sa zadanim minimalnim pragom uvjerenosti. Ako je taj izraz zadovoljen, pravilo spremamo u rječnik kojemu ključ predstavlja antecedent, a par konsekvens i pripadajuća uvjerenost predstavljaju vrijednost. Implementacija navedenoga prikazana je na slici 14.

Konsekvens	Antecedent
voda	mlijeko, kruh
mlijeko	voda, kruh
kruh	voda, mlijeko
voda, mlijeko	kruh
voda, kruh	mlijeko
kruh, mlijeko	voda

Tablica 4. Generirani antecedent i konsekvens

```python
def generiraj_asocijativna_pravila(obrasci,min_confidence):
    """
    confindence(A=>B)=P(B|A)=support_count(AuB)/support_count(A)
    l=all_non_empty subsets(itemset)  ,s=non_empty subset
    rule "s_rule => (l-s)" if support_count(l)/support_count(s) >= min_confidence
    s = antecedent
    c = konsekvens
    """
    pravila= {}
    for itemset in obrasci.keys():
        sup_count_l = obrasci[itemset]

        for i in range(1,len(itemset)):
            for s in itertools.combinations(itemset,i):
                s=tuple(sorted(s))
                c = tuple(sorted(set(itemset)-set(s)))

                if s in obrasci:
                    sup_count_s = obrasci[s]
                    confindence = float(sup_count_l/sup_count_s)

                    if confindence >= min_confidence:
                        pravila[s] =(c,confindence)

    return pravila
```

Slika 14. Generiranje asocijativnih pravila

5. EKSPERIMENTALNO VREDNOVANJE ALGORITMA

5.1 Manji skup podataka

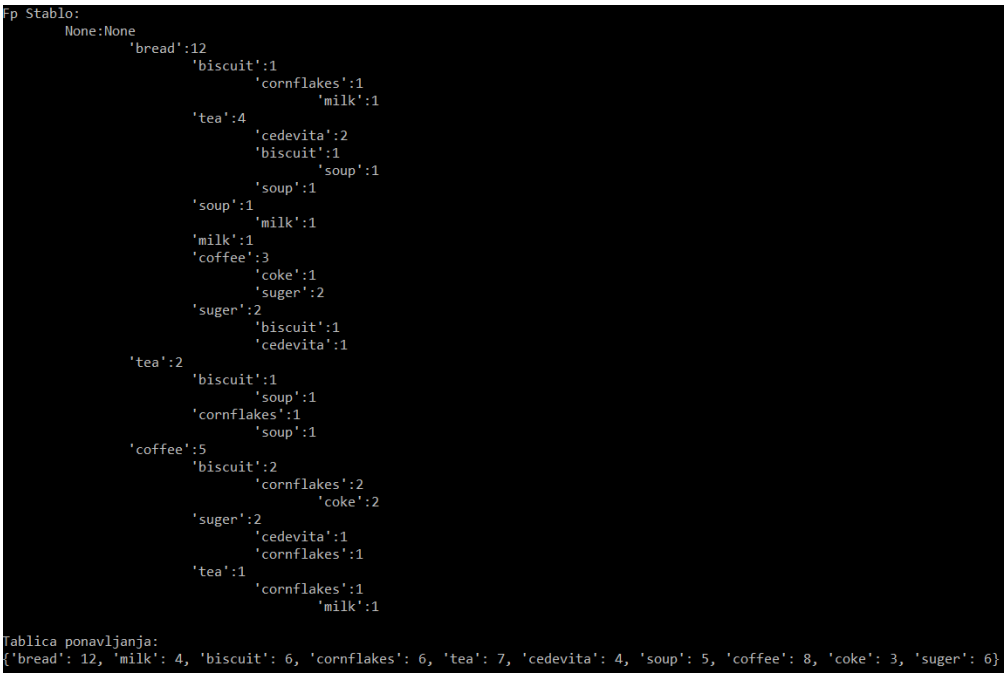
Implementirani algoritam prvo ćemo testirati na manjem skupu podataka da bi se prikazao i ispis generiranog stabla i svi generirani česti obrasci [15].

```
Ulazni set podataka:

0      [bread, milk, biscuit, cornflakes]
1                  [bread, tea, cedevita]
2               [jam, soup, bread, milk]
3                  [soup, tea, biscuit]
4                [bread, tea, cedevita]
5               [soup, tea, cornflakes]
6            [soup, bread, tea, biscuit]
7               [jam, soup, bread, tea]
8                           [bread, milk]
9     [coffee, coke, biscuit, cornflakes]
10    [coffee, coke, biscuit, cornflakes]
11             [coffee, suger, cedevita]
12                [bread, coffee, coke]
13              [bread, suger, biscuit]
14           [coffee, suger, cornflakes]
15              [bread, suger, cedevita]
16                [bread, coffee, suger]
17                [bread, coffee, suger]
18        [tea, milk, coffee, cornflakes]
Name: itemsets, dtype: object
```

Slika 15. Ulazni podaci

Niz transakcija koje će algoritam obraditi prikazane su na slici 15. Skup se sastoji od 19 transakcija od kojih svaka sadrži po nekoliko artikala, a minimalni prag podrške je 3. Na temelju ovih transakcija algoritam će generirati FP-stablo te će ispisati broj ponavljanja svakog pojedinog artikla koji je veći od zadanog minimalnog praga. Sve to je prikazano na slici 16.



Slika 16. FP-stablo i broj ponavljanja artikala

Nakon toga algoritam će pretražiti stablo i generirati najčešće obrasce prikazane na slici 17. Generirane obrasce te minimalni broj uvjerenosti (eng. *minimal confidence*) predat ćemo funkciji za generiranje asocijativnih pravila koja će onda generirati jaka asocijativna pravila prikazana na slici 18. Na temelju generiranih pravila zaključujemo da je u 100 posto slučajeva kada je kupac kupio kolu također kupio i kavu. U 75 posto slučajeva kada je kupac kupio mlijeko ili cedevitu također je kupio i kruh.

```
Cesto ponavljani obrasci:
('coke',) : 3
('coffee', 'coke') : 3
('milk',) : 4
('bread', 'milk') : 3
('cedevita',) : 4
('bread', 'cedevita') : 3
('bread', 'soup') : 3
('soup', 'tea') : 4
('biscuit',) : 6
('biscuit', 'bread') : 3
('biscuit', 'cornflakes') : 3
('coffee', 'cornflakes') : 4
('coffee', 'suger') : 4
('bread', 'suger') : 4
('tea',) : 7
('bread', 'tea') : 4
('coffee',) : 8
('bread', 'coffee') : 3
('bread',) : 12
```

Slika 17. Česti obrasci

```
Jaka asocijativna pravila:
('coke',) : (('coffee',), 1.0)
('milk',) : (('bread',), 0.75)
('cedevita',) : (('bread',), 0.75)
```

Slika 18. Generirana asocijativna pravila

5.2 Veći skup podataka

Sada ćemo algoritam testirati na nekoliko većih skupova transakcija. U idućem primjeru kao ulazni skup koristimo skup od 1000 transakcija prikazan na slici 19 [14].

```
Ulazni set podataka:

0      [baguette, soda, hering, cracker, heineken, ol...
1      [avocado, cracker, artichok, heineken, ham, tu...
2      [olives, bourbon, coke, turkey, ice_cream, ham...
3      [hering, corned_b, apples, olives, steak, avoc...
4      [sardines, heineken, chicken, coke, ice_cream,...
5      [olives, bourbon, coke, turkey, ice_cream, hei...
6      [corned_b, peppers, bourbon, cracker, chicken,...
7      [soda, olives, bourbon, cracker, heineken, pep...
8      [corned_b, peppers, bourbon, cracker, chicken,...
9      [baguette, sardines, apples, peppers, avocado,...
10     [baguette, hering, avocado, artichok, heineken...
11     [hering, corned_b, apples, olives, steak, sard...
12     [baguette, sardines, apples, peppers, avocado,...
13     [hering, corned_b, olives, ham, turkey, coke, ...
14     [olives, bourbon, coke, turkey, ice_cream, art...
15     [baguette, hering, avocado, artichok, heineken...
16     [sardines, heineken, chicken, coke, ice_cream,...
17     [soda, olives, bourbon, cracker, heineken, bag...
18     [soda, olives, bourbon, cracker, heineken, ste...
19     [hering, corned_b, olives, ham, turkey, apples...
20     [baguette, hering, avocado, artichok, heineken...
21     [soda, olives, bourbon, cracker, heineken, art...
22     [soda, olives, bourbon, cracker, heineken, cor...
23     [sardines, heineken, chicken, coke, ice_cream,...
24     [sardines, heineken, chicken, coke, ice_cream,...
25     [olives, bourbon, coke, turkey, ice_cream, app...
26     [olives, bourbon, coke, turkey, ice_cream, sod...
27     [soda, olives, bourbon, cracker, heineken, art...
28     [olives, bourbon, coke, turkey, ice_cream, sar...
29     [olives, bourbon, coke, turkey, ice_cream, sod...
                           ...
970    [corned_b, peppers, bourbon, cracker, chicken,...
```

Slika 19. Ulazni podaci

Generirano FP-stablo je u ovom slučaju dosta veliko pa ga nećemo ispisivati. Iz stabla su generirani česti podskupovi prikazana na slici 20. Minimalni prag podrške u ovom primjeru postavljen je na 200 jer se radi o puno većem broju ponavljanja pojedinih artikala.

```
Cesto ponavljani obrasci:
('steak',) : 237
('turkey',) : 285
('olives', 'turkey') : 220
('sardines',) : 298
('coke',) : 298
('coke', 'ice_cream') : 220
('peppers',) : 303
('ham',) : 306                          ('corned_b', 'hering') : 244
('artichok', 'avocado') : 212           ('baguette', 'hering') : 249
('artichok', 'heineken') : 253          ('baguette', 'heineken', 'hering') : 214
('ice_cream',) : 317                    ('baguette', 'heineken') : 261
('chicken',) : 318                      ('bourbon', 'heineken') : 212
('chicken', 'heineken') : 203           ('bourbon', 'cracker') : 240
('cracker', 'soda') : 251               ('bourbon', 'olives') : 247
('cracker', 'heineken', 'soda') : 234   ('heineken', 'olives') : 207
('heineken', 'soda') : 257              ('hering', 'olives') : 258
('apples',) : 319                       ('cracker', 'hering') : 203
('avocado', 'baguette') : 216           ('heineken', 'hering') : 292
('avocado', 'heineken') : 252           ('cracker',) : 490
('corned_b', 'olives') : 237            ('cracker', 'heineken') : 367
('corned_b', 'hering', 'olives') : 201  ('heineken',) : 602
```

Slika 20. Česti obrasci

Na temelju čestih obrazaca koje je algoritam generirao i za zadanu minimalnu uvjerenost 0,75 generirana su sljedeća jaka asocijativna pravila, slika 21.

```
Jaka asocijativna pravila:
('turkey',) : (('olives',), 0.7719298245614035)
('cracker', 'soda') : (('heineken',), 0.9322709163346613)
('heineken', 'soda') : (('cracker',), 0.9105058365758755)
('corned_b', 'hering') : (('olives',), 0.8237704918032787)
('corned_b', 'olives') : (('hering',), 0.8481012658227848)
('hering', 'olives') : (('corned_b',), 0.7790697674418605)
('baguette', 'heineken') : (('hering',), 0.8199233716475096)
('baguette', 'hering') : (('heineken',), 0.8594377510040161)
```

Slika 21. Generirana asocijativna pravila

Iz generiranih pravila možemo primjetiti da u 91 posto slučajeva, ukoliko kupac kupi pivo (*heineken)* i sok, kupit će i krekere. Također, možemo primjetiti da u 77,9 posto slučajeva kada je kupac kupio ribu (haringa, eng. *hering*) i maslinovo ulje, uz to je još kupio i kukuruzni kruh. Generirano je još nekoliko pravila koja se lako mogu iščitati sa slike na isti način.

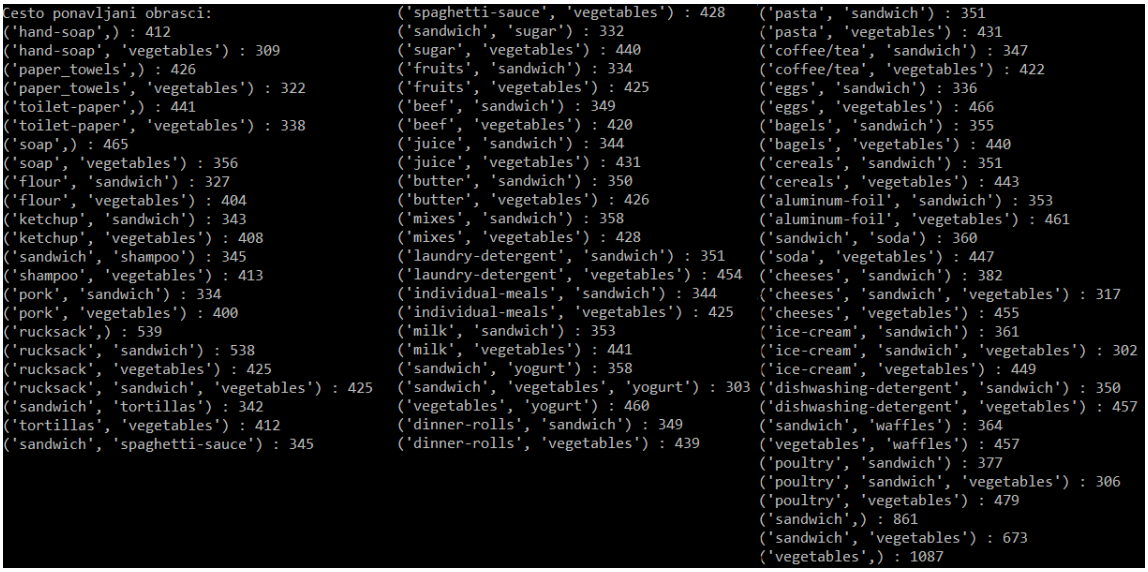
U idućem primjeru algoritam je primijenjen na skupu od oko 1500 transakcija [16]. Zadani minimalni prag podrške je 300. Skup transakcija prikazan je na slici 22.

```
Ulazni set podataka:

0       [toilet-paper, shampoo, hand-soap, waffles, ve...
1       [soda, pork, soap, ice-cream, toilet-paper, di...
2                  [cereals, juice, soda, toilet-paper, ]
3       [sandwich, pasta, tortillas, mixes, hand-soap,...
4       [laundry-detergent, toilet-paper, eggs, vegeta...
5       [individual-meals, paper_towels, tortillas, ve...
6       [ice-cream, juice, paper_towels, waffles, soda...
7       [juice, poultry, coffee/tea, dishwashing-deter...
8       [ketchup, coffee/tea, toilet-paper, pork, flou...
9       [sandwich, ice-cream, soda, bagels, dishwashin...
10      [pork, tortillas, shampoo, pasta, juice, bagel...
11      [sugar, fruits, aluminum-foil, laundry-deterge...
12      [fruits, dinner-rolls, individual-meals, shamp...
13      [individual-meals, ice-cream, cereals, paper_t...
14      [sugar, sandwich, rucksack, flour, juice, milk...
15      [milk, hand-soap, pasta, individual-meals, spa...
16      [sandwich, rucksack, toilet-paper, bagels, sha...
17      [individual-meals, laundry-detergent, coffee/t...
18      [shampoo, dishwashing-detergent, yogurt, juice...
19          [waffles, fruits, pork, juice, bagels, mixes]
20      [cheeses, vegetables, cereals, sugar, bagels, ...
21      [vegetables, aluminum-foil, bagels, shampoo, d...
22      [fruits, pasta, cheeses, juice, sandwich, ruck...
23      [bagels, sugar, pork, sandwich, tortillas, ice...
24      [fruits, sandwich, vegetables, coffee/tea, alu...
25      [laundry-detergent, pork, pasta, cheeses, frui...
26      [pork, bagels, poultry, pasta, butter, shampoo...
27      [pasta, butter, sandwich, spaghetti-sauce, jui...
28       [flour, bagels, cheeses, sandwich, toilet-paper]
29      [aluminum-foil, eggs, ice-cream, pasta, juice,...
                              ...
1468    [waffles, poultry, soap, mixes, fruits, dishwa...
1469    [poultry, tortillas, mixes, cheeses, toilet-pa...
1470    [soap, sandwich, pork, bagels, sugar, juice, f...
```

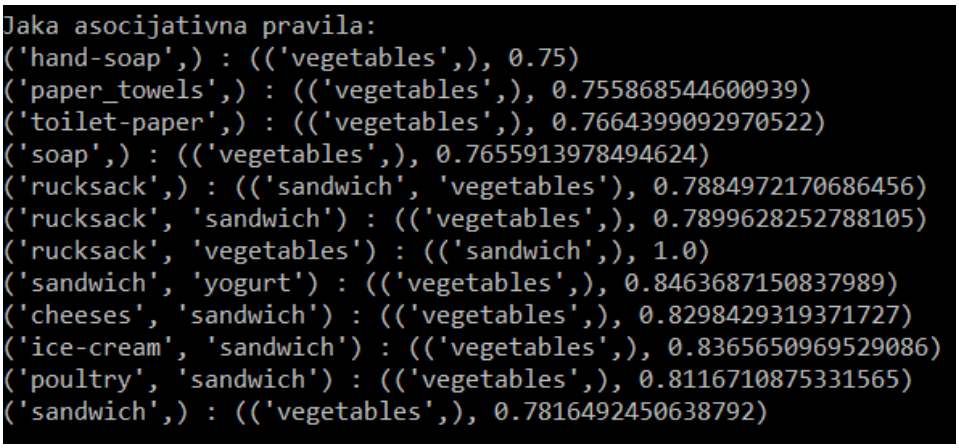
Slika 22. Ulazni podaci

Na temelju predanih ulaznih podataka algoritam je generirao sljedeće česte obrasce, slika 23.



Slika 23. Česti obrasci

Iz navedenih čestih obrazaca i uz zadanu minimalnu uvjerenost 0,75 generirana su jaka asocijativna pravila prikazana na slici 24.

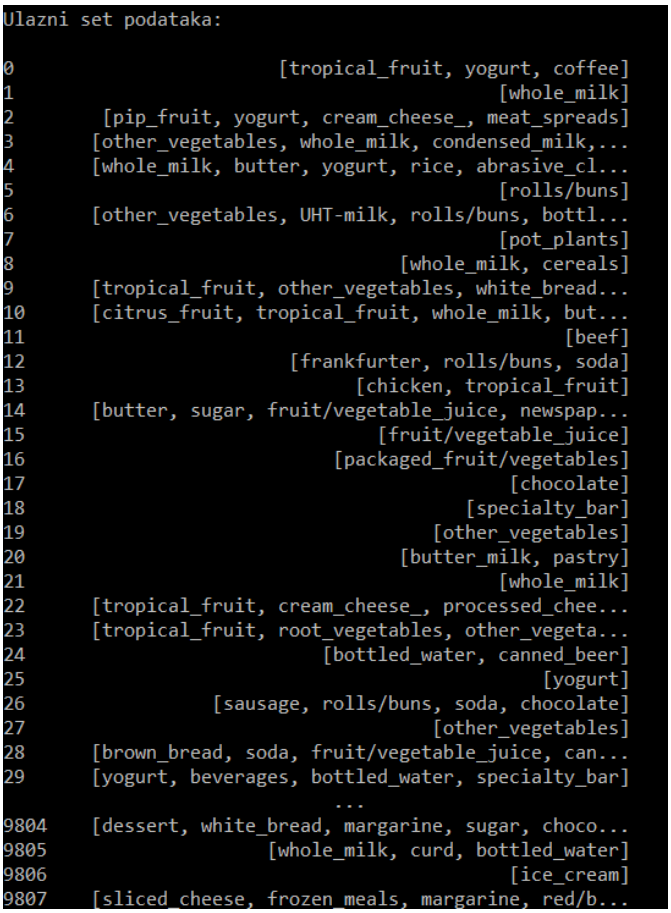


Slika 24. Generirana asocijativna pravila

U ovom skupu transakcija na temelju generiranih pravila možemo zaključiti da u slučaju kupovine ruksaka i sendviča, kupac je u 78 posto slučajeva kupio i

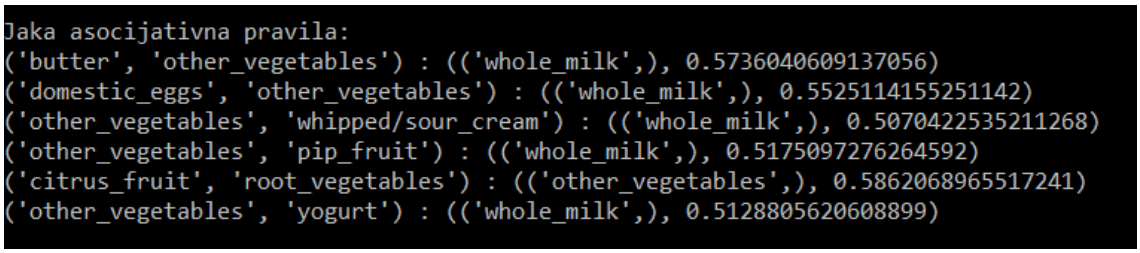
povrće. U 82,9 posto slučajeva kupac je sa sirom i sendvičem također kupio povrće.

U posljednjem primjeru algoritam je primijenjen na skupu od oko 10 tisuća transakcija [17]. Zadani minimalni prag podrške u ovom slučaju je samo 100. Ovaj skup podataka realnije prikazuje stvarnu situaciju zbog pune veće razlike u transakcijama uz velik broj različitih artikala. Skup transakcija prikazan je na slici 25.



Slika 25. Ulazni skup podataka

Zbog velike raznolikosti i niskog minimalnog praga podrške generiran je velik broj čestih obrazaca pa oni nisu prikazani. Na temelju čestih obrazaca i uz zadanu vrijednost minimalne uvjerenosti 0,50 generirana su jaka asocijativna pravila prikazana na slici 26.



Slika 26. Generirana asocijativna pravila

U ovom skupu transakcija na temelju generiranih pravila možemo zaključiti da u slučaju kupovine povrća i maslaca kupac je u 57,4 posto slučajeva kupio i punomasno mlijeko. U slučaju kupovine citrusnog voća i korjenastog povrća kupac je u 58,6 posto slučajeva kupio i neko drugo povrće.

6. ZAKLJUČAK

Generiranje asocijativnih pravila koristi se za otkrivanje zanimljivih i korisnih veza između određenih artikala i skupova artikala. U svom završnom radu sam analizirao skupove podataka u obliku potrošačke košarice, tj. skupove transakcija. Na tim skupovima podataka primjenio sam algoritam FP-Growth koji uz zadani minimalni prag podrške generira česte podskupove, odnosno obrasce, koji se pojavljuju u skupu transakcija. Analizom tih rezultata generirao sam asocijativna pravila koja sam na temelju mjere uvjerenosti filtrirao te dobio jaka asocijativna pravila.

Informacije koje dobijemo iz generiranih pravila jako su korisne raznim trgovcima i velikim kompanijama jer ih oni koriste kako bi manipulirali tržištem, davajući određene kupone i popuste na pojedine artikle mušterijama za koje vjeruju da će vjerojatno kupiti te proizvode. Ovi podaci također pridonose i raspodjeli proizvoda po supermarketima jer možemo postaviti proizvode koje trgovci često kupuju zajedno jedan pored drugoga.

Kroz svoj rad pokazao sam detaljno svaki korak algoritma FP-Growth te kako se generiraju asocijativna pravila. Demonstrirao sam rad na nekim jednostavnim

primjerima, a nakon toga sam i na konkretnim primjerima pokazao da su konačna dobivena asocijativna pravila zaista korisna u stvarim situacijama u svijetu.

7. LITERATURA

[1] 'Data Mining, Practical Machine Learning Tools and Techniques, Fourth edition', Morgan Kaufmann, (Ian H. Witten, Eibe Frank, Mark A. Hall, Christopher J. Pal, 2016.)

[2] 'Association Analysis: Basic Concepts and Algorithms, Chapter 6 from Introduction to Data Mining' (Tan, Steinbach, Kumar, 2004., https://www-users.cs.umn.edu/~kumar001/dmbook/ch6.pdf)

[3] 'CS 176 Programming Assignment 1' (Isabelle Tingzon, University of the Philippines, ožujak 2015.)

[4] 'Association Rule Mining: A Survey' (Qiankun Zhao, Sourav S. Bhowmick, Technical Report, CAIS, Nanyang Technological University, Singapore, No. 2003116, 2003.)

[5] 'Explanation of the Market Basket Model' (Information Builders, SAD, https://webfocusinfocenter.informationbuilders.com/wfappent/TLs/TL_rstat/sourcs/marketbasket49.htm, zadnji pristup 2. lipnja 2019.)

[6] 'A Gentle Introduction on Market Basket Analysis – Association Rules' (Susan Li, rujan 2017, https://towardsdatascience.com/a-gentle-introduction-on-market-basket-analysis-association-rules-fa4b986a40ce)

[7] 'Market Basket Analysis' (Techopedia - The IT Education Site, https://www.techopedia.com/definition/32063/market-basket-analysis, zadnji pristup 2. lipnja 2019.)

[8] 'Apriori vs FP-Growth for Frequent Item Set Mining' (Singularities, svibanj 2018., https://www.singularities.com/blog/our-blog-1/post/apriori-vs-fp-growth-for-frequent-item-set-mining-11, zadnji pristup 27. svibnja 2019.)

[9] 'Frequent Pattern (FP)-Growth algorithm' (Tobias Hardes, http://thardes.de/big-data-englisch/frequent-pattern-fp-growth-algorithm/)

[10] 'MachineX: Understanding FP-Tree Construction' (Akshansh Jain, lipanj 2018., https://dzone.com/articles/machinex-understanding-fp-tree-construction)

[11] 'FP Growth Algorithm Implementation' (Narina Thakur, International Journal of Computer Applications, Volume 93 – No.8, svibanj 2014.)

[12] 'Coding FP-growth algorithm in Python 3' (Piush Vaish, https://adataanalyst.com/machine-learning/fp-growth-algorithm-python-3/)

[13] 'Rudarenje podataka' (https://hr.wikipedia.org/wiki/Rudarenje_podataka)

Izvori ulaznih skupova podataka:

[14] 'MBA.txt' (Pranit Bose, 2017., https://github.com/pranitbose/market-basket-analysis/blob/master/dataset/MBA.txt)

[15] 'GroceryStoreDataSet.csv' (Shazad Udwadia, 2016., https://www.kaggle.com/shazadudwadia/supermarket)

[16] 'dataset.csv' (https://www.kaggle.com/fanatiks/shopping-cart#dataset.csv)

[17] 'groceries.csv' (Pranit Bose, 2017., https://github.com/pranitbose/market-basket-analysis/blob/master/dataset/groceries.csv)

8. SAŽETAK I KLJUČNE RIJEČI

8.1 Sažetak i ključne riječi na hrvatskom jeziku

Ovaj rad obrađuje dubinsku analizu podataka, tj. analizu potrošačke košarice uporabom FP-Growth algoritma te generiranje jednostavnih asocijativnih pravila. Potrošačka košarica sastoji od skupa transakcija koje se predaju FP-Growth algoritmu. Na temelju tog skupa transakcija algoritam uz zadani minimalni prag podrške (eng. *minimal support*) generira najčešće podskupove, tj. česte obrasce. Pomoću dobivenih čestih obrazaca i uz zadanu minimalnu povjerljivost, (eng. *minimal confidence*) generiraju se stroga asocijativna pravila. U ovom radu detaljno je objašnjen taj cijeli postupak te je nakon toga demonstrirana njegova primjena na konkretnim primjerima kako bi se pokazalo da su dobiveni rezultati jako korisni, zbog čega ih velike kompanije često koriste i analiziraju kako bi unaprijedili svoje poslovanje.

Ključne riječi:

- dubinska analiza podataka, analiza potrošačke košarice, skup transakcija, podatak, kupac, trgovac, artikl

- FP-Growth algoritam, FP-stablo, čvor, obrasci, česti podskupovi, prefiksni putevi, podskup, problem, podproblem, sufiks, minimalni prag podrške, ažuriranje, uvjetno FP-stablo

- asocijativna pravila, nepovezani skupovi, podrška, povjerljivost, formula, vjerojatnost

- implementacija, razred, lista, korijen, obrađivanje, eksperimentalno vrednovanje, podstablo, rječnik, sortiranje

8.2 Sažetak i ključne riječi na engleskom jeziku (Summary and keywords in english)

This paper processes data mining, i.e. market basket analysis using FP-Growth algorithm for frequent patterns mining and generating simple association rules. Market basket contains a set of transactions that are submitted to the algorithm. Based on the given set of transactions and minimum support count algorithm generates frequent itemsets, i.e. frequent patterns. With these frequent patterns and given minimum confidence value we can generate strict association rules. In this paper, the entire procedure is explained in detail and its use is demonstrated on a concrete example. The results obtained by analysis are very useful and are often used by large companies to improve their business.

Keywords:

- data mining, market basket analysis, set of transactions, data, buyer, merchant, item

- FP-Growth algorithm, FP-tree, node, patterns, frequent subsets, prefix paths, subset, problem, subproblem, suffix, minimal support count, update, conditional FP-tree

- association rules, disjoint itemsets, support, confidence, formula, probability

- implementation, class, list, root, processing, experimental evaluation, subtree, dictionary, sorting