UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

ZAKLJUČNA NALOGA

PRIMERJAVA RAZLIČNIH ALGORITMOV ZA GRADNJO ASOCIACIJSKIH PRAVIL ZA NAPOVEDOVANJE

JAN MARTINČIČ

2014
ZAKLJUČNA NALOGA
MARTINČIČ

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

Primerjava različnih algoritmov za gradnjo asociacijskih pravil za napovedovanje

(Using association rules for classification - A comparison of different algorithms)

Ime in priimek: Jan Martinčič
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Branko Kavšek

Koper, avgust 2014

Ključna dokumentacijska informacija

Ime in PRIIMEK: Jan MARTINČIČ

Naslov zaključne naloge: Primerjava različnih algoritmov za gradnjo asociacijskih pravil za napovedovanje

Kraj: Koper

Leto: 2014

Število listov: 49 Število slik: 2

Število referenc: 67

Mentor: doc. dr. Branko Kavšek

Ključne besede: Apriori, asociacijska pravila, klasifikacija, podpora, zaupanje

Izvleček:

V zaključni nalogi je obravnavana tematika asociacijskih in klasifikacijskih pravil ter ali so asociacijska pravila lahko primerna za klasifikacijo. Razložena sta pojma asociacijska pravila in klasifikacija ter predstavljeni primeri algoritmov za iskanje asociacijskih pravil (Apriori, CBA) in klasifikacijskih pravil (C4.5, PART). Algoritme smo primerjali med seboj tako po klasifikacijski točnosti, velikosti modelov kot tudi časovni učinkovitosti. Ugotovitve so lahko koristne pri ugotavljanju smiselnosti uporabe algoritmov za iskanje asociacijskih pravil v namen klasifikacije. Primerjave smo izvedli na podatkovnih množicah iz spletnega podatkovnega skladišča UCI ML Repository. Izbrali smo take podatkovne množice, ki predstavljajo probleme klasifikacijskega tipa. Za testiranje smo uporabljali javanske implementacije algoritmov iz odprtokodnega orodja za podatkovno rudarjenje WEKA. Rezultati testov so pokazali, da klasifikacija z asociacijskimi pravili potrebuje veliko količino spominskega prostora. Ker je bila točnost napovedi algoritmov Apriori in CBA primerljiva s klasifikacijskimi algoritmi C4.5 in PART lahko zaključimo, da je klasifikacija z asociacijskimi neprimerna.

Key words documentation

Name and SURNAME: Jan MARTINČIČ

Title of final project paper: Using association rules for classification - A comparison of different algorithms

Place: Koper

Year: 2014

Number of pages: 49 Number of figures: 2

Number of references: 67

Mentor: Assist. Prof. Branko Kavšek, PhD

Keywords: Apriori, association rules, classification, confidence, support

Abstract:

In this paper we discuss whether the association rules mining algorithms are effective when mining for classification rules. We present the definition and differences of association rules and classification mining. We have tested the effectivness of algorithms such as Apriori and CBA compared to classification algorithms like C4.5 and PART. For comparison we used mesaures like time needed to generate model, number of generated rules and prediction accuracy. The mesurements were made on datasets from the UCI ML Repository, and all datasets default tasks are clasification based. For testing we used Weka, a data mining software implemented in java, that contains a collection of machine learning algorithms for data mining. The test results showed, that asociation algorithms use al lot of memory space. Because the effectiveness of association algorithms Apriori and CBA is comparable to the effectiveness of classification algorithms PART and C4.5, we have concluded that classification with association is not recommended.

Zahvala

Iskrene zahvale namenjam mentorju, doc. dr. Branku Kavšku, za potrpljenje, korektnost in pomoč pri pisanju naloge. Zahvaljujem se vsem profesorjem in asistentom, s katerimi sem imel možnost sodelovati v času mojega študija. Zahvale gredo moji ožji družini in punci za moralno in finančno podporo, ter za dano motivacijo, ki sem jo potreboval pri pisanju. Zahvalil bi se tudi svojim prijateljem, ki so mi stali ob strani.

Kazalo vsebine

Kazalo slik

Kazalo tabel

Kazalo algoritmov

Seznam kratic

- CAR Class Association Rules
- CBA Classification Based on Associations
- $RIPPER$ Repeated Incremental Pruning to Produce Error Reduction
- $TDIDT$ Top Down Induction of Decision Trees

1 Uvod

Podatkovno rudarjenje (ang. Data Mining) je tehnika informatike in statistike, ki nam pomaga iz podatkov zajeti vzorec, pravila ali celo znanje, ki ga lahko učinkovito uporabimo. Pri podatkovnem rudarjenju združeno uporabljamo metode umetne inteligence in upravljanja podatkovnih baz za analizo digitalnih sklopov podatkov, ki jim pravimo podatkovne množice (ang. data set) [28].

Pri podatkovnem rudarjenju uporabljamo klasifikacijske metode za napovedovaje pripadajoče skupine instancam podatkovne množice. Kateri skupini pripada določena instanca nam pove razredni atribut oz. razred. Napoved sestoji iz določenih pravil, ki jih dobimo iz atributov, na podlagi učenja na učnih množicah.

Iskanje asociacijskih pravil, je prav tako ena izmed tehnik podatkovnega rudarjenja. Iskanje asociacijskih pravil predstavlja iskanje razmerij med atributi v dejanski podatkovni množici. Pravila lahko vsebujejo dva ali več atributov hkrati in količina pravil je odvisna od raznolikosti množice. Naša naloga je ugotoviti ali je smiselno tovrstna pravila uporabiti kakor klasifikatorje pri napovedi pripadajoče skupine oz. razreda.

S podobnim problemom so se srečali že B. Liu, W. Hsu in Y. Ma v [39]. Problem klasificiranja z asociacijskimi pravili se pojavi pri razmerju med razredom in atributi. Pri običajnih klasifikacijskih metodah je eno pravilo pripisano natanko eni vrednosti razreda, kar pa ni vedno res pri asociacijski klasifikaciji. V kolikor ne nastavimo pravih vrednosti parametrov za iskanje, je lahko eno pravilo v razmerju z več vrednostmi razreda in obratno. Optimalne nastavitve parametrov pa dobimo z večkratnim poskušanjem. Med parametri ki jih nastavljamo sta predvsem *zaupanje* in *podpora*. Heravi in Zaïne sta v [32] predstavila alternative tem meram, vendar se nobena od predlaganih mer ni izkazala za bistveno boljšo. Zaradi neenakomernih razmerij med pravili in razredom, je število generiranih pravil večje, kakor bi bilo, če bi bila razmerja 1 : 1. Število pravil lahko sicer naknadno zmanjšamo z rezanjem, vendar lahko v takih primerih porežemo pravilo, ki bi lahko bilo zanimivo. Ne glede na to, so asociacijska pravila bolj podrobna kakor klasifikacijska, saj imamo večje število pravil katera načeloma zajamejo manj instanc. Posledično, če so napake zaradi razmerij zanemarljive, bi lahko bila asociacijska pravila boljši napovedovalec kakor klasifikacijska pravila. Naš namen je bil ugotoviti ali to drži, zato smo izvedli poskuse in ugotovili, ali

so asociacijska pravila boljše, slabše ali enakovredno merilo za napovedovanje razreda v primerjavi s klasifikacijskimi pravili.

V drugem poglavju bomo pregledali nekaj do sedaj opravljenih raziskav iz področja asociacijskih pravil. Tretje poglavje podrobneje opisuje kaj so asociacijska pravila in kako jih dobimo, kaj je klasifikacija in kaj so odločitvena drevesa ter bistvene razlike med iskanjem asociacijskih pravil in klasifikacijo. V četrtem poglavju bomo spoznali kako lahko asociacijska pravila uporabimo za klasifikacijo, ter algoritem CBA, ki klasificira na podlagi asociacijskih pravil. V petem poglavju je opisano kako smo testirali, naša pričakovanja in rezultati. Zadnje, šesto poglavje vsebuje naš razmislek in možne smernice za nadaljnje raziskave.

2 Sorodna dela

Iskanje asociacijskih pravil je v zadnjem obdobju dokaj aktualna tematika. Pogosto iskanje asociacijskih pravil zamenjujemo z iskanjem pogostih vzorcev, saj nas ti privedejo do asociacijskih pravil. Na temo iskanja pravil ali vzorcev obstaja veliko raziskav, kot npr. iskanje asociacijskih pravil brez generiranja kandidatov [29], [30].

Algoritem za iskanje pogostih vzorcev z verti kalnim formatom podatkov *Eclat* (EquivalenceCLASSTransformation) je leta 2000 predstavil Zaki [64] [67]. .

Iskanje zaprtih pogostih vzorcev so predstavili Pasquier, Bastide, Taouil in Lakhal leta 1999, kot rešitev problema ogromnega števila pogostih vzorcev [46]. Algoritem *A-Close* poišče zaprte pogoste vzorce $x \in H$, kjer so x taki pogosti vzorci v podatkovni množici H, ki nimajo nad-vzorca $y \in H$ z enako podporo. S časom je bilo predstavljenih še kar nekaj učinkovitejših algoritmov za iskanje zaprtih pogostih vzorcev. *CLOSET* algoritem, ki uporablja FP-drevo [49], *CHARM* algoritem, ki išče zaprte množice z uporabo dvojnega drevesa množic zapisov v podatkovni bazi in tehnike *diffset*, ki zmanjša porabo spomina ter hiter *hash-based* pristop za odstranjevanje ne zaprtih množic [66]. Algoritmi *CLOSET+* [60], *FPClose* [27] in *AFOPT* [41] prav tako uporabljajo *FP-drevesu* podobno strukturo za hranjenje odkritih vzorcev.

Podobno iskanju zaprtih pogostih vzorcev je iskanje maksimalnih pogostih vzorcev. Vzorec $x \in H$ je maksimalen pogost vzorec v podatkovni množici H, če je pogost vzorec v H in ne obstaja nad-vzorec $y \in H$, ki je pogost. *MaxMiner* je prvi algoritem za iskanje maksimalnih pogostih vzorcev [8]. Posebej učinkovit algoritem je tudi *MAFIA*, predvsem kadar so vzorci zelo dolgi [13].

Precej pogosti so primeri, kadar težko najemo dobra asociacijska pravila na nižjih nivojih abstrakcije. Za primer vzemimo podatkovno množico trgovine, ki prodaja komponente in ima dva nivoja abstrakcije: komponenta (višji nivo) in proizvajalec (nižji nivo). Sklepamo lahko, da bodo asociacijska pravila boljša na višjem nivoju, kajti razpršenost podatkov je manjša. Iskanje več nivojskih asociacijskih pravil sta med prvimi preučevala Srikant in Agrawal [55]. Če predpostavimo, da za vse nivoje abstrakcije velja, da imajo enako podporo, potem lahko na nižjih nivojih iščemo le vzorce, ki ustrezajo vzorcem na višjih nivojih. Za obraten primer, kadar se z vsakim nadaljnjim nivojem podpora zmanjšuje, je bil predstavljen pristop z variabilno podporo [28].

Iskanje več dimenzionalnih asociacijskih pravil je nadgradnja iskanja več nivojskih pravil. V primeru trgovine imamo samo en predikat "kupil". Prav tako bi lahko v podatkovni bazi bil podatek o spolu ali starosti, ki bi predstavljala dva dodatna predikata. Iskanje tovrstnih asociacijskih pravil so raziskovali M. Kamber, J. Han in Chiang, ki so predlagali štiri algoritme, ki uporabljajo strukturo podatkovne kocke (angl. data cube structure) [33].

Do sedaj predstavljene tehnike iskanja asociacijskih pravil veljajo za diskretne atribute. V kolikor bi skušali z njimi rudariti na zveznih atributih, bi bili rezultati nesmiselni. Lent, Swami in Widom so raziskovali rudarjenje zveznih atributov z rojenjem (ang. clustering), tako da zvezne atribute diskretiziramo [36]. Diskretiziramo lahko tudi z uporabo rojenja na podlagi razdalje med zveznimi atributi [44].

J. Mata et al. so predstavili evolucijski algoritem, kateri poskrbi, da se izognemo statični diskretizaciji atributov [42]. Aumann in Lindell sta uporabila statistično teorijo za iskanje kvantitativnih asociacijskih pravil [7]. Nadgradnja tega algoritma je genetski algoritem (*QuantMiner*) za iskanje zanimivih kvantitativnih asociacijskih pravil [53].

Iskanje sekvenčnih vzorcev sta predlagala Agrawal in Srikant [6]. Sekvenčni podatki so zaporedja urejenih dogodkov, ki se zgodijo pri nekem procesu. Dogodki so urejeni po nekem ključu (npr. čas) in lahko vsebujejo različne pod-elemente, ki ne vplivajo na urejenost sekvence. Sekvenčne podatke si lahko predstavljamo kot televizijski program, sekvence pri bioloških procesih ali potek igre. Pogosti sekvenčni vzorci so pod-sekvence sekvenčnih podatkov, ki zadostijo pogojem pogostosti. Algoritem *SPADE* [65] uporablja vertikalni format za iskanje sekvenčnih vzorcev, algoritem PrefixSpan [50] pa bistveno zmanjša potrebo po generiranju kandidatov. *CloSpan* je prvi algoritem ki išče zaprte sekvenčne vzorce [62].

Iskanje vzorcev z omejitvami je iskanje, ki ustreza omejitvam uporabnika (angl. constraint-based mining). Iskanje lahko omejimo na različne načine [28]. Ena od možnih omejitev je definiranje predloge pravil (angl. metarule or rule templates) [34]. Iskanje lahko omejimo tudi z logičnimi izrazi [56]. Algoritem *CAP* zmore iskanje z več omejitvami hkrati [45]. Pei, Han in Lakshmanan so predstavili tehniko uporabe omejitev med iskalno fazo FP-growth algoritma [48]. Tovrstne omejitve niso monotone, lahko pa jih z uporabo ustrezno sortiranih elementov preoblikujemo. Takim omejitvam pravimo preoblikovane omejitve (angl. convertible constraints) [9].

Iskanje stisnjenih pogostih vzorcev v splošnem delimo na dve veji. Take kjer ne izgubljamo informacij (zaprti pogosti vzorci) in take kjer imamo izgube informacij. Iskanje brez izgub je raziskovano v [14] [15] [40]. Iskanje z izgubami pa v [2] [54] [59] [61] [63].

Uporaba asociacijskih pravil za klasifikacijo je tudi dokaj raziskano področje, ki je podrobneje opisano v poglavju 4 in v [20] [37] [39] [38] [40].

3 Teoretično ozadje

V tem poglavju bomo podrobneje predstavili asociacijska pravila in algoritem Apriori. Predstavili bomo tudi klasifikacijska pravila in algoritme C4.5 in PART ter pogledali bistvene razlike med iskanjem asociacijskih pravil in klasifikacijo.

3.1 Asociacijska pravila

Asociacijska pravila so mera pogostosti pojavljanja določenih vzorcev v podatkovnih množicah [4]. Za razlago asociacijskih pravil si bomo pomagali z analizo nakupovalne košarice. Zaradi nazornejšega prikaza smo originalno tabelo iz [4] poenostavili s tabelo 1. Tabela 1 prikazuje nakupe kupcev, iz katerih lahko razberemo verjetnost da bo kupec kupil hkrati mleko kot tudi kruh in za dan primer znaša $\frac{3}{4}$. Odkritje tovrstnih vzorcev lahko pripomore prodajalcem k boljši organizaciji polic in učinkovitejšem marketingu, zato se veliko podjetji odloča za tovrstne analize. Asociacijska pravila lahko uporabimo tudi za iskanje pod-sekvenc. Primer, ko v danem zaporedju kupci najprej kupijo telefon, nato prenosnik in zatem tablični računalnik. Asociacijska pravila se uporabljajo tudi za iskanje podstruktur v grafih in drevesnih strukturah [28].

Vrnimo se na tabelo 1. Ogledali smo si rezultate analize kupcev mleka in kruha. Vidimo lahko, da je veliko kupcev kupilo tako mleko kot kruh, zato se ti dve vrednosti večkrat pojavljata v tabeli. Večkratnim pojavitvam enakih vzorcev, v našem primeru mleka in kruha, pravimo pogosti vzorci (*ang. Frequent paterns*). Pogosti vzorci so temelji za gradnjo asociacijskih pravil. Osnovni patent za iskanje asociacijskih pravil sestoji iz dveh korakov. Najprej poiščemo vse pogoste vzorce v dani podatkovni množici, nato iz njih sestavimo asociacijska pravila.

Z asociacijskimi pravili so se med prvimi ubadali R. Agrawal, T. Imielinski, in A. Swami [4] in leta 1993 definirali asociacijska pravila s pomočjo analize nakupovalne košarice. Predstavljajmo si, da imamo podatke o vseh nakupih kupcev v neki trgovini. Naj bo $I = \{i_1, i_2, ..., i_y\}$ množica vseh atributov enega nakupa (npr. i_1-mleko, i_2-kruh, itd.) binarne oblike {1,0} (1 v primeru nakupa, 0 sicer). Recimo da je $T \subset I$ podmnožica I. Vsak element $t[k] \in T$ je predstavljen kot $t[k] = 1$, če je oseba kupila izdelek in $t[k] = 0$ sicer ($k \in \mathbb{N}$). Za lažjo predstavo bomo uvedli še množico $H = \{I_1, I_2, ..., I_n\}$, ki je množica vseh možnih meritev oziroma nakupov. Denimo tudi,

da je $MT_1 = P(I_1)$ potenčna množica od I_1 in naprej sledi, da je $MT = \bigcup_1^n MT = \{T_1, T_2, ..., T_l\}$ množica vseh možnih potenčnih množic od vseh I $[MT = \bigcup_1^n P(I)]$.

Pogost vzorec je neprazna množica atributov $V \in MT$ in je podmnožica vsaj S-tim elementom množice H ($S \in \mathbb{N}$). S je minimalni prag podpore, določen s strani izvajalca analize. V primeru da je podpora 20% pomeni, da je v množici H 20% elementov I takih, ki vsebujejo V $[H_q = \{I|V \subseteq I\}$ in $H_q \subseteq H]$

Tabela 1: Nakupovalna košarica

ID	**mleko**	**kruh**	**sir**
1	1	1	0
2	1	1	0
3	1	1	1
4	0	1	0

Asociacijsko pravilo je implikacija oblike $X \implies X \cup \{i_j\}$ ali $X \implies V$, kjer je $X \subset V$ podmnožica V in moč množice X je natanko za ena manjše od moči množice V $[M(X) = M(V) - 1]$. Element i_j je element V $[i_j \in V]$ in ni element X $[i_j \notin X]$.

Ker imamo v večini primerov opravka z velikimi podatkovnimi množicami, je veliko tudi število meritev, posledično je veliko tudi število asociacijskih pravil. Ker nas ne zanimajo vsa pravila, se poslužujemo zaupanja c. Zaupanje je definirano kot

$$c = \frac{podpora(V)}{podpora(X)} \tag{1}$$

Asociacijsko pravilo je zanimivo in nam poda bistveno informacijo v primeru da zadovoljuje pogoj minimalnega zaupanja c. Meja minimalnega zaupanja ni vnaprej določena s strani algoritmov. Vsaka podatkovna množica ima drugačne elemente, posledično tudi različna asociacijska pravila zato sami določimo minimalno mejo za c glede na naše interese.

Podatek, da je zaupanje 70% pomeni, da je od takih primerov I, ki vsebujejo X (npr. 100), 70% takih ki hkrati vsebujejo tudi i_j oziroma V(npr. 70).

Pogoste vzorce lahko še dodatno omejimo, tako da morajo vsebovati specifičen element $i \in I$. S tem smo prilagodili iskanje asociacijskih pravil našim potrebam in se hkrati izognili nekaterim pravilom, ki nam zagotovo ne bi bila v pomoč. Recimo, da je naša množica $V = \{v_1, v_2, ..., v_k\}$. Če predpostavimo, da je na desni strani pravila lahko samo en element, dobimo natanko k pravil.

$$\begin{aligned} v_1 &= v_2 + v_3 + ... + v_k \\ v_2 &= v_1 + v_3 + ... + v_k \\ &\vdots \\ v_k &= v_1 + v_2 + ... + v_{k-1} \end{aligned} \tag{2}$$

Na levi strani pravila imamo množico $X \subset V$, ki je podmnožica množice V, z natanko $k-1$ elementi. Na desni strani pravila je element i_j, ki je element V ni pa element X. Asociacijska pravila so s časom nadgradili na taka, ki imajo lahko na obeh straneh implikacije več elementov. Če zapišemo podporo in zaupanje s pomočjo verjetnosti P, dobimo enačbi:

$$\begin{aligned} podpora(X \implies Y) &= P(X \cup Y) \\ zaupanje(X \implies Y) &= P(Y|X) \end{aligned} \tag{3}$$

kjer je $X \subset I$, $Y \subset I$ in $X \cap Y = \emptyset$ ter $P(Y|X)$ je pogojna verjetnost. Verjetnost $P(X \cup Y)$ je število pojavitev množic X ali Y v množici H. Sledi torej

$$zaupanje(X \implies Y) = P(Y|X) = \frac{podpora(X \cup Y)}{podpora(X)} \tag{4}$$

3.1.1 Apriori algoritem

Apriori je algoritem za iskanje pogostih vzorcev. Deluje po principu piramide, kar pomeni da najprej najde manjše ponavljajoče vzorce v podatkovni množici, ki jih nato po korakih povečuje, do točke, ko nadaljnjo povečevanje nima več smisla, oziroma je podpora vzorca manjša od željene [5].

Algoritem uporablja tako imenovano "level-wise" iskanje. Tako iskanje uporablja $k-vzorcev$ za iskanje $(k+1)-vzorcev$. Najprej poišče vse $1-vzorce$ (eno atributne vzorce, $k=1$), ki ustrezajo pogoju minimalne podpore, s skeniranjem celotne podatkovne množice H. Takim vzorcem bomo dali oznako L_1. Z drugim skeniranjem podatkovne množice, L_1 vzorce razširimo v $2-vzorce$ (dvo-atributne vzorce, $k=2$), z oznako L_2, ki še zmeraj zadovoljujejo pogoj minimalne podpore. L_2 vzorce nato razširimo v L_3 vzorce itd. Postopek nadaljujemo do točke, ko je k-vzorec prazna množica, oziroma kadar nadaljnja razširitev v L_X privede do praga podpore, ki je manjši od zahtevane minimalne podpore.

Ker je za vsako razširitev L potrebno eno skeniranje podatkovne baze, za L_k torej k skeniranj, je v primerih, kadar je podatkovna množica zelo velika oziroma kadar so pogosti vzorci zelo obširni (več atributni), algoritem časovno zelo potraten. R. Agrawal in R. Srikant sta v članku [5] za optimizacijo algoritma uporabila Apriori lastnost, ki

pravi, da morajo biti podmnožice pogostega vzorca tudi pogoste. Pri kreiranju $k+1$ množice pogostih vzorcev lahko torej uporabimo lastnost, ki pravi, da če vzorec A ne presega minimalne podpore, posledično vzorec $A \cup B$ zagotovo tudi ne bo presegal minimalnega praga podpore. Iz tega lahko sklepamo, da so za kreiranje množice $k+1$ pogostih vzorcev, potrebne le kombinacije $k-vzorcev$, ki zadoščajo minimalnemu pragu podpore. (Dobljenim $k+1$ vzorcem moramo nato preveriti ali je njihova podpora nad minimalno določeno mejo.)

Apriori lastnost uporabimo dvakrat. Prvič, pri koraku združevanja kadar iz množice L_{k-1} pogostih vzorcev generiramo množico kandidatov C_k. Drugič pa pri koraku rezanja. Množica C_k vsebuje tako pogoste, kot tudi nepogoste vzorce. Ker vemo, da je L_k podmnožica C_k, lahko uporabimo Apriori lastnost in lahko iz množice C_k odstranimo vse kandidate, za katere velja, da katerakoli podmnožica dolžine $k-1$ ni element L_k. Ostanejo nam tako samo kandidati dolžine $k-1$, ki so elementi L_k, to pa so pogosti vzorci. Množica C_k po rezanju tako vsebuje samo kandidate, ki so elementi L_k. Poglejmo si kako deluje algoritem 1, Apriori.

Algoritem 1: Apriori

```
1 L_1 = 1_vzorci; za k = 2; L_{k-1} ≠ ∅; k++
2     C_k = apriori_gen(L_{k-1}); za vsako transakcijo t ∈ H
3         C_t = podmnožica (C_k, t) za vsak kandidat c ∈ C_t
4             c.števec++;;
5         L_k = {c ∈ C_k | c.števec ≥ minsup};
6 vrni ⋃_k L_k;
```

Funkcija *apriori_gen* najprej združi množico L_{k-1}, ki ima elemente velikosti $k-1$, samo s seboj, da nastane nova množica C_k, ki ima elemente velikosti k. Nato obreže elemente (ang. pruning), katerih vsaj ena podmnožica dolžine $k-1$ ni element L_{k-1}. Koda za *apriori_gen* je prikazana v algoritmu 2. Primer če je:

$L_2 = \{\{1,2\},\{1,3\},\{1,4\},\{3,4\}\}$, bo množica $C_3 = \{\{1,2,3\},\{1,2,4\},\{1,3,4\}\}$. Po končanem rezanju bo množica $C_3 = \{\{1,3,4\}\}$, saj je element $\{1,3,4\}$ edini element, za katerega velja, da so vse možne podmnožice dolžine 2, vsebovane v L_2: $\{1,3\},\{1,4\},\{3,4\} \in L_2$.

Množica C_k vsebuje možne kandidate za pogoste vzorce. Preveriti moramo, ali vsi zadovoljujejo pogoju minimalne podpore. Korak stika L_{k-1} si lahko predstavljamo tudi kot združitev množice L_{k-1} z vsakim elementom v podatkovni množici H in nato brisanjem takih elementov, katerih $(k-1)$-*vzorec*, dobljen z brisanjem $(k-1)ga$ elementa ni v množici L_{k-1}. Z drugimi besedami, če katerakoli podmnožica kateregakoli

Algoritem 2: apriori_gen

```
1 Vstavi v C_k
2 Izberi  p.člen_1, p.člen_2, ..., p.člen_{k-1}, q.člen_{k-1}
3 Iz  L_{k-1} p, L_{k-1} q
4 Kjer  p.člen_1 = q.člen_1, p.člen_2 = q.člen_2, ..., p.člen_{k-2} = q.člen_{k-2},
  p.člen_{k-1} < q.člen_{k-1}
5 za vsak  vzorec c ∈ C_k
6     za vsako  (k − 1) − podmnožico s iz c
7         če s ∉ L_{k-1} potem
8             izbriši c iz C_k;
```

dobljenega elementa ni prisotna v L_{k-1}, element izbrišemo, podobno kakor pri drugem koraku funkcije *apriori_gen*. Pogoj $p.\text{člen}_{k-1} < q.\text{člen}_{k-1}$ poskrbi da nimamo dvojnih elementov. Iz tega lahko sklepamo da je $L_k \subseteq C_k$. Po podobnem razmišljanju opazimo, da se enako zgodi pri koraku rezanja C_k. Rezanje ne izbriše elementov, ki bi lahko bili v L_k.

Generiranje kandidatov lahko še dodatno razširimo. Iz množice C_k, dobljene v k-tem koraku, direktno generiramo množico kandidatov C'_{k+1}. Tak pristop se obnese v kasnejših fazah algoritma, kadar je cena shranjevanja dodatnih množic kandidatov $C'_{k+1} - C_{k+1}$ manjša kakor cena pregleda celotne podatkovne množice. Zavedati pa se moramo dejstva, da je $C_{k+1} \subseteq C'_{k+1}$, saj množica C'_{k+1} ni generirana iz L_{k+1}.

Funkcija *podmnožica* poišče vse kandidate $c \in C_k$, ki so vsebovani v transakcijah $t \supseteq c$, z drugimi besedami preveri, ali ima določen c vsaj kakšno pojavitev v podatkovni množici T. Kandidate shranjuje v razpršilno drevesno strukturo. Kandidatni vzorci $c \in C_k$ so shranjeni v listih drevesa, notranja vozlišča so razpršilne tabele, ki vsebujejo kazalce na vozlišča višje stopnje. Koren drevesa je stopnje 1. Notranja vozlišča na globini m imajo kazalce usmerjene na vozlišča, ki so na globini $m + 1$. Vsako vozlišče je kreirano kot list in se, kadar je število vzorcev večje od določene meje, pretvori v notranje vozlišče. Kandidate $c \in C_k$ vstavljamo v koren in se premikamo do listov s pomočjo razpršilne funkcije. V prvem koraku vstavljanja izvedemo razpršilno funkcijo na prvem elementu $i \in c$, v m-tem koraku pa na m-tem elementu ($i_m \in c$). Postopek nadaljujemo dokler ne pridemo do lista.

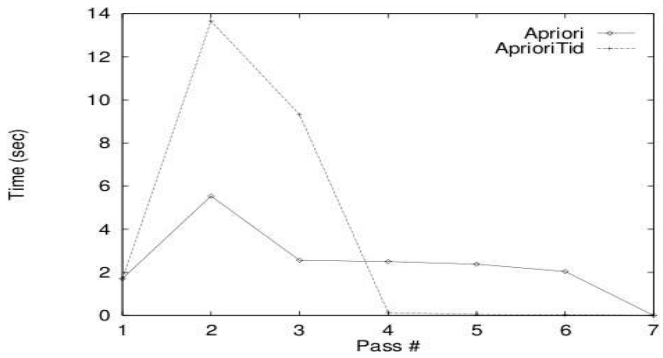
Iskanje $c \subseteq t$ v drevesu izvedemo na naslednji način. Če je element c vsebovan v transakciji t, mora biti prvi element $i_c \in c$ tudi vsebovan v $i_c \in t$. Razpršilno funkcijo v prvem koraku izvedemo na vsakem elementu $i \in t$. Na ta način ignoriramo vse kandidate $c' \in C_k$, za katere velja da prvi element $i_1 \in c'$ ni vsebovan v t, posledično tudi $c' \not\subseteq t$. V primeru da koren ni list in je i' prvi element nekega kandidata c, bo

razpršena vrednost $i^{'}$-ja zapisana v korenu drevesa oziroma je vrednost v tabeli kar kazalec na naslednje vozlišče. Število takih elementov $i^{'}$ nam pove koliko je možnih vzorcev c, ki bi lahko bili vsebovani v tej transakciji. Za vsako poddrevo, na katero smo dobili kazalec, postopek rekurzivno ponovimo in preverimo, ali je razpršena vrednost katerega od $i \in t$ v tabeli korena poddrevesa. Ker so kandidatni vzorci c urejeni, nam v notranjih vozliščih (korenih poddreves) ni potrebno izvesti razpršilne funkcije na vseh elementih transakcije t, temveč le na tistih, ki so večji od $i^{'}$. Ko se nahajamo v listu, pogledamo kateri vzorci $c \in C_k$ so vsebovani v transakciji t $[c \subseteq t]$ in te dodamo v množico C_t.

3.1.2 Različice in izboljšave algoritma Apriori

Poleg osnovnega algoritma, ki smo ga spoznali v prejšnjem razdelku, obstaja tudi algoritem AprioriTid [5]. Za iskanje kandidatnih vzorcev je v tem algoritmu uporabljena enaka funkcija kakor v osnovnem in sicer *apriori_gen*. Razlika je v tem, da za preverjanje podpore vzorcev pregledujemo množico $\overline{C}_k$ in ne celotne podatkovne množice H. Vsak element množice $\overline{C}_k$ je oblike $< TID, \{X_k\} >$, kjer je X_k vzorec vsebovan v transakciji predstavljeni z TID. Če je $k = 1$, je množica $\overline{C}_k$ kar enaka H, le da je element i predstavljen kot vzorec $\{i\}$. Za večje $k > 1$ je množica $\overline{C}_k$ generirana s strani algoritma iz množic C_k in $\overline{C}_{k-1}$. Elementi množice $\overline{C}_k$ v razmerju s transakcijo t so oblike $< t.TID, \{c \in C_k | c \; je \; vsebovan \; v \; t\}$. Za velike vrednosti k-ja je lahko število zapisov iz $\overline{C}_k$ manjše od števila transakcij v podatkovni množici H, za zapise pa velja, da so lahko manjši od ustrezne transakcije. Nasprotno velja za majhne vrednosti k, saj zapisi iz C_k vsebujejo vse kandidate dolžine k iz ustrezne transakcije.

V začetnih fazah se Apriori izkaže za časovno učinkovitejši algoritem. V kasnejših fazah se načeloma število kandidatnih vzorcev zmanjša. Kadar je množica $\overline{C}_k$ dovolj majhna, da lahko zapišemo vse njene elemente direktno v pomnilnik, se hitrost algoritma AprioriTid bistveno zmanjša. Primer je na sliki 1, ko je $k = 4$ se hitrost AprioriTid algoritma občutno izboljša v primerjavi z hitrostjo algoritma Apriori. Algoritem AprioriHibrid [5] je algoritem, ki izkorišča prednosti algoritmov Apriori in AprioriTid. V začetnih fazah je za iskanje vzorcev uporabljen Apriori, ko pa je celotna množica $\overline{C}_k$ dovolj majhna, da jo lahko zapišemo direktno v pomnilnik, se iskanje izvaja z algoritmom AprioriTid [31].



Slika 1: Časovna učinkovitost algoritmov Apriori in AprioriTid na podatkovni množici T10.I4.D100K

3.2 Odločitvena pravila in drevesa

Vsak element podatkovne množice $I \in H$ je vektor oblike $I = \{i_1,\ i_2, \ldots, i_y\}$. Element $i_y \in I$ je ciljni atribut ali razred, katerega vrednost želimo napovedati. Za napovedovanje se poslužujemo klasifikacijskih pravil, ki jih dobimo na podlagi ujemanja i_y z ostalimi atributi. Poglejmo si primer iz tabele 2. Vreme, Temp, Vlaga, Veter in Razred predstavljajo atribute. Za odločitvena drevesa velja, da mora biti ciljni atribut, razred diskreten. V nasprotnem primeru imamo opravka z regresijskimi drevesi.

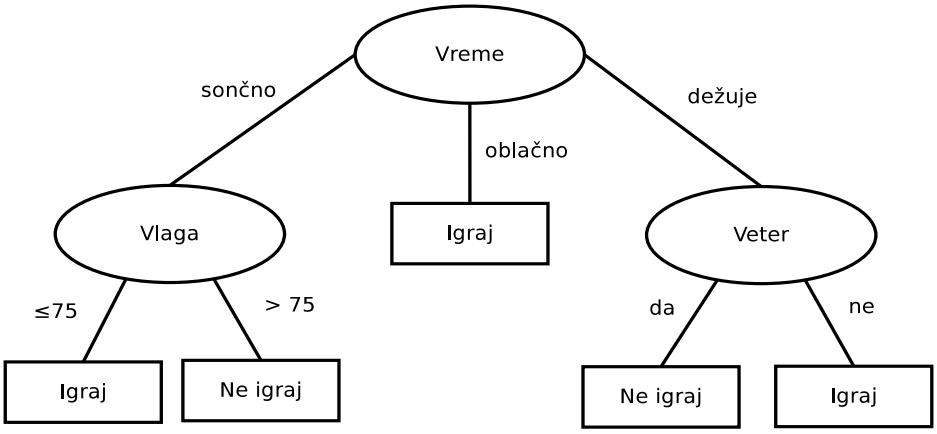
Tabela 2: Atributna predstavitev

Vreme	**Temp (°F)**	**Vlaga(%)**	**Veter**	**Razred**
sončno	75	70	da	Igraj
sončno	80	90	da	Ne igraj
sončno	85	85	ne	Ne igraj
sončno	72	95	ne	Ne igraj
sončno	69	70	ne	Igraj
oblačno	72	90	da	Igraj
oblačno	83	78	ne	Igraj
oblačno	64	65	da	Igraj
oblačno	81	75	ne	Igraj
deževno	71	80	da	Ne igraj
deževno	65	70	da	Ne igraj
deževno	75	80	ne	Igraj
deževno	68	80	ne	Igraj
deževno	70	96	ne	Igraj

Odločitvena pravila lahko predstavimo v obliki implikacije. Če velja nekaj potem velja nekaj drugega. Poglejmo si katera pravila lahko sestavimo za napoved razreda.

če Vreme=sončno **in** Vlaga ≤ 75	$\Rightarrow$	Igraj
če Vreme=sončno **in** Vlaga > 75	$\Rightarrow$	Ne igraj
če Vreme=oblačno	$\Rightarrow$	Igraj
če Vreme=dežuje **in** Veter = da	$\Rightarrow$	Ne igraj
če Vreme=dežuje **in** Veter = ne	$\Rightarrow$	Igraj

Dobljena odločitvena pravila smo povzeli iz odločitvenega drevesa na sliki 2.



Slika 2: Odločitveno drevo za primer iz tabele 2

Vsaka pot od korena drevesa do lista predstavlja eno odločitveno pravilo. Notranja vozlišča predstavljajo atribute, veje drevesa ponazarjajo različne vrednosti atributa, listi pa predstavljajo razredne vrednosti [11].

3.2.1 Odločitvena drevesa

Osnovnemu modelu generiranja odločitvenih dreves pravimo TDIDT (ang. Top Down Induction of Decision Trees). Model je tipa deli in vladaj. V vsaki iteraciji skušamo množico razpoložljivih učnih primerov razbiti na podmnožice glede na atribut, ki ima največje ujemanje z razredom (najboljši atribut). Iteracije se ponavljajo dokler ni zadovoljen nek ustavitveni pogoj, ki je lahko:

- velika večina primerov pripada istemu razredu,
- premalo primerov za nadaljevanje ali

- ni več dobrih atributov

Za izbiro atributa, ki ima najboljše ujemanje z razredom, si pomagamo z informacijskimi merami, kot so informacijski prispevek (ang. information gain), Gini-indeks in ReliefF, ki so podrobneje opisani v [35]. Kolikšno je število primerov, da ni zadovoljen ustavitveni pogoj določi uporabnik. Kadar je število primerov manjše ali enako določeni vrednosti, algoritem 3 ustvari list in ustavi nadaljnjo gradnjo.

Algoritem 3: Osnovni algoritem učenja odločitvenega drevesa

```
1 če ustavitveni pogoj potem
2     postavi list, ki vključuje vse učne primere;
3 sicer
4     izberi najboljši atribut i_i;
5     označi naslednike z vrednostmi atributa i_i;
6     za vsako vrednost v_{i,j} atributa i_i
7         rekurzivno zgradi poddrevo z ustrezno podmnožico učnih primerov;
```

Nižji nivoji odločitvenega drevesa znajo biti nezanesljivi saj, vozliščem ustreza majhno število učnih primerov. Taka vozlišča ali celo poddrevesa po potrebi porežemo (ang. pruning). Poznamo dva osnovna načina rezanja: predhodno rezanje (ang. pre-pruning) in naknadno rezanje (ang. post-pruning). Predhodno rezanje izvajamo med samo gradnjo odločitvenega drevesa. Ko je nadaljnja gradnja nezanesljiva ali nepotrebna, jo skušamo ustaviti z ustavitvenimi pogoji. Med samo gradnjo je težko oceniti kdaj le-ta postane nezanesljiva, zato se pogosto poslužujemo naknadnega rezanja (algoritem 4), ki ga izvajamo po končani gradnji odločitvenega drevesa [16].

Algoritem 4: Osnovni algoritem za naknadno rezanje

```
1 Ponavljaj za vsa vozlišča (razen listov) od spodaj navzgor
2     oceni povprečno pričakovano klasifikacijsko napako v poddrevesih;
3     oceni pričakovano klasifikacijsko napako v trenutnem vozlišču;
4     če povprečna pričakovana napaka poddreves > pričakovane napake vozlišča
      potem
5         poreži poddrevesa;
6         spremeni vozlišče v list;
```

3.2.2 Algoritem C4.5

C4.5 je algoritem, ki gradi odločitvena drevesa in je nadgradnja ID3 (Iterative Dichotomiser 3) algoritma [52]. Za razliko od Apriori algoritma, ki išče asociacijska pravila na podlagi pogostih vzorcev, algoritem C4.5 išče odločitvena pravila za napoved razreda tako, da izbira najboljši atribut za razvejitev na podlagi informacijskega prispevka. V vsaki fazi se izbere natanko en atribut i_a, ki ima največji informacijski prispevek, po naslednji enačbi:

$$\begin{aligned} IP(T, i_a) &= H(T) - H(T|i_a) \\ IP(T, i_a) &= H(T) - \sum_{v \in vals(i_a)} \frac{|\{I \in T|a = v\}|}{|T|} \cdot H(\{I \in T|a = v\}) \end{aligned} \tag{5}$$

kjer je T množica tesnih primerov oblike $(I, y) = (i_1, i_2, \ldots, i_k, y)$ in $a \in vals(i_a)$ je vrednost a-tega atributa vzorca I, $y \in vals(i_y)$ pa je vrednost razreda, ki pripada temu primeru. $H(T)$, entropija, je povprečna informacija na vzorec in je podrobneje opisana v [57] [10]. Na podlagi dobljenega atributa se učni primeri razdelijo na več podmnožic.

Če si zamislimo podatkovno množico kakor H in instance kakor $I_1, I_2, \ldots, I_n \in H$, je vsaka instanca I_x y-dimenzionalen vektor ($I = \{i_1, i_2, ..., i_y\}$), kjer i_a predstavlja atribut na a-tem mestu. V vsaki iteraciji algoritem izbere atribut i_a, ki ima najvišji informacijski prispevek, in razdeli preostalo množico na podlagi vrednosti atributa (npr. vse instance, ki imajo vrednost atributa $x_a = 0$ so v eni podmnožici, $x_a = 1$ v drugi itd.). V naslednjih fazah tega atributa ne pregledujemo več, kajti vsak atribut je lahko izbran le enkrat. Postopek nadaljujemo dokler ni zadovoljen kak ustavitveni pogoj.

Za podatkovne množice, nad katerimi želimo pognati algoritem C4.5 mora veljati, da je razred nominalen (diskreten), atributi pa poljubni. V primeru, da je razred zveznega tipa, se izvede diskretizacija [1] [51].

C4.5, podobno kot ostala odločitvena drevesa, velikokrat kaže manjšo zanesljivost na nižjih nivojih. Algoritem C4.5 uporablja dva načina naknadnega rezanja:

- naknadno *rezanje z zmanjšano napako* (ang. reduced-error pruning)
- *pesimistično* naknadno rezanje (ang. pessimistic pruning)

Rezanje z zmanjšano napako ocenjuje klasifikacijsko napako na ločeni testni množici. Če imamo dovolj veliko podatkovno množico, da si lahko privoščimo veliko testno množico, so ocene nepristranske in dokaj zanesljive. V primeru da temu ni tako, se lahko poslužujemo pesimističnega rezanja, ki klasifikacijsko napako ocenjuje kar nad

[1] Diskretizacijske metode so podrobneje opisane v [17]

učno množico. Predpostavka je, da je množica binomsko porazdeljena, zgornja meja pa je aproksimirana s pomočjo intervala zaupanja (stopnja zaupanja je nastavljiv parameter). V podatkovni množici z n instancami, kjer ima vsaka po y atributov, je zahtevnost algoritma, ki nam vrne k pravil:

$$O(y \cdot n \cdot log(n)) \tag{6}$$

3.2.3 PART algoritem

PART algoritem je kombinacija algoritmov C4.5 in RIPPER [19], ki sta ga leta 1998 predstavila Eibe Frank in Ian Witten [24]. Algoritem deluje po principu deli in vladaj. Za vsako pravilo generira porezano odločitveno drevo. List, kateremu pripada največ instanc, se nato uporabi za pravilo (pot od korena do lista), drevo pa se izbriše in postopek ponovi, dokler nismo obdelali vseh instanc. Instance, ki pripadajo *listu pravila*, v naslednjih iteracijah ne upoštevamo.

Sprotna gradnja in izbris dreves je lahko zamuden postopek, vendar nam tako ni potrebno skrbeti za problem prekomernega rezanja, kakor pri algoritmu C4.5. Ideja je sestaviti delno odločitveno drevo, ki ga ni več mogoče poenostaviti. Delno odločitveno drevo se od običajnega odločitvenega drevesa razlikuje v tem, da nimajo vse veje poddreves. Tako drevo dobimo z integracijo rezanja in sestavljanja drevesa do točke, ko ni več možno poenostavljati. Psevdokoda algoritma je sledeča:

Algoritem 5: Del PART algoritma, ki zgradi odločitveno drevo iz dane množice

```
1 izberi razdelitev danega niza primerov v podmnožice;
2 dokler velja obstajajo nerazširjene podmožice in vse do sedaj razširjene
  podmnožice so listi naredi
3 |  izberi naslednjo podmnožico in jo razširi;
4 če vse razširjene podmnožice so listi potem
5 |  poskusi zamenjati vozlišče z listom;
```

V prvem koraku algoritem razdeli podatkovno množico na podmnožice po enakem principu kakor algoritem C4.5. Vsako podmnožico nato razdeli na manjše podmnožice glede na entropijo. Atribut z manjšo entropijo ima prednost, saj je večja verjetnost, da bomo v manj iteracijah prišli do lista (če bi izvedli še eno iteracijo bi se število instanc, ki pripada pravilu zmanjšalo, pravilo bi bilo bolj specifično). Kadar nadaljnja razdelitev podmnožic ni več mogoča oz. kadar se podmnožica razširi v list se izvede vračanje (ang. backtracking) in pregleda vsa vozlišča drevesa. Če obstaja kako notranje vozlišče, ki ima vse otroke razširjene do listov, se izvede rezanje. Za vsakega otroka

se algoritem odloči (enako kakor algoritem C4.5) ali je smiselno zamenjati celotno poddrevo z listom. Če se izvede zamenjava, algoritem nadaljuje s procesom vračanja. V primeru, da med vračanjem naletimo na vozlišče, ki ima vse otroke take, ki niso listi, bo algoritem ustavil delo na tem drevesu. Tak primer nastane, kadar se algoritem ne odloči zamenjati poddrevesa z listom. Ko ni več možno rezanje, se iz končne verzije drevesa izbere tisto pravilo, ki ustreza največ instancam. Prednost PART algoritma v primerjavi z C4.5 je, da ni potrebno globalno optimizirati.

Če podatkovna množica, nad katero poganjamo algoritem, nima manjkajočih vrednosti ter vsebuje dovolj veliko število instanc, da algoritmu ni potrebno rezanje, je potrebno pregledati le eno izmed poti od lista do korena. To lahko pripomore k izboljšanju učinkovitosti algoritma. V nasprotnem primeru se instance z manjkajočimi vrednostmi razdelijo po podmnožicah s težo proporcionalno številu instanc, ki pripada podmnožici, normalizirano s številom instanc brez manjkajočih vrednosti. Med testiranjem se postopek ponovi za vsako pravilo posebej, teža pa se ponovno preračuna na podlagi uporabnosti pravila (bolj kot je bilo pravilo uporabljeno na testni množici, večjo težo nosi vrednost atributa). Od prvotne teže instance se odšteje teža uporabnosti in ko pridemo do vrednosti nič, so napovedane vrednosti razreda združene v končno pravilo.

Gradnja odločitvenega drevesa ima časovno kompleksnost $O(y \cdot n \cdot log(n))$. Za generiranje množice pravil velikosti k, bomo torej porabili $O(k \cdot y \cdot n \cdot log(n))$. Če predpostavimo, da je število odločitvenih pravil konstanta kakor v analizah [19, 26], potem je časovna kompleksnost algoritma PART kar enaka:

$$O(y \cdot n \cdot log(n)) \tag{7}$$

3.3 Razlikovanje med iskanjem asociacijskih pravil in klasifikacijo

Iskanje asociacijskih pravil poteka po točno definiranih smernicah in lahko rečemo, da je deterministične narave. Razlog bi lahko bil, da iskanje asociacijskih pravil poteka na eni podatkovni množici in ni napovedovanja ali testiranja. Asociacijska pravila vemo točno katera so, na podlagi dane podpore in zaupanja. Pravila lahko opredelimo z minimalno dovoljeno natančnostjo (lahko tudi 100%). Vsi algoritmi za iskanje asociacijskih pravil na podlagi mer podpora in zaupanje morajo po definiciji podati enaka pravila in enako število le teh. Razlikujemo jih po njihovi strukturi (spominski zahtevnosti) in času, ki je potreben da se algoritem izvede [25].

Klasifikacija načeloma temelji na napovedi in trdimo lahko, da je nedeterministične

narave. Klasifikacijska pravila v splošnem generiramo na učni množici U, testiramo pa na testnih množicah, različnih od U. Ker poteka iskanje in preverjanje na različnih podatkovnih množicah, ne moremo trditi da je klasifikacija na učni množici enako točna kakor na testni preden smo izvedli testiranje. Velikokrat pride do napak zaradi narave podatkov, kajti učno množico uporabimo za gradnjo klasifikatorja, testno pa za testiranje, zato ne moremo vedno tako zanesljivo (npr. Z vsaj enako točnostjo kakor pri učni množici) napovedati rezultata, saj testne množice v naprej ne poznamo. Klasifikacijska pravila si lahko predstavljamo tudi kot intuitivne napovedi. Napovedi so nedeterministične narave, saj ne moremo zagotovo napovedati pravilnega dogodka, če se ta še ni zgodil.

Kot zanimivost si poglejmo primer zaporednih številk: 1, 4, 9, 16, ?. če premislimo katera bi bila naslednja številka, bi verjetno odgovorili z? Sam sem sprva dejal 25, saj številke nakazujejo polinom n^2, dejansko pa je lahko rešitev drugačna. Freitas je v [25] uporabil polinom $(-5n^4 + 50n^3 - 151n^2 + 250n - 120)/24$, ki da enaka začetna števila. Naslednja številka je v tem primeru 20. Skušali smo pokazati, da obstaja več različnih polinomov (neskončno), ki dajo enake začetne številke (1, 4, 9, 16). Če to posplošimo na primer klasifikacijskih pravil ter učne(1, 4, 9, 16) in testne (?) množice opazimo nedeterminizem.

Pri iskanju razlik med asociacijskimi pravili in klasifikacijskimi pravili lahko omenimo tudi prekomerno prileganje (ang. Overfitting) in njegovo nasprotje prekomerno poenostavitev (ang Underfitting), ki so podrobneje opisani v [22]. Algoritmi za iskanje asociacijkih pravil poiščejo vsa pravila, ki ustrezajo dani podpori in zaupanju, neglede na to, ali pravila prekomerno poenostavijo podatke ali povzročijo prekomerno prileganje. Lahko bi celo rekli, da prekomerno poenostavitev oziroma prekomerno prileganje pri iskanju asociacijskih pravil zanemarimo. Še več, ker asociacijskia pravila ne iščemo na testni množici, temveč na učni, ne moremo ugotoviti ali se prekomerna poenostavitev oziroma prileganje dejansko pojavi.

Popolnoma nasprotno lahko trdimo za iskanje klasifikacijskih pravil. Prekomerno prileganje in prekomerna poenostavitev sta dva od vzrokov zakaj je iskanje klasifikacijskih pravil tako zelo zapleteno, saj ne smemo dopustiti da pride do prekomernega prileganja oziroma poenostavitve. Večina algoritmov za klasifikacijo ima že implementiran proces, kako se izogniti prekomernemu prileganju oziroma poenostavitvi, v nasprotnem primeru bi pomenilo degradacijo pri učinkovitosti napovedi na testni množici (saj je ne poznamo) [25].

Pristranskost (ang. Inductive bias) je pomemben faktor pri klasifikaciji, ne pa pri asociacijskih pravilih. Michalski [43] pravi, da je potencialno število predpostavk (klasifikacijskih pravil), za dano podatkovno množico, neskončno. Pristranskost nekega algoritma lahko opredelimo kakor skupek predpostavk, ki jih uporabimo za učenje.

Naučeno nato algoritem uporabi za napovedovanje rezultatov na množicah, ki jih ne pozna. Lahko bi tudi rekli, da je pristranskost osnova, ki da prednost nekaterim predpostavkam pred ostalimi. V primeru, da pristranskosti ne bi upoštevali, bi se lahko algoritmi za klasifikacijo naučili le najenostavnejše predpostavke in nobena predpostavka ne bi imela prednosti pred ostalimi. Iz tega povzamemo, da ima vsak algoritem za iskanje klasifikacijskih pravil pristranskost. Primerjava dveh različnih algoritmov za klasifikacijo je tako mogoča le, če oba testiramo na enakih podatkovnih množicah.

Algoritmi za iskanje asociacijskih pravil nimajo pristranskosti, saj algoritem ne vrne predpostavk (napovedi) temveč pravila. Lahko bi rekli, da minimalna podpora in zaupanje definirta pristranskost nekega algoritma za iskanje asociacijskih pravil. Minimalna podpora in zaupanje pa sta definirani s strani uporabnika, tako da je v tem primeru bolj pristranskost s strani uporabnika kakor pa algoritma.

Razlika med iskanjem asociacijskih pravil in klasifikacijo je tudi na ravni sintakse. Za iskanje odločitvenega pravila se na vsaki stopnji poslužujemo le enega atributa, medtem ko pri iskanju asociacijskih pravil že v drugi iteraciji uporabljamo več atributov hkrati. Iskanje pravil lahko ločimo tudi po simetriji. Pri iskanju odločitvenega pravila se osredotočimo na iskanje točno definiranega atributa (npr. Razred), ki ga dobimo z rudarjenjem ostalih atributov, zato lahko rečemo, da je iskanje klasifikacijskih pravil asimetrično iskanje glede na atribute. Pri iskanju asociacijskih pravil ne določimo atributa v naprej in vsak atribut se lahko pojavi večkrat, v več različnih pravilih, zato bi iskanje asociacijskih pravil lahko opredelili kot simetrično glede na atribute.

Vse do sedaj navedene razlike veljajo, če imamo v mislih običajno iskanje asociacijskih pravil. Če bi dobljena asociacijska pravila posplošili ali celo rezali, potem lahko govorimo tudi o nedeterminizmu. V naslednjem poglavju si bomo pogledali ali so lahko asociacijska pravila nedeterminističa, odločitvena.

4 Asociacijska pravila za klasifikacijo

Asociacijska klasifikacija je klasifikacija na podlagi asociacijskih pravil, ki jih dobimo z iskanjem (odkrivanjem) asociacij med atributi in razredom [32]. Za gradnjo klasifikacijskega modela uporabimo asociacijska pravila, ki vsebujejo razred. Osnovni meri za iskanje in odkrivanje asociacijskih pravil sta zaupanje in podpora. Čeprav sta ti meri dobri, še ne moremo reči da sta idealni, saj je v veliko primerih, generirano ogromno število asociacijskih pravil, ki lahko negativno vplivajo na učinkovitost klasifikacije in v takih primerih pravila porežemo.

Vsi algoritmi za iskanje asociacijskih pravil na osnovi podpore in zaupanja delujejo po enakem principu. Algoritmi, kot je npr. Apriori, najprej poiščejo vse pogoste vzorce, ki zadovoljujejo pogoju minimalne podpore, nato se s pomočjo zaupanja sestavijo asociacijska pravila. Za klasifikacijo se izmed vseh pravil izlušči tista, ki vsebujejo razred. Dobljeni pogosti vzorci so tako naslednje oblike: $V = \{X, i_y\}$, kjer X predstavlja atribute, i_y pa razred. Iz takih pogostih vzorcev nato sestavimo klasifikacijska pravila oblike $X \Rightarrow i_y$.

4.1 CAR

CAR (ang. Class Association Rules) so pravila, ki združujejo asociacijska pravila s klasifikacijo. CAR so predstavili Bing Liu, Wynne Hsu in Yiming Ma leta 1998 [39]. Ideja je integracija klasifikacijskih metod in asociacijskega iskanja brez izgub zmogljivosti in z asociacijskimi (ne odločitvenimi) pravili.

Prvi korak je iskanje asociacijskih pravil, zato se v ta namen uporabi algoritem, kot je npr. Apriori. Podatkovne množice so po navadi ogromne in imajo veliko asociacij, zato se s pomočjo rezanja poišče le tiste vzorce, ki vsebujejo razred. Pri tem nam pomaga kar sam algoritem z nastavljivim parametrom, ki zahteva prisotnost določenega atributa [58] (Razred). Rudarjenje zveznih atributov potrebuje še veliko raziskav, zato v takih primerih atribute diskretiziramo. Obstaja veliko dobrih algoritmov za diskretizacijo [21,23], zato ne bomo podrobneje obdelali tematike. Iskanje CAR sestoji iz treh korakov:

- diskretizacija zveznih atributov, če ti obstajajo,
- generiranje vseh asociacijskih pravil, ki imajo na desni strani razred,
- gradnja klasifikatorja na podlagi dobljenih pravil.

Model v splošnem pripomore k večji natančnosti napovedi kakor algoritem C4.5 in klasifikacija poteka z uporabo asociacijskih pravil. Tak pristop pripomore tudi pri reševanju problemov, kot je problem razumljivosti [18, 47].

Problem razumljivosti (ang. understandability problem) nastane, kadar so dobljena pravila pri običajnih klasifikacijskih metodah za nas nerazumna, saj algoritmi uporabljajo domensko neodvisno pristranskost (ang. domain indipendent bias) in hevristiko za gradnjo pravil. Zgodi se lahko, da so nekatera pravila, ki se nam zdijo logična izpuščena, spet druga pravila, ki nimajo smisla, pa uporabljena za klasifikacijo s strani algoritma. Iskanje razumnih pravil je s pristopom CAR poenostavljeno, saj najprej poiščemo vsa pravila, kar nam pomaga identificirati razumna pravila [37–39].

Poleg razumnih pravil si želimo, da so najdena pravila zanimiva. Iskanje zanimivih pravil je dokaj zahtevna naloga in velikokrat se zgodi, da algoritmi, kot je C4.5, taka pravila izpustijo, saj klasificirajo z omejenim številom pravil in prednost imajo pravila z večjo podporo.

CAR pravilo je implikacija oblike $X \Rightarrow i_y$, kjer je $X \subseteq I$ in ima podporo s. Za iskanje pravil CAR uporabljamo algoritem CBA (ang. Classification Based on Associations).

4.1.1 CBA algoritem

Algoritem delimo na dva dela. Prvi del je generator pravil (CBA-RG), ki temelji na Apriori algoritmu, drugi del je gradnja klasifikatorjev (CBA-CB).

CBA-RG

Glavna naloga CBA-RG je poiskati pogoste vzorce. To stori tako da najprej poišče vse vzorce oblike $< condset, y >$, ki imajo podporo večjo od minimalne po enakem principu kakor Apriori. *Condset* predstavlja določene vrednosti atributov ali z drugimi besedami, *condset* predstavlja vzorec X, y pa razred. Število pojavitev *condset* v podatkovni množici bomo poimenovali *condsupŠtevec*. Od vseh pojavitev (*condsupŠtevec*) je število takih, ki imajo razred $y-$*rulesupŠtevec*. Vsaka tak vzorec predstavlja pravilo, ki ima podporo:

$$s = (\frac{rulesupŠtevec}{n}) \cdot 100\% \tag{8}$$

in zaupanje:

$$c = (\frac{rulesupŠtevec}{countsupŠtevec}) \cdot 100\%, \tag{9}$$

kjer je n število instanc v podatkovni množici H.

V primeru da nam enaki nabor *condset* vrne različni vrednosti za razred y, za generiranje pravila uporabimo tisto vrednost, ki ima večjo podporo. V skrajnem primeru ko sta podpori enaki pa izberemo naključno.

Z večkratnim pregledovanjem podatkovne množice algoritem generira vse pogoste vzorce. S prvim pregledom pregleduje posamezne vzorce in izloči vse, ki imajo podporo manjšo od določene. V drugem pregledu algoritem išče nove morebitne pogoste vzorce iz pogostih vzorcev dobljenih iz prejšnje iteracije (enako kakor Apriori). Ko je pregled zaključen izberemo kandidatne vzorce, ki so dejansko pogosti in iz njih sklepamo pravila (CARs).

Poglejmo si psevdo kodo algoritma CBA-RG:

Algoritem 6: CBA-RG algoritem

```
1  F_1 = {veliki 1 − pogostiVzorci};
2  CAR_1 = genPravila(F_1);
3  prCAR_1 = reziPravila(CAR_1);
4  za (k = 2; F_{k−1} ≠ ∅; k++)
5      C_k = candidateGen(F_{k−1});
6      za vsak primer I ∈ H
7          C_I = podmnPravil(C_k, I);
8          za vsak kandidat c ∈ C_d
9              c.countsupŠtevec++;
10             če I.razred = c.razred potem
11                 c.rulesupŠtevec++;
12     F_k = {c ∈ C_k | c.rulesupŠtevec ≥ minsup};
13     CAR_k = genPravila(F_k);
14     prCAR_k = reziPravila(CAR_k);
15 CARs = ⋃_k CAR_k;
16 prCARs = ⋃_k prCAR_k
```

Recimo da je $k-vzorec$ tak vzorec, katerega *condset* premore k elementov. Množica

tovrstnih $k - vzorcev$ je F_k ($k - vzorec \in F_k$). Elementi množice F_k so oblike:

$$< (condset, countsupŠtevec), (y, rulesupŠtevec) > .$$

C_k je množica kandidatnih $k - vzorcev$. V prvi vrstici algoritem poišče pogoste vzorce. Funkcija *genPravila* generira *CAR-1* in zatem funkcija *prunePravila(CAR-1)* pravila poreže. Rezanje je prisotno pri vsaki naslednji iteraciji in v principu poteka enako kakor rezanje pri algoritmu C4.5. Če ima pravilo p_1 večjo pesimistično napako kakor pravilo p_2 (dobljeno z izbrisom enega od pogojev p), potem je p zoperstavljen rezanju.

V vsaki iteraciji algoritem izvede štiri operacije:

- pogosti vzorci F_{k-1}, ki jih dobimo v $k-1$ koraku, so uporabljeni za generiranje kandidatnih vzorcev C_k,
- z vnovičnim pregledom ponovno nastavimo minimalno podporo kandidatov v C_k,
- algoritem nato z novimi pogostimi vzorci generira *CAR-k*,
- na koncu *CAR-k* obrežemo.

CBA-CB gradnja klasifikatorja

Izbira optimalnega klasifikatorja, ki nam na tesni množici najmanjkrat zgreši napoved, zahteva pregled vseh podmnožic pravil. Takih podmnožic je 2^m, kjer je m število pravil. Število pravil je lahko ogromno, zato je tako iskanje velikokrat v praksi neizvedljivo. Algoritem CBA-CB, ki si ga bomo ogledali je hevrističen, vendar se obnese bolje kakor klasifikator pri C4.5.

Definicija 1. Če imamo dve pravili r_i in r_j velja, da ima r_i prednost pred r_j ($r_i \succ r_j$) če velja:

- zaupanje od r_i je večje od zaupanja r_j,
- ob enakem zaupanju ima r_i večjo podporo kakor r_j,
- ob enaki podpori in zaupanju, je bil r_i generiran pred r_j.

Označimo vsa pravila CAR s črko R in podatkovno množico s črko H. Ideja je, da izbiramo taka pravila $r_i \in R$, ki imajo visoko prednost in pokrijejo čim večji del podatkovne množice H. Klasifikator je oblike:

$$< r_1, r_2, ..., r_n, privzet\_razred > .$$

Pri klasifikaciji še neznanega primera, za klasifikacijo uporabimo prvo pravilo, ki ustreza primeru. V primeru da nobeno pravilo ne ustreza danemu primeru, klasifikator vrne

privzet razred (enako kakor C4.5). Naivna verzija algoritma za gradnjo klasifikatorja M1 je sledeča:

Algoritem 7: CBA-CB M1 algoritem

```
1  R = sort(R);
2  za vsako pravilo r ∈ R v zaporedju
3      temp = ∅;
4      za vsak primer I ∈ H
5          če I zadovoljuje pogoje r potem
6              shrani I.id v temp in označi r, če pravilno klasificira I;
7      če r je označen potem
8          vstavi r na konec C; izbriši vse primere iz H, ki imajo id-je shranjene v
           temp;
9          izberi privzet razred za trenutni C;
10         izračunaj število napak za C;
11 Poišči prvo pravilo p ∈ C z najmanjšim številom napak in izbriši vsa naslednja
   pravila od p;
12 Dodaj ustrezen privzet razred za vse primere iz C, ki so nasledniki p;
13 vrni klasifikator C
```

V prvi vrstici psevdo kode razvrstimo klasifikatorje iz R glede na prednost, kar nam zagotavlja, da bomo za dan klasifikacijski primer izbrali klasifikator z največjo natančnostjo. Do enajste vrstice (v.11) algoritem za vsak $r \in R$ pregleda H in išče primere ki ustrezajo r. Če r pravilno klasificira vsaj en primer $I \in H$, ga označimo z $I.id$, ki je edinstvena identifikacijska številka za vsak pravilno klasificiran I. Pravilno klasificirane primere nato začasno odstranimo iz H. Iz preostale podatkovne množice nato izberemo tisti razred, ki se največkrat pojavi (algoritem ZeroR). Nato se izračuna kolikšna je napaka pri napovedi na celotni podatkovni množici, ki je enaka seštevku napak pri klasifikaciji z r in napak pri napovedi razreda. Nepotrebna pravila, tista ki ne doprinesejo večje natančnosti klasifikatorju, nato odstranimo (v.11). Pravila ki jih nismo odstranili in privzet razred iz C tvorijo končni klasifikator.

Vsak testni primer je pokrit s pravilom, ki ima največjo prednost in vsako pravilo iz C klasificira vsaj eno instanco. Algoritem je kljub enostavnosti neučinkovit, zlasti v primerih, kadar se podatkovna množica ne nahaja v glavnem pomnilniku, saj algoritem velikokrat pregleduje celotno podatkovno množico, kar je zelo zamuden postopek. Naslednji algoritem je nadgradnja algoritma M1 in sicer M2. Namesto da za vsako pravilo pregledamo podatkovno množico, pri M2 poiščemo najboljše pravilo, ki zadosti vsakemu primeru.

Algoritem 8: CBA-CB M2 algoritem: Faza-1

```
1  Q = ∅;
2  U = ∅;
3  A = ∅;
4  za vsak primer I ∈ H
5      cPravilo = maxPokritPravila(C_c, I);
6      wPravilo = maxPokritPravila(C_w, I);
7      U = U ∪ {cPravilo};
8      cPravilo.pokritiPrimeri[I.razred] + +;
9      če cPravilo ≻ wPravilo potem
10         Q = Q ∪ {cPravilo};
11         označi cPravilo;
12     sicer
13         A = A ∪ < I.id, I.razred, cPravilo, wPravilo >
```

V prvi fazi za vsak primer I poiščemo pravili, ki imata največjo prednost, pogoj pa je, da eno pravilo pravilno klasificira I ($cPravilo$), drugo pa ne ($wPravilo$). V primeru, da je $cPravilo \succ wPravilo$, $cPravilo$ klasificira I, v nasprotnem primeru pa algoritem hrani strukturo oblike

$$< I.ID, i_y, cPravilo, wPravilo >,$$

kjer je $I.ID$ identifikacijska številka od primera I, i_y pa razred.

Recimo da A označuje zbirko vseh vzorcev oblike $< I.ID, i_y, cPravilo, wPravilo >$, U množico vseh $cPravil$ in Q množico takih $cPravil$, ki imajo prednost pred ustreznimi $wPravili$.

Funkcija $maxPokritPravila$ poišče pravilo z največjo prednostjo, ki pokrije primer I. C_c je skupek pravil, ki imajo enak razred kakor I (C_w nasprotno). Za vsako $cPravilo$ si z uporabo polja $pokritiPrimeri$ algoritem zapomni število primerov, ki jih pravilo pokriva za vsak posamezen razred.

Vsak I, ki ga nismo uspeli pripisati nobenemu pravilu, ponovno obdelamo in poiščemo vsa pravila, ki nepravilno klasificirajo I in imajo večjo prednost kakor ustrezno $cPravilo$. Za to operacijo je potreben ponoven pregled podatkovne množice.

Algoritem 9: CBA-CB M2 algoritem: Faza-2

```
1  za vsak primer < I.ID, i_y, cPravilo, wPravilo >∈ A
2      če wPravilo je označeno potem
3          cPravilo.pokritiPrimeri[i_y] − −;
4          wPravilo.pokritiPrimeri[i_y] + +;
5      sicer
6          wPravila = vsipokritiPrimeri(U, I.ID.primer, cPravilo);
7          za vsako pravilo w ∈ wPravila
8              w.zamenjaj = w.zamenjaj ∪ {< cPravilo, I.ID, i_y};
9              w.pokritiPrimeri[i_y] + +;
10         Q = Q ∪ wPravila
```

Če je $wPravilo$ označeno pomeni, da je to pravilo za nek drugi I opredeljeno kakor $cPravilo$. V takem primeru vemo, da bo pravilo pokrilo primer $I.ID$. Pogoja, da je vsak primer pokrit s pravilom, ki ima največjo prednost in vsako pravilo iz C je klasifikator vsaj dvema primeroma, sta zadovoljena. Število primerov, ki jih pokrivata $cPravilo$ in $wPravilo$ se posodobi (v.3-4). Funkcija $vsipokritiPrimeri$ poišče vsa pravila, ki napačno klasificirajo $I.ID$ vendar imajo večjo prednost kakor ustrezno $cPravilo$ (v.5). Vsako posamezno pravilo ima polje zamenjav, v katerega shranjujemo taka pravila. Povečanje števca $w.pokritiPrimeri[i_y]$ oznani da pravilo w lahko pokrije dan primer. Q vsebuje vsa pravila, ki jih potrebujemo za gradnjo klasifikatorja.

Izbira končnega nabora pravil klasifikatorja ima dva koraka. Prvi korak je izbira niza potencialnih pravil za gradnjo klasifikatorja, v drugem koraku pa odstranimo pravila, ki ne doprinesejo k večji točnosti klasifikatorja in vrnemo končni klasifikator C. Poglejmo si tretjo fazo algoritma M2:

Algoritem 10: CBA-CB M2 algoritem: Faza-3

```
1  classDistr = compClassDistri(H);
2  napakePravil = 0;
3  Q = sort(Q);
4  za vsako pravilo r ∈ Q v zaporedju
5      če r.pokritiPrimeri[r.razred] ≠ 0 potem
6          za vsak < rul, dID, y >∈ r.replace
7              če I.ID pokrit s strani drugega r potem
8                  r.pokritiPrimeri[i_y] − −;
9              sicer
10                 rul.pokritiPrimeri[i_y] − −;
11             napakePravil = napakePravil + napakeOdPravila(r);
12             classDistr = obnovi(r, classDistr);
13             privzetRazred = izberiPrivzet(classDistr);
14             privzeteNapake = privNap(privzetRazred, classDistr);
15             skupneNapake = napakePravil + privzeteNapake;
16             Vstavi < r, privzet − razred, skupneNapake > nakonecC
17 Poišči prvo pravilo p ∈ C, ki ima najmanjšo vrednost skupneNapake in zavrzi
   vsa naslednja pravila od p v C;
18 Vstavi privzet razred vezan na p na konec C;
19 vrni C brez skupneNapake in privzetRazred
```

Glede na prednost pravila sortiramo Q, kar nam zagotavlja izpolnjenje prvega pogoja. Funkcija *compClassDistr* prešteje število testnih primerov za vsak posamezen razred. Števec *napakePravil* hrani napake, ki smo jih naredili na testni množici do tega trenutka. Če pravilo r ne klasificira pravilno nobenega testnega primera, ga zavržemo (v.5). V nasprotnem primeru r uporabimo kot pravilo za klasifikator. Pogoj 2 je s tem izpolnjen. Če je bil $I.ID$ predhodno pokrit s strani kakega pravila, tisto pravilo (rul) ostane nespremenjeno, v nasprotnem primeru r zamenja vsa pravila (rul), ki imajo manjšo prednost (v.6). Vrednosti $r.pokritiPrimeri[i_y]$ ali $rul.pokritiPrimeri[i_y]$ se ustrezno posodobita (v7-10). Za vsako izbrano pravilo posodobimo tudi *napakePravil* in *classDistr* (v.11-12). Privzet razred je večinski razred preostale podatkovne množice, ki ga dobimo na podlagi izračuna *classDistr* (v.13). Kadar poznamo privzet razred lahko izračunamo *privzeteNapake*, ki jih naredimo na preostanku testne množice (v.14). Iz tega podatka lahko nato izračunamo *skupneNapake*, ki jih dobimo pri klasifikaciji s pravili iz C in privzetim razredom (v.15).

V naslednjem poglavju bomo do sedaj predstavljene algoritme primerjali med seboj.

5 Primerjava algoritmov

Naš cilj je primerjati uspešnost napovedi algoritmov kot sta *Apriori* in *CBA* v primerjavi z algoritmi za klasifikacijo kot sta *PART* in *C4.5*. Opazovali bomo čas izvedbe algoritma, natančnost napovedi razreda in število generiranih pravil. Primerjave bomo izvajali s pomočjo odprtokodnega, v javi implementiranega programa, *WEKA 3.7.10*[1]. Nad vsako podatkovno množico bomo pognali tudi algoritem *ZeroR*, ki nam bo služil za določitev privzete najmanjše sprejemljive natančnosti. Vsako podatkovno množico bomo testirali večkrat, z uporabo različnih nastavitev in upoštevali najboljšo meritev. Testirali bomo z 10-kratnim prečnim preverjanjem (ang. 10-fold cross-validation). Privzete nastavitve algoritmov so sledeče:

1. C4.5:
   - Faktor zaupanja: 0,25
   - Minimalno število instanc na list: 2
2. PART:
   - Faktor zaupanja: 0,25
   - Minimalno število instanc na list: 2
   - Seme pri rezanju z zmanjšano napako: 1
3. Apriori: Nastavitve bomo prilagajali za čim natančnejšo napoved
4. CBA: Nastavitve bomo prilagajali za čim natančnejšo napoved

[1] *http://www.cs.waikato.ac.nz/ml/weka/index.html*

5.1 Podatkovne množice

Testiranje bomo izvedli nad dvaindvajsetimi podatkovnimi množicami iz spletnega repozitorija UCI-ML[2]. Izbrane množice so opredeljene kot klasifikacijske, razredni atributi so v vseh primerih nominalni. Nekatere množice imajo zvezne atribute, katere bomo s pomočjo programa Weka pri testiranju diskretizirali. Opisi podatkovnih množic so objavljeni v UCI-ML repozitoriju, imena pa so vidna v tabeli 3 skupaj z rezultati meritev.

5.2 Testiranje

Za namene testiranja smo z ukazom $-Xmx4000M$ programu $Weka$ dodelili 4GB spomina. Privzete nastavitve algoritmov PART in C4.5 ne bomo spreminjali, zato pa bomo za vse podatkovne množice prilagodili iskanje z algoritmoma Apriori in CBA. Parametra, ki ju bomo optimizirali sta pri obeh algoritmih enaka in sicer maksimalno število generiranih pravil in spodnja meja podpore. Parameter zaupanja med testiranjem nismo spreminjali, nastavljen pa je bil na vrednost 90%. Po nekaj poskusih na podatkovnih množicah smo ugotovili, da je optimalna meja števila generiranih pravil 2000, kar pomeni, da v kolikor algoritem generira več pravil kolikor je meja, se vsa nadaljnja pravila ne upoštevajo. Izjeme, kjer je bila meja presežena, sto bili podatkovne množice *Ionosphere*, *Nursery* in *Car-evaluation-database*. Parameter število pravil smo zato v teh primerih zvišali na 3500 in tako nekoliko povečali natančnost.

Drugi nastavljiv parameter je bil meja podpore. Odločili smo se za dokaj nizko mejo, kajti želeli smo več pravil za bolj natančno napoved. Privzeta vrednost je bila $0,1\%$. Testiranja so pokazala, da v kolikor smo mejo povečali, je natančnost padla. V naslednjih primerih smo morali zvišati prag podpore, saj program $Weka$ ni uspel analizirati množic v kolikor je bila podpora manjša od navedene v spodnjem seznamu. Dobljeni rezultati so nastali na podlagi naslednjih nastavitev podpore:

- *Wisconsin-breast-cancer* 0,1
- *Credit-rating* 0,11594
- *Lymphography* 0,145
- *Mushroom* 0,225
- *Nursery* 0,005
- *Cleveland-14-heart-disease* 0,12
- *Ionosphere* 0,44

[2] *http://archive.ics.uci.edu/ml/datasets.html*

V primerih da smo bodisi nastavili število pravil na več kakor 3500, bodisi zmanjšali minimalno mejo zaupanja na manj kakor $0,1\%$, nam je program *Weka* javil napako. Program bi za izračun potreboval več spomina, kakor smo ga imeli na razpolago. Taki primeri so v tabeli 3 prazni. V nekaterih primerih smo problem rešili s povečanjem meje zaupanja. Kljub temu nismo uspeli izračunati vseh vrednosti, zato smo polja v tabeli 3 pustili prazna.

5.3 Pričakovanja

Glede na dosedanje raziskave [39] [38] bi morale biti napovedi vsaj tako točne kakor pri običajnih klasifikacijskih algoritmih. Zavedati se moramo, da je pri klasifikaciji z asociacijskimi pravili generirano ogromno število pravil. Če pravila režemo se zna zgoditi, da bo točnost napovedi padla, prav tako, če zmanjšamo mejo podpore ali zaupanja. Optimalno mejo je težko določiti v prvem poskusu zato bomo vsako podatkovno množico večkrat obdelali z različnimi nastavitvami in upoštevali najboljše.

Za generiranje velikega števila pravil potrebujemo tudi zelo zmogljiv sistem. Lahko se zgodi, da bomo zaradi preobremenjenosti sistemov morali prekiniti izvajanje algoritma nad določenimi podatkovnimi množicami, ki imajo večje število instanc oz. atributov. V takih primerih bomo morali nastavitve iskanja prirediti sistemu in posledično bodo napovedi lahko manj zanesljive.

V splošnem pričakujemo, da bodo napovedi razredov natančnejše kakor pri klasifikacijskih algoritmih, saj bo generirano večje število pravil, kar pomeni da z enim pravilom zajamemo enako ali manj možnosti.

5.4 Primerjava rezultatov

Tabela 3: Rezultati meritev

	Apriori			PART			C4.5			ZeroR	CBA		
Množica	čas	št. pravil	točnost	čas	št. pravil	točnost	čas	št. pravil	točnost	točnost	čas	št. pravil	točnost
balance-scale-weight	0.05	825	69.28	0.05	34	77.28	0	33	64.48	45	1.23	303	65.6
wisconsin-breast-cancer	0.03	874	94.4206	0.03	10	93.8484	0	14	94.5637	65.5222	0.23	32	95.279
car-evaluation-database	0.17	3280	89.7569	0	68	95.7755	0	131	92.3611	70.0231	0.5	23	91.1458
credit-rating	0.28	1962	86.3768	0.02	23	85.3623	0	30	86.087	55.5072	0.88	46	79.4203
hepatitis				0	8	84.5161	0	11	83.871	79.3548	0.08	17	82.5806
cris	0.2	157	97	0.02	8	97	0	10	97	33.3333	0.02	8	97
king-rook vs king				8.99	3706	54.3484	0.08	4128	56.5868	33.3333			
lymphography	0.11	1609	78.3784	0	13	76.3514	0	21	77.027	54.7297	8.19	2	70.2703
mushroom	1.03	1432	98.9168	0.02	13	100	0	25	100	51.7971	17.48	8	93.4023
nursery	2.17	2085	87.1836	0.14	220	99.2052	0.02	359	97.0525	33.3333	115.08	4	70.7099
primary-tumor				0.02	43	40.708	0	88	39.823	24.7788			
$08 Solar flare CLASS_X$	0.14	623	99.1361	0	7	98.8481	0	1	99.1361	99.1361	0,08	0	99.1361
tic-tac-toe	0.24	1884	97.8079	0.02	50	94.4676	0	95	85.0731	65.3445	4.11	28	98.8518
zoo				0	8	92.0792	0	9	92.0792	40.5941	0.48	1	60.396
anneal				0.02	17	98.2183	0.02	35	98.441	76.1693			
cleveland-14-heart-disease	0.09	1808	82.8383	0.06	19	79.868	0	30	77.5578	54.4554	2.05	73	82.1782
sick				0.06	20	98.6214	0.05	34	98.807	93.8759	4.34	0	93.8759
sonar				0.02	8	80.2885	0.02	18	71.1538	53.3654			
vehicle				0.05	29	71.513	0.02	98	72.4586	25.6501	5.64	12	49.1726
waveform				2.06	93	77.42	0.38	330	75.08	33.84			
ionosphere	0.78	2332	64.1026	0.03	10	91.7379	0.02	18	91.453	64.1026	0.27	6	84.6154

Dobljeni rezultati se v večini primerov ne skladajo s pričakovanji. Algoritem Apriori se je izkazal za neučinkovitega ko gre za klasifikacijo. Skoraj polovico testnih množic Apriori algoritem ni uspel klasificirati. Točnost je sicer v izvedljivih primerih zelo primerljiva z algoritmoma PART in C4.5 vendar je čas izvedbe povprečno skoraj 10 krat večji. Kljub temu lahko časovno razliko zanemarimo saj ni tako očitna, kajti časi so zelo majhni v razponu od 0,001 do 2 sekunde. Najbolj opazna razlika je v številu generiranih pravil algoritma Apriori, kar pa smo nekako pričakovali. V enajstih primerih je zaradi prevelikega števila pravil prišlo do napake v testnem okolju. Naprave na katerih smo testiranje izvajali (4GB RAM) niso prenesle tako velike obremenitve spomina. Podobnih problemov z algoritmi PART in C4.5 nismo imeli. Slednja algoritma sta za napoved potrebovala bistveno manjše število pravil in spomina.

V prvem testiranju z algoritmom Aprioti nismo uporabili rezanja. V drugem poskusu smo pravila rezali vendar se rezultati točnosti in časovne zahtevnosti niso spremenili na boljše, manjše je bilo le število generiranih pravil. Odločili smo se, da bomo upoštevali meritve brez rezanja, ki so tudi zapisane v tabeli 3.

Iz tabele 3 lahko opazimo, da so izvedljive meritve dokaj primerljive. Imeli smo primere kot npr. pri množici *Nursery*, kjer sta algoritma Apriori in CBA negativno izstopala. Po drugi strani smo imeli tudi množico *Cleveland-14-heart-disease*, kjer sta se bolje odrezala algoritma Apriori in CBA.

6 Zaključek

Model, ki temelji na podpori in zaupanju je kritiziran s strani mnogih avtorjev [1] [3] [12]. Eden od razlogov je težavnost nastavitve parametrov (podpore in zaupanja). Če izberemo visoko mejo za minimalno zaupanje, dobimo le osnovna pravila, izpustimo pa nekatera pravila, ki bi lahko bila zanimiva. Po drugi strani, če izberemo nižjo mejo za minimalno zaupanje, dobimo ogromno število pravil, lahko tudi redundantnih. Pri zaupanju ne upoštevamo nič drugega kakor pogojno verjetnost pravil, ki vodijo do asociacij. Pokazano je bilo, da imajo dobljena pravila z visoko podporo in zaupanjem negativno povezavo s predhodniki in nasledniki [12].

Poleg podpore in zaupanja obstajajo še druge mere za iskanje asociacijskih pravil. Uporabljajo se predvsem kot filtri ali razvrščevalniki za nezanimiva pravila. Za ugotovitev, ali so ostale mere boljše kakor podpora in zaupanje, je bila narejena raziskava v [32]. Izkazalo se je da so nekatere mere lahko boljše, vendar nobena ni vidno izstopala pri vseh poskusih. Če je bila na določenih podatkih neka mera najboljša, je bila na drugih podatkih ta ista mera slabša od ostalih. Zanimiva ugotovitev je, da mera, ki vrne največjo natančnost pri celotnem seznamu odločitvenih pravil (bodisi odločitveno drevo C4.5 ali seznam PART), ni najnatančnejš pri porezanem seznamu pravil. Za naše raziskovalno delo smo uporabili model na podlagi zaupanja in podpore. Kljub temu da ima nekaj slabosti, je še zmeraj najbolj uporabljen postopek za iskanje asociacijskih pravil, saj nobena druga mera vidno ne izstopa.

Iz naših poskusov je razvidno, da sta podpora in zaupanje dobri meri in vprašanje je, ali bi z drugimi merami dosegli boljše rezultate. Res, da smo bili pred poskusi optimisti in pričakovali boljše rezultate, vendar nismo imeli zadostne sistemske zmogljivosti za generiranje večjega števila pravil. Kljub temu moramo poudariti da to ni bila ovira pri klasifikacijskih algoritmih PART in C4.5. Naš poskus je tako pokazal, da za klasifikacijo z asociacijskimi algoritmi potrebujemo veliko spominskega prostora. Posledično lahko izjavimo, da se načeloma ne priporočamo izvajanja klasifikacije z asociacijskimi algoritmi, če nimamo dovolj zmogljivih računalnikov.

V primeru da bi imeli boljšo opremo, bi bili rezultati najverjetneje drugačni. Kljub temu pa ne moremo trditi, da bi bila klasifikacija z asociacijskimi pravili zagotovo bolj natančna od standardne klasifikacije. Običajne klasifikacijske metode, katere uporabljata tudi algoritma PART in C4.5 so občutno manj časovno potratne, kakor metode

iskanja asociacijskih pravil in klasifikacije z le temi, ki jih uporabljata algoritma Apriori in CBA.

Če potegnemo črto, smo med celotnim poskusom spoznali asociacijska in klasifikacijska pravila, ter kako najdemo tovrstna pravila v podatkovni množici. Odgovorili smo na vprašanje, ali so algoritmi za iskanje asociacijskih pravil primerljivi z algoritmi za klasifikacijo. Za nadaljno raziskovanje bi bilo zanimivo, če bi poskuse ponovili na bolj zmogljivih računalnikih, takih, ki premorejo bistveno večje obremenitve spomina.

7 Literatura

[1] J.M. Adamo, *Data mining for association rules and sequential patterns: sequential and parallel algorithms*, Springer-Verlag. (2001)

[2] F. Afrati, A. Gionis in H. Mannila, Approximating a collection of frequent sets, *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (2004), 12–19. *(Citirano na strani 31.)*

[3] C.C. Aggarwal in P.S. Yu, A new framework for itemset generation, *PODS: Proceedings of the 17th symposium on Principles of Database Systems* ACM (1998), 18–24. *(Citirano na strani 4.)*

[4] R. Agrawal, T. Imielinski in A. Swami, Mining association rules between sets of items in large databases, *Proceedings of the ACM SIGMOD International Conference on Management of Data* (Washington D.C., Maj 1993), 207–216. *(Citirano na strani 31.)*

[5] R. Agrawal in R. Srikant, Fast Algorithms for Mining Association Rules, *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB* (Santiago, Chile, September 1994), 487–499. *(Citirano na strani 5.)*

[6] R. Agrawal in R. Srikant, Mining sequential patterns, *Data Engineering, 1995, Proceedings of the Eleventh International Conference* (1995), 3–14. *(Citirano na straneh 7 in 10.)*

[7] Y. Aumann in Y. Lindell, A statistical theory for quantitative association rules, *Journal of Intelligent Information Systems* 20(3) (2003), 255–283. *(Citirano na strani 4.)*

[8] R.J. Bayardo, Efficiently mining long patterns from databases, *ACM Sigmod Record* 27(2) (1998), 85–93. *(Citirano na strani 4.)*

[9] F. Bonchi in C. Lucchese, On condensed representations of constrained frequent patterns, *Knowledge and Information Systems* 9(2) (2006), 180–201. *(Citirano na strani 3.)*

[10] M. Borda, *Fundamentals in Information Theory and Coding*, Springer. (2011) str. 11 *(Citirano na strani 4.)*

[11] L. Breiman, J. Freidman, R. Olshen in C. Stone, Classification and Regression Trees, *Wadsworth International Group* (Belmont, CA. 1984), . *(Citirano na strani 14.)*

[12] S. Brin, R. Motwani in C. Silverstein, IGMOD '97: Proceedings of the 1997 ACM SIGMOD international conference on Management of data, *ACM* (1997), 265-276. *(Citirano na strani 12.)*

[13] D. Burdick, M. Calimlim in J. Gehrke, MAFIA: A maximal frequent itemset algorithm for transactional databases, *IN ICDE* (2001), 443-452. *(Citirano na strani 31.)*

[14] T. Calders in B. Goethals, Mining All Non-derivable Frequent Itemsets, *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery* Springer-Verlag (2002), 74–85. *(Citirano na strani 3.)*

[15] T. Calders in B. Goethals, Depth-first non-derivable itemset mining, *Proc. of the 2005 SIAM International Data Mining Conference* Newport Beach (2005), 250-261. *(Citirano na strani 4.)*

[16] B. Cestnik in I. Bratko, On estimating probabilities in tree pruning, *V Proceedings of the European Working Session on Learning* (Kodratoff Y., ur.), Springer-Verlag (Porto 1991), 138–150. *(Citirano na strani 4.)*

[17] M.R. Chmielewskihal in J.W. Grzymala-Busse, Global discretization of continuous attributes as preprocessing for machine learning, *International Journal of Approximate Reasoning* V. 15 (November 1996), 319–331. *(Citirano na strani 13.)*

[18] P. Clark in S. Matwin, Using qualitative models to guide induction learning, *ICML-93* (1993), 49–56. *(Citirano na strani 14.)*

[19] W.W. Cohen, Fast effective rule induction, *Procedings of the 12th International Conference on Machine Learning* Morgan Kaufman Publishers (San Francisco, CA, 1995), 115–123. *(Citirano na strani 20.)*

[20] G. Dong in J. Li, Efficient mining of emerging patterns: Discovering trends and differences, *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining* KDD'99 (1999), 43–52. *(Citirano na straneh 15 in 16.)*

[21] J.R. Dougherty, R. Kohavi in M. Sahami, Supervised and unsupervised discretization of continuous features, *A. Prieditis & S. Russell (Eds.), Proceedings of the Twelfth International Conference on Machine Learning,* Morgan Kaufmann (San Francisco, CA, 1995), 194–202. *(Citirano na strani 4.)*

[22] B.S. Everitt, *The Cambridge Dictionary of Statistics*, Cambridge University Press. (2002) *(Citirano na strani 19.)*

[23] U.M. Fayyad in K.B. Irani, Multi-interval discretization of continuous-valued attributes for classification learning, *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence,* Morgan Kaufmann (San Francisco, CA, 1993), 1022–1027. *(Citirano na strani 17.)*

[24] E. Frank in I. Witten, Generating accurate rule sets without global optimization, *Machine Learning: Proceedings of the Fifteenth International Conference,* Morgan Kaufmann Publishers (San Francisco, CA, 1998), . *(Citirano na strani 19.)*

[25] A.A. Freitas, Understanding the crucial differences between classification and discovery of association rules, *SIGKDD Explorations* (2000), 65–69. *(Citirano na strani 15.)*

[26] J. Fürnkranz, Pruning algorithms for rule learning, *Technical Report TR-96-07* Australian Research Institute for Artificial Intelligence (Dunaj, 1996), https://www.ai.univie.ac.at/papers/oefaitr-96-07.ps.Z. *(Citirano na straneh 16 in 17.)*

[27] G. Grahne in J. Zhu, Efficiently using prefix-trees in mining frequent itemsets, *Proceedings of the ICDM Workshop on Frequent Itemset Mining Implementations* (2003), . *(Citirano na strani 16.)*

[28] R. Han, M. Kamber in J. Pei, *Data Mining: Concepts and Techniques, Second Edition*, Morgan Kaufmann. 2006 *(Citirano na strani 3.)*

[29] J. Han, J. Pei in Y. Yin, Mining frequent patterns without candidate generation, *Proceedings of the 2000 ACM SIGMOD international conference on Management of data* ACM SIGMOD Record, V.29 (New York, 2000), 1–12. *(Citirano na straneh 1, 3, 4 in 5.)*

[30] J. Han, J. Pei, Y. Yin in R. Mao, Mining frequent patterns without candidate generation: A frequent-pattern tree approach., *Data Mining and Knowledge Discovery* V.8 (2003), 53–87. *(Citirano na strani 3.)*

[31] M. Hegland, The Apriori Algorithm - a Tutorial, *Canberra ACT 0200* CMA, Australian National University John Dedman Building, Canberra ACT 0200 (2005), www2.ims.nus.edu.sg/preprints/2005-29.pdf. *(Citirano na strani 3.)*

[32] M.J. Heravi in O. R. Zaïane, A study on interestingness measures for associative classifiers, *Proceedings of the 2010 ACM Symposium on Applied Computing* (SAC 2010), 1039–1046. *(Citirano na strani 10.)*

[33] M. Kamber, J. Han in J.Y. Chiang, Metarule-guided mining of multidimensional association rules using data cubes, *KDD* (1997), 207–210. *(Citirano na straneh 1, 19 in 31.)*

[34] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen in A.I. Verkamo, Finding interesting rules from large sets of discovered association rules, *Proceedings of the third international conference on Information and knowledge management* CIKM'94 (1994), 401–408. *(Citirano na strani 4.)*

[35] I. Kononenko, *Strojno učenje*, Založba FE in FRI. Ljubljana, 1997 *(Citirano na strani 4.)*

[36] B. Lent, A. Swami in J. Widom, Clustering association rules, *Data Engineering, 1997. Proceedings. 13th International Conference* (1997), 220–231. *(Citirano na strani 13.)*

[37] B. Liu in W. Hsu, Post-analysis of learned rules, *AAAI-96, Vol. 1* (1996), 828–834. *(Citirano na strani 4.)*

[38] B. Liu, W. Hsu in S. Chen, Using general impressions to analyze discovered classification rules, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining* (Newport Beach, CA, 1997), 31–36. *(Citirano na straneh 4 in 20.)*

[39] B. Liu, W. Hsu in Y. Ma, Integrating classification and association rule mining, *Proceedings of the 4rd International Conference Knowledge Discovery and Data Mining (KDD-98), AAAI Press* (1998), 80–86. *(Citirano na straneh 4, 20 in 29.)*

[40] G. Liu, J. Li, L. Wong in W. Hsu, Positive Borders or Negative Borders: How to Make Lossless Generator-based Representations Concise, *Proceeding of the 2006 SIAM international conference on data mining (SDM'06)* Bethesda, MD (2006), 467–471. *(Citirano na straneh 1, 4, 19, 20 in 29.)*

[41] G. Liu, H. Lu, W. Lou in J.X. Yu, On computing, storing and querying frequent patterns, *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (2003), 612. *(Citirano na strani 4.)*

[42] J. Mata, J.L. Alvarez in J.C. Riquelme, Discovering Numeric Association Rules via Evolutionary Algorithm, *6TH CONF. ON KNOWLEDGE DISCOVERY AND DATA MINING* 2336 (2002), 40–51. *(Citirano na strani 3.)*

[43] R.W. Michalski, A theory and methodology of inductive learning, *Artificial Inteligence 20* (1983), 111–161. *(Citirano na strani 4.)*

[44] R.J. Miller in Y. Yang, Association rules over interval data, *ACM SIGMOD Record* 26(2) (1997), 461. *(Citirano na strani 17.)*

[45] R.T. Ng, L.V. Lakshmanan, J. Han in A. Pang, Exploratory mining and pruning optimizations of constrained associations rules, *ACM SIGMOD Record* 27(2) (1998), 13–24. *(Citirano na strani 4.)*

[46] N. Pasquier, Y. Bastide, R. Taouil in L. Lakhal, Discovering frequent closed itemsets for association rules, *Database Theory ICDT 99* (1999), 398–416. *(Citirano na strani 4.)*

[47] M. Pazzani, S. Mani in W.R. Shankle, Beyond concise and colorful: learning intelligible rules, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining* (Newport Beach, CA, 1997), 235–238. *(Citirano na strani 3.)*

[48] J. Pei, J. Han in L.V. Lakshmanan, Mining frequent itemsets with convertible constraints, *Proc. 2001 Int' l Conf. Data Eng. (ICDE 01)* (April, 2001), 433–332. *(Citirano na strani 20.)*

[49] J. Pei, J. Han in R. Mao, CLOSET: An efficient algorithm for mining frequent closed itemsets, *CM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery* (2000), 21–30. *(Citirano na strani 4.)*

[50] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal in M.C. Hsu, PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth, *Proceedings of the International Conference on Data Engineering* (2001), 215–226. *(Citirano na strani 3.)*

[51] J.R. Quinlan, Improved use of continuous attributes in C4.5, *Journal of Artificial Intelligence Research* 4 (1996), 77–90. *(Citirano na strani 4.)*

[52] J.R. Quinlan, *C4.5: Programs for machine learning*, Morgan Kaufman Publishers. (California, 1993) *(Citirano na strani 14.)*

[53] A. Salleb-Aouissi, C. Vrain in C. Nortet, Quantminer: A genetic algorithm for mining quantitative association rules, *Proceedings of the 2007 International Join Conference on Artificial Intelligence* (2007), 1035–1040. *(Citirano na strani 14.)*

[54] A. Siebes, J. Vreeken in M. van Leeuwen, Item sets that compress, *Proceedings of SDM'06* (2006), 393–404. *(Citirano na strani 4.)*

[55] R. Srikant in R. Agrawal, Mining generalized association rules, *Future Gener Comput Syst* 13(2) (1997), 161–180. *(Citirano na strani 4.)*

[56] R. Srikant, Q. Vu in R. Agrawal, Mining association rules with item constraints, *KDD* vol. 97 (1997), 67–73. *(Citirano na strani 3.)*

[57] C.E. Shannon, A Mathematical Theory of Communication, *Bell System Technical Journal* 27 (July/October 1948), 379–423. *(Citirano na strani 4.)*

[58] C.A. Shoemaker in C. Ruiz, Association Rule Minning Algorithms for Set-Valued Data, *Department of Computer Science, Worcester Polytechnic Institute* Lecture Notes in Computer Science Vol. 2690 (Worcester USA, 2003), 669–676. *(Citirano na strani 14.)*

[59] M. Steinbach, P.N. Tan in V. Kumar, Support envelopes: a technique for exploring the structure of association patterns, *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (2004), 296–305. *(Citirano na strani 19.)*

[60] J. Wang, J. Han in J. Pei, Closet+: Searching for the best strategies for mining frequent closed itemsets, *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (2003), 245. *(Citirano na strani 4.)*

[61] X. Yan, H. Cheng J. Han in D. Xin, ummarizing itemset patterns: a profile-based approach, *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining* KDD'05 (2005), 314–323. *(Citirano na strani 3.)*

[62] X. Yan, J. Han in R. Afshar, CloSpan: Mining closed sequential patterns in large datasets, *Proc. of SIAM Int. Conf. on Data Mining* SDM '03 (2003), 166–177. *(Citirano na strani 4.)*

[63] C. Yang, U. Fayyad in P.S. Bradley, Efficient discovery of error-tolerant frequent itemsets in high dimensions, *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining* (2001), 194–203. *(Citirano na strani 4.)*

[64] M.J. Zaki, Scalable algorithms for association mining, *IEEE Transactions on Knowledge and Data Engineering* V.12(3) (2000), 327–390. *(Citirano na strani 4.)*

[65] M.J. Zaki, SPADE: An efficient algorithm for mining frequent sequences, *Machine Learning* 42(1) (2001), 31–60. *(Citirano na strani 3.)*

[66] M.J. Zaki in C.J. Hsiao, CHARM: An efficient algorithm for closed itemset mining, *2nd SIAM International Conference on Data Mining* (2002), 457–473. *(Citirano na strani 4.)*

[67] M.J. Zaki in K. Gouda, Fast vertical mining using diffsets, *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining* (Washington, D.C, USA, 2003), 326–335. *(Citirano na strani 3.)*

(Citirano na strani 3.)