UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

MAGISTRSKO DELO

UPORABA ASOCIACIJSKIH PRAVIL ZA KLASIFIKACIJO

JAN ŠKORJANC

2021
MAGISTRSKO DELO
JAN ŠKORJANC

UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Magistrsko delo

Uporaba asociacijskih pravil za klasifikacijo

(Use of association rules for classification)

Ime in priimek: Jan Škorjanc
Študijski program: Računalništvo in informatika, 2. stopnja
Mentor: doc. dr. Branko Kavšek
Somentor: izr. prof. dr. Jernej Vičič

Koper, junij 2021

Ključna dokumentacijska informacija

Ime in PRIIMEK: Jan ŠKORJANC

Naslov magistrskega dela: Uporaba asociacijskih pravil za klasifikacijo

Kraj: Koper

Leto: 2021

Število listov: 61 Število slik: 13 Število tabel: 14

Število referenc: 34

Mentor: doc. dr. Branko Kavšek

Somentor: izr. prof. dr. Jernej Vičič

UDK: 004.021(043.2)

Ključne besede: klasifikacija, strojno učenje, algoritmi, primerjava, asociacijska pravila

Izvleček:

Dve izmed bolj poznanih področij podatkovnega rudarjenja sta klasifikacija in iskanje asociacijskih pravil. V magistrskem delu obe področji združimo. To storimo tako, da analiziramo delovanje klasifikatorjev, ki uporabljajo asociacijska pravila. Med te spadajo CBA, bCBA, wCBA, RCAR, CMAR in CPAR. Naštete algoritme primerjamo z algoritmi J48, PART in JRip, ki sodijo med standardne algoritme, ki uporabljajo pravila. Algoritme zaženemo na podatkovnih zbirkah pridobljenih iz repozitorija UCI ML. Pri tem sledimo metodologiji CRISP-DM, ki je ena izmed najpogosteje uporabljenih metodologij na področju podatkovnega rudarjenja. Pri analizi si pomagamo s programskim okoljem Weka in izbranimi knjižnicami v programskem jeziku R. Ugotovimo, da so uporabljeni algoritmi na podatkovnih zbirkah, ki imajo v povprečju 6892 primerov, v 43,17 sekundah sposobni pravilno klasificirati 74,99% primerov. Pri tem ustvarijo 913,57 pravil. Edini algoritem, ki izstopa je CPAR, ki spada med slabše tako po času, kot po uspešno klasificiranih primerih. Algoritmi, ki uporabljajo asociacijska pravila so v primerjavi s standardnimi algoritmi, ki uporabljajo pravila nekoliko slabši zgolj pri meritvi uspešno klasificiranih primerov.

Key document information

Name and SURNAME: Jan ŠKORJANC

Title of the thesis: Use of association rules for classification

Place: Koper

Year: 2021

Number of pages: 61 Number of figures: 13 Number of tables: 14

Number of references: 34

Mentor: Assist. Prof. Branko Kavšek, PhD

Co-Mentor: Assoc. Prof. Jernej Vičič, PhD

UDC: 004.021(043.2)

Keywords: classification, machine learning, algorithms, comparison, association rules

Abstract:

Two of the most known fields of data mining are classification and association rules discovery. In the master thesis we combine both fields, by analyzing the performance of classifiers that use association rules; for example, there are CBA, bCBA, wCBA, RCAR, CMAR and CPAR. We compare them with standard algorithms that use rules such as J48, PART and JRip. Algorithms are run on datasets retrieved from the UCI ML repository. We follow the CRISP-DM methodology, which is one of the most used methodologies in data mining. For the analysis we use the Weka software and chosen R libraries. We find out that on average the selected algorithms are capable of correctly classifying 74,99% of the instances in 43,17 seconds when the average dataset size is 6892 instances, and they generate on average 913,57 rules. The algorithm which stands out is CPAR, which had the highest running times and lowest classification accuracy. Classifiers that use association rules are not as good as standard algorithms that use rules only at correctly classified instances.

KAZALO VSEBINE

KAZALO PREGLEDNIC

KAZALO SLIK IN GRAFIKONOV

SEZNAM KRATIC

KDD – Odkrivanje znanja in podatkovno rudarjenje (angl. Knowledge Discovery and Data Mining)
CMC – Kombinacija več klasifikatorjev (angl. Combination of Multiple Classifiers)
RAM – Bralno-pisalni pomnilnik (angl. random access memory)
CRISP-DM – Medindustrijski standardni proces za podatkovno rudarjenje (angl. Cross-Industry Standard Process for Data Mining)
IDE – Integrirano razvojno okolje (angl. Integrated development environment)
TIOBE – Pomembnost resnosti (angl. The Importance of Being Earnest)
CRAN – Obširna mreža R arhivov (angl. The Comprehensive R Archive Network)
ARFF – Datotečni format relacij in atributov (angl. Attribute-Relation File Format)
CSV – Vrednosti ločene z vejico (angl. Comma separated values)
URL – Enolični krajevnik vira (angl. Uniform Resource Locator)
CBA – Klasifikacija na podlagi asociacij (angl. Classification Based on Associations)
CBA-RG – Klasifikacija na podlagi asociacij-generiranje pravil (angl. Classification Based on Associations-Rule Generation)
CBA-CB – Klasifikacija na podlagi asociacij-gradnja klasifikatorja (angl. Classification Based on Associations-Classifier Building)
CPAR – Klasifikacija na podlagi napovednih asociacijskih pravil (angl. Classification Based on Predictive Association Rules)
CMAR – Klasifikacija na podlagi več asociacijskih pravil (angl. Classification based on Multiple Association Rules)
RCAR – Regulariziran algoritem za klasifikacijo razrednih asociacijskih pravil (angl. Regularized Class Association Rules algorithm for classification)
FOIL – Induktivni učenec prvega reda (angl. First-Order Inductive Learner)
RIPPER – Ponavljajoče rezanje za zmanjšanje napak (angl. Repeated Incremental Pruning to Produce Error Reduction)
UCI ML – Univerza v Kaliforniji za podatkovno rudarjenje (angl. University of California, Irvine Machine Learning)
JRip – JRipper (RIPPER napisan v programskem jeziku Java)
IREP – Rezanje z namenom zmanjšanja napak (angl. Incremental reduced error pruning)
SQL – Strukturirani povpraševalni jezik (angl. Structured Query Language)
GNU – Zbirka programov (angl. GNU's Not Unix)
LibSVM – Knjižnica za metodo podpornih vektorjev (angl. Library for Support Vector Machines)

FP(-rast) – Pogosti vzorci (angl. frequent pattern)

SPECT – Enofotonska emisijska računalniška tomografija (angl. Single-photon emission computed tomography)

ZAHVALA

Zahvaljujem se mentorju doc. dr. Branku Kavšku in somentorju izr. prof. dr. Jerneju Vičiču za svetovanje in pomoč pri izdelavi magistrskega dela. Prav tako se zahvaljujem celotni družini za vso podporo, ki so mi jo nudili med samim študijem.
Hvala!

1 UVOD

Klasifikacija in iskanje asociacijskih pravil sodita med najbolj uporabljene naloge podatkovnega rudarjenja. Za klasifikacijo je značilno, da na učni množici podatkov zgradi model, s katerim napove enega izmed atributov, ki mu pravimo razred. Asociacijska pravila razrednega atributa nimajo, končna množica pravil pa je lahko izpeljana za katerikoli atribut. Pri algoritmih asociacijskih pravil je pomembno, da na podlagi primerov v podatkovni zbirki najdejo vsa pravila, ki zadovoljijo določene vrednosti. Te vrednosti največkrat predstavljata podpora (angl. support) in zaupanje (angl. confidence). Ena izmed pomembnih razlik med klasifikacijo in asociacijskimi pravili je deterministočnost. Za iskanje asociacijskih pravil se načeloma uporabljajo algoritmi, ki so deterministični glede na zaupanje in podporo, pri klasifikaciji pa nedeterministični.

Kljub nekaterim razlikam med klasifikacijo in asociacijskimi pravili poznamo tudi postopek klasifikacije, pri katerem se uporabljajo asociacijska pravila. Takim pravilom, ki temeljijo na majhni množici pravil, s katerimi ustvarijo klasifikator, pravimo tudi klasifikacijska pravila.

Pri algoritmih, ki uporabljajo pravila se pojavljata dve glavni težavi. Prvo predstavlja postopek, s katerim se določeno pravilo označi kot uporabno in učinkovito. Nekatere metode v ta namen uporabijo posebno številsko vrednost, ki jo določi uporabnik. Med najbolj znane spadata zaupanje in podpora, ki ju je potrebno za vsako podatkovno zbirko prilagoditi. Drugi izmed problemov je velikost podatkov, pri katerem se pojavljajo vprašanja predvsem o načinu shranjevanja pravil ter podatkov, ki so pomembni za pravilno delovanje algoritma. Problem se najlažje reši s prilagajanjem podpore in zaupanja ali krčenjem podatkovne zbirke. Obstajajo tudi algoritmi, ki so namenjeni prav za delo z velikimi množicami podatkov, vendar so zaradi tega običajno manj učinkoviti. Eden izmed izzivov podatkovnega rudarjenja je implementacija algoritmov, ki bi bili na tako velikih množicah podatkov hitri in učinkoviti [19][18][11].

V magistrskem delu se osredotočimo na primerjavo nekaterih algoritmov, ki podatke klasificirajo na podlagi pravil. Med seboj primerjamo algoritme:

- CBA,
- bCBA,
- wCBA,
- RCAR,
- CMAR,
- CPAR,
- J48,
- PART,
- JRip

Algoritmi J48, PART in JRip sodijo med standardne algoritme, ki uporabljajo pravila, medtem ko sodijo CBA, bCBA, wCBA, RCAR, CMAR in CPAR med algoritme, ki uporabljajo asociacijska pravila. Primerjavo naredimo tako, da algoritme zaženemo na več podatkovnih zbirkah ter preverimo njihovo hitrost, točnost in učinkovitost. Pomagamo si s programskim jezikom R in programsko opremo Weka.

V magistrskem delu je najprej predstavljeno podatkovno rudarjenje v splošnem. Poglavje zajema njegovo zgodovino, vrste podatkov in nekaj o komponentah algoritmov, ki so del podatkovnega rudarjenja. Sledi opis CRISP-DM modela, predstavitev strojnega učenja ter njegove prednosti. V tretjem poglavju je opisana uporabljena programska oprema, kateri sledi opis uporabljenih algoritmov. Namen petega poglavja je spoznati podatkovne zbirke, ki so bile pri magistrskem delu uporabljene. Sledi šesto poglavje, v katerem so predstavljeni rezultati algoritmov in njihova analiza. Magistrsko delo se zaključi s sedmim poglavjem, ki je namenjen diskusiji rezultatov.

2 OPIS PODATKOVNEGA RUDARJENJA

Magistrsko delo temelji na podatkovnem rudarjenju. To je proces iskanja določenih pravil in zakonitosti v velikih množicah podatkov. Vključuje področja strojnega učenja, dela s podatkovnimi bazami, statistike in umetne inteligence, pomaga pa nam pri iskanju določenih povezav in vzorcev v podatkovnih zbirkah. Pomembno je, da algoritmi podatkovnega rudarjenja delujejo samostojno, brez intervencije uporabnika in da so zmožni obdelave velikih količin podatkov. Danes se podatkovno rudarjenje uporablja na veliko različnih področjih. Primer predstavljajo podjetja, ki strmijo k čim večjem dobičku, h kateremu pripomore opazovanje trga in trendov. Iz preteklih podatkov lahko določeno podjetje, ki se ukvarja s trženjem napove, kateri produkt bo v določenem obdobju pritegnil največ ljudi k nakupu. Če podjetje takšen izdelek ponudi na trg med prvimi, je verjetnost zaslužka večja. Podatkovno rudarjenje se uporablja tudi v medicini, kjer se z njegovo pomočjo lahko hitreje in učinkovitejše napove diagnozo pacienta. Na medijskem področju se lahko z algoritmi podatkovnega rudarjenja ugotovi, kaj ljudje ob določenih urah najraje gledajo, kar televizijskim hišam pomaga pri sestavljanju sporeda določenega televizijskega programa. Med ostale konkretne primere, kjer se podatkovno rudarjenje uporablja, sodijo detekcija nezaželenih sporočil, napovedovanje vremena in zaznavanje spletnih prevar [33] [6].

Bistvo podatkovnega rudarjenja niso samo dobri algoritmi za napovedovanje, ampak tudi širše in globlje razumevanje podatkov. Pri tem lahko težave povzročajo porazdelitve v podatkih, ki v vsakodnevnih realnih problemih niso nujno Gaussove, poleg tega pa je lahko podatkovna zbirka sestavljena iz primerov, kjer podatki manjkajo ali primerov, kjer vrednosti preveč izstopajo. Podatki so lahko tudi premalo natančni ali netočni. Take podatke je zelo težko odkriti, vendar lahko bistveno vplivajo na končni rezultat, zato jih je potrebno izločiti. Od tradicionalne statistike se podatkovno rudarjenje razlikuje predvsem v količini podatkov, saj pri problemih podatkovnega rudarjenja podatkovne zbirke vsebujejo tudi do nekaj tisoč terabajtov. S tem se pojavijo problemi pri shranjevanju, dostopu in analizi podatkov, s katerimi se statistiki ne soočajo [8].

Velik del razvoja podatkovnega rudarjenja je prispevala KDD konferenca, katere začetki segajo v devetdeseta leta prejšnjega stoletja. V tistem obdobju je bila to majhna delavnica namenjena ožjemu krogu ljudi, danes pa je to vsakoletni dogodek, ki je med najbolj obiskanimi dogodki v svetu računalništva. V začetkih delovanja konference se je podatkovno rudarjenje definiralo kot proces avtomatskega pridobivanja veljavnih, potencialno uporabnih, razumljivih informacij iz velikih podatkovnih baz [8].

2.1 ZGODOVINA PODATKOVNEGA RUDARJENJA

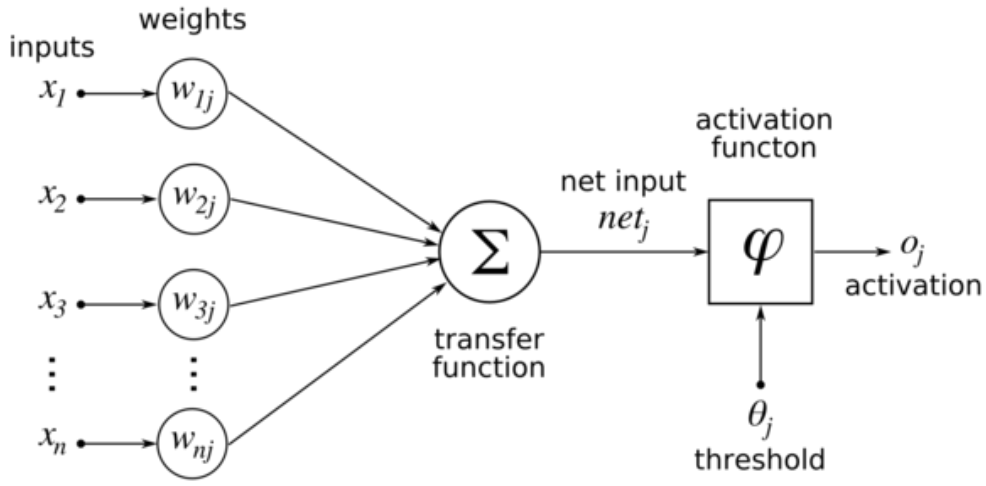
Čeprav se je podatkovno rudarjenje kot ga poznamo danes, razširilo v zadnjih 20 letih, njegovi začetki segajo že v 18. stoletje. Takrat termina podatkovno rudarjenje niso poznali in je področje temeljijo zgolj na statistiki. Bistvena razlika je v tem, da se podatkovno rudarjenje ukvarja z velikimi množicami podatkov, ki pa jih v obdobju pred razvojem računalnika ni bilo moč obdelati.

Prvi izmed mejnikov, ki je vplival na razvoj podatkovnega rudarjenja se je zgodil leta 1763. V tem letu je posthumno izšel članek angleškega matematika Thomasa Bayesa, v katerem je opisal Bayesov izrek oz. Bayesov obrazec. Ta opisuje verjetnost nekega dogodka A, če upoštevamo vse dogodke, ki lahko na dogodek A vplivajo. Bayesov izrek je s tem postal eno izmed pomembnih odkritij statistike, še danes pa se uporablja v enem izmed glavnih algoritmov za klasifikacijo, ki se imenuje Bayesov klasifikator [17].

Leta 1805 sta Adrien-Marie Legendre in Carl Friedrich Gauss raziskovala telesa v vesolju. Uporabila sta algoritem, ki je danes eden izmed ključnih v podatkovnem rudarjenju. To je regresija, ki pomaga določiti povezave med določenimi spremenljivkami. Metoda, ki sta jo omenjena matematika uporabila, se imenuje metoda najmanjših kvadratov [17].

Po skoraj stoletju stagnacije se je zgodila velika prelomnica v svetu računalništva. Leta 1936 je Alan Turing v svojem članku opisal stroj, ki je zmožen operacij, na katerih temelji računalnik še danes. Kljub temu, da je takrat Turing pisal zgolj o teoretičnem stroju, brez njega tako računalnikov, kot podatkovnega rudarjenja še dandanes najbrž ne bi poznali [17].

Tako kot je bil Turing pionir na področju računalništva, sta bila Warren McCulloch in Walter Pitts pionirja na področju nevronskih mrež. Leta 1943 sta v članku prva opisala idejo o delovanju nevronov v mreži. Vsak nevron je bil sposoben vhodne podatke sprejeti, jih obdelati in generirati izhodne podatke. Njuna ideja delovanja nevronov je prikazana na sliki 1.



Slika 1: Prikaz delovanja nevronov po ideji Warrena McCullocha in Walterja Pittsa (vir:[27])

Leta 1965 je Lawrence J. Fogel ustanovil podjetje Decision Science. To je bilo prvo podjetje, ki se je ukvarjalo z evolucijskem računanjem. Glavni cilj tega področja je razvoj algoritmov, ki delujejo po principu naravnega izbora in evolucije [17].

V 70. letih prejšnjega stoletja so se počasi začeli razvijati sistemi za upravljanje podatkovnih baz, ki so bili zmožni shranjevanja nekaj terabajtov podatkov. To je sprožilo potrebo po njihovi analizi, kar je privedlo do razvoja kompleksnejših algoritmov. Z obstoječimi algoritmi namreč takih podatkov ni bilo mogoče obdelati. V istem obdobju so se razvili tudi prvi genetski algoritmi, ki posnemajo procese genetike v organizmih. Tako posnemanje nudi nove ideje pri optimizaciji algoritmov podatkovnega rudarjenja. Primer enega izmed takih algoritmov predstavlja CMC [20] [17].

V 80. letih je podjetje HNC kot blagovno znamko zaščitilo besedno zvezo rudarjenje podatkovnih baz, S tem so želeli zaščititi svoj produkt, ki je bil namenjen grajenju nevronskih mrež. V istem obdobju je leta 1989 Gregory Piatetsky-Shapiro ustanovil delavnico KDD, ki je kasneje postala ena izmed najbolj znanih konferenc podatkovnega rudarjenja [17].

V 90. letih so podatkovno rudarjenje začela množično uporabljati tudi večja podjetja, ki so se ukvarjala s prodajo. Njihov cilj je bila analiza podatkov in prepoznavanje njihovih navad, saj so tako k nakupu svojih izdelkov privabili čim več ljudi. Leta 1992 se je zgodil pomemben korak na področju algoritmov metode podpornih vektorjev. Bernhard E. Boser, Isabelle M. Guyon in Vladimir N. Vapnik so predlagali izboljšavo, s katero se je začel razvoj nelinearnih klasifikatorjev. Do tedaj so bili namreč vsi klasifikatorji linearni. Leto kasneje je prvič izšlo glasilo KDnuggets, ki je bilo sprva namenjeno zgolj obiskovalcem KDD konference. Sčasoma je glasilo prešlo v spletno stran Kdnuggets.com, ki je še danes namenjena širši javnosti [17].

Čeprav je besedna zveza "podatkovna znanost" obstajala že od 60. let 20. stoletja, je bila šele leta 2001 predstavljena kot samostojna disciplina. Prvi so jo uporabili pri opisovanju vlog na Linkedinu in Facebooku. Leta 2003 je podatkovno rudarjenje doseglo tudi šport. Tega leta je namreč izšla knjiga z naslovom MoneyBall, ki je spremenila način iskanja prostih igralcev v baseballu. Knjiga opisuje, kako je ekipa iz Oaklanda v Kaliforniji kot prva uporabila statistični pristop pri iskanju podcenjenih igralcev. Takim igralcem so kljub njihovi kakovosti ponudili nižje plače in se tako v sezoni 2003 uvrstili v končnico z zgolj tretjino stroškov povprečja preostalih ekip [17].

Leta 2015 se podatkovno rudarjenje preseli tudi v Belo hišo, leta 2016 pa postane ena izmed najbolj raziskovanih tehnik globoko učenje (angl. deep learning), ki je zmožna ugotavljanja kompleksnejših vzorcev [17].

Podatkovno rudarjenje se danes širi zelo hitro in je prisotno v podjetništvu, znanosti, inženirstvu, medicini, delnicah, kartičnih transakcijah in še mnogo drugih področjih [17].

2.2 VRSTE PODATKOV

Proces podatkovnega rudarjenja se lahko aplicira na eno izmed vrst podatkov. Vsaka ima svoje lastnosti in zakonitosti, ki imajo velik vpliv na izbiro algoritmov. Poznamo 4 glavne skupine oblik podatkov. To so:

- Podatkovne baze,
- Podatkovna skladišča,
- Transakcijski podatki,
- Druge vrste podatkov

2.2.1 Podatkovna baza

Podatkovna baza je shramba podatkov, katero lahko upravljamo s sistemom za upravljanje podatkovnih baz. Ta sestoji iz samih podatkov in programske opreme, preko katere se lahko do podatkov dostopa ali jih spreminja. Vsaka podatkovna baza ima svojo strukturo. Najpogostejše so relacijske podatkovne baze, ki so sestavljene iz tabel, v katerih vsaka vrstica predstavlja instanco, vsak stolpec pa atribut. Do podatkovne baze se lahko dostopa z ukazi napisanih v obliki SQL stavkov. Na enak način se lahko pripravi tudi datoteka, ki hrani podatke namenjene za naslednjo obdelavo. Primer SQL stavka prikazuje slika 2.

```
SELECT *
  FROM [AdventureWorks].[Person].[Contact]
```

100 %

Results Messages

	ContactID	NameStyle	Title	FirstName	MiddleName	LastName	Suffix	EmailAddress	EmailPromotion	Phone
1	1	0	Mr.	Gustavo	NULL	Achong	NULL	gustavo0@adventure-works.com	2	398-555-0132
2	2	0	Ms.	Catherine	R.	Abel	NULL	catherine0@adventure-works.com	1	747-555-0171
3	3	0	Ms.	Kim	NULL	Abercrombie	NULL	kim2@adventure-works.com	0	334-555-0137
4	4	0	Sr.	Humberto	NULL	Acevedo	NULL	humberto0@adventure-works.com	2	599-555-0127
5	5	0	Sra.	Pilar	NULL	Ackerman	NULL	pilar1@adventure-works.com	0	1 (11) 500 555-0132
6	6	0	Ms.	Frances	B.	Adams	NULL	frances0@adventure-works.com	1	991-555-0183
7	7	0	Ms.	Margaret	J.	Smith	NULL	margaret0@adventure-works.com	0	959-555-0151
8	8	0	Ms.	Carla	J.	Adams	NULL	carla0@adventure-works.com	0	107-555-0138
9	9	0	Mr.	Jay	NULL	Adams	NULL	jay1@adventure-works.com	1	158-555-0142
10	10	0	Mr.	Ronald	L.	Adina	NULL	ronald0@adventure-works.com	0	453-555-0165
11	11	0	Mr.	Samuel	N.	Agcaoili	NULL	samuel0@adventure-works.com	0	554-555-0110
12	12	0	Mr.	James	T.	Aguilar	Jr.	james2@adventure-works.com	1	1 (11) 500 555-0198
13	13	0	Mr.	Robert	E.	Ahlering	NULL	robert1@adventure-works.com	0	678-555-0175

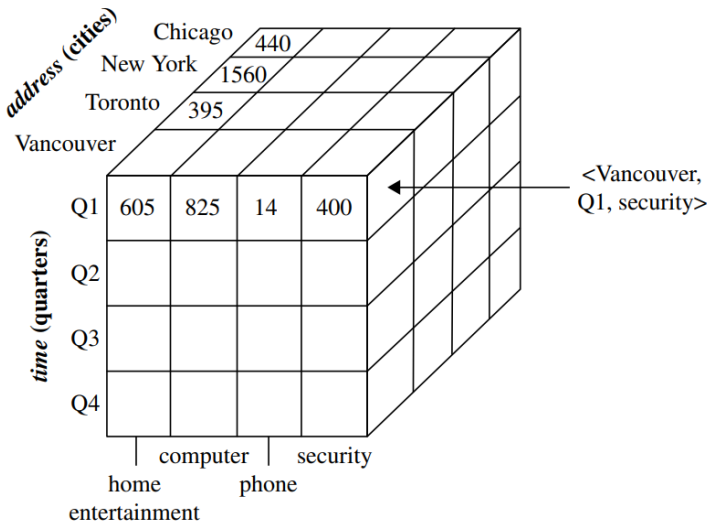
Slika 2: Primer SQL stavka in njegovih rezultatov

Podatkovne baze so najbolj razširjena in uporabljena oblika podatkov, zato se z njimi največkrat srečujemo tudi pri podatkovnem rudarjenju [1].

2.2.2 Podatkovno skladišče

Podatkovno skladišče predstavlja zbirko informacij, ki so pridobljene iz različnih virov, shranjene pa v isti shemi na isti lokaciji. Nastanejo med procesom čiščenja in združevanja podatkov, namenjena pa so predvsem lažji analizi. Podatkovna skladišča uporabljajo velika podjetja, ki so sestavljena iz več prodajaln. Namesto, da bi vsaka prodajalna shranjevala podatke na svojih strežnikih, se podatki pošljejo na enoten strežnik, saj se tako z njimi lažje upravlja.

Podatkovna skladišča so lahko strukturirana v obliki podatkovnih kock (primer na sliki 3) in so večdimenzionalna, kar pomeni, da vsaka dimenzija predstavlja atribut ali zbirko atributov. Z njimi lahko dosežemo hiter izračun nekaterih splošnih lastnosti podatkov kot so vsota ali števec vseh primerov [1].



Slika 3: Primer podatkovnega skladišča v obliki podatkovne kocke (vir: [1])

Kljub temu da orodja, s katerimi si lahko pomagamo pri podatkovnem rudarjenju podatkovnih skladišč že obstajajo, ta pogosto potrebujejo globljo analizo. V ta namen se je razvilo večdimenzionalno podatkovno rudarjenje, ki se ukvarja prav s tehnikami rudarjenja večdimenzionalnih podatkov [1].

2.2.3 Transakcijski podatki

Transakcijski podatki so podatki sestavljeni iz transakcij. To so lahko vse oblike nakupov ali na primer kliki na spletnih straneh. Vsaka transakcija je sestavljena iz identifikatorja in ene ali več postavk. Kljub temu da so transakcijski podatki zgolj posebna vrsta podatkov iz podatkovnih baz, jih mnogi uvrščajo v svojo skupino podatkov. Eden izmed glavnih razlogov za to je njihova popularnost [1].

2.2.4 Druge vrste podatkov

Poleg naštetih pa poznamo še druge, v podatkovnem rudarjenju manj razširjene vrste podatkov. To so:

- Časovni ali zaporedni podatki (rast delnic, zgodovinski podatki),
- Tok podatkov (podatki nadzornih kamer in senzorjev, ki neprestano oddajajo),
- Prostorski podatki (zemljevidi),
- Inženirski podatki (načrti),
- Hipertekst in multimedijski podatki (besedilo, slike, zvok in videoposnetki),
- Podatki v mreži ali grafu (družabna omrežja),
- Splet

Našteti podatki pri obdelavi predstavljajo večji izziv, saj nimajo organizirane strukture in jih je potrebno najprej predobdelati. Čeprav je za rudarjenje takih podatkov potrebno posebno znanje, se v njih lahko skriva veliko zanimivih vzorcev. Z njimi se največkrat ukvarjajo ljudje, ki imajo s podatkovnim rudarjenjem nekaj več izkušenj [1].

2.3 KOMPONENTE ALGORITMOV PODATKOVNEGA RUDARJENJA

Vsak algoritem podatkovnega rudarjenja je sestavljen iz več komponent. To so:

- Struktura modela ali vzorca,
- Točkovna funkcija,
- Optimizacija in metode iskanja,
- Strategija upravljanja s podatki

2.3.1 Struktura vzorca ali modela

Model je definiran kot globalni povzetek podatkovne zbirke. Z njim lahko nekaj trdimo o kateremkoli podatku, ki se nahaja v podatkovni zbirki. Če si podatkovno zbirko predstavljamo kot množico p-dimenzionalnih vektorjev (p točk v prostoru), lahko z modelom nekaj povemo o kateremkoli vektorju in posledično tudi vsaki točki tega prostora. Z modelom lahko po drugi strani vsaki točki določimo skupino ali napovemo njeno vrednost. Prednost določanja modela je ta, da se lahko manjkajočim podatkom v podatkovni zbirki s pomočjo modela priredi vrednost. Primer preprostega modela ima obliko $Y = aX + c$, kjer sta Y in X spremenljivki, a in c pa parametra modela. To je primer linearnega modela, obstajajo pa tudi nelinearni modeli.

Vzorec, za razliko od modela, ki je globalen, opisuje omejen prostor. Preprost primer vzorca predstavlja izjava "če je $X > x_1$ potem je verjetnost, da je $Y > y_1 = p_1$". V tem primeru o podatkih nekaj izvemo samo, če zadostijo pogoju $X > x_1$. Struktura vzorca temelji na pravilih verjetnosti, z njo pa lahko ugotovimo, če v podatkovni zbirki obstajajo izjeme ali dobimo pravila, ki veljajo za zgolj določeno podmnožico. Banke so z analizo vzorcev v preteklosti že ugotovile, da imajo nekateri pokojni ljudje pri njih še vedno odprte bančne račune in jih tako odstranili iz sistema. Za določitev vzorca je v nekaterih primerih potrebna tudi določitev modela. Tak primer predstavlja iskanje osamelcev v podatkovni zbirki, saj osamelec predstavlja vrednost, ki močno odstopa od modela. Če model ni definiran, tudi odstopanja ni mogoče definirati.

Model in vzorec vsebujeta tudi svoje parametre. Vsak algoritem podatkovnega rudarjenja ugotavlja ali na določeni podatkovni zbirki lahko aplicira določen model ali vzorec in

kakšne vrednosti parametrov so zanj najbolj optimalne. Modelu z najbolj optimalnimi parametri pravimo ustrezen model [13].

2.3.2 Točkovna funkcija

Naloga točkovne funkcije (angl. score function) je ovrednotenje ustreznega modela na določeni podatkovni zbirki. V najboljšem primeru se rezultat točkovne funkcije izrazi v uporabnosti in koristnosti uporabljenega modela. Velika težava pa je takšno funkcijo najti, saj se lastnosti, kot so uporabnost in koristnost razlikujejo glede na nalogo in podatkovno zbirko. Kljub temu obstajajo nekatere splošne točkovne funkcije, ki so se izkazale kot dobre. To so klasifikacijska točnost, vsota kvadratov napake in nekatere druge.

Pri točkovnih funkcijah je pomembna tudi njihova hitrost, saj se na velikih podatkovnih zbirkah vrednost točkovne funkcije izračuna velikokrat. Bilo bi namreč nekoristno, da bi na podatkovni zbirki, ki analizira podatke izbruha določenega virusa na rezultat točkovne funkcije čakali več tednov. V tem primeru bi bilo boljše, da kot točkovno funkcijo izberemo drug izračun, ki je hitrejši, čeprav manj učinkovit. Včasih lahko na hitrost izračuna točkovne funkcije vplivajo tudi anomalije v podatkih, na kar je potrebno biti zelo pozoren [13].

2.3.3 Optimizacija in metode iskanja

Modeli in vzorci so v osnovi opisani brez vrednosti spremenljivk. Naloga optimizacije in metod iskanja je za določen model ali vzorec najti take vrednosti spremenljivk, ki bodo dale točkovni funkciji najbolj optimalne vrednosti (minimume ali maksimume odvisno od točkovne funkcije). Optimizacija se osredotoči na pridobivanje najboljših modelov, metode iskanja pa se uporabijo pri iskanju najboljših vzorcev. Pri linearni regresiji se v največ primerih uporablja točkovna funkcija po metodi najmanjših kvadratov. Enako kot pri točkovni funkciji, tudi proces optimizacije ni v celoti objektiven, saj je odvisen predvsem od podatkovnih zbirk in naloge, ki jo na podatkovni zbirki želimo rešiti [13].

2.3.4 Strategija upravljanja s podatki

Zadnja izmed naštetih komponent algoritmov je strategija upravljanja s podatki. Ko govorimo o podatkovnem rudarjenju, se ukvarjamo z velikimi podatkovnimi zbirkami, zato upravljanje z njimi predstavlja prav poseben izziv. Sprašujemo se, kako podatke shraniti, jih indeksirati in do njih dostopati. Večina najbolj uporabljenih algoritmov podatkovnega rudarjenja predpostavlja, da je vsak podatek hitro dostopen in shranjen v spominu računalnika (RAM). Kljub velikim izboljšavah na področju delovanja spomina računalnika, pa večinoma temu ni tako, saj je količina podatkov prevelika, da bi jo v celoti shranili v RAM. Čas, ki ga porabi računalnik, da iz diska prenese podatke v spomin,

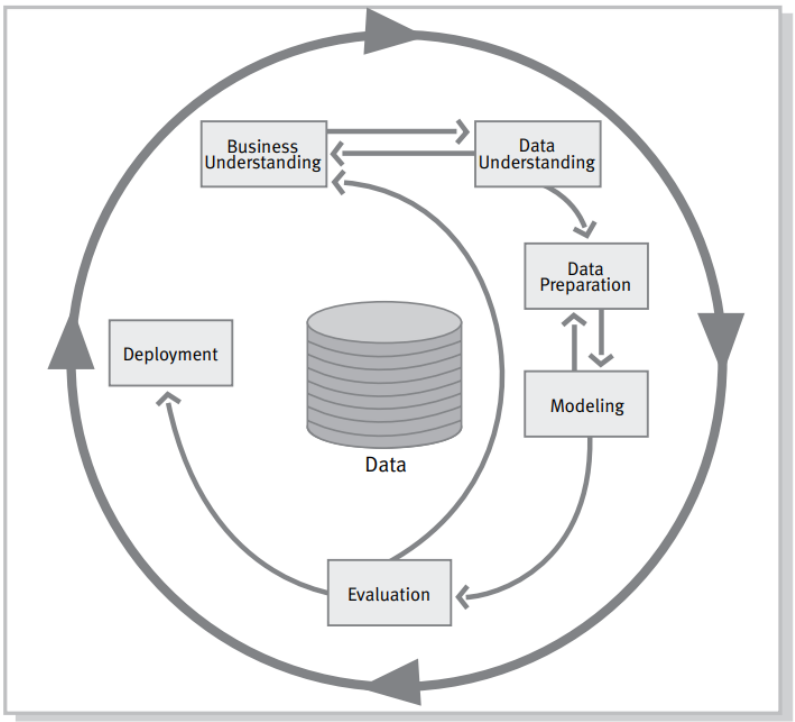
predstavlja omejitev, katere se jo razvijalci algoritmov podatkovnega rudarjenja morajo zavedati [13].

2.4 CRISP-DM MODEL

CRISP-DM model [3][5] predstavlja življenjski cikel projekta podatkovnega rudarjenja. Nastal je leta 1996 in je še danes eden izmed najbolj uporabljenih modelov. Uspeh mu lahko pripišemo zaradi modela, ki ne temelji teoriji, ampak na praktičnih primerih. Projekt je bil financiran tudi s strani Evropske Unije. Razdeljen je na več nalog, ki jih lahko združimo v 6 večjih faz. To so:

- **Razumevanje problema** (angl. Business understanding),
- **Razumevanje podatkov** (angl. Data understanding),
- **Priprava podatkov** (angl. Data preparation),
- **Modeliranje** (angl. Modeling),
- **Vrednotenje** (angl. Evaluation),
- **Uporaba** (angl. Deployment)

Fazam naj bi se sledilo po naštetem vrstnem redu. Kljub temu pa se lahko po nekaterih opravljenih fazah vrne na prejšnjo fazo in se jo z novimi ugotovitvami ponovi, kar prikazuje slika 4. Skozi vse faze sem med izdelavo magistrskega dela šel tudi sam.



Slika 4: CRISP-DM model (vir: [5])

2.4.1 Razumevanje problema

Začetna faza temelji na razumevanju ciljev projekta in določanju poslovnih zahtev. Razumeti je potrebno, kaj stranka želi s projektom doseči in katere omejitve je potrebno pri tem upoštevati. Pomembno je, da se vse nejasnosti glede projekta razrešijo na začetku, saj lahko to ob koncu projekta precej več stane. V tej fazi se med drugim ugotovi koliko ljudi lahko na projektu dela, kakšno znanje imajo ter kaj lahko k projektu doprinesejo. Določi se programska oprema, ki bo potrebna za analizo, pregleda se s kakšno vrsto podatkov bomo imeli opravka, na koncu pa se določi tudi natančne cilje in naredi projektni plan [3][5].

2.4.2 Razumevanje podatkov

V fazi razumevanja podatkov, je potrebno najprej pridobiti dostop do podatkovnih zbirk. Te lahko priskrbi stranka, ki je projekt naročila ali pa jih je potrebno najti sami. Dobro je, da se v tej fazi podatke združi in jih hrani na enem mestu. Uporabi se lahko tudi orodja, ki pomagajo pri samem razumevanju podatkov. Faza se nadaljuje z grobim in hitrim opisom podatkovne zbirke, ki ponavadi vključuje število in vrsto atributov ter število instanc. V naslednjem koraku se s pomočjo tehnik vizualizacije podatkov podatke analizira in preveri ali zbirka vsebuje manjkajoče vrednosti. V zadnjem koraku faze se ovrednoti ali je podatkovna zbirka smiselna in dovolj dobra za nadaljnjo analizo [3][5].

2.4.3 Priprava podatkov

Pri procesu priprave podatkov se je najprej potrebno odločiti, katere atribute in instance je smiselno uporabiti za nadaljnjo analizo. Nekateri primeri lahko vsebujejo veliko manjkajočih vrednosti ali zaradi napak v podatkih izstopajo, zato je take primere najbolje odstraniti. Če je potrebno, se lahko v tej fazi iz že obstoječih atributov izpelje nove. Tak primer je recimo površina območja, ki se jo lahko pridobi iz širine in dolžine stranic. Podatkovne zbirke se lahko tudi združi, številske vrednosti sešteje ali odšteje, podobne vrednosti pa razvrsti v skupine. Faza priprave podatkov obsega tudi krčenje prevelikih podatkovnih zbirk. V zadnjem koraku se podatke uredi tako, da so kompatibilni z orodjem definiranim v eni izmed prejšnjih faz [3][5].

2.4.4 Modeliranje

Modeliranje je sestavljeno iz štirih korakov. V prvem je potrebno izbrati tehniko in algoritme, ki bodo uporabljeni na podatkovnih zbirkah pripravljenih v prejšnji fazi. Algoritme se sicer lahko definira v prvi fazi, vendar včasih tega ni mogoče storiti še posebej, ko ne vemo natančno s kakšnimi podatki se bomo ukvarjali. V drugem koraku je potrebno določiti na kakšen način se bo ovrednotilo in primerjalo izbrane algoritme. V naslednjem koraku se definira vhodne parametre algoritmov ter se jih na podatkih zažene, zadnji korak pa vsebuje grobo oceno uspešnosti modeliranja [3][5].

2.4.5 Vrednotenje

Po končanem modeliranju je potrebno modele natančno ovrednotiti in jih znati razložiti. Ugotoviti je potrebno ali se rezultati ujemajo s hipotezami, ki so bile zastavljene na začetku in določiti uspešnost rezultatov. V fazi vrednotenja se lahko uporabijo globlje analize in statistični izračuni, ob nenavadnih rezultatih pa se je potrebno vprašati ali je bil izbrani model smiseln. Ob ugotovitvi nesmiselnosti modela je fazo modeliranja najbolje ponoviti z drugimi modeli. Projekt se prenese v naslednjo fazo, ko so rezultati dovolj dobri, da smo z njimi zadovoljni [3][5].

2.4.6 Uporaba

Cilj zadnje faze je prenos vsega znanja, ki smo ga pri procesu pridobili, na stranko, ki je projekt naročila. Najprej je potrebno narediti načrt, ki opisuje na kakšen način in s kakšnimi orodji bodo podatki predstavljeni. Sledi izvedba pripravljenega načrta in sestava končnega poročila, v katerem so vsi rezultati in postopki tudi natančno opisani. V zadnjem koraku modela CRISP-DM se projekt v celoti analizira, zapiše kaj je bilo pri projektu storjeno dobro in kje bi lahko projekt izboljšali. Ta korak je pomemben predvsem za nadaljnje delo, saj se tako lahko nekaterim napakam v prihodnosti izognemo [3][5].

2.5 STROJNO UČENJE

Računalnik sam po sebi nima zmožnosti učenja iz izkušenj, zato se ne more učiti iz preteklih napak. Ker bi omenjeno učenje računalnika ljudem koristilo in ker se ustvarjanje takih algoritmov bistveno razlikuje od ostalih algoritmov, se je razvilo področje imenovano strojno učenje. Strojno učenje raziskuje, kako se lahko računalniki s pomočjo podatkov nekaj naučijo. Z drugimi besedami bi lahko rekli, da računalnik ob svojem delovanju spremeni svojo strukturo, program ali podatke na tak način, da se njegovo delovanje izboljša. Pri implementaciji se je potrebno zavedati, da morajo algoritmi strojnega učenja pričakovati interakcijo z uporabnikom izvajanega programa. Interakcija poteka večinoma v obliki dodajanja novih podatkov v bazo znanja, program pa se mora v tem primeru ustrezno odzvati in se v najboljšem primeru nekaj novega naučiti. Pri algoritmu učenja sta pomembni dve lastnosti. Ena je ta, da je algoritem učinkovit pri reševanju definiranega problema. Druga pa, da se ob naučenem modelu z njim da učinkovito sklepati tudi na podatkih, s katerimi bo algoritem imel opravka v prihodnosti. Pomembno je tudi, da je algoritem hiter in ob delovanju ne zasede veliko prostora [21].

Strojno učenje je del sorodnega področja imenovanega umetna inteligenca, hkrati pa predstavlja tudi velik del podatkovnega rudarjenja. Pomaga nam najti rešitve na področjih prepoznavanja govora, robotike in računalniškega vida. Ena izmed bolj poznanih sedanjih

uporab računalniškega vida je prepoznavanje obraza. Če se vprašamo na kakšen način socialna omrežja prepoznajo točno določeno osebo na določeni sliki, si lahko predstavljamo, da je to nemogoče storiti s programom, ki bi za vsako osebo na svetu definiral značilnosti obraza. V tem primeru se najbolj izkažejo algoritmi strojnega učenja. Programi prepoznavanja obraza so narejeni tako, da se na podlagi preteklih slik, ki so označene z imenom in priimkom oseb, naučijo njihove obrazne lastnosti. Ko se na socialnem omrežju pojavi nova slika, se uporabi algoritem, ki primerja lastnosti na sliki z vsemi naučenimi lastnostmi. Algoritem določi, da je na sliki oseba, ki se na podlagi prejšnjih naučenih slik najbolj ujema z osebo, ki je na analizirani sliki.

Strojno učenje predstavlja izvajanje določenega algoritma, z namenom optimizacije parametrov modela na podlagi izkušenj. Model je lahko napovedni, kar pomeni, da se z njim da napovedati prihodnje dogodke ali opisni, s katerim se določene podatke opiše [21].

2.5.1 Prednosti strojnega učenja

Največja prednost strojnega učenja je, da olajša delo uporabnikom računalnika. Poleg tega pa ima še nekaj drugih pozitivnih lastnosti:

- Nekateri problemi se težko rešijo brez uporabe konkretnih primerov. Tak primer je že zgoraj omenjeno prepoznavanje osebe na podlagi slike, katerega se najlažje reši z določanjem vhodnih in izhodnih podatkov. Vhodni podatek je v omenjenem primeru slika ljudi, izhodni pa imena oseb na sliki. Z algoritmom strojnega učenja se lahko omogoči prepoznavanje oseb zgolj na podlagi predhodnih slik.
- V veliki količini podatkov se lahko s strojnim učenjem najdejo povezave med podatki, ki so s hitrim pregledom podatkovne zbirke neopazne. To prednost se uporablja predvsem pri podatkovnem rudarjenju.
- V času programiranja naprave je v nekaterih primerih okolje, v katerem bo naprava delovala neznano. S strojnim učenjem se lahko napravo naredi prilagodljivo in problem nepoznanega okolja minimizira.
- Količina znanja o določenih stvareh je lahko prevelika, da bi se jo lahko optimalno vneslo v program. V takem primeru se algoritem implementira tako, da med izvajanjem postopoma polni bazo znanja, s čimer se programiranju velike količine znanja izogne.
- Kljub poznanem okolju v katerem deluje naprava, se ta lahko čez čas spremeni. S strojnim učenjem naredimo napravo takšno, da se ob spremembi okolja prilagodi, kar znatno zmanjša stroške, ki bi nastali s ponovnim programiranjem naprave [21].

2.5.2 Vrste učenja

Pri strojnemu učenju se srečamo z različnimi vrstami učenja [2]. Poznamo:

- Nadzorovano učenje,
- Nenadzorovano učenje,
- Polnadzorovano učenje

2.5.2.1 Nadzorovano učenje

Nadzorovano učenje lahko enačimo s klasifikacijo. Gre za problem, v katerem je podatkovna zbirka sestavljena iz primerov, ki so označeni. To pomeni, da učenje poteka na množici, v kateri imamo podan razredni atribut, ki ga želimo na drugi množici z enako strukturo napovedati. Matematično lahko nadzorovano učenje predstavimo kot iskanje hipoteze *h*, ki velja za množico *S*, pri čemer so vrednosti funkcije *f* za primere iz množice *S* že znane. Če nam uspe najti hipotezo *h* za množico *S*, ki je zelo podobna funkciji *f* in je primerov v množici *S* veliko, pravimo, da je hipoteza dobra. Najbolj znani algoritmi nadzorovanega učenja so naivni Bayes in odločitvena drevesa [1][21][25].

2.5.2.2 Nenadzorovano učenje

Pri nenadzorovanem učenju za razliko od nadzorovanega učenja podatkovna zbirka ne vsebuje označenih primerov. Nenadzorovano učenje se uporablja za iskanje skritih vzorcev v podatkovni zbirki, med najpogostejše algoritme pa sodijo razvrščanje v skupine, zaznavanje anomalij in asociacijska pravila. Primer vhodnih podatkov nenadzorovanega učenja predstavlja podatkovna zbirka fotografij na roko napisanih števk. Kot eden izmed ciljev nenadzorovanega učenja bi lahko algoritem take fotografije razvrstil v 10 skupin, kjer bi vsaka skupina vsebovala zgolj fotografije enake števke [1][21][25].

2.5.2.3 Polnadzorovano učenje

Za polnadzorovano učenje velja, da se za učenje uporabi tako označene kot neoznačene podatke. V večini primerov je označenih podatkov manj kot neoznačenih, saj jih je teh na spletu največ. Uporablja se predvsem v primerih, ko je označenih podatkov premalo ali pa v primerih, ko tako algoritmi nadzorovanega kot algoritmi nenadzorovanega učenja ne dajo zadovoljivih rezultatov. Za take primere se lahko ustvari nov algoritem ali pa se algoritme nadzorovanega in nenadzorovanega učenja zgolj prilagodi [1][21][25].

3 UPORABLJENA PROGRAMSKA OPREMA

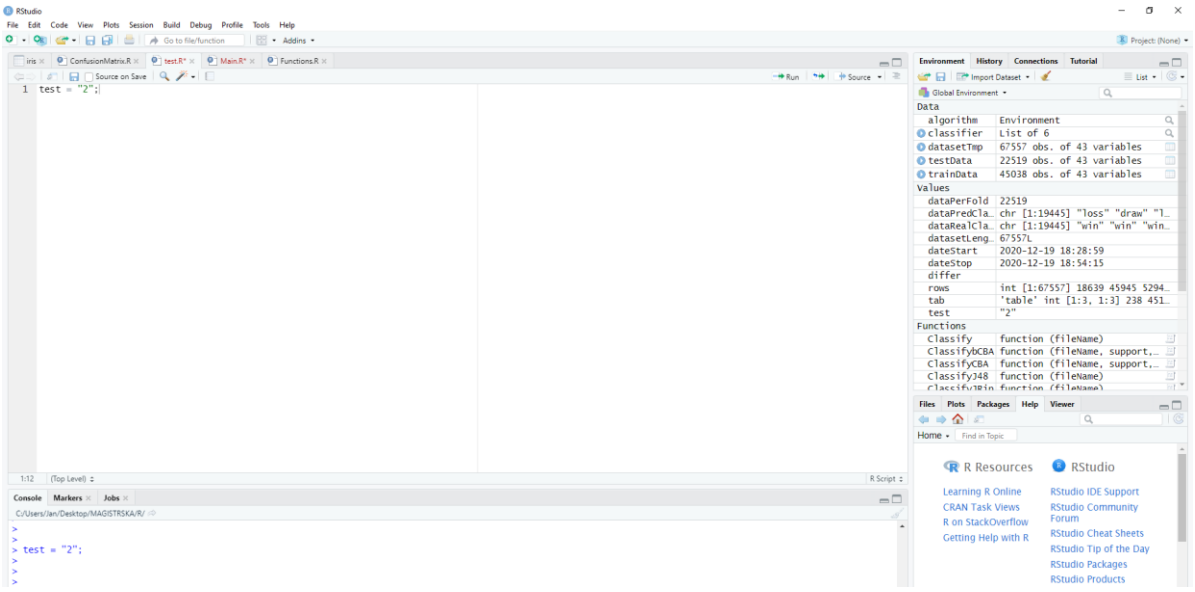
Za potrebe magistrskega dela je bilo uporabljeno programsko okolje RStudio [23], programski jezik R in orodje Weka [31].

3.1 RSTUDIO

RStudio je integrirano razvojno okolje (angl. IDE) namenjeno programskemu jeziku R. Glede na način uporabe je bil razvit v dveh različicah. Prva se imenuje RStudio Desktop, katero si lahko vsakdo namesti na osebni računalnik, druga pa je RStudio Server, ki deluje na strežniku, do katerega se lahko dostopa preko spletnega brskalnika. Obe različici sta odprtokodni, izdani pod licenco GNU Affero General Public License, namenjeni pa sta tako Windows kot tudi Linux in macOS uporabnikom. Sam sem uporabljal namizno različico aplikacije na operacijskem sistemu Windows, ki je prikazana na sliki 5 [23].

Okolje RStudio je sestavljeno več komponent, med katerimi sta najbolj uporabljena urejevalnik besedil in konzola. Omogoča tudi izvajanje in razhroščevanje kode ter nekatere osnovne načine vizualizacije. Pretežno je napisan v programskem jeziku Java, delno pa tudi v jeziku C++ in JavaScript. Razvoj okolja se je pričel z letom 2010, prva beta različica pa je bila izdana leto dni kasneje. Prva stabilna različica je izšla leta 2016. Trenutna različica, ki je bila uporabljena pri magistrskem delu ima oznako 1.2.5001 [23] [24].

RStudio je eno izmed bolj poznanih in uporabljenih programskih okolij predvsem zaradi njegove enostavne uporabe in podpore paketov. Ti vsebujejo dele kode, ki jih lahko vsi razvijalci uporabljajo, dostopni pa so preko že vgrajenega upravljalnika paketov.



Slika 5: Grafični vmesnik RStudia

3.2 R

R je programski jezik in brezplačno programsko okolje namenjeno statističnemu programiranju ter vizualizaciji podatkov. Je eden izmed najbolj pogosto uporabljenih jezikov med statistiki in osebami, ki se ukvarjajo s podatkovnim rudarjenjem. Oktobra 2020 je po TIOBE indeksu, ki meri popularnost programskih jezikov pristal na 9. mestu [29]. Na voljo je pod GNU licenco, prvič pa se je pojavil leta 1993 kot naslednik programskega jezika S. Ime je dobil po začetnicah imen Rossa Ihake in Roberta Gentlemana, ki sta programski jezik tudi razvila. Prva stabilna verzija namenjena množični uporabi je izšla leta 2000 [16].

R omogoča linearno in nelinearno modeliranje, vizualizacijo, statistične teste, časovno analizo podatkov, klasifikacijo, razvrščanje v skupine in še mnoge druge vrste nalog. Večina funkcij je napisana v jeziku R, za računsko zahtevne naloge pa se uporablja koda napisana v C, C++ ali Fortranu. Prednost programskega jezika R je širok izbor paketov, ki so dostopni na portalu CRAN. Zadnja stabilna različica ima oznako 4.0.3 [22].

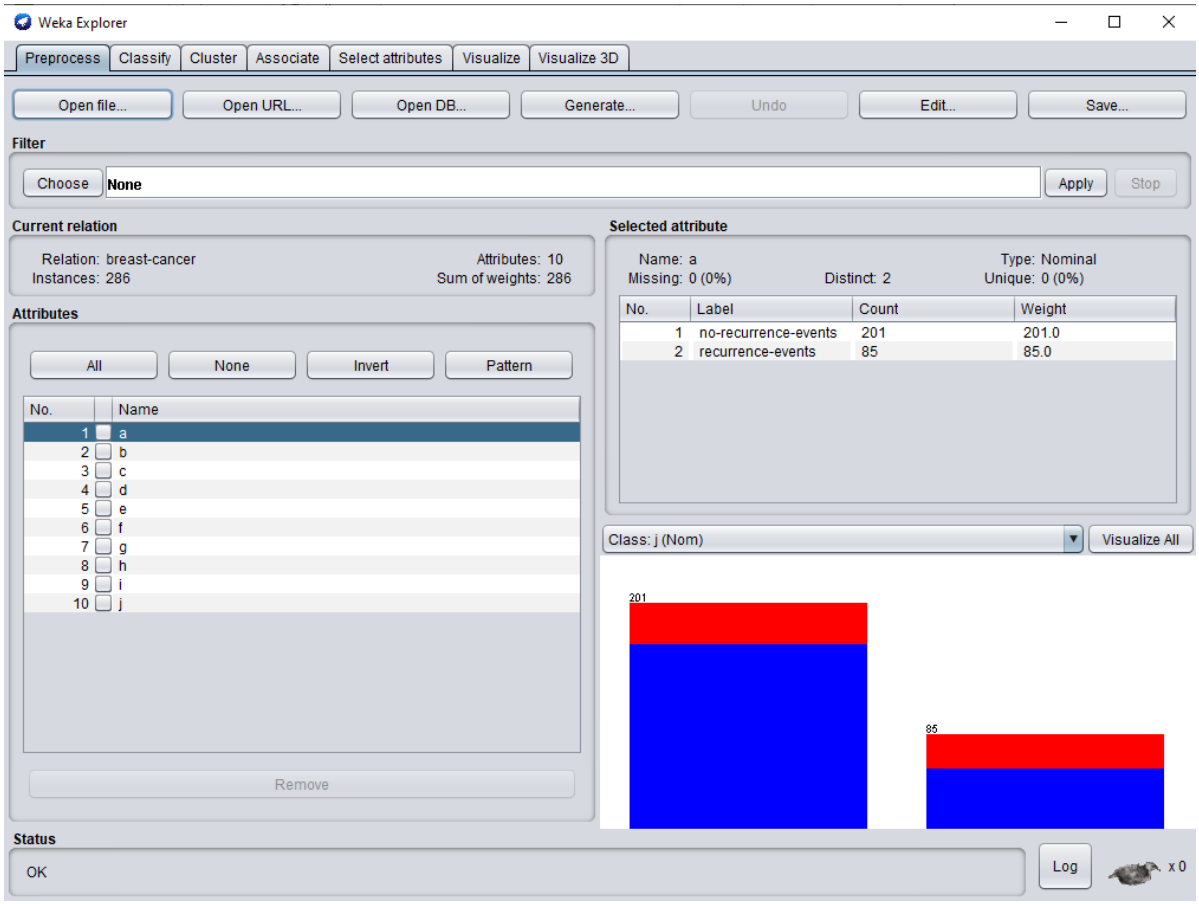
3.3 WEKA

Weka je odprtokodna programska oprema namenjena strojnemu učenju in podatkovnemu rudarjenju. Z njo se lahko v določeni podatkovni zbirki podatke predobdela ali na njih zažene že implementirane algoritme klasifikacije, regresije, razvrščanja v skupine, asociacijskih pravil, izbire atributov in vizualizacije. Omogoča tudi primerjavo uporabljenih algoritmov na različnih podatkovnih zbirkah. Vsi omenjeni algoritmi in programski vmesnik so napisani v Javi [9].

Ideja izdelave programa sega v leto 1992. Njen cilj je bil razvoj aplikacije, ki bi bila zgled vsem aplikacijam na področju strojnega učenja in pomagala pri ključnih vprašanjih novozelandske ekonomije. Projekt je bil leta 1993 s strani novozelandske vlade tudi financiran. Prva različica za množično uporabo je izšla leta 1996. Napisana je bila v programskem jeziku C. Zaradi čedalje težjega vzdrževanja so program leta 1999 prepisali v Javo in tako je ostalo vse do najnovejše različice (3.9.4), ki je bila uporabljena tudi za potrebe izdelave magistrskega dela [12].

Glavni uporabniški vmesnik programa Weka predstavlja raziskovalec (angl. explorer). Ta vsebuje zavihke za vsakega izmed prej omenjenih algoritmov. Podatkovne zbirke, ki jih lahko v Weko naložimo, morajo biti v formatu ARFF, CSV, LibSVM ali C4.5. V program pa lahko naložimo tudi podatkovno bazo ali podatke prenesemo s pomočjo URL. Poleg raziskovalca lahko v Weki uporabimo še preizkuševalec (angl. experimenter), ki je

namenjen primerjavi učinkovitosti algoritmov na različnih podatkovnih zbirkah, tok znanja (angl. knowledge flow), s katerim lahko spremljamo obdelavo toka podatkov, delovno okolje (angl. workbench), ki vsebuje vse naštete komponente združene v enem oknu ter vmesnik z ukazno vrstico. Za magistrsko delo je bil uporabljen zgolj raziskovalec, ki je prikazan na sliki 6 [31].



Slika 6: Grafični vmesnik programa Weka

4 OPIS UPORABLJENIH ALGORITMOV

Kot cilj magistrskega dela smo si zadali, da bomo primerjali kakovost algoritmov, ki jih bomo zagnali nad različnimi podatkovnimi zbirkami. Uporabili bomo algoritme klasifikacije, ki med izvajanjem uporabljajo pravila. V nadaljevanju so podrobneje opisani klasifikacija, asociacijska pravila ter vsak izmed uporabljenih algoritmov.

4.1 KLASIFIKACIJA

Klasifikacija je oblika analize podatkov, katere cilj je pridobitev modelov za opis podatkov v obliki razredov. Takim modelom pravimo klasifikatorji. Kot primer se lahko iz bančnih podatkov stranke ustvari klasifikator, ki pove, če je stranka kreditno sposobna. Algoritmi klasifikacije se največ uporabljajo na področju strojnega učenja, prepoznavanja vzorcev in statistike. Med najbolj znana področja konkretne uporabe spadajo ugotavljanje prevar, ciljno trgovanje ter napoved učinkovitosti in diagnoze bolnikov v zdravstvu. Pri svojem delovanju morajo zato algoritmi pričakovati velike količine vhodnih podatkov.

Klasifikacija je sestavljena iz koraka učenja, kjer se klasifikacijski model ustvari in iz koraka klasifikacije, kjer je ta model uporabljen za napoved razreda na določenih podatkih.

Pri koraku učenja algoritem na podlagi analize učne množice, ki je sestavljena iz podatkov z že dodeljenim razredom, ustvari klasifikator. Ta podatke predstavi v obliki n-dimenzionalnega vektorja atributov $X = (x_1, x_2, \ldots, x_n)$, kjer je n število meritev oziroma število lastnosti. Vsak vektor X ima definiran svoj razredni atribut, ki mu pravimo tudi razred. Ta je ponavadi v obliki diskretne, kategorične vrednosti. Več takih vektorjev skupaj sestavlja učno množico. Proces koraka učenja lahko matematično predstavimo tudi z iskanjem preslikave $y = f(X)$, s katero napovemo razred vektorja X [1].

Pri koraku klasifikacije se klasifikacijski model ovrednoti. To se najpogosteje naredi z izračunom natančnosti klasifikatorja, ki predstavlja odstotek pravilno napovedanih razredov na podatkovni zbirki. Za ovrednotenje je pomembno, da se ne uporabijo podatki iz učne množice, saj bi tak način lahko vplival na objektivnost rezultatov. Korak klasifikacije se tako opravi na množici podatkov, ki ji pravimo testna množica. V primeru, da testne množice ni mogoče pridobiti, se lahko uporabijo druge tehnike ovrednotenja algoritma. Eno izmed takih je k-kratno prečno preverjanje.

Ker je razred vsakega vektorja pri klasifikaciji znan, spada klasifikacija med metode nadzorovanega učenja [1].

4.2 ASOCIACIJSKA PRAVILA

Asociacijska pravila so pravila, ki jih pridobimo iz asociacijske analize in predstavljajo skrite povezave v velikih množicah podatkov. Pri tem so podatki predstavljeni v obliki transakcij in množic predmetov. Množica predmetov je zbirka nič ali več predmetov, ki so del podatkovne zbirke, najlažje pa se jo predstavi na primeru nakupa v trgovini. Če stranka kupi pivo, moko in mleko sta primera množic predmetov tako množica *S = {pivo, mleko},* kot tudi *R = {moka, mleko}.* Definicija asociacijskih pravil se lahko zapiše v obliki $X \rightarrow Y$, kjer sta X in Y disjunktni množici predmetov (njun presek je prazna množica).

Pri ustvarjanju asociacijskih pravil se soočamo z dvema večjima izzivoma. Prvi je način, kako iz velikih podatkov najti povezave, pri drugem izzivu pa je glavno vprašanje smiselnost najdene povezave. Pri določanju ali je pravilo dobro si lahko pomagamo s podporo (angl. support) in zaupanjem (angl. confidence), ki ju uporabnik določi sam. Podpora je meritev, ki pove, kako pogosto se določeno pravilo pojavi v podatkovni zbirki, zaupanje pa pove, kako pogosto se predmeti iz zgoraj definirane množice Y pojavijo v primerih iz množice X [28] [14].

Formalno se podpora in zaupanje zapišeta po enačbah, ki so zapisane na sliki 7, kjer je T množica instanc v podatkovni zbirki.

$$\sigma(X) = \left|\{t_i | X \subseteq t_i,\ t_i \in T\}\right|,$$

$$\text{Support, } s(X \longrightarrow Y) = \frac{\sigma(X \cup Y)}{N}; \qquad (6.1)$$

$$\text{Confidence, } c(X \longrightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}. \qquad (6.2)$$

Slika 7: Enačbe za izračun podpore in zaupanja (vir: [28])

S podporo in zaupanjem se lahko hitro določi smiselnost pravila. Če je podpora nekega pravila zelo nizka, pomeni, da tako pravilo ne velja za veliko primerov v podatkovni zbirki. Takemu pravilu pravimo nezanimivo pravilo in ga je najbolje odstraniti. Zaupanje pa meri zanesljivost nekega pravila. Če pravilo ni zanesljivo, je neuporabno, zato se tudi tako pravilo odstrani. Podporo in zaupanje lahko pri večini algoritmov asociacijskih pravil definira uporabnik kot vhodni parameter. To pomeni, da bo končna množica pravil sestavljena zgolj iz pravil, ki imajo večjo ali enako podporo in zaupanje kot ju je določil uporabnik.

Kljub dobri podpori in dobrem zaupanju je treba vsako pravilo postaviti v kontekst. Včasih se namreč zgodi, da so v podatkovni zbirki slabe ali pomanjkljive meritve, ki na algoritme asociacijskih pravil vplivajo negativno. V tem primeru se lahko v končni množici pravil znajdejo pravila, ki so kljub dobri podpori in zaupanju nesmiselna [28] [14] .

Težavo pri algoritmih asociacijskih pravil predstavlja tudi generiranje pravil. Največje število pravil *R*, ki ga dobimo v podatkovni zbirki z *d* elementi lahko izračunamo po enačbi

$$R = 3^d - 2^{d+1} + 1$$

R število je ponavadi zelo veliko, kar pomeni, da bi z zgolj naključnim generiranjem bili algoritmi asociacijskih pravil neučinkoviti. Zato je večina algoritmov razdeljenih na dva dela:

- Generiranje množic pogostih predmetov in
- Generiranje pravil

4.2.1 Generiranje množic pogostih predmetov

Podatkovna zbirka s *k* elementi ima lahko največ $2^k - 1$ različnih množic predmetov. Ker je *k* v praksi zelo velik, je prostor iskanja takih množic eksponentno velik. V tem primeru je čas delovanja za najbolj enostaven algoritem generiranja množic pogostih predmetov, ki primerja vse množice predmetov med seboj, $O(2^n)$. Tak čas delovanja je tudi za najboljše računalnike prevelik, zato je potrebno algoritem poenostaviti. Pri tem obstajata dve možni splošni rešitvi. V prvi se kandidate, ki lahko pridejo v množico pogostih predmetov zmanjša. Tak primer uporablja Apriori način algoritma. Druga rešitev pa se osredotoča na zmanjšanje števila primerjav, katero se lahko doseže z uporabo naprednih podatkovnih struktur ali stiskanjem podatkovne zbirke [28] [14].

4.2.2 Generiranje pravil

Iz množice pogostih predmetov je potrebno v naslednji fazi generirati pravila. Iz take množice, ki vsebuje *k* predmetov, se lahko izpelje največ $2^k - 2$ asociacijskih pravil. Z uporabo elementov iz množic pogostih predmetov je pri generiranju pravil dobra podpora zagotovljena. V nalogi generiranja pravil je zato pomembno, da se najdejo pravila, ki imajo dovolj visoko tudi zaupanje. Najlažje se to naredi tako, da se množico predmetov *Y* razdeli na dve ne prazni podmnožici *X* in $Y - X$, kjer velja da $X \rightarrow Y - X$ zadovolji dodeljeno zaupanje. Na primer za množico $X = \{1,2,3\}$, ki je že označena kot pogosta množica predmetov lahko izpeljemo pravila $\{1,2\} \rightarrow \{3\}$, $\{1,3\} \rightarrow \{2\}$, $\{2,3\} \rightarrow \{1\}$, $\{1\} \rightarrow \{2,3\}$, $\{2\} \rightarrow \{1,3\}$ in $\{3\} \rightarrow \{1,2\}$. Na kakšen način je to narejeno v praksi, pa je odvisno od posameznega algoritma [28] [14].

4.3 KLASIFIKACIJA Z UPORABO PRAVIL

V magistrskem delu so bili uporabljeni sledeči algoritmi klasifikacije z uporabo pravil:

- CBA,
- bCBA (boosted CBA),
- wCBA, (weighted CBA),
- CMAR,
- CPAR,
- RCAR,
- J48,
- JRip,
- PART

Algoritmi CBA, bCBA, wCBA, RCAR, CMAR in CPAR sodijo med algoritme, ki uporabljajo asociacijska pravila, medtem ko algoritmi J48, PART in JRip sodijo med standardne algoritme, ki uporabljajo pravila. Vsem algoritmom je skupno to, da so sestavljeni iz faze, kjer se pravila generirajo in iz faze, kjer se generirana pravila ovrednotijo.

4.3.1 CBA

CBA je algoritem klasifikacijsko asociacijskih pravil, ki je nastal je leta 1998. Sestavljen je iz CBA-RG faze, ki generira množico pravil in CBA-CB faze, ki ustvari klasifikator.

CBA-RG je generator asociacijskih pravil, ki deluje na podlagi Apriori algoritma. Njegova naloga je najti vsa pravila, ki imajo podporo višjo od najnižje določene podpore (določena s strani uporabnika). Takim pravilom pravimo pogosta pravila. Za vsa pravila, ki imajo enak pogojni stavek, algoritem določi možno pravilo. To je tisto pravilo, ki ima največje zaupanje. Za pravila, ki imajo zaupanje večje, kot je najnižje določeno zaupanje (določeno s strani uporabnika) pravimo, da so natančna pravila. Cilj CBA-RG algoritma je tako najti vsa možna pravila, ki so tako pogosta kot natančna.

CBA-RG naredi pri ustvarjanju pravil več sprehodov po podatkih. V prvem sprehodu vsem pravilom izračuna podporo in določi ali je pravilo pogosto. Iz pogostih pravil v naslednjem koraku ustvari možne kandidate za končna pravila, ki jim nato še enkrat preveri pogostost. Če so ta pravila pogosta, so dodana v končno množico pravil. Med izvajanjem uporabi tudi rezanje, ki deluje na enak način kot pri C4.5 algoritmu. Psevdokoda celotnega CBA-RG algoritma je predstavljena na sliki 8 [19].

```
1   F_1 = {large 1-ruleitems};
2   CAR_1 = genRules(F_1);
3   prCAR_1 = pruneRules(CAR_1);
4   for (k = 2; F_{k-1} ≠ ∅; k++) do
5       C_k = candidateGen(F_{k-1});
6       for each data case d ∈ D do
7           C_d = ruleSubset(C_k, d);
8           for each candidate c ∈ C_d do
9               c.condsupCount++;
10              if d.class = c.class then c.rulesupCount++
11          end
12      end
13      F_k = {c ∈ C_k | c.rulesupCount ≥ minsup};
14      CAR_k = genRules(F_k);
15      prCAR_k = pruneRules(CAR_k);
16  end
17  CARs = ⋃_k CAR_k;
18  prCARs = ⋃_k prCAR_k;
```

Slika 8: Psevdokoda CBA-RG algoritma (vir: [19])

Po uspešno ustvarjenih pravilih se zažene CBA-CB algoritem, katerega vhodni podatek predstavlja množica pravil iz prejšnje faze, izhodni pa klasifikator. Za izdelavo najboljšega klasifikatorja je potrebno na učni množici ovrednotiti vsa pravila, ki so bila ustvarjena v CBA-RG algoritmu in izbrati tisto podmnožico, ki naredi najmanj napačnih klasifikacij. Ker je za *m* pravil lahko vseh možnih podmnožic 2^m, je to v praksi težko izvedljivo, zato se v tem primeru uporabi hevristični pristop. Glavna ideja tega pristopa je izbrati množico pravil z visoko prednostjo, kar pomeni, da se najdejo pravila, ki imajo v primerjavi z ostalimi pravili večjo zaupanje ali večjo podporo. V primeru, da sta zaupanje in podpora enaka se izbere pravilo, ki je bilo prej dodano v množico pravil. Za tak pristop poznamo dve različici algoritma. Prva se imenuje M_1 in je enostavnejša, vendar počasnejša, druga pa M_2, ki je hitrejša, učinkovitejša, ampak tudi bolj kompleksna. V magistrskem delu je bila uporabljena enostavna M_1 različica, ki je predstavljena na sliki 9 [15].

```
1   R = sort(R);
2   for each rule r ∈ R in sequence do
3       temp = ∅;
4       for each case d ∈ D do
5         if d satisfies the conditions of r then
6           store d.id in temp and mark r if it correctly
              classifies d;
7       if r is marked then
8           insert r at the end of C;
9           delete all the cases with the ids in temp from D;
10          selecting a default class for the current C;
11          compute the total number of errors of C;
12      end
13  end
14  Find the first rule p in C with the lowest total number
        of errors and drop all the rules after p in C;
15  Add the default class associated with p to end of C,
        and return C (our classifier).
```

Slika 9: Psevdokoda M_1 algoritma (vir: [19])

V magistrskem delu sta bili uporabljeni tudi modificirani različici CBA algoritma. To sta bCBA (boosted CBA) in wCBA (weighted CBA). [19]

4.3.2 CMAR

CMAR je algoritem, ki se za določanje razreda namesto na eno samo pravilo zanaša na množico pravil. Sestavljen je iz dveh faz. V prvi se generirajo pravila v obliki $R : P \rightarrow c$, kjer R predstavlja pravilo, P vzorec v podatkovni zbirki, c pa razred. Pri generiranju je pomembno, da sta podpora in zaupanje pravila R znotraj omejitev, ki jih je določil uporabnik ter, da je R kvalitetno pravilo. Namesto Apriori metode, se uporablja FP-rast metoda, ki algoritem pohitri predvsem pri velikemu številu pravil, veliki učni množici ali dolgimi pravili [18].

Pomembna lastnost CMAR algoritma je tudi, da se pravila za izboljšanje natančnosti in učinkovitosti shranjujejo v posebno strukturo imenovano CR-drevo. Največja prednost te strukture je, kompaktnost drevesa, kar pomeni, da v povprečju prihrani med 50% in 60% prostora. Drevo deluje podobno kot indeksi v tabelah, zato je dostop do pravil hiter, s čimer se omogoči enostavno rezanje pravil. Shranjevanje pravil v drevo poteka v prvi fazi [18].

V naslednji fazi algoritma se shranjena pravila uporabijo za klasifikacijo. Za vsako množico pravil se tako preveri ali na testnih podatkih določa pravi razred. V primeru pravilnega določanja razreda se taka množica pravil uporabi kot eno izmed končnih množic pravil, v nasprotnem primeru, pa se tako množico razdeli na skupine in izmeri njihova kombinirana učinkovitost. Za take primere se najpogosteje uporabi hi kvadrat teste, v primeru CMAR algoritma pa je na tem mestu uporabljen obtežen hi kvadrat test. Pravila, ki v testu dajo najboljše rezultate, se uporabijo kot končna pravila klasifikatorja [18].

4.3.3 CPAR

CPAR se za generiranje pravil zgleduje po ideji FOIL algoritma. V primerjavi z ostalimi algoritmi ima tri glavne prednosti. Prva je ta, da je že v fazi generiranja pravil množica pravil znatno manjša kot pri ostalih algoritmih, druga je preprečevanje generiranja redundantnih pravil, tretja pa, da pri napovedovanju razreda namesto zgolj enega najboljšega pravila uporabi nekaj k najboljših pravil. Poleg tega algoritem pri svojem delovanju uporablja dinamično programiranje, s čimer se izogne večkratnemu izračunu enakih pravil. Njegovi slabosti sta, da je tako za generiranje pravil, kot za klasifikacijo časovno zelo potraten ter, da lahko njegova evalvacija, ki temelji na zaupanju privede do preprileganja. Za klasifikacijo uporablja izračun Laplaceove pričakovane natančnosti po formuli $(n_c + 1)/(n_{tot} + k)$, kjer je k število razredov, n_{tot} število vseh primerov, ki

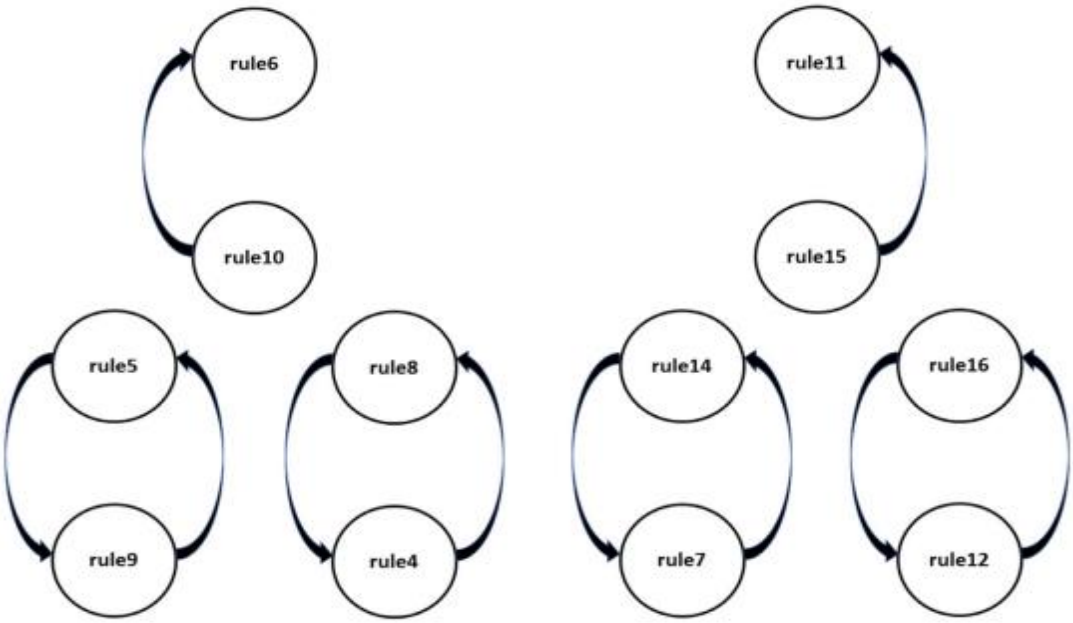
zadovoljijo pogojni stavek pravila, n_c pa število primerov, ki zadovoljijo pogojni stavek pravila in pripadajo napovedanemu razredu [34].

Klasifikacija pravilnega razreda za primer *x* tako deluje na način:

1. Najdi vsa pravila, za katere *x* zadovolji njihov pogojni stavek.
2. Izmed pravil v prejšnji točki najdi nekaj najboljših pravil vsakega razreda.
3. Za vsak razred izračunaj pričakovano natančnost in kot pravilni razred določi razred z najvišjo povprečno pričakovano natančnostjo.

4.3.4 RCAR

RCAR je najnovejši izmed uporabljenih algoritmov, saj je nastal leta 2019. Ima tri glavne korake. Prvi je rudarjenje velikih množic klasifikacijsko asociacijskih pravil glede na definirano zaupanje in podporo. Za drugi korak je značilna uporaba Lasso regularizirane logistične regresije. Ta z nadzorovanjem kompleksnosti modela s pomočjo parametrov preprečuje preprileganje in napove pogojno verjetnost obstoja nekega izida. Postopek poteka s pomočjo metapravil, ki predstavljajo povezave med pravili in imajo bistveno vlogo pri preprečevanju preprileganja. Metapravila so predstavljena na sliki 10.



Slika 10: Grafični prikaz metapravil (vir: [4])

V zadnjem od glavnih korakov se opravi analiza in optimizacija pridobljenih klasifikacijsko asociacijskih pravil. Pri pridobivanju pravil se uporabi metoda podobna Apriori algoritmu. RCAR spada med bolj kompleksne algoritme, ki klasifikacije, ki uporabljajo asociacijska pravila [4].

4.3.5 J48

J48 je Java različica C4.5 algoritma, ki ga je razvil Ross Quinlan in je eden izmed najbolj uporabljenih algoritmov podatkovnega rudarjenja. Deluje tako, da na podlagi informacijskega prispevka ustvari odločitveno drevo, ki ga je mogoče brati kot skupek pravil. Postopek izdelave odločitvenega drevesa se prične z razdelitvijo podatkovne zbirke na več podmnožic. Razdelitev podmnožic se vedno opravi na podlagi atributa, ki množice najbolj enakomerno porazdeli, pri čemer se upoštevajo vrednosti informacijskega prispevka. Tista podmnožica, ki vsebuje najboljši informacijski prispevek ostane, saj bo predstavljala del klasifikatorja, ostale pa v klasifikatorju ne bodo prisotne. Na preostalih podmnožicah se opisani postopek rekurzivno ponovi, dokler ne ostanejo zgolj listi drevesa, ki pripadajo enemu razredu. Izdelano drevo je sestavljeno iz več vozlišč. Vsako vozlišče je sestavljeno iz ene ali več vej, ki predstavljajo odločitev. Pri klasifikaciji je za vsak primer je potrebno narediti sprehod po drevesu, dokler ne pridemo do listov, ki vsebujejo pravilno klasificirano vrednost razreda [26].

4.3.6 JRip

JRip predstavlja implementacijo algoritma RIPPER, ki jo je ustvaril William W. Cohen kot izboljšavo IREP algoritma. Izvedba algoritma se začne z definicijo množice pravil, ki je na začetku prazna. Nato se za vsak razred po vrsti od najredkejšega do najpogostejšega določi pravila. Postopek določanja pravil poteka v dveh stopnjah [7].

Prva stopnja je stopnja gradnje (angl. Building stage), ki je sestavljena iz dveh faz. To sta faza rasti (angl. Grow phase) in faza rezanja (angl. Prune phase). V fazi rasti se pravilo dopolnjuje s požrešno metodo dodajanja pogojev, dokler ni perfektno, kar pomeni, da ima 100% natančnost. Pri postopku se v množico pravil izbere tisto pravilo, ki ima največji informacijski prispevek. Sledi faza rezanja, ki zmanjša število pravil na tak način, da ohrani njihovo učinkovitost. Omenjeni fazi se ponavljata dokler ni dolžina opisa množice pravil ali stopnja napak dovolj velika. Sledi druga stopnja imenovana stopnja optimizacije, pri kateri se kot mera rezanja pravil uporablja formula $(TP+TN)/(P+N)$, kjer so TP resnično pozitivni primeri (angl. true positive), TN resnično negativni primeri (angl. true negative), P pozitivni primeri in N negativni primeri. V množico pravil se doda pravilo z najmanjšo dolžino opisa. V zadnjem koraku se pravila, ki bi povečala dolžino opisa izloči ter nadomesti s pravili, ki so nastala v fazi optimizacije [32].

4.3.7 PART

PART je kombinacija J48 in JRip algoritma, ki za razliko od večine algoritmov pri svojem delovanju ne opravi globalne optimizacije. To algoritem poenostavi, kar je njegova glavna prednost. Deluje po principu deli in vladaj, pri katerem najprej ustvari pravila za nekatere

izmed primerov, nato uporabljene primere odstrani ter postopek ponovi na preostalih primerih. Potek algoritma traja, dokler v podatkovni zbirki ne zmanjka primerov. Pri iskanju pravil si PART algoritem pomaga z algoritmom odločitvenih dreves. To stori tako, da v vsaki iteraciji izbere tisto pravilo, ki je dalo najboljše rezultate (največji informacijski prispevek) v odločitvenem drevesu. Odločitveno drevo se v vsaki iteraciji generira od začetka, kar algoritem izboljša. Ker deluje po principu deli in vladaj, vsakokratna izdelava odločitvenega drevesa na hitrost bistveno ne vpliva [10].

5 UPORABLJENE PODATKOVNE ZBIRKE

Podatkovne zbirke uporabljene v magistrskem delu so bile pridobljene iz podatkovnega repozitorija UCI ML [30].

5.1 PODATKOVNI REPOZIROTIJ UCI ML

UCI ML je zbirka podatkovnih baz in podatkovnih generatorjev, ki jih z namenom analize algoritmov podatkovnega rudarjenja, uporablja skupnost podatkovnih rudarjev. Nastala je leta 1987 s strani Davida Ahe in nekaj študentov kalifornijske univerze. Podatkovne zbirke UCI ML je do danes uporabljajo že veliko študentov, učiteljev in raziskovalcev iz vsega sveta. Citirane so bile že več kot tisočkrat, kar jo uvršča med 100 najbolj citiranih del v računalništvu. Trenutna spletna različica je bila ustvarjena leta 2007 s pomočjo univerze v Massachusettsu in podporo ameriške fundacije za državno znanost. Trenutno obsega 559 podatkovnih zbirk, do katerih lahko vsak dostopa preko iskalnika, svoje podatkovne zbirke pa je mogoče tudi donirati [30].

5.2 OPIS UPORABLJENIH PODATKOVNIH ZBIRK

Vse uporabljene podatkovne zbirke so bile pridobljene iz podatkovnega repozitorija UCI ML. Mapa, v kateri so bili podatki shranjeni je sestavljena iz datoteke s podatki (končnica .data) in datoteke z opisom pomena atributov (končnica .names). V magistrskem delu je bilo uporabljenih 14 podatkovnih zbirk, ki so prvotno namenjene klasifikaciji, lahko pa se jih uporabi tudi za druge vrste nalog. Zaradi izbire algoritmov, ki ne omogočajo obdelave drugih tipov podatkov, so vse podatkovne zbirke sestavljene iz atributov, ki so nominalni. Uporabljene podatkovne zbirke so v nadaljevanju tudi opisane.

Balance Scale
Podatkovna zbirka Balance Scale vsebuje vzorce psihološkega eksperimenta na področju kognitivnega razvoja. Sestavljena je iz 625 instanc in 5 atributov. Cilj podatkovne zbirke je na podlagi nekaterih psiholoških lastnosti ugotoviti nagnjenost kognitivnega razvoja. Razred ima 3 možne vrednosti: levo, desno in uravnoteženo. Razmerje med njimi je 288:288:49. Podatkovna zbirka ne vsebuje manjkajočih vrednosti.

Breast Cancer
Podatkovna zbirka Breast Cancer vsebuje primere pacientov onkološkega inštituta v Ljubljani. Sestavljena je iz 286 instanc in 10 atributov. Cilj podatkovne zbirke je na podlagi nekaterih lastnosti kot so starostna skupina in določene meritve pri prebolevanju

raka ugotoviti ali se bo rak ponovil ali ne. Razred ima 2 možni vrednosti: ponovljivi dogodek, neponovljivi dogodek. Razmerje med njima je 218:68. Podatkovna zbirka vsebuje 9 manjkajočih vrednosti, kar predstavlja 0,31% vseh podatkov.

Car Evaluation

Podatkovna zbirka Car Evaluation vsebuje opise avtomobilov. Sestavljena je iz 1728 instanc in 7 atributov. Cilj podatkovne zbirke je ugotoviti ali je avto vreden svoje cene na podlagi udobja, ki ga nudi. Razred ima 4 možne vrednosti: nesprejemljiv, sprejemljiv, dober, zelo dober. Razmerje med njimi je 1210:384:69:65. Podatkovna zbirka ne vsebuje manjkajočih vrednosti.

Chess (King-Rook vs. King-Pawn)

Podatkovna zbirka Chess (King-Rook vs. King-Pawn) vsebuje vse možne šahovske poteze zaključne faze, pri kateri na šahovnici ostaneta samo kralja, kmet in trdnjava. Sestavljena je iz 3196 instanc in 37 atributov. Cilj podatkovne zbirke je na podlagi vseh možnih kombinacij preučiti zaključke igre v določeni poziciji. Razred ima 2 možni vrednosti: zmaga, poraz (ali neodločeno). Razmerje med njima je 1669:1527. Podatkovna zbirka ne vsebuje manjkajočih vrednosti.

Connect4

Podatkovna zbirka Connect4 vsebuje vse možne kombinacije potez popularne igre štiri v vrsto. Sestavljena je iz 67557 instanc in 43 atributov (42 za vsako lokacijo, kamor lahko igralec postavi svoj žeton + razred). Cilj podatkovne zbirke je na podlagi lokacije žetonov določiti zmagovalca. Razred ima 3 možne vrednosti: zmaga, neodločeno, poraz. Razmerje med njimi je 44473:6449:16635. Podatkovna zbirka ne vsebuje manjkajočih vrednosti.

Hayes Roth

Podatkovna zbirka Hayes Roth je namenjena testiranju algoritmov podatkovnega rudarjenja. Ker je avtor podatkovne zbirke menil, da je pomenska vrednost določenega atributa v tem primeru nepomembna, je vrednosti atributov šifriral tako, da je prave vrednosti zamenjal s šifriranimi vrednostmi. Podatkovna zbirka je sestavljena iz 132 instanc in 6 atributov. Razred ima 3 možne vrednosti: ena, dva, tri. Razmerje med njimi je 51:51:30. Podatkovna zbirka ne vsebuje manjkajočih vrednosti.

House Votes

Podatkovna zbirka House Votes vsebuje glasove predstavnikov kongresov v ZDA. Sestavljena je iz 435 instanc in 17 atributov. Cilj podatkovne zbirke je ugotoviti, kateri stranki pripada določena oseba na podlagi glasovanja pri nekaterih političnih vprašanjih. Razred ima 2 možni vrednosti: republikanci, demokrati. Razmerje med njima je 168:267.

Podatkovna zbirka vsebuje 392 manjkajočih vrednosti, kar predstavlja 5,30% vseh podatkov.

MONKs

Podatkovna zbirka je MONKs je razdeljena na 3 podzbirke MONKs1, MONKs2 in MONKs3. Podobna je podatkovni zbirki Hayes Roth, saj vsebuje zgolj številske vrednosti. Podatkovne podzbirke so sestavljene iz skupno 415 instanc (124+169+122), vsaka pa ima 8 enakih atributov. Razred ima 2 možni vrednosti: nič, ena. Razmerje med njima v vsaki podatkovni zbirki je 62:62, 105:64, 62:60. Podatkovna zbirka ne vsebuje manjkajočih vrednosti.

Mushroom (agaricus-lepiota)

Podatkovna zbirka Mushroom vsebuje primere opisov različnih vrst gob. Sestavljena je iz 8124 instanc in 23 atributov. Cilj podatkovne zbirke je na podlagi opisa posameznih delov gobe, kot so velikost, barva in oblika napovedati ali je goba užitna ali ne. Razredni atribut ima 2 možni vrednosti: užitna, strupena. Razmerje med njima je 3916:4208. Podatkovna zbirka vsebuje 2480 manjkajočih vrednosti, kar predstavlja 1,33% vseh podatkov.

Nursery

Podatkovna zbirka Nursery vsebuje lastnosti staršev otrok vpisanih v vrtec. Sestavljena je iz 12960 instanc in 9 atributov. Cilj podatkovne zbirke je iz socialnega statusa staršev določiti prednostno listo, na podlagi katere bodo otroke sprejemali v vrtec. Razred ima 5 možnih vrednosti: nepriporočljivo, priporočljivo, zelo priporočljivo, prednost, posebna prednost. Razmerje med njimi je 4320:2:328:4266:4044. Podatkovna zbirka ne vsebuje manjkajočih vrednosti.

SPECT

Podatkovna zbirka SPECT vsebuje vzorce enofotonske emisijske računalniške tomografije (angl. SPECT). Sestavljena je iz 80 instanc in 23 atributov. Cilj podatkovne zbirke je na podlagi točk slike, ki predstavljajo določeni del SPECT slike, diagnosticirati srčno bolezen. Posebnost podatkovne zbirke je, da je sestavljena iz zgolj atributov binarnega tipa. Razred ima prav tako 2 možni vrednosti: nič, ena. Razmerje med njima je 40:40. Podatkovna zbirka ne vsebuje manjkajočih vrednosti.

TIC-TAC-TOE

Podatkovna zbirka TIC-TAC-TOE vsebuje vse možne kombinacije potez priljubljene igre križec krožec. Sestavljena je iz 958 instanc in 10 atributov (9 za vsako lokacijo, kamor lahko igralec postavi križec ali krožec + razred). Cilj podatkovne zbirke je na podlagi lokacije križcev in krožcev določiti zmagovalca. Razred ima 2 možni vrednosti: zmaga in

poraz (ali neodločeno). Razmerje med njima je 626:332. Podatkovna zbirka ne vsebuje manjkajočih vrednosti.

Povzetek vseh uporabljenih podatkovnih zbirk je prikazan v preglednici 1.

Preglednica 1: Povzetek uporabljenih podatkovnih zbirk

Ime zbirke	**Št. instanc**	**Št. atributov**	**Manjkajoče vrednosti**
Balance Scale	625	5	NE
Breast Cancer	286	10	DA (0,31%)
Car Eval.	1728	7	NE
Chess	3196	37	NE
Connect4	67557	43	NE
Hayes Roth	132	6	NE
House Votes	435	17	DA (5,30%)
MONKs1	124	8	NE
MONKs2	169	8	NE
MONKs3	122	8	NE
Mushroom	8124	23	DA (1,33%)
Nursery	12960	9	NE
SPECT	80	23	NE
TIC-TAC-TOE	958	10	NE

6 PRIMERJALNA ANALIZA

6.1 METODOLOGIJA DELA

Magistrsko delo se zgleduje po CRISP-DM metodologiji, ki je opisana v poglavju 2.4. Sledi opis posameznih faz metodologije na primeru mojega magistrskega dela.

Prva faza modela CRISP-DM predstavlja razumevanje problema. V tej fazi sva z mentorjem našla problem, ki bi se ga dalo preučiti in se pogovorila o programski opremi ter podatkih, ki bodo v magistrskem delu uporabljeni. Določila sva, da bo glavni cilj magistrskega dela preučevanje učinkovitosti algoritmov, ki pri izvajanju uporabljajo asociacijska pravila ter da bom podatke pridobil na spletu, saj za takšne naloge obstaja repozitorij UCI ML, ki vsebuje nekaj lepo urejenih in dobro opisanih podatkovnih zbirk. Za programsko opremo sva določila, da bom uporabil program Weka in programski jezik R, ki ga bom uporabil s pomočjo okolja RStudio. K taki odločitvi je pripomoglo dejstvo, da sem vsa omenjena orodja spoznal že med časom študija.

V fazi razumevanja podatkov sem podatkovne zbirke prenesel na svoj računalnik in ocenil njihovo smiselnost. Izbirati je bilo potrebno zgolj podatkovne zbirke z nominalnimi atributi, saj nekateri algoritmi ostalih vrst atributov ne zmorejo obdelati. Pomembno vlogo pri izbiri sem namenil tudi lastnosti vsake podatkovne zbirke, saj sem želel, da so si med seboj čimbolj različne. Tako sem dobil množico štirinajstih podatkovnih zbirk.

Sledila je priprava podatkov, v kateri sem podatke pretvoril v CSV format. To je namreč eden izmed formatov, ki ga lahko prebereta in uporabita tako R kot Weka.

V fazi modeliranja sem določil, katere meritve bom pri analizi algoritmov spremljal. Izbral sem čas izvajanja, uspešnost klasificiranih primerov in število pravil, ki jih je določeni algoritem uporabil med klasifikacijo. Evalvacija algoritmov je potekala na način, kjer se podatkovna zbirka razdeli na učno in testno množico. V fazi vrednotenja sem vse algoritme in njihovo izvajanje podrobno analiziral in preučil.

Zadnja v CRISP-DM modelu je faza uporabe, v kateri se vso pridobljeno znanje predstavi. Rezultati so tako predstavljeni v poglavju 6.2.

6.2 REZULTATI ANALIZE ALGORITMOV

Analiza algoritmov je bila opravljena s pomočjo programske opreme Weka in programskega jezika R. V Weki sem uporabil algoritme J48, PART in JRip, ki so v programu že implementirani. V R-ju sem uporabil algoritme CBA, bCBA, wCBA, RCAR, CMAR in CPAR. Pri izvedbi sem si pomagal s knjižnicami ArulesCBA, caret, RKeel. Pred začetkom sem s programom, ki sem ga napisal v R, vsako podatkovno zbirko razdelil na učno in testno množico. Algoritmi so klasifikator zgradili na učni množici, testirali pa na testni množici v razmerju 2:1. Z namenom pravične analize podatkov je bila pri vseh algoritmih za vsako podatkovno zbirko uporabljena enaka učna in testna množica. Omenjeni postopek sem ponovil 5-krat, rezultati v preglednicah pa predstavljajo povprečja vseh petih poskusov ter njihove standardne odklone. Uspešnost algoritma sem meril z odstotkom pravilno klasificiranih primerov, časom izvajanja in številom pravil. Dober algoritem je hiter, ima visok odstotek pravilno klasificiranih primerov in pri izvajanju generira majhno množico pravil. Vse rezultate sem zaradi preglednosti zaokrožil na 2 decimalni mesti.

6.2.1 Skupna statistika

Skupni povprečni rezultat in standardni odklon sta prikazana v preglednici 2. V povprečju je bila statistična uspešnost vseh algoritmov na vseh podatkovnih zbirkah 74,99%. Algoritmi so se izvedli v povprečnem času 43,17 sekund, generirali pa so v povprečju 913,57 pravil. Glede na ne ravno nizek standardni odklon lahko sklepamo, da so imele predvsem pri času in številu pravil podatkovne zbirke velik pomen. Rezultati so za vsako podatkovno zbirko in posamezno meritev natančneje predstavljeni v nadaljevanju. Pri tem velja omeniti, da algoritma CMAR in CPAR podatkov o številu pravil nista vsebovala, zato so podatki za omenjena primera manjkajoči.

Preglednica 2: Skupna statistika algoritmov

	Čas (s)	**Uspešnost (%)**	**Št. pravil**	**Velikost zbirke**
Povprečje	43,17	74,99	913,57	6892,57
St. odklon	62,45	7,95	1333,50	17280,97

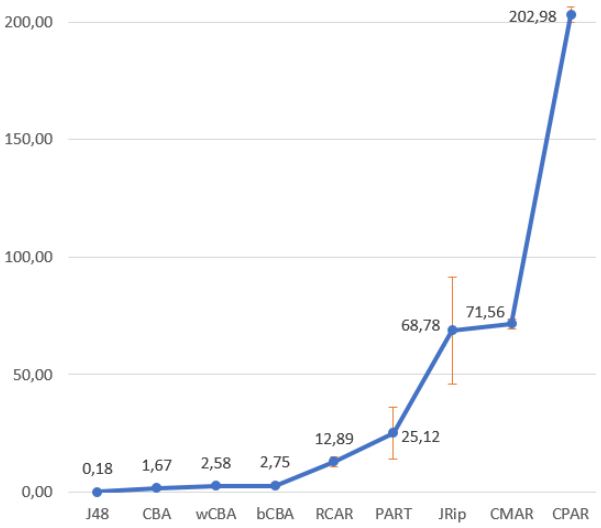
6.2.2 Najboljši algoritmi po posameznih meritvah

Preglednica 3 prikazuje povprečni čas algoritmov. Pri tej se je najbolj izkazal algoritem J48, ki je za izvajanje potreboval v povprečju 0,18 sekund za podatkovno zbirko. Sledita mu CBA in wCBA s povprečnimi časi 1,67 in 2,58 sekund. Kot najslabšega se lahko tukaj izpostavi algoritem CPAR, ki je za svoje delovanje v povprečju porabil več časa kot vsi preostali algoritmi skupaj. Največji standardni odklon opazimo pri JRip algoritmu (22,74),

najmanjši pa pri J48 algoritmu (0,02). Na sliki 11 so vsi rezultati predstavljeni v obliki grafa.

Preglednica 3: Algoritmi razvrščeni po povprečnem času

Algoritem	Povprečen čas (s)
J48	0,18 ± 0,02
CBA	1,67 ± 0,11
wCBA	2,58 ± 0,08
bCBA	2,75 ± 0,10
RCAR	12,89 ± 1,97
PART	25,12 ± 11,19
JRip	68,78 ± 22,74
CMAR	71,56 ± 2,25
CPAR	202,98 ± 3,22

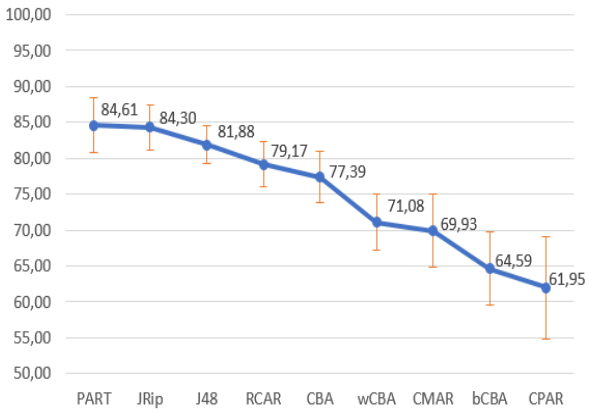


Slika 11: Graf povprečnega časa algoritmov

Preglednica 4 prikazuje povprečno uspešnost algoritmov. Med najbolj uspešnimi algoritmi so se znašli algoritmi programskega okolja Weka. Na samem vrhu je algoritem PART, ki je v povprečju pravilno klasificiral 84,61% primerov. Na drugem in tretjem mestu najdemo algoritma JRip in J48. Najmanj pravilno klasificiranih primerov sta dosegla bCBA in CPAR. Največji standardni odklon opazimo pri CPAR algoritmu (7,13), najmanjši pa pri J48 algoritmu (2,63). Vse podatke lahko na sliki 12 vidimo v grafični podobi, kjer opazimo, da nobeden izmed algoritmov v tej meritvi glede na povprečje ne izstopa.

Preglednica 4: Algoritmi razvrščeni po povprečni uspešnosti

Algoritem	Povprečna uspešnost (%)
PART	84,61 ± 3,76
JRip	84,30 ± 3,19
J48	81,88 ± 2,63
RCAR	79,17 ± 3,10
CBA	77,39 ± 3,61
wCBA	71,08 ± 3,91
CMAR	69,93 ± 5,06
bCBA	64,59 ± 5,11
CPAR	61,95 ± 7,13

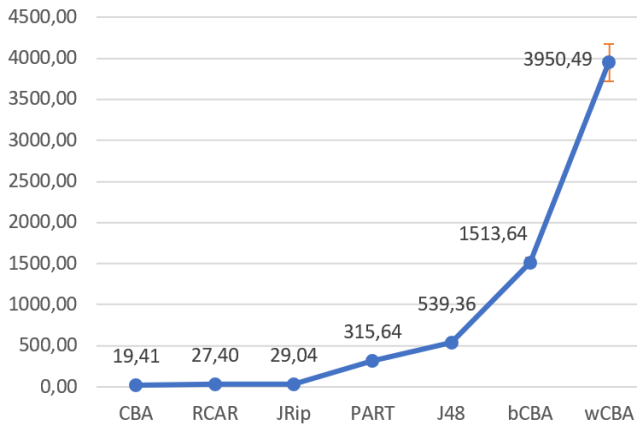


Slika 12: Graf povprečne uspešnosti algoritmov

Preglednica 5 prikazuje povprečno generirano število pravil. V primeru, da želimo kot izhodne podatke dobiti najmanj pravil, je najbolje, da izberemo algoritem CBA, ki v povprečju ustvari 19,41 pravil. Podobno dober rezultat sta dosegla tudi RCAR in JRip, ki sta med izvajanjem v povprečju generirala 27,40 in 29,04 pravil. Zanimiv je podatek, da sta na dnu razvrstitve pristala oba algoritma različice algoritma CBA, čeprav je bil algoritem CBA v tej meritvi najboljši. bCBA je tako v povprečju ustvaril 1513,64 pravil, wCBA pa kar 3950,49 pravil. Največji standardni odklon opazimo pri wCBA algoritmu (228,88), najmanjši pa pri algoritmih PART in J48 (0). Slika 13 vse opisano prikaže v obliki grafa.

Preglednica 5: Algoritmi razvrščeni glede na povprečno generirano število pravil

Algoritem	**Povprečno št. pravil**
CBA	19,41 ± 2,09
RCAR	27,40 ± 3,79
JRip	29,04 ± 2,56
PART	315,64 ± 0,00
J48	539,36 ± 0,00
bCBA	1513,64 ± 59,14
wCBA	3950,49 ± 228,88
CMAR	/
CPAR	/



Slika 13: Graf povprečnega števila pravil algoritmov

6.2.3 Statistika za posamezni algoritem

CBA algoritem podatke s 77,39% uspešnostjo klasificira v povprečnem času 1,67 sekund in pri tem v povprečju ustvari 19,41 pravil. Podrobna statistika algoritma je prikazana v preglednici 6.

Preglednica 6: Statistika CBA algoritma

CBA	**Čas (s)**	**Uspešnost (%)**	**Št. pravil**	**Velikost zbirke**
Balance Scale	0,08 ± 0,00	71,01 ± 3,85	7,00 ± 1,41	625
Breast Cancer	0,25 ± 0,02	75,42 ± 1,69	30,20 ± 4,45	286
Car Eval.	0,09 ± 0,00	80,42 ± 0,87	4,20 ± 0,40	1728
Chess	2,80 ± 0,09	94,41 ± 0,25	26,60 ± 5,00	3196
Connect4	16,47 ± 0,54	67,69 ± 0,28	96,40 ± 1,36	67557
Hayes Roth	0,08 ± 0,01	46,36 ± 9,81	3,00 ± 1,10	132
House Votes	1,42 ± 0,48	95,03 ± 2,20	26,40 ± 1,20	435
MONKs1	0,25 ± 0,20	85,72 ± 9,16	7,60 ± 4,03	124
MONKs2	0,10 ± 0,01	58,95 ± 5,94	7,60 ± 2,33	169
MONKs3	0,12 ± 0,02	92,19 ± 4,47	7,00 ± 2,28	122
Mushroom	1,04 ± 0,06	99,95 ± 0,04	25,60 ± 0,49	8124
Nursery	0,21 ± 0,06	76,27 ± 0,41	3,40 ± 0,49	12960
SPECT	0,23 ± 0,04	66,67 ± 8,44	8,80 ± 2,14	80
TIC-TAC-TOE	0,15 ± 0,03	73,31 ± 3,14	18,00 ± 2,61	958
Povprečje	**1,67 ± 0,11**	**77,39 ± 3,61**	**19,41 ± 2,09**	

bCBA algoritem podatke z 64,59% uspešnostjo klasificira v povprečnem času 2,75 sekund in pri tem v povprečju ustvari 1513,64 pravil. Podrobna statistika algoritma je prikazana v preglednici 7.

Preglednica 7: Statistika bCBA algoritma

bCBA	**Čas (s)**	**Uspešnost (%)**	**Št. pravil**	**Velikost zbirke**
Balance Scale	0,04 ± 0,00	16,65 ± 4,22	2,00 ± 0,00	625
Breast Cancer	0,08 ± 0,01	73,96 ± 5,47	240,80 ± 62,95	286
Car Eval.	0,05 ± 0,00	72,88 ± 4,21	18,20 ± 1,17	1728
Chess	5,88 ± 0,18	92,91 ± 1,19	5217,00 ± 93,43	3196
Connect4	25,72 ± 0,84	61,91 ± 0,67	2407,60 ± 148,21	67557
Hayes Roth	0,04 ± 0,00	39,55 ± 10,43	4,20 ± 1,47	132
House Votes	1,04 ± 0,06	92,69 ± 2,49	3925,00 ± 218,87	435
MONKs1	0,04 ± 0,01	60,47 ± 10,06	8,60 ± 1,50	124
MONKs2	0,04 ± 0,00	60,70 ± 9,32	7,60 ± 3,26	169
MONKs3	0,04 ± 0,00	91,71 ± 5,69	11,60 ± 1,96	122
Mushroom	5,08 ± 0,15	98,42 ± 0,34	8805,00 ± 74,56	8124
Nursery	0,13 ± 0,00	33,31 ± 0,42	12,00 ± 0,00	12960
SPECT	0,20 ± 0,06	51,85 ± 9,37	524,40 ± 216,42	80
TIC-TAC-TOE	0,05 ± 0,02	57,31 ± 7,63	7,00 ± 4,15	958
Povprečje	**2,75 ± 0,10**	**64,59 ± 5,11**	**1513,64 ± 59,14**	

wCBA algoritem podatke z 71,08% uspešnostjo klasificira v povprečnem času 2,58 sekund in pri tem v povprečju ustvari 3950,49 pravil. Podrobna statistika algoritma je prikazana v preglednici 8.

Preglednica 8: Statistika wCBA algoritma

wCBA	**Čas (s)**	**Uspešnost (%)**	**Št. pravil**	**Velikost zbirke**
Balance Scale	0,05 ± 0,00	68,33 ± 4,51	6,60 ± 0,80	625
Breast Cancer	0,07 ± 0,00	76,46 ± 3,64	270,80 ± 9,93	286
Car Eval.	0,04 ± 0,01	69,62 ± 1,44	20,00 ± 0,89	1728
Chess	9,24 ± 0,26	86,30 ± 0,55	23926,60 ± 1054,21	3196
Connect4	18,68 ± 0,25	65,64 ± 0,18	5177,80 ± 48,26	67557
Hayes Roth	0,03 ± 0,01	43,64 ± 10,51	3,40 ± 1,50	132
House Votes	1,59 ± 0,06	89,24 ± 1,54	6578,00 ± 204,51	435
MONKs1	0,03 ± 0,00	67,62 ± 5,35	22,80 ± 5,15	124
MONKs2	0,03 ± 0,00	52,98 ± 6,31	13,60 ± 4,59	169
MONKs3	0,02 ± 0,00	86,83 ± 8,81	24,80 ± 4,71	122
Mushroom	4,46 ± 0,02	88,63 ± 0,37	12338,20 ± 31,22	8124
Nursery	0,12 ± 0,00	71,63 ± 0,56	17,40 ± 0,49	12960
SPECT	1,73 ± 0,51	58,52 ± 6,37	6855,60 ± 1825,08	80
TIC-TAC-TOE	0,04 ± 0,02	69,63 ± 4,64	51,20 ± 12,92	958
Povprečje	**2,58 ± 0,08**	**71,08 ± 3,91**	**3950,49 ± 228,88**	

RCAR algoritem podatke s 79,17% uspešnostjo klasificira v povprečnem času 12,89 sekund in pri tem v povprečju ustvari 27,40 pravil. Podrobna statistika algoritma je prikazana v preglednici 9.

Preglednica 9: Statistika RCAR algoritma

RCAR	**Čas (s)**	**Uspešnost (%)**	**Št. pravil**	**Velikost zbirke**
Balance Scale	0,62 ± 0,05	75,12 ± 1,63	6,60 ± 1,20	625
Breast Cancer	0,07 ± 0,00	77,29 ± 4,44	3,20 ± 0,98	286
Car Eval.	1,39 ± 0,11	78,89 ± 0,69	2,20 ± 0,40	1728
Chess	70,05 ± 3,53	98,93 ± 0,46	117,40 ± 5,78	3196
Connect4	10,77 ± 18,32	66,13 ± 0,25	9,00 ± 1,67	67557
Hayes Roth	0,26 ± 0,02	45,91 ± 9,90	3,20 ± 0,98	132
House Votes	2,47 ± 0,11	96,41 ± 1,10	11,80 ± 1,33	435
MONKs1	0,25 ± 0,02	86,67 ± 11,72	9,80 ± 1,72	124
MONKs2	0,22 ± 0,01	60,35 ± 3,06	7,80 ± 1,94	169
MONKs3	0,26 ± 0,02	93,66 ± 3,31	5,80 ± 4,17	122
Mushroom	79,60 ± 1,50	99,88 ± 0,07	132,20 ± 24,38	8124
Nursery	10,48 ± 2,72	71,24 ± 0,46	18,00 ± 0,00	12960
SPECT	1,48 ± 0,48	66,66 ± 4,68	4,80 ± 4,26	80
TIC-TAC-TOE	2,51 ± 0,69	91,19 ± 1,65	51,80 ± 4,21	958
Povprečje	**12,89 ± 1,97**	**79,17 ± 3,10**	**27,40 ± 3,79**	

CMAR algoritem podatke z 69,93% uspešnostjo klasificira v povprečnem času 71,56 sekund. Podrobna statistika algoritma je prikazana v preglednici 10.

Preglednica 10: Statistika CMAR algoritma

CMAR	**Čas (s)**	**Uspešnost (%)**	**Velikost zbirke**
Balance Scale	1,83 ± 0,01	47,69 ± 0,17	625
Breast Cancer	1,81 ± 0,01	76,96 ± 2,39	286
Car Eval.	2,45 ± 0,03	67,09 ± 2,44	1728
Chess	18,97 ± 1,42	90,62 ± 0,49	3196
Connect4	903,35 ± 18,57	63,82 ± 2,19	67557
Hayes Roth	1,69 ± 0,01	32,09 ± 9,42	132
House Votes	14,08 ± 0,89	89,10 ± 11,90	435
MONKs1	1,79 ± 0,11	59,36 ± 15,55	124
MONKs2	1,76 ± 0,01	60,57 ± 3,39	169
MONKs3	1,73 ± 0,01	66,36 ± 14,78	122
Mushroom	10,36 ± 0,13	99,11 ± 0,15	8124
Nursery	34,75 ± 7,36	92,18 ± 0,48	12960
SPECT	5,12 ± 2,93	69,63 ± 6,37	80
TIC-TAC-TOE	2,21 ± 0,06	64,39 ± 1,18	958
Povprečje	**71,56 ± 2,25**	**69,93 ± 5,06**	

CPAR algoritem podatke z 61,95% uspešnostjo klasificira v povprečnem času 202,98 sekund. Podrobna statistika algoritma je prikazana v preglednici 11.

Preglednica 11: Statistika CPAR algoritma

CPAR	**Čas (s)**	**Uspešnost (%)**	**Velikost zbirke**
Balance Scale	0,70 ± 0,02	51,19 ± 6,26	625
Breast Cancer	0,64 ± 0,04	76,46 ± 4,35	286
Car Eval.	1,46 ± 0,13	71,08 ± 16,49	1728
Chess	7,15 ± 0,17	52,74 ± 0,97	3196
Connect4	2790,10 ± 44,01	54,70 ± 0,65	67557
Hayes Roth	0,49 ± 0,00	75,91 ± 10,02	132
House Votes	0,83 ± 0,06	57,37 ± 6,27	435
MONKs1	0,58 ± 0,11	74,07 ± 14,93	124
MONKs2	0,57 ± 0,01	56,41 ± 8,56	169
MONKs3	0,52 ± 0,01	84,39 ± 15,78	122
Mushroom	13,60 ± 0,13	51,51 ± 1,27	8124
Nursery	23,43 ± 0,23	33,85 ± 1,12	12960
SPECT	0,62 ± 0,12	60,00 ± 7,91	80
TIC-TAC-TOE	1,05 ± 0,03	67,61 ± 5,23	958
Povprečje	**202,98 ± 3,22**	**61,95 ± 7,13**	

J48 algoritem podatke z 81,88% uspešnostjo klasificira v povprečnem času 0,18 sekund in pri tem v povprečju ustvari 539,36 pravil. Podrobna statistika algoritma je prikazana v preglednici 12.

Preglednica 12: Statistika J48 algoritma

J48	**Čas (s)**	**Uspešnost (%)**	**Št. pravil**	**Velikost zbirke**
Balance Scale	0,00 ± 0,00	66,88 ± 2,12	41,00 ± 0,00	625
Breast Cancer	0,00 ± 0,00	75,62 ± 2,14	30,00 ± 0,00	286
Car Eval.	0,00 ± 0,00	90,40 ± 1,28	182,00 ± 0,00	1728
Chess	0,01 ± 0,00	99,21 ± 0,36	59,00 ± 0,00	3196
Connect4	2,50 ± 0,21	80,10 ± 0,12	6445,00 ± 0,00	67557
Hayes Roth	0,00 ± 0,00	73,63 ± 4,68	25,00 ± 0,00	132
House Votes	0,00 ± 0,00	96,27 ± 0,94	16,00 ± 0,00	435
MONKs1	0,00 ± 0,00	74,14 ± 6,65	18,00 ± 0,00	124
MONKs2	0,00 ± 0,00	55,00 ± 10,31	31,00 ± 0,00	169
MONKs3	0,00 ± 0,00	90,24 ± 2,67	12,00 ± 0,00	122
Mushroom	0,01 ± 0,00	100,00 ± 0,00	28,00 ± 0,00	8124
Nursery	0,02 ± 0,00	96,06 ± 0,15	511,00 ± 0,00	12960
SPECT	0,00 ± 0,00	65,92 ± 4,32	11,00 ± 0,00	80
TIC-TAC-TOE	0,00 ± 0,00	82,81 ± 1,10	142,00 ± 0,00	958
Povprečje	**0,18 ± 0,02**	**81,88 ± 2,63**	**539,36 ± 0,00**	

PART algoritem podatke z 84,61% uspešnostjo klasificira v povprečnem času 25,12 sekund in pri tem v povprečju ustvari 315,64 pravil. Podrobna statistika algoritma je prikazana v preglednici 13.

Preglednica 13: Statistika PART algoritma

PART	**Čas (s)**	**Uspešnost (%)**	**Št. pravil**	**Velikost zbirke**
Balance Scale	0,00 ± 0,00	73,58 ± 3,36	28,00 ± 0,00	625
Breast Cancer	0,00 ± 0,00	72,29 ± 3,87	22,00 ± 0,00	286
Car Eval.	0,00 ± 0,00	95,87 ± 0,54	64,00 ± 0,00	1728
Chess	0,02 ± 0,00	98,80 ± 0,16	23,00 ± 0,00	3196
Connect4	351,44 ± 156,59	77,99 ± 0,50	3973,00 ± 0,00	67557
Hayes Roth	0,00 ± 0,00	73,18 ± 7,25	13,00 ± 0,00	132
House Votes	0,00 ± 0,00	95,58 ± 1,72	6,00 ± 0,00	435
MONKs1	0,00 ± 0,00	87,31 ± 16,50	8,00 ± 0,00	124
MONKs2	0,00 ± 0,00	58,21 ± 5,48	21,00 ± 0,00	169
MONKs3	0,00 ± 0,00	93,66 ± 2,93	5,00 ± 0,00	122
Mushroom	0,01 ± 0,00	100,00 ± 0,00	13,00 ± 0,00	8124
Nursery	0,14 ± 0,03	98,81 ± 0,15	187,00 ± 0,00	12960
SPECT	0,00 ± 0,00	68,14 ± 7,63	7,00 ± 0,00	80
TIC-TAC-TOE	0,01 ± 0,00	91,06 ± 2,53	49,00 ± 0,00	958
Povprečje	**25,12 ± 11,19**	**84,61 ± 3,76**	**315,64 ± 0,00**	

JRip algoritem podatke z 84,30% uspešnostjo klasificira v povprečnem času 68,78 sekund in pri tem v povprečju ustvari 29,04 pravil. Podrobna statistika algoritma je prikazana v preglednici 14.

Preglednica 14: Statistika JRip algoritma

JRip	**Čas (s)**	**Uspešnost (%)**	**Št. pravil**	**Velikost zbirke**
Balance Scale	0,03 ± 0,01	72,45 ± 3,51	15,00 ± 3,35	625
Breast Cancer	0,00 ± 0,00	73,33 ± 4,59	1,80 ± 0,40	286
Car Eval.	0,23 ± 0,07	83,56 ± 2,29	45,80 ± 4,53	1728
Chess	0,21 ± 0,06	98,93 ± 0,56	17,00 ± 1,67	3196
Connect4	943,52 ± 315,95	74,38 ± 0,57	151,80 ± 18,54	67557
Hayes Roth	0,00 ± 0,00	74,09 ± 8,21	7,00 ± 0,00	132
House Votes	0,00 ± 0,00	94,62 ± 1,34	3,60 ± 0,80	435
MONKs1	0,03 ± 0,02	96,10 ± 3,96	5,00 ± 0,00	124
MONKs2	0,00 ± 0,00	60,71 ± 4,66	1,60 ± 0,49	169
MONKs3	0,00 ± 0,00	93,66 ± 3,96	5,00 ± 0,00	122
Mushroom	0,12 ± 0,04	99,98 ± 0,04	9,60 ± 0,49	8124
Nursery	18,80 ± 2,11	95,97 ± 0,31	129,60 ± 3,14	12960
SPECT	0,00 ± 0,00	65,18 ± 10,10	3,00 ± 1,26	80
TIC-TAC-TOE	0,03 ± 0,02	97,25 ± 0,60	10,80 ± 1,17	958
Povprečje	**68,78 ± 22,74**	**84,30 ± 3,19**	**29,04 ± 2,56**	

7 DISKUSIJA

Iz analize podatkov ugotovimo, da pri upoštevanju vseh treh meritev skupaj izstopa zgolj CPAR algoritem. Ta je bil najslabši tako pri uspešnosti kot pri času izvajanja algoritma. V povprečju so bili sicer algoritmi 74,99% uspešni, kar je zadovoljivo, saj bi lahko rekli, da bi ne glede na izbiro podatkovne zbirke v vsakem primeru uspešno klasificirali 3 izmed 4 primerov. Tudi čas je zadovoljiv, saj je 43,17 sekund glede na povprečno velikost podatkovne zbirke, ki je bila 6892,57 dober rezultat.

Med najhitrejše algoritme s povprečnimi časi 0,18, 1,67, 2,58 in 2,75 sekunde sodijo J48, CBA, wCBA in bCBA, med najpočasnejše pa sodita CMAR s povprečnim časom 71,56 sekund in CPAR s povprečnim časom 202,98 sekund. Pri tem je potrebno poudariti, da je algoritem CPAR samo za obdelavo podatkovne zbirke Connect4 potreboval 2790,10 sekund, kar močno pokvari njegovo povprečje. Glede na to, da je podatkovna zbirka Connect4 največja ter da je pri ostalih podatkovnih zbirkah potreboval bistveno manj časa (manj kot 30 sekund) lahko sklepamo, da je algoritem CPAR bolje uporabiti na manjših podatkovnih zbirkah, saj ga velika količina podatkov zelo upočasni.

Najuspešnejši algoritmi po odstotku uspešno klasificiranih primerov so bili algoritmi, ki pri delovanju ne uporabljajo asociacijskih pravil, temveč spadajo med standardne klasifikatorje, ki uporabljajo pravila. To so PART, JRip in J48 in so v povprečju edini pravilno klasificirali več kot 80% vseh primerov. Med najslabše pri tej meritvi sodita bCBA in CPAR z nekaj več kot 60% uspešnostjo.

Pri meritvi števila pravil za algoritma CMAR in CPAR ni bilo mogoče dobiti podatkov, zato omenjena algoritma pri analizi nista upoštevana. Izmed preostalih so se najbolje izkazali algoritmi CBA, RCAR in JRip, ki so v povprečju generirali manj kot 30 pravil. Taki algoritmi se lahko uporabijo predvsem pri interpretaciji povezav med podatki in postavljanju podatkov v določen kontekst, saj so njihova pravila lahko predstavljena v drevesni strukturi, ki je razumljiva širši množici ljudi. Največ pravil pri izvajanju v povprečju generira wCBA s povprečno 3950,49 pravili na podatkovno zbirko, kar je več kot jih povprečno generirajo vsi preostali algoritmi skupaj.

Pri času izvajanja in številu uporabljenih pravil vrsta uporabljenega algoritma nima pomembne vloge, saj bistvene razlike med standardnimi algoritmi, ki uporabljajo pravila in med algoritmi, ki uporabljajo asociacijska pravila ni. Pri odstotku uspešno klasificiranih primerov, pa se razlika opazi, saj se ravno na prvih treh mestih nahajajo PART, JRip in J48, ki sodijo med standardne algoritme, ki uporabljajo pravila.

Povzamemo lahko, da se je večina algoritmov izkazala v vsaj eni izmed izbranih meritev. Kateri algoritem bi pri podatkovnem rudarjenju izbrali je odvisno predvsem od cilja, ki ga z nalogo želimo doseči in velikosti podatkovne zbirke. Pri delu z veliko množico podatkov, bi izbral J48, saj je v povprečju najhitrejši, za nalogo, kjer bi moral povezave med podatki tudi predstaviti bi izbral CBA, saj v povprečju generira najmanj pravil, v primerih, kjer je pa pomembna zgolj natančnost algoritma pa bi izbiral med standardnimi algoritmi, ki uporabljajo pravila, saj so uspeli pravilno klasificirati največ primerov.

8 ZAKLJUČEK

V magistrskem delu sem analiziral algoritme, ki za klasifikacijo uporabljajo pravila. Pomagal sem si z metodologijo, ki je najbolj priljubljena na področju podatkovnega rudarjenja CRISP-DM. Najprej sem moral določiti algoritme in podatke na katerih bom algoritme lahko uporabil. Nato je sledila priprava podatkov in sama implementacija algoritmov. Pri tem sem si pomagal s knjižnicami programskega jezika R ter programskim okoljem Weka, ki vsebuje implementacije najbolj znanih algoritmov podatkovnega rudarjenja. Za analizo sem uporabil 14 različnih podatkovnih zbirk, nad katerimi sem izvedel algoritme CBA, bCBA, wCBA, CMAR, CPAR, JRip, PART in J48.

Pri analizi sem za vsak algoritem izmeril povprečni čas, povprečni odstotek uspešno klasificiranih primerov in povprečno število pravil, ki jih algoritem generira. Z rezultati vseh algoritmov sem bil v splošnem zadovoljen, saj so algoritmi v povprečnem času 43,17 sekund uspešno klasificirali 74,99% primerov in pri tem generirali 913,57 pravil. Odstopanja v pozitivno smer v povprečju ni bilo, so se pa za vsako meritev najbolje odrezali drugi algoritmi. Pri času je bil to J48, pri uspešnosti klasifikacije je bil to PART, pri številu pravil pa CBA. Kot slabši algoritem lahko označimo algoritem CPAR, ki se tako po uspešnosti, kot po času nahaja na dnu lestvice. Razlika med standardnimi algoritmi, ki uporabljajo pravila in med algoritmi, ki uporabljajo asociacijska pravila je bila opazna zgolj pri meritvi uspešno klasificiranih primerov, kjer so bili algoritmi, ki uporabljajo asociacijska pravila nekoliko slabši.

Magistrsko delo lahko služi kot dobra osnova za primerjavo nekaterih že obstoječih klasifikatorjev, ki med izvajanjem uporabljajo asociacijska pravila. V bodoče bi se lahko omenjene algoritme izboljšalo tako na področju uspešnosti klasifikacije kot tudi hitrosti in številu pravil. Hkrati pa bi lahko uporabljeni algoritmi predstavljali idejo za izdelavo popolnoma novih, boljših algoritmov. Ena izmed raziskav, ki v magistrskem delu ni opravljena in bi lahko bila zanimiva za nadaljnje delo, je analiza omenjenih algoritmov glede na lastnosti podatkovne zbirke.

9 LITERATURA IN VIRI

[1] S. Agarwal, *Data mining: Data mining concepts and techniques*, 2014.

[2] T. O. Ayodele, Types of Machine Learning Algorithms, *New Advances in Machine Learning*, 2010

[3] A. Azevedo in M. F. Santos, Kdd, Semma and Crisp-Dm:a Parallel Overview, *Proceedings of Informatics 2008 and Data Mining 2008*, 2008, 182-185

[4] M. Azmi, G. C. Runger, in A. Berrado, Interpretable regularized class association rules algorithm for classification in a categorical data space, *Information Sciences (Ny)*, 2019, 313–331

[5] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, R. Wirth, Crisp-Dm 1.0, *CRISP-DM Consortium*, 2000

[6] M. Chen, J. Han in P. S. Yu, Data Mining: An Overview from a Database Perspective, *IEEE Transaction On Knowledge Data Engineering*, 1996, 866–883

[7] W. W. Cohen, Fast Effective Rule Induction, *Machine Learning? Proceedings of the Twelfth International Conference*, 1995, 115-123

[8] M. J. Embrechts, B. Szymanski, K. Sternickel, Chapter 10 : Introduction to Scientific Data Mining : Direct Kernel Methods & Applications, *The Fusion of Soft Computing and Hard Computing*, 2005, 317–365

[9] E. Frank, M. Hall, L. Trigg, G. Holmes, in I. H. Witten, Data mining in bioinformatics using Weka, *Bioinformatics*, št. 15, 2004, 2479–2481

[10] E. Frank in I. H. Witten, Generating accurate rule sets without global optimization, *Proc. Fifteenth Int. Conf. Mach. Learn.*, 1998, 144–151

[11] A. A. Freitas, Understanding the crucial differences between classification and discovery of association rules - a position paper, *ACM SIGKDD Explorations Newsletter*, 2000, 65–69

[12] M. Hall, The WEKA Data Mining Software: An Update, *SIGKDD Explorations*, št. 4, 2017, 452–465

[13] D. Hand, H. Mannila, in P. Smyth, *Principles of Data Mining*, 2001

[14] M. Hahsler, B. Grün, K. Hornik in C. Buchta, Introduction to Arules - A computational environment for mining association rules and frequent item sets, *Journal of Statistical Software*, 2005, 1–25

[15] M. Hahsler, I. Johnson, T. Kliegr, in J. Kuchar, Associative classification in R: Arc, arulesCBA, and rCBA, *R Journal*, št. 2, 2019, 254–267

[16] R. Ihaka, R: Past and Future History, *Proceedings of the 30th Symposium on the Interface*, 1998, 392–396

[17] History of Data Mining, *KDNuggets* [Online]. Dosegljivo: https://www.kdnuggets.com/2016/06/rayli-history-data-mining.html [Dostopano: 13. 10. 2020]

[18] W. Li, J. Han, in J. Pei, CMAR: Accurate and efficient classification based on multiple class-association rules, *Proceedings - IEEE International Conference on Data Mining, ICDM*, 2001, 369–376

[19] B. Liu, W. Hsu, in Y. Ma, Integrating Classification and Association Rule Mining, *Kdd*, 1998, 80–86

[20] B. Minaei-Bidgoli in W. F. Punch, Using genetic algorithms for data mining optimization in an educational web-based system, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2003, 2252–2263

[21] N. J. Nilsson, *Introduction to machine learning*, št. 2, 2005

[22] R FAQ, *R* [Online]. Dosegljivo: https://cran.r-project.org/doc/FAQ/R-FAQ.html. [Dostopano: 25. 10. 2020] {18}

[23] RStudio osnovna stran, *RStudio* [Online]. Dosegljivo: https://rstudio.com/products/rstudio/. [Dostopano: 25. 10. 2020]

[24] RStudio programska koda, *RStudio* [Online]. Dosegljivo: https://github.com/rstudio/rstudio/. [Dostopano: 25. 10. 2020]

[25] S. J. Russell in P. Norvig, *Artificial Intelligence A Modern Approach.*, št. 2. 1995

[26] N. Saravanan in V. Gayathri, Performance and Classification Evaluation of J48 Algorithm and Kendall's Based J48 Algorithm (KNJ48), *International. Journal of Computer Trends and Technology*, št. 2, 2018, 73–80

[27] Walter Pitts and the Mathematical Model of a Neural Network, *Sci-Hi* [Online]. Dosegljivo: http://scihi.org/walter-pitts-and-the-mathematical-model-of-a-neural-network/. [Dostopano: 20. 12. 2020]

[28] P. N. Tan, M. Steinbach, in V. Kumar, Association Analysis: Basic Concepts and Algorithms, *Introduction to Data mining*, 2005, 327–414

[29] TIOBE indeks, *TIOBE* [Online]. Dosegljivo: https://www.tiobe.com/tiobe-index/ [Dostopano: 25. 10. 2020]

[30] UC Irvine Machine Learning Repository, *UCI ML* [Online]. Dosegljivo: https://archive.ics.uci.edu/ml/index.php. [Dostopano: 2. 11. 2020].

[31] Weka 3: Data Mining Software in Java, *Weka* [Online]. Dosegljivo: https://www.cs.waikato.ac.nz/ml/weka/. [Dostopano: 25. 10. 2020]

[32] Class JRip, *Weka* [Online]. Dosegljivo: https://weka.sourceforge.io/doc.dev/weka/classifiers/rules/JRip.html. [Dostopano: 30. 10. 2020]

[33] I. H. Witten, E. Frank, M. A. Hall, *Data Mining Practical Machine Learning Tools and Techniques*, 2011

[34] X Yin in M. J. Han, Classification based on predictive association rules, *Proceeding of the SDM* , 2003 , 369–376