SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5459

Implementacija i vrednovanje algoritma RIPPER za izgradnju pravila prekrivanja

Romano Barilar

Zagreb, 06. 2018.

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5459

Implementacija i vrednovanje algoritma RIPPER za izgradnju pravila prekrivanja

Romano Barilar

Zagreb, 06. 2018.

IZVORNIK

Sadržaj

1. Uvod

Cilj ovog završnog rada je implementacija algoritma strojnog učenja *RIPPERk*, vrednovanje točnosti i efikasnosti na više različitih skupova podataka te usporedba napisanog algoritma sa sličnim implementacijama i algoritmima.

Za početak, pokrit ću teorijski dio rada te odgovoriti na pitanja što je strojno učenje, dubinska analiza podataka, koje vrste učenja postoje te gdje se sve strojno učenje koristi. Zatim ću prikazati algoritme *RIPPERk* i *IREP* te objasniti glavna načela rada algoritma *RIPPERk* kao što su podrezivanje, *MDL*, koraci optimizacije, čišćenja te ću se dotaknuti i različitih kriterija za vrednovanje pravila i različitih uvjeta za prestanak stvaranja pravila unutar algoritma *RIPPERk*.

Nakon toga ću prijeći na skupove podataka koje ću koristiti za vrednovanje implementacije. O njima ću napisati koje su im značajke: broj instanci, broj atributa, vrstu vrijednosti atributa te prikazati raspodjelu po razredima.

Zatim ću objasniti kako instalirati i koristiti napisanu implementaciju, te opisati rješenja nekih problema koji su se ukazali tijekom pisanja koda. Na kraju ću objasniti kako koristiti implementacije s kojim ću uspoređivati svoju, te prikazati rezultate usporedbe.

2. Strojno učenje

2.1. Što je strojno učenje?

Današnje doba se može zvati informacijsko doba. Nekad prije samo su kompanije stvarale velike količinu podataka, koji su se spremali u posebnim računalnim centrima gdje bi se analizirali i koristili. Danas, zahvaljujući globalizaciji i sveopćem napretku tehnologije, svaki čovjek u svom svakodnevnom životu stvara podatke. Bilo to običnom kupovinom u trgovini, pozivom na mobilni uređaj ili surfanjem internetom podaci se stvaraju i spremaju. [2]

Strojno učenje, korištenjem raznih statističkih metoda i algoritama, analizira podatke iz stvarnog svijeta i omogućuje računalima da donose zaključke o svijetu oko sebe. Krajnji cilj strojnog učenja, kao podgrane umjetne inteligencije, je osposobiti računala da mogu „učiti“ iz iskustva te mijenjati svoje ponašanje sukladno naučenom. Iako još ne mogu učiti, može se reći da računala mogu bit podučena/istrenirana od strane ljudi da rješavaju probleme vanjskog svijeta. [3]

Za ovaj rad vrlo su nam bitni algoritmi strojnog učenja. Algoritmi analiziraju skupove podataka te stvaraju model na temelju analize. Stvoreni modeli se mogu koristiti za predviđanje raznih parametara kod novih podataka ili stjecanje znanja o samim podacima. Također, budući da nisu eksplicitno napisani, uz male ili nikakve promjene se mogu koristiti na raznim skupovima podataka. [2]

2.2. Vrste učenja

Vrste strojnog učenja se dijele u tri kategorije: nadzirano, nenadzirano i potporno učenje. Nadzirano učenje bavi se predviđanjem razreda novih instanci podataka. Kod nadziranog učenja postoji takozvani “nadglednik“ koji ulaznim podacima pridaje određeni razred te je cilj samog učenja pronaći model koji opisuje dane podatke. Dok kod nenadziranog učenja ne postoji očita podjela u razreda niti nikakav “nadglednik“ koji će klasificirati podatke, nego postoje samo ulazni podaci koji se koriste za

pronalazak različitih pravilnosti i struktura podataka. Iako sve pronađene strukture i pravilnosti neće biti od koristi, cilj je pronaći one najvrednije pomoću kojih se podaci mogu grupirati u različite skupove. Problemi vezani uz nadzirano učenje mogu biti regresijski ili klasifikacijski, gdje su regresijski problemi predviđanja kontinuiranih vrijednosti a klasifikacijski predviđanje pripadnosti konačnom broju razreda. Za nenadzirano učenje najvažniji su problemi razvrstavanja podataka u različite grupe te problem pronalaska anomalija u podacima. Na kraju, potporno učenje se ne bavi skupovima podataka niti njihovom analizom, nego se bavi ponašanjem agenata umjetne inteligencije, tj. ponašanjem izvršitelja pojedinih zadataka u nekoj određenoj okolini. Stoga je cilj pronaći što bolji redoslijed poteza koji bi omogućio što veći uspjeh ili što manji neuspjeh. Algoritmi potpornog učenja moraju moći na temelju već odrađenih dobrih sekvenci biti u stanju odrediti ili barem smanjiti moguću domenu odabira budućih poteza. [2]

2.3. Dubinska analiza podataka

U današnjem vremenu, kada su ljudi okruženi velikim količinama podataka, vrlo je važno moći brzo i efikasno te iste podatke analizirati. Kao što rudari odlaze u dubine Zemlje pronaći vrijedne rude, tako i naša računala odlaze u dubine podataka pronaći vrijedne informacije. Dubinskom analizom podataka naziva se primjena algoritama strojnog učenja (točnije algoritama nadziranog ili nenadziranog učenja) kao tehnička podloga samoj analizi. Cilj dubinske analize je pronaći skrivene (ne)regularnosti, modele i iznimke u podacima radi veće iskoristivosti te lakšeg razumijevanja samih podataka. Iako će neke informacije biti korisne, većina njih neće, stoga se mora biti u stanju razaznati korisne informacije i odbaciti one nebitne. [3]

3. Algoritmi prekrivanja

Rad algoritma *RIPPERk* i sličnih algoritama prekrivanja može se svesti na generiranje pravila koja će najbolje opisivati zadani skup podataka. Jedan od najvećih problema s kojim se susreću takvi algoritmi je problem prenaučenosti. Problem prenaučenosti (eng. *overfitting*) predstavlja specijaliziranost pravila za klasifikaciju podataka koji su korišteni za njihovo generiranje. Dakle, na podacima koji nisu dosad „viđeni“ ostvaruju se lošiji rezultati od očekivanih. Prenaučenost samog pravila smanjuje se podrezivanjem, dok prenaučenost cijelih skupova se smanjuje uz pomoć heuristika za evaluaciju pravila kao što je *MDL*.

Algoritmi prekrivanja koriste tzv. *seek-and-conquer* pristup. Prilikom pretraživanja najboljeg pravila za pojedini dio podataka, algoritam odvaja samo podatke koji ga zanimaju za trenutačno pravilo te ovisno o njima stvara pravilo. [3]

3.1. Pravila

Klasificiranje unutar dubinske analize podataka odvaja se u dvije velike skupine. Prva skupina je klasificiranje stablima odluke a druga je klasificiranje pravilima. Iako se povijesno više koristilo klasificiranje stablima odluke, ovaj rad bavi se klasificiranjem pravilima. [3]

Pravila se sastoje od antecedenta i konzekventa, gdje antecedens predstavlja skup tvrdnji (uvjeta) povezanih logičkim operatorom I, a konzekvens predstavlja tvrdnju ili skup tvrdnji koje se predviđaju. Kao i kod logičke implikacije, kada je antecedens istinit smatra se da je istinit i konzekvens. Postoje dvije vrste pravila – asocijacijska i klasifikacijska. Klasifikacijska pravila određuju samo razred nekog podataka ovisno o njegovim atributima, a asocijacijska pravila mogu određivati bilo koji atribut pa tako i skup više atributa. [3]

3.2. Podrezivanje

Kao što je već naglašeno prilikom stvaranja pravila pojavljuje se problem prenaučenosti. Svrha podrezivanja je pravila poopćiti te ih testirati na podatke koje algoritam još nije "vidio". Stoga je vrlo bitno da se podrezivanje provodi nad podskupom koji nema iste jedinke kao i podskup za učenje, jer u suprotnom algoritam bi preferirao prenaučena pravila. Taj skup će se odsad nazivati skup za podrezivanje. Pravilo se podrezuje tako da se odbacuju pojedini uvjeti čije uvažavanje smanjuje vrijednost heuristike za evaluaciju pravila.

3.3. MDL

MDL (engl. minimum description length) je princip koji pomaže u odabiru najbolje teorije za opis pojedinih podataka. Princip je nastao na temelju Ockhamove britve te kaže da u slučaju više različitih teorija treba odabrati onu najjednostavniju. Ako se uzme da su *D* podaci a *H* jedna od hipoteza (modela) kojom se opisuju ti podaci, cilj *MDL*-a je pronaći što manju opisnu dužinu teorije ovisnu o opisnoj dužini hipoteze *L(H)* i o opisnoj dužini podataka opisanih danom hipotezom *L(D|H)*. Stoga je i princip dobio ime *minimum description length*.

$$L(H) + L(D|H) \quad (1)$$

Također krenuvši od Bayesovog pravila (2) može se doći do principa *MDL*-a.

$$P(H|D) = \frac{P(H)P(D|H)}{P(D)} \quad (2)$$

Na desnoj strani se nalazi *P(H|D)* koji predstavlja vjerojatnost da hipoteza H objašnjava podatke *D*, dok je *P(D)* konstantna jer cijelo vrijeme se radi s istim podacima a hipoteze se mijenjaju. Kad se negativno logaritmira (2) dobije se sljedeće:

$$-\log_2 P(H|D) = -\log_2 P(H) - \log_2 P(D|H) + C \quad (3)$$

Kada se uspoređuju dvije hipoteze slobodno se može odbaciti konstanta *C* te se dolazi do izraza (1). Također, iz desne strane izraza (3) se može zaključiti da u slučaju najveće vjerojatnosti se dobije najmanja vrijednost logaritma, stoga uz najjednostavniju traži se ujedno i najvjerojatnija teorija.

Primjenom ovog principa u indukciji pravila dolazi se do heuristike koja ovisi o/ evaluira kompleksnost skupa pravila koji se generira. Kad se problem klasificiranja proučava iz perspektive teorije informacija može ga se opisati kao problem pošiljatelja i primatelja. Ako pošiljatelj i primatelj imaju iste podatke ali pošiljateljevi podaci su i klasificirani, kako će pošiljatelj poslati primatelju model kojim može klasificirati svoje podatke? Može mu poslati sve svoje podatke, ali to bi bilo neefikasno. Također može stvoriti model koji opisuje podatke te mu poslati kodirana pravila i njihove iznimke. Za odabiranje najboljeg modela se koristi *MDL* princip, tj. traži se najmanja duljina računajući po danom izrazu (4). [3] [9]

$$exception\ bits + w * theory\ bits \quad (4)$$

3.4. RIPPERk

Repeated Incremental Pruning to Produce Error Reduction ili kraće *RIPPERk* je algoritam strojnog učenja koji se koristi za klasifikaciju skupova podataka koji predstavljaju situacije iz stvarnog svijeta. Nastao je na temelju algoritma *IREP* (*Incremental Reduced Error Pruning*) kojeg je William W. Cohen pokušao optimirati te mu povećati točnost, ali umjesto toga razvio je novi, efikasniji algoritam. Prvi put je objavljen u sklopu zbornika znanstvenih radova s dvanaeste međunarodne konferencije strojnog učenja pod nazivom *Fast Effective Rule Induction*. Cohen na temelju *RIPPERk-a* kasnije razvija i algoritam *SLIPPER* koji ima još veću točnost od *RIPPERk-a*. *RIPPERk* spada u algoritme prekrivanja (algoritme temeljene na pravilima). Oni na temelju ulaznih podataka stvaraju razna pravila koji opisuju (prekrivaju) pojedine podgrupe podataka. [1] [5] [10]

Sam algoritam se može podijeliti u četiri faze: faza izgradnje, faza generiranja varijanti, faza završetka i faza čišćenja. Faza generiranja varijanti, završetka i čišćenja također spadaju u fazu optimizacije. Faza izgradnje stvara inicijalni skup pravila koji pokriva podatke pojedinog razreda, dok se ostatak algoritma bavi rješavanjem problema prenaučenosti i pronalaskom što jednostavnije teorije za opis podataka. Rješavanje problema se provodi kroz više koraka optimizacije, čišćenja i podrezivanja. Slovo *k* u imenu algoritma predstavlja broj ponavljanja koraka optimizacije te se najčešće koristi algoritam *RIPPER2*. Ulazni skup podataka se sastoji od atributa i njihovih vrijednosti te razreda kojem pripadaju, a atributi mogu biti kategorički ili numerički. Algoritam prihvaća i nedostatak vrijednosti tj. ne mora svaka instanca imati vrijednosti svih atributa, neke mogu i nedostajati. [1]

Prije početka faze izgradnje, predani ulazni podaci se grupiraju po razredima kojima pojedine instance pripadaju. Faze se odvijaju jedna za drugom te se izmjenjuju razredi za koje se stvaraju pravila. Razredi su sortirani uzlazno po broju instanci koje pokrivaju. To znači da se prvo stvaraju pravila za razred s najmanjim brojem instanci a na kraju za razred s najvećim brojem instanci. Prije prelaska u prvu fazu, za svaki razred podaci se dijele u pozitivne i negativne, gdje su pozitivni podaci koji pripadaju radnom razredu dok su negativni svi koji pripadaju drugim razredima. [3]

```
Za svaki razred od najvećeg do najmanjeg:
    Podijeli podatke u pozitivne i negativne
    IZGRADI:
    Podijeli podatke u skup za učenje i podrezivanje
    Ponavljaj dok se ne ispuni uvjet za prestanak (a):
        NAUČI:
        Stvori pravilo koristeći pohlepni algoritam
        PODREŽI:
        Podreži pravilo na skupu za podrezivanje
    OPTIMIRAJ k puta:
        GENERIRAJ VARIJANTE:
        Za svako izgrađeno pravilo:
            Podijeli podatke u skup za učenje i podrezivanje
            Stvori prvu varijantu
            Stvori drugu varijantu
            Odaberi najbolje od tri pravila
            Izbaci podatke pokrivene zadnjim odabranim pravilom
        ZAVRŠI:
        Stvori pravila za sve podatke koji nisu pokriveni
        OČISTI:
        Izračunaj DL za skup pravila
        Za svako pravilo:
            Izračunaj DL za skup pravila bez danog pravila ako je DL skupa bez
            pravila manji, izbaci pravilo
KRAJ
```

Slika 2: Pseudokod za algoritam RIPPERk

3.4.1. Faza izgradnje

Na početku faze izgradnje predani podaci se nasumično promiješaju pa se dijele na dio za učenje i dio za podrezivanje pravila. Uobičajena je praksa podijeliti u omjeru dva naprema jedan (2:1)(učenje:podrezivanje). Skup za učenje predaje se nekom od pohlepnih algoritama koji stvaraju pravila na temelju točnosti, količine informacija ili neke druge vrijednosti. Pravila nastaju jedno po jedno te ih se nakon stvaranja podrezuje kako bi se smanjio utjecaj prenaučenosti.

Funkcija za podrezivanje prima pravilo u kojem su uvjeti poredani od zadnjeg dodanog uvjeta do prvog dodanog, te uzima u obzir izbacivanje bilo koje zadnje sekvence uvjeta u svrhu povećanja kvalitete pravila nad skupom za podrezivanje. Jedna od komponenti *IREP*-a koju je Cohen promijenio dok je razvijao *RIPPERk* je heuristika za evaluaciju pravila, *IREP* je koristio tzv. *Accuracy* (5) heuristiku dok RIPPERk koristi tzv. *Precision* (6) heuristiku. [1] [3] [4]

$$acc = \frac{p + (N - n)}{P + N} \quad (5)$$

$$prec = \frac{p - n}{p + n} \quad (6)$$

Unutar (5) i (6) *P* i *N* predstavljaju ukupan broj pozitivnih i negativnih instanci u podskupu za podrezivanje, dok *p* i *n* predstavljaju broj pozitivnih i negativnih instanci koje pokriva pravilo. Cohen je utvrdio da (5) može ponekad dovesti do odabira lošijih pravila, npr. (5) bi preferiralo pravilo koje pokriva dvije tisuće (2000) pozitivnih instanci i tisuću (1000) negativnih instanci nad pravilom koje pokriva tisuću (1000) pozitivnih i jednu (1) negativnu instancu (uz uvjet da su *P* i *N* jednaki u oba slučaja). [1] [4]

Postupak generiranja pravila treba se prekinuti u tri slučaja. Prvi slučaj je kad u pozitivnim podacima više nema radnog razreda tj. nema više pozitivnih instanci. Drugi slučaj je kada opisna duljina dosadašnjeg skupa pravila je za neki prag veća od

najmanje opisne duljine skupa pravila koje je dosad pronađeno, uobičajeno prag iznosi šezdeset i četiri bita te se bilo koji drugi broj može predati kao ulazni parametar. Treći slučaj je kada izgrađeno pravilo ima postotak pogreške veći od pedeset posto na skupu podataka za podrezivanje. Ovaj uvjet je naslijeđen od *IREP-a*, te ga Cohen u svojoj implementaciji izbacuje dok ga se u drugim implementacijama može koristiti ili ne koristiti, ovisno o ulaznim parametrima. [1] [3] [4]

3.4.2. Faza optimizacije

Nakon stvaranja inicijalnog skupa pravila za pojedini razred, faza izgradnje prosljeđuje listu pravila fazi optimizacije. Cohen u svom radu naglašava kako je cilj faze optimizacije smanjenje stope pogreške skupa kao cjeline. Sama faza optimizacije se ponavlja *k* puta. [1] [3] [4]

3.4.2.1. Faza generiranja varijanti

Unutar faze generiranja varijanti stvaraju se dva nova pravila za svako pravilo iz inicijalnog modela. Pravila su poredana kako su nastajala, te se prvo pravilo stvara na temelju svih podataka, dok se zadnje stvara na temelju svih podataka koji nisu dosad pokriveni. Prva varijanta pravila se stvara od početka, a druga varijanta se stvara dodavanjem uvjeta na pravilo za koje se trenutačno stvaraju varijante. Na kraju pravila se pojedinačno stavljaju nazad u skup podataka te se odabire ono pravilo koje zajedno sa skupom ima najmanju opisnu duljinu. Nakon odabira izbacuju se svi podaci koje pravilo pokriva.

3.4.2.2. Faza završetka

Za početak faze završetka izbace se sve instance koje su pokrivene dosadašnjim skupom pravila. Ako nakon izbacivanja postoji još pozitivnih instanci koje nisu pokrivene, za te instance se vraća u fazu izgradnje te se stvaraju dodatna pravila. [1] [3] [4]

3.4.2.3. Faza čišćenja

Kao što je već naglašeno, cilj je pronaći što bolja pravila te nije poželjno imati pravila koja ne pridonose kvaliteti skupa. Stoga se u fazi čišćenja izračunava opisna duljinu cijelog skupa pravila. Zatim se izbacuje pravilo po pravilo iz skupa te se računaju nove opisnu duljine. Ako se opisna duljina smanjila, pravilo koje je izbačeno se odbacuje, a u suprotnom se vraća u skup. Nakon završetka iteriranja, skup pravila se sprema te se odbacuju svi podaci koji su pokriveni generiranim skupom. [1] [3] [4]

Nakon što se odradi k iteracija, skup pravila se ispisuje/ pohranjuje te je spreman za evaluaciju.

3.5. IREP

IREP, kao i *RIPPERk*, je algoritam baziran na pravilima. Nastao je 1994. na temelju algoritma *REP* te su mu autori Johannes Fürnkranz i Gerhard Widmer. Njihov cilj je bio unaprijediti algoritam *REP* na temelju problema koje je Cohen iznio u radu 1993. Neki od problema su bili neefikasnost, podjela podataka te utjecaj podrezivanja na cijeli skup podataka.

```
Dok ima pozitivnih podataka:
        Podijeli podatke u set za učenje i set za podrezivanje:
                Dok ima negativnih:
                        Dodavaj uvjete u pravilo
                Podreži pravilo
                Ako je pravilo lošije od praznog pravila
                        Završi generiranje
                Izbaci podatke pokrivene pravilom
KRAJ
```

Slika 3: Pseudokod algoritma IREP

Algoritam *REP* podrezuje pravila nakon generiranja svih pravila. Velika promjena uvedena kod *IREP-a* je podrezivanje svakog pravila nakon što je generirano, te se time povećala efikasnost i riješio problem i utjecaj podrezivanja na cijeli skup. Kako je *REP* podrezivao nakon stvaranja pravila, podaci koji nisu nakon više pripadali podrezanom pravilu stvarali su razliku kod drugih generiranih pravila. Budući da *IREP* odmah stvori i podrezuje pravilo, iz skupa podataka se izbacuju samo oni podaci koji su pokriveni podrezanim pravilom. Također uzastopnim raspoređivanjem podataka u skup za podrezivanje i učenje, sveli su *REP-ov* problem loše podjele na samo pravilo a ne na čitav skup. [10]

4. Skupovi podataka

Za vrednovanje implementacije algoritma *RIPPERk* koristit će se sedam različitih skupova podataka. Skupovi se sastoje od različitog broja instanci. Instance su opisane nekolicinom atributa i razredom kojem pripadaju. Atributi mogu imate kategoričke (diskretne) ili numeričke (kontinuirane) vrijednosti, dok su razredi kategoričke vrijednosti.

4.1. Skup podataka Iris

Skup podataka Iris se sastoji od sto pedeset instanci koje predstavljaju tri različite vrste biljke perunike. Skup ima četiri numerička atributa i tri moguća razreda.

Broj	Postotak	Razred
50	33.33%	Iris-setosa
50	33.33%	Iris-verscicolor
50	33.33%	Iris-verginica

4.2. Skup podataka Abalone

Skup podataka Abalone se sastoji od četiri tisuće sto sedamdeset i sedam (4177) instanci koje prikazuju obilježja životinje zvane Petrovo uho. Postoji jedan kategorički i sedam numeričkih atributa te je cilj odrediti broj godina (krugova) pojedinog Petrovog uha.

Skup podataka posjeduje 28 različita razreda te se ne može prikazati u tablici.

4.3. Skup podataka Ionosphere

Skup podataka Ionosphere se sastoji od tristo pedeset i jedne (351) instance podataka koji prikazuju sedamnaest pulsnih očitavanja pojedinog radara. Svaki puls je opisan dvama atributima stoga postoje trideset i četiri numerička atributa i dva kategorička razreda koji predstavljaju dobar ili loš radar.

Broj	Postotak	Razred
225	64%	good
126	36%	bad

4.4. Skup podataka Wine

Skup podataka Wine sastoji se od sto sedamdeset i osam (178) instanci podataka koji predstavljaju vina koja se razlikuju po kemijskom sastavu i mjestu podrijetla. Sastav je opisan s trinaest numeričkih atributa, a pogađaju se tri različita razreda podrijetla.

Broj	Postotak	Razred
59	33%	1
71	40%	2
48	27%	3

4.5. Skup podataka Glass Identification

Skup podataka Glass Identification sastoji se od dvjesto četrnaest (214) instanci podatka koji opisuju kemijski sastav raznih vrsta stakla. Skup ima devet numeričkih atributa te je podijeljen u sedam različitih tipova stakla.

Broj	Postotak	Razred
70	33%	1
76	35%	2
17	8%	3
13	6%	5
9	4%	6
29	14%	7

4.6. Skup podataka Adult

Skup podataka Adult se sastoji od trideset dvije tisuće petsto šezdeset i jednu (32 561) instancu podataka koji prikazuju privatna i poslovna obilježja raznih ljudi. Skup ima četrnaest atributa od kojih je šest numeričkih a osam kategoričkih, cilj je odrediti zarađuje li pojedini čovjek više ili manje od pedeset tisuća američkih dolara godišnje.

Broj	Postotak	Razred
24720	76%	<=50k
7841	24%	>50k

4.7. Skup podataka Car evaluation

Skup podataka Car evaluation sastoji se od tisuću sedamsto dvadeset i osam (1728) instanci koje opisuju obilježja automobila. Instance se sastoje od šest kategoričkih atributa i podijeljene su u četiri razreda.

Broj	Postotak	Razred
1210	70%	unacc
384	22%	acc
65	4%	vgood
69	4%	good

5. Implementacija

Implementacije ovog algoritma je napisana u Python programskom jeziku, točnije u Python 3.6 verziji. Sastoji se od dvije datoteke: ripper.py i utils.py. U prvoj datoteci se nalazi sam algoritam *RIPPERk* i njegove pojedine faze, dok se u drugoj nalaze pomoćne metode i strukture podataka. Implementacija prihvaća numeričke i nepostojeće vrijednosti.

5.1. Upute za korištenje

Kao što je već navedeno za pokretanje algoritma potrebno je na računalu imati instaliran python 3.6. Algoritam se može pozvati iz naredbenog retka ili nekog razvojnog okruženja kao što je pycharm, netbeans ili slični. U sklopu ove upute objasnit ću kako ga pokrenuti iz naredbenog retka.

Nakon pozicioniranja u direktorij u kojem se nalazi ripper.py algoritam te željeni podaci za validaciju potrebno je upisati:

```
python_interpreter ripper.py -f data [dodatni atributi]
```

gdje `python_interpreter` predstavlja naziv pod kojim je python instaliran, `ripper.py` datoteku koja se pokreće (algoritam), `-f` datoteku u kojoj se nalaze podaci te dodatni parametri koji mogu biti i njihova značenja su:

`-h`, `--help` – prikaz pomoći pri korištenja na engleskom jeziku

`-r` – prihvaća omjer u obliku 5:6, te predstavlja omjer podataka za učenje i podrezivanje, zadana vrijednost je 2:1

`-d` – predstavlja najveću moguću razliku opisne duljine između starog i novog pravila prilikom stvaranja inicijalnog modela podataka, zadana vrijednost je 64

`-e` – predstavlja odabir metode za evaluaciju, može biti 10fold za deseterostruku križnu validaciju ili 66:33 za dvije trećine podataka za učenje i jednu trećinu podataka za evaluaciju, zadano je 10fold

`-n` – odabire način diskretizacije koji može biti *class* za podjelu po razredima, *all* za granice između svake vrijednosti, i zadana vrijednost *information* za podjelu po količini informacija

`-l` – predstavlja argument za stvaranje datoteke u koju se zapisuje tok rada algoritma, ako se želi stvoriti datoteka izabire se True, ako se ne želi izabire se False što je ujedno i zadana vrijednost

Od parametara jedino je `-f` potreban jer svi ostali posjeduju zadanu vrijednost.

```
D:\roman\Zavrsni rad>python36 ripper.py --help
usage: ripper.py [-h] -f F [-r R] [-d D] [-e E] [-n N] [-l L]

Get parameters.

optional arguments:
  -h, --help  show this help message and exit
  -f F        Data file to use.
  -r R        Growing set size: pruning set size ratio.
  -d D        Bit threshold for mdl usage.
  -e E        Evaluation method 10fold or 66:33.
  -n N        Numeric discretization all or class.
  -l L        To create or not create log file.
```

Slika 4: Prikaz poziva algoritma uz help parametar[1]

```
D:\roman\Zavrsni rad>python36 ripper.py -f car.data -l True -d 128
```

Slika 5: Prikaz poziva algoritma

Ulazni podaci moraju biti u .csv (engl. comma-separated values) formatu, gdje se u prvi redak sprema nazive atributa a u drugi njihovu vrstu, *cat* predstavlja kategoričke vrijednosti, a *num* predstavlja numeričke vrijednosti. Također vrlo je bitno razred koji se želi klasificirati označiti s *class*, da bi algoritam znao na kojem mjestu u instanci očekivati razred. Na kraju zapis treba izgledati kao na slici 5 te ne smiju biti bespotrebni razmaci oko zareza jer algoritam u suprotnom neće moći prepoznati oznaku *class*.

[1] Odabirom „custom“ instalacije pythonu se može promijeniti ime (kao python36) dok prilikom „quick“ instalacije se uobičajeno zove python

```
RI,Na,Mg,Al,Si,K,Ca,Ba,Fe,class
num,num,num,num,num,num,num,num,num,cat
1.52101,13.64,4.49,1.10,71.78,0.06,8.75,0.00,0.00,1
1.51761,13.89,3.60,1.36,72.73,0.48,7.83,0.00,0.00,1
1.51618,13.53,3.55,1.54,72.99,0.39,7.78,0.00,0.00,1
1.51766,13.21,3.69,1.29,72.61,0.57,8.22,0.00,0.00,1
1.51742,13.27,3.62,1.24,73.08,0.55,8.07,0.00,0.00,1
```

Slika 6:Prikaz izlgeda zapisa podataka

5.2. Numeričke vrijednosti

Prilikom stvaranja pravila, kod kategoričkih vrijednosti algoritam provjerava jedino je li pojedini atribut određene vrijednosti ili nije. Dok kod rada s numeričkim atributima nije bitno je li neka vrijednost jednaka pojedinim brojevima, nego kojem rasponu brojeva pripada. Također, uvjeti koji se tiču numeričkih vrijednosti mogu imati vrijednost koja predstavlja donju granicu i vrijednost koja predstavlja gornju granicu raspona. Dakle, pred algoritmom se nalazi problem pronalaska granica za usporedbu numeričkih vrijednosti. Rješenje tog problema se nalazi u diskretizaciji numeričkih vrijednosti. Ovisno o ulaznim podacima pronalazi se skup vrijednosti koji može predstavljati razne raspone unutar podataka. Jedan od načina je da se učitaju svi različiti brojevi koji se pojavljuju u listu. Lista se sortira te se kao granice uzimaju vrijednosti jednake izrazu (7):

$$\frac{v_n + v_{n+1}}{2} \quad (7)$$

U izrazu (7), *v* predstavlja vrijednost člana na *n*-tom ili (*n*+1)-om članu u listi.

Ovakav pristup je vrlo jednostavan za implementirati, ali je vrlo spor i neefikasan. Za skupove podataka kao što je Iris (mali broj instanci, mali broj atributa, najviše 40 različitih numeričkih vrijednosti) to ne predstavlja problem pa se učenje odvija u razumnom vremenu. Dok se u skupovima kao Abalone koji su u usporedbi s Iris veliki (28 puta više instanci, duplo više atributa, najviše 2428 različitih vrijednosti) učenje odvija jako sporo.

Druga opcija je uz sortirane numeričke vrijednosti u listi također spremiti kojem razredu vrijednost pripada. U tom slučaju za granice usporedbe se ne uzimaju svi

susjedni elementi nego samo oni nakon kojih se mijenja pojedini razred. Ako dva razreda imaju jednaku vrijednost i nju se uzme za usporedbu. Iako u nekim slučajevima ovakav pristup pomaže, s obzirom na prošli objašnjeni, kod skupa Abalone to opet nije istina s obzirom na to da su mu vrlo rasute vrijednosti atributa.

Treća opcija je pronaći podjelu na instance s najvećom količinom informacije. Za početak se spremaju u listu sve numeričke vrijednosti i broj instanci koje one pokrivaju. Nakon toga lista se sortira po vrijednosti te se iterira po članovima od početka do kraja. Za svakog člana se pronalazi broj instanci koju su veće i koje su manje od njega. Zatim se izračuna količina informacije za pronađene grupe instanci te se na kraju za uvjet odabire vrijednost čija grupa ima najveću količinu informacija. Ovakav pristup daje najbolje rezultate te se algoritam najbrže izvodi.

Od tri prikazane opcije, preporučuje se korištenje diskretizacije na temelju količine informacije.

5.3. MDL i skupovi pravila

Kao što je objašnjeno u 3.3., *MDL* se koristi za odabir najjednostavnije teorije za opis podataka. Točnije, kao što je Cohen napisao u svome radu, koristi se formula za kodiranje podskupova koja glasi (8):

$$S(n,k,p) = k\ \log_2 \frac{1}{p} + (n-k) \log_2 \frac{1}{1-p} \quad (8)$$

Ovdje *n* predstavlja broj poznatih elemenata, *k* broj elemenata koji se želi kodirati, a *p* se računa kao *k/n* i poznat je od strane primatelja. Također kada se računaju bitovi potrebni za zapis teorije, ukupna suma se množi s koeficijentom između 0 i 1 (uobičajeno 0.5) da bi se prilagodila mogućoj redundanciji kod atributa. Izraz (8) se koristi za kodiranje skupova i podskupova pravila te podataka i iznimki. Kada se koristi za kodiranje pravila, n je broj mogućih kombinacija uvjeta za atribute, *k* je broj uvjeta koji se koriste. Broj mogućih kombinacija uvjeta se računa na početku rada algoritma, tako da se ovisno o atributima sumiraju njihove moguće vrijednosti. Za kategoričke atribute se zbrajaju samo moguće vrijednosti, dok se za numeričke

atribute množi broj granica za usporedbu s 2 jer instance mogu biti veće ili manje od svake pojedine granice. Kodiranje podataka i iznimki se odrađuje tako da se logaritmira broj ukupnih podataka te se pronađe broj bitova za pokrivene i nepokrivene iznimke. Pokrivenim iznimkama se smatraju lažno pozitivni podaci koji nastaju kada algoritam instancu nekog drugog razreda prepozna kao instancu radnog razreda. Nepokrivenim iznimkama se smatraju lažno negativne instance tj. podaci koji pripadaju radnom razredu ali ih trenutačni skup pravila ne pokriva. Za kodiranje jednog podskupa potrebno je izračunati opisnu dužinu pravila, podataka i iznimki.

Tijekom rada algoritma, potrebno je više puta izračunati opisnu dužinu skupova i podskupova pravila. Radi bržeg rada algoritma, svakom pravilu se nakon što je generiran zapisuje koje sve podatke pokriva. Kada treba izračunati opisnu duljinu iznimki iterirajući po zapisu pokrivenih podataka, lako se izračuna broj lažnih pozitivnih i negativnih instanci, dok je u suprotnom potrebno svaki put provjeravati koje pravilo prekriva koje instance. Tijekom razvoja ove implementacije, prešlo se na iteriranje po zapisu pokrivenih podataka te se algoritam ubrzao skoro 10 puta s obzirom na prošli pristup.

6. Vrednovanje

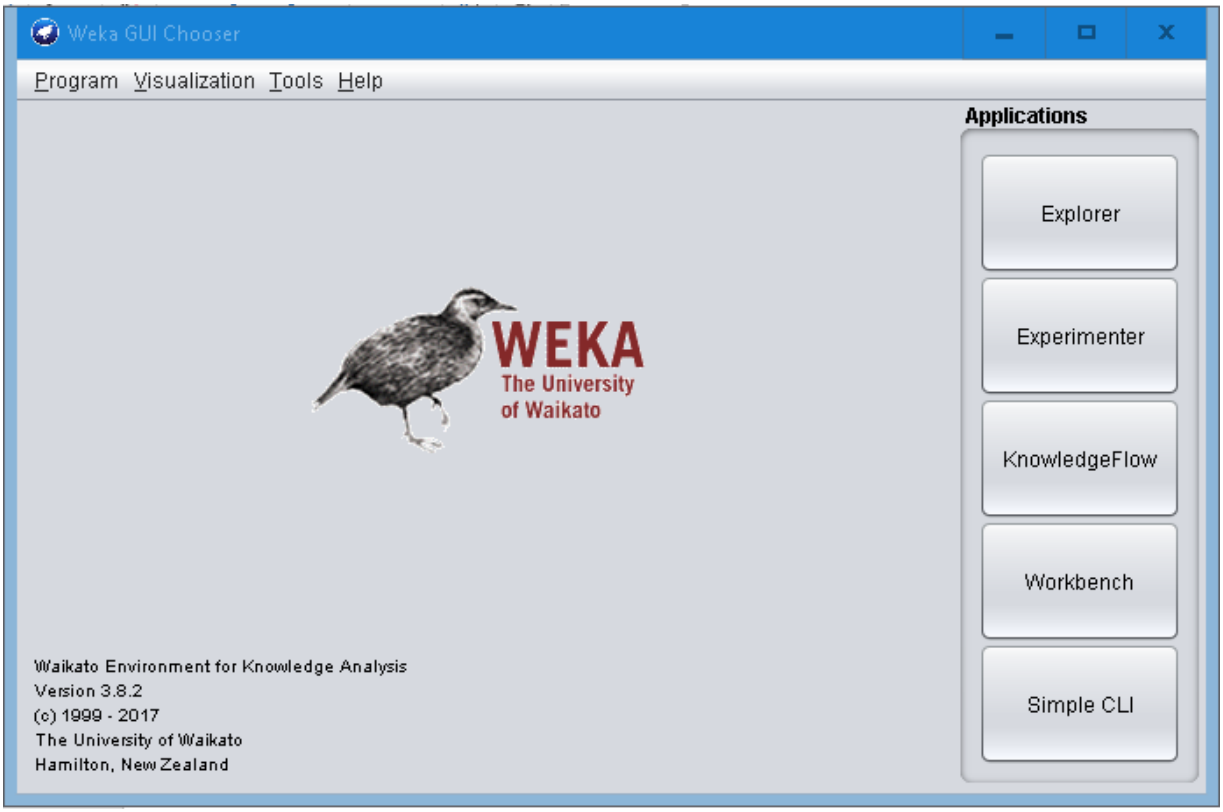
Vrednovanje će se provesti na sedam već prikazanih skupova podataka uz pomoć implementacija algoritma *RIPPERk* u javno dostupnim izvorima kao što su Weka i RapidMiner.

6.1. Weka

Weka je kolekcija algoritama strojnog učenja nastala na fakultetu Waikato u Novom Zelandu. Algoritmi su pisani u Javi te rješavaju razne probleme strojnog učenja poput problema regresije, grupiranja, klasifikacije i slično. U sklopu ovog rada bitan je *JRIP* tj. *RIPPERk* napisan u Javi. [7]

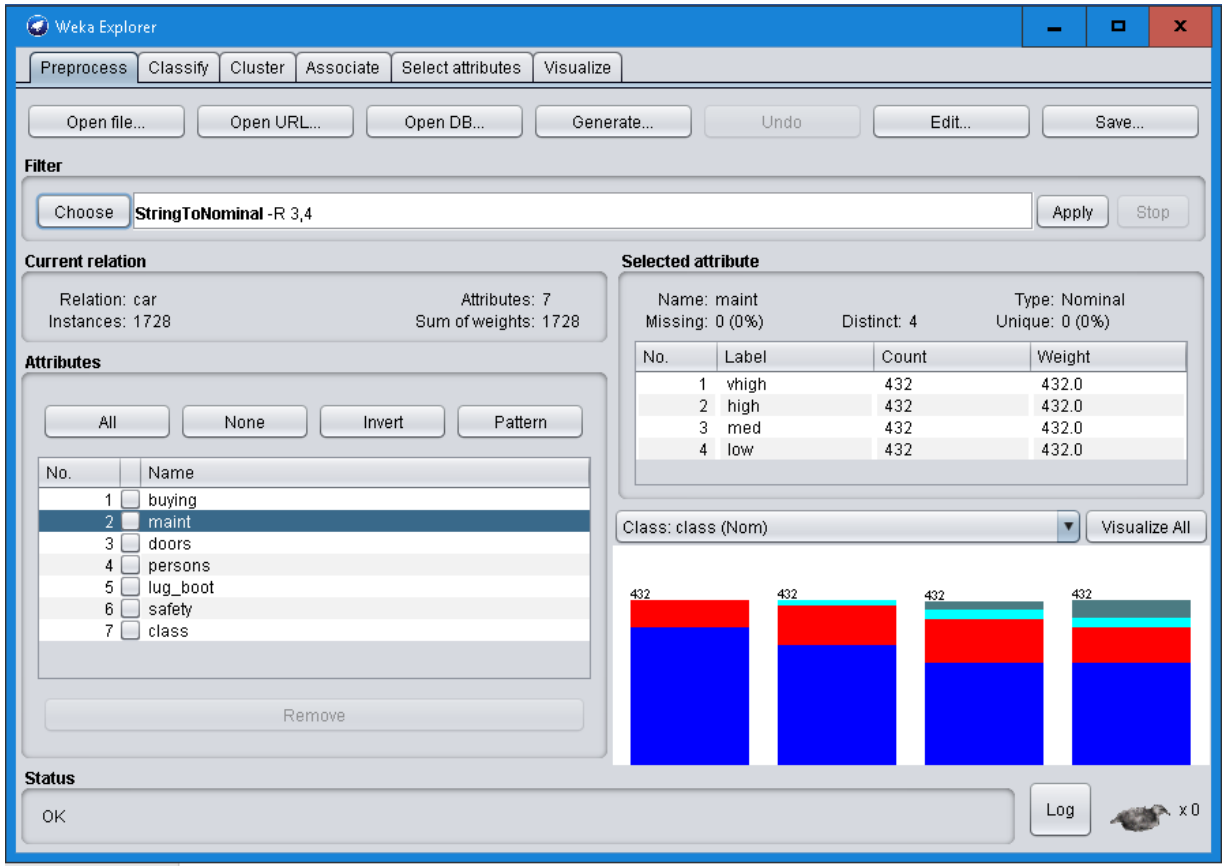
6.1.1. Upute za korištenje

Nakon instalacije Weka programskog paketa, te početka korištenja prikazuje se sljedeći izbornik.



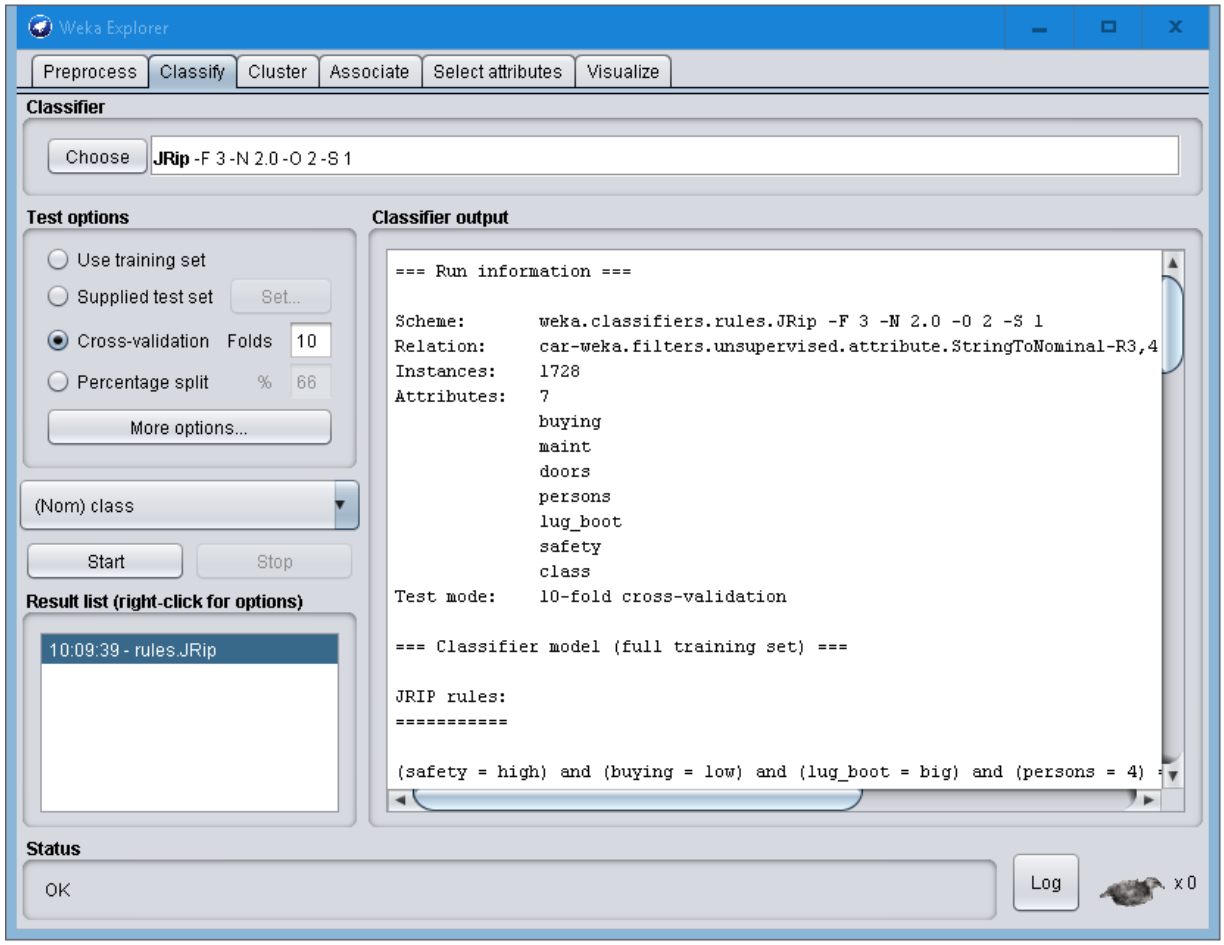
Slika 7: Prikaz glavnog izbornika

Odabirom tipke *Explorer*, dolazi se do dijela za rad s podacima. Na početku se učitavaju podaci te se mogu promijeniti atribute i vrijednosti pojedinih instanci ako je to potrebno. To se obavlja tako da se odaberu pojedini filtri za promjenu tipova atributa, ili pak dodaju i brišu instance te cijeli skupovi atributa. U sklopu ovog rada ta mogućnost se koristi kada je program krivo označio razne atribute kao tip string, dok su oni zapravo nominalnog tipa. Također se mogu vidjeti razni grafovi i podjele koje opisuju skup podataka koji je predan programu.



Slika 8: Prikaz manipuliranja podacima

Nakon učitavanja podataka dolazi se do samog klasificiranja. Odabirom kartice *Classify* dolazi se do dijela za odabir algoritma i prikaz rezultata klasificiranja. Zatim se odabire algoritam *JRIP* te testiranje deseterostrukom unakrsnom validacijom.



Slika 9: Prikaz odabira algoritma i pregleda rezultata

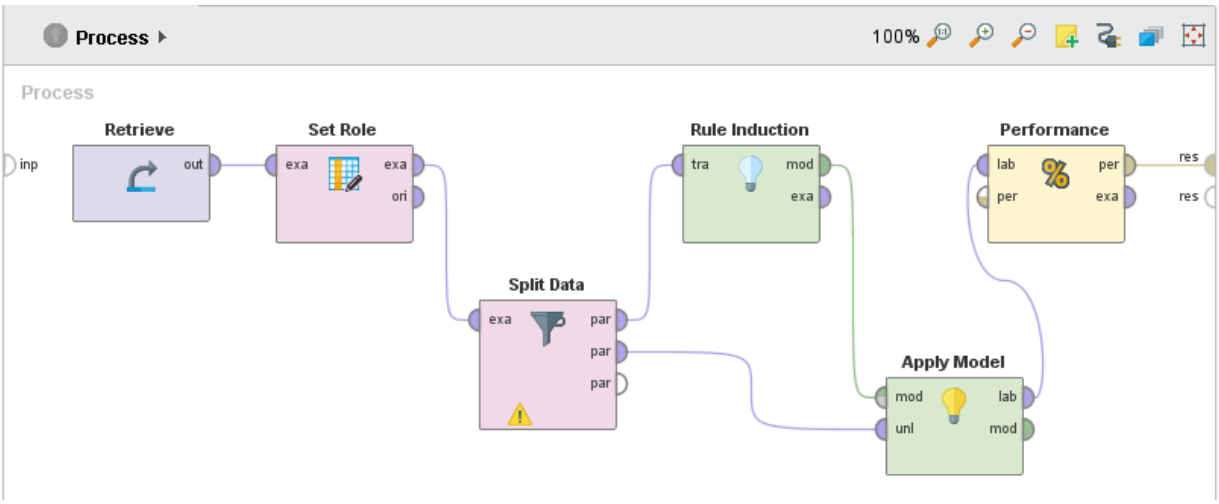
6.2. RapidMiner

RapidMiner je programska platforma koja se također bavi raznim analizama i manipulacijom podataka, strojnim učenjem, prediktivnom analitikom i sličnim. Nastao je 2001. pod imenom YALE na Dortmundskom sveučilištu te 2007. mijenja ime u RapidMiner. RapidMiner nudi tri različite godišnje pretplate i jednu besplatnu verziju u trajanju od mjesec dana, ali besplatna verzija ima ograničen broj podataka i smanjene performanse s obzirom na plaćene verzije. Za ovaj rad će se koristiti besplatna verzija. [8]

6.2.1. Upute za korištenje

Rad s RapidMinerom svodi se na stvaranje procesa koji se puni raznim operatorima. Operatori se vrlo jednostavno spajaju ulaznim i izlaznim kanalima. Spajanjem različitih operatora se zapravo stvara slijed akcija koje treba obaviti nad podacima. Neki operatori kao operator *Cross Validation* sadrže unutarnje procese koji se također mogu puniti operatorima.

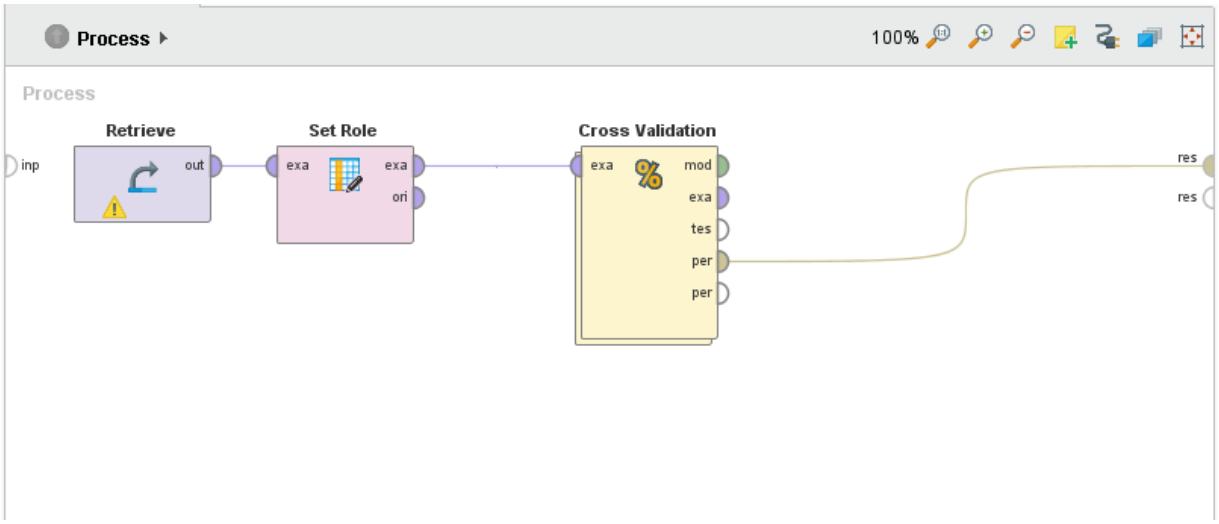
Krenuvši od praznog procesa, prvo se mora dodati operator koji će unijeti željene podatke. Za to se koristi operator *Retrieve*, te se njegov izlaz spaja na operater *Set Role* koji će promijeniti naziv atributa kojeg se želi klasificirati. To se čini jer razred/atribut kojeg se klasificira mora biti označen s imenom *label*. Nakon toga ovisno o načinu evaluacije se dodaje operator *Split Data* ili *Cross Validation*.



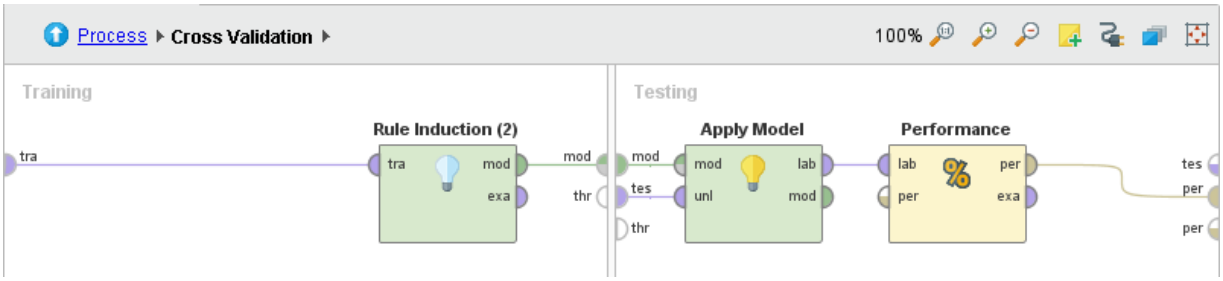
Slika 10: Prikaz slijeda operatora uz *Split Data* podjelu

Operator *Split Data* se koristi u slučaju kada se želi evaluirati podatke na temelju podjele podataka 66% za učenje i 33% za evaluaciju. Prva particija se spaja na operator *Rule Induction* koji predstavlja implementaciju algoritma *RIPPERk*, dok se druga particija spaja na operator *Apply Model*. Na operator *Apply Model* se također spaja i model kojeg je generirao operator *Rule Induction* te on ovisno o ulaznim podacima iznosi svoje predviđanje. Za kraj se podatke iz operatora *Apply Model*

zajedno s predviđanjima šalje na operator *Performance* koji proračunava statistiku i postotak točnosti.



Slika 11: Prikaz slijeda operatora uz *Cross Validation* podjelu



Slika 12: Unutrašnjost operatora *Cross Validation*

Operatoru *Cross Validation* se predaju podaci s izlaza operatora *Set Role* te se unutar njegovog *Training* dijela postavlja operator *Rule Induction* koji kao i kod prošle upotrebe šalje model u *Testing* dio. Unutar *Testing* dijela nalazi se ponovno kombinacija operatora *Apply Model* i *Performance* koji se na kraju spajaju s izlazom operatora *Cross Validation.* Opisani sklop predstavlja deseterostruku unakrsnu validaciju.

6.3. Rezultati

Za usporedbu će se koristiti tri vrijednosti: *precision*, *recall* i točnost. *Precision*(6) je već objašnjen u 3.4.1., ali u ovom slučaju se računa *precision* skupa pravila za određeni razred. *Recall*(9) se računa na sljedeći način:

$$rec = \frac{tp}{tp+fn} \quad (9)$$

Ovdje *tp* predstavlja podatke određenog razreda koji su točno klasificirani, dok *fn* predstvalja podatke istog razreda koji sukrivo klasificirani. U tablicama su prikazane vrijednosti za cijeli skup a ne za pojedine razrede te se svaka implementacija poziva tri puta sa nasumično poredanim podacima.

6.3.1. Skup podataka Iris

	RapidMiner			Weka			ripper.py		
1.	0.921	0.920	0.920	0.937	0.940	0.940	0.927	0.927	0.927
2.	0.946	0.946	0.946	0.956	0.953	0.960	0.940	0.940	0.940
3.	0.940	0.940	0.940	0.932	0.933	0.933	0.947	0.947	0.947
	P	R	T	P	R	T	P	R	T

6.3.2. Skup podataka Abalone

	RapidMiner			Weka			ripper.py		
1.	0.126	0.096	0.230	0.104	0.071	0.192	0.054	0.065	0.177
2.	0.098	0.084	0.223	0.065	0.131	0.192	0.062	0.072	0.181
3.	0.106	0.094	0.231	0.101	0.070	0.192	0.049	0.059	0.175
	P	R	T	P	R	T	P	R	T

6.3.3. Skup podataka Ionosphere

	RapidMiner			Weka			ripper.py		
1.	0.873	0.864	0.880	0.894	0.894	0.903	0.864	0.864	0.875
2.	0.872	0.874	0.883	0.894	0.887	0.900	0.863	0.866	0.875
3.	0.854	0.846	0.863	0.877	0.874	0.886	0.882	0.875	0.889
	P	R	T	P	R	T	P	R	T

6.3.4. Skup podataka Wine

	RapidMiner			Weka			ripper.py		
1.	0.882	0.871	0.876	0.945	0.947	0.943	0.954	0.945	0.949
2.	0.870	0.872	0.865	0.916	0.919	0.915	0.943	0.944	0.944
3.	0.877	0.878	0.876	0.945	0.941	0.938	0.930	0.924	0.927
	P	R	T	P	R	T	P	R	T

6.3.5. Skup podataka Glass Identification

	RapidMiner			Weka			ripper.py		
1.	0.665	0.701	0.696	0.640	0.610	0.682	0.618	0.644	0.687
2.	0.639	0.598	0.691	0.612	0.639	0.654	0.626	0.604	0.659
3.	0.621	0.665	0.705	0.632	0.594	0.672	0.661	0.663	0.673
	P	R	T	P	R	T	P	R	T

6.3.6. Skup podataka Adult

Skup podataka Adult neće biti evaluiran na RapidMiner imeplementaciji s obzirom da ima više podataka od dozvoljene granice.

	Weka			ripper.py		
1.	0.806	0.742	0.845	0.798	0.714	0.835
2.	0.798	0.744	0.842	0.794	0.718	0.835
3.	0.805	0.742	0.844	0.784	0.721	0.832
	P	R	T	P	R	T

6.3.7. Skup podataka Car evaluation

	RapidMiner			Weka			ripper.py		
1.	0.770	0.751	0.885	0.701	0.781	0.883	0.690	0.742	0.865
2.	0.772	0.724	0.886	0.694	0.781	0.879	0.711	0.750	0.871
3.	0.745	0.724	0.889	0.709	0.742	0.875	0.700	0.755	0.858
	P	R	T	P	R	T	P	R	T

7. Zaključak

RIPPERk je moćan algoritam za klasificiranje podataka. Može se koristiti s numeričkim i kategoričkim vrijednostima te prihvaća nedostatak pojedinih atributa. Koristi se poznatim principima strojnog učenja kao što su podrezivanje, *MDL* i slični.

Implementacija algoritma se može činiti jednostavnom, ali postoji mnogo detalja koji nisu napisani u knjigama a vrlo su bitni za rad algoritma. Stoga je vrlo važno imati originalni algoritam ili neku pouzdanu implementaciju koja može biti vodilja tijekom pisanja nove implementacije.

Ova implementacija ima vrlo sličnu točnost kao i implementacije Weka i RapidMiner programskih paketa. Kada se uspoređuju brzine izvođenja, u slučaju kada skup podataka ima samo numeričke atribute također su brzine vrlo slične, ali kada skup ima kategoričke atribute, brže su Weka i RapidMiner implementacije.

Iako su implementirane tri različite diskretizacije, preporučuje se korištenje diskretizacije temeljene na količini informacija granica za podjelu. Diskretizacije po razredima i svim numeričkim vrijednostima su neefikasne i mogu prouzročiti neočekivane rezultate.

8. Literatura

[1] Cohen W. W., Fast effective rule induction, Machine Learning Proceedings of the Twelfth International Conference, Tahoe City (1995), 115 - 123

[2] Alpaydin E., Introduction to Machine Learning, USA: Massachusetts Institute of Technology, 2014.

[3] Witten I. A., Frank E., Hall M. A., Data Mining - Practical Machine Learning Tools and Techniques, Brulington: Elsevier, 2011.

[4] Fürnkranz J., Gamberger D., Lavrač N., Foundations of Rule Learning, Springer, 2012.

[5] Cohen W. W., Singer Y., A Simple, Fast, and Effective Rule Learner, New York (1999)

[7] Waikato Universety, Weka, https://www.cs.waikato.ac.nz/ml/weka/, 01.06.2018.

[8] RapidMiner, RapidMiner, https://rapidminer.com/, 01.06.2018.

[9] Quinlan J. R., C4.5: Programs for Machine Learning, San Mateo: Morgan Kaufmann publishers, 1998.

[10] Fürnkranz J., Widmer G., Incremental Reduced Error Pruning, Machine Learning Proceedings of the Eleventh International Conference, New Brunswick (1994), 70 - 77

9. Naslov, sažetak i ključne riječi

Naslov: Implementacija i vrednovanje algoritma RIPPER za izgradnju pravila prekrivanja

Ključne riječi: dubinska analiza podataka, strojno učnenje, opisna duljina, pravila, algoritmi prekrivanja, skupovi podataka, podrezivanje, heuristika, validacija

U ovom radu je opisan algoritam *RIPPERk* i njegov prethodnik *IREP* te najvažniji principi po kojima rade. Također je opisana teorijska podloga za rad alogirtama strojnog učenja. Nakon teorijskog dijela opisana je implementacija algoritma *RIPPERk* te su napisane upute za instalaciju. Zatim je opisan način korištenja sličnih algoritama s javno dostupnih izvora, te su na kraju prikazani i uspoređeni rezultati različitih implementacija.

10. Abstract

Title: Implementation and evaluation of the RIPPER algorithm for covering rules induction

Keywords: data mining, machine learning, description length, rules, covering rules algorithms, data sets, pruning, heuristics, validation

This paper describes the *RIPPERk* algorithm , its predecessor *IREP* and the most important principles by which they work. Theoretical background for the work of machine learning algorithms is also described. After the theoretical section, the implementation of the *RIPPERk* algorithm is described and the installation instructions are written. Next, the usage of similar algorithms which are found in publicly available sources is explained. Finally the results of the various implementations are presented and compared.