SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 4849

# Postupci za učenje klasifikacijskih pravila

Matija Haničar

Zagreb, lipanj 2017.

Ovdje dođe original zadatka koji je preuzet od djelovođe diplomskih i završnih radova

Sadržaj

# 1. Uvod

Danas postoje brojni pristupi u strojnome učenju. Različiti pristupi koriste se u različitim situacijama. Neki od poznatijih pristupa su: klasifikacijsko učenje, asocijativna pravila, grupiranje, numerička predikcija. Cilj klasifikacijskog učenja je naučiti kako klasificirati podatke u određene kategorije. S ovim znanjem možemo buduće podatke klasificirati, što nam može biti korisno u daljnjoj analizi podataka. Metode u strojnome učenju se dijele na nadzirano, nenadzirano i polunadzirano učenje. Klasifikacijsko učenje spada pod kategoriju nadziranog učenja, jer za svaku uzorak (primjerak, instancu) unutar skupa za učenje znamo kojoj klasi zapravo pripada.

Jedna od metoda klasifikacije su klasifikacijska pravila. Klasifikator koji koristi ovu metodu za ulazni skup, odnosno skup za učenje vraća, skup tzv. AKO-ONDA pravila koja klasificiraju uzorke. Danas postoje brojni algoritmi koji realiziraju ovaj klasifikator, kao što su na primjer: FURIA [1], PRISM [2], IREP++ [3]. Trenutno jedan od vodećih algoritama je RIPPER [4]. U sklopu ovog završnog rada napravit ćemo vlastitu implementaciju tog algoritma i usporediti ga s postojećom implementacijom unutar biblioteke `Weka.jar`.

# 2. Podaci

Prije nego što počnemo ulaziti dublje u temu završnog rada, trebamo prvo definirati ulazne podatke. Ključni pojmovi su uzorak (engl. *sample*) i atribut (engl. *attribute*). Skup podataka je zapravo skup uzoraka, dok se uzorak sastoji od atributa. Ako skup podataka prikažemo tablicom, tada su atributi stupci, a uzorci reci.

Atributi mogu biti numerički ili kategorički. Numerički atributi mogu biti kontinuirani (realni), ili diskretni (cjelobrojni). Kategorički podaci označavaju da atribut umjesto numeričke vrijednosti poprima jednu od unaprijed određenih kategorija, koje dolaze iz konačnog skupa diskretnih vrijednosti. Primjer realnog atributa je duljina latica unutar skupa podataka *Iris*, koji je jedan od najpoznatijih skupova podataka unutar strojnog učenja. Primjer kategoričkog atributa je astigmatičanost u skupu podataka *Lenses*. Vrijednosti koje postiže su: *da* i *ne*. Svaki uzorak ima labelu koju nazivamo klasom. Klasa je zapravo kategorički atribut, no zbog važnosti često se navodi kao poseban element uzorka. Uzorak može pripadati u više klasa, no u ovome radu koristit ćemo uzorke koji pripadaju samo jednoj klasi. Također moguće je da i za neke uzorke atribut nema vrijednost. U nekim slučajevima previše je drakonski jednostavno izbrisati te uzorke, jer mogu sadržavati važne informacije za izgradnju novih klasifikacijskih pravila. Način na koji to funkcionira bit će objašnjen prilikom objašnjavanja same implementacije algoritma RIPPER.

Skupove podataka *Iris* i *Lenses*, uz druge, koristit ćemo i dalje u radu. Treba napomenuti da kategorički podaci uglavnom imaju numeričke vrijednosti kao ključeve za stvarne vrijednosti. Primjerice atribut astigmatičnost poprima vrijednosti 1 ili 2, kao šifre za *ne*, odnosno *da*. Podaci korišteni u ovome radu, javno su dostupni i preuzeti su sa stranica sveučilišta Sveučilišta u Irvineu, u Kaliforniji (http://archive.ics.uci.edu/ml/).

Jedan od implementacijskih problema kod strojnog učenja je činjenica da podaci dolaze u različitim oblicima. Preuzeti podaci dolazili su u različitim oblicima, pa ih je bilo potrebno pretvoriti u jednak oblik. Odabran je oblik datoteke s vrijednostima odvojenim zarezom (.csv).

Postoje i drugi oblici zapisa podataka, kao što je datoteka u obliku .arff, koju koristi Weka, alat koji ćemo koristiti kasnije pri usporedbi implementacija algoritma RIPPER. Primjer .arff-datoteke vidimo na slici 1. U datoteci su jasno definirani atributi, kao i vrijednosti koje mogu poprimiti. Ovdje su brojke zapravo šifre za određene vrijednosti. Primjerice atribut *age* poprima vrijednosti: 1, 2 i 3, što su zapravo šifre za *young*, *pre-presbyopic*, odnosno *presbyopic.* Ovako strukturirani podaci znatno su lakši za korištenje jer su standardizirani. Nažalost, repozitorij s kojeg su preuzeti podaci nije sadržavao podatke u ovome formatu, pa je bilo potrebno ručno ih pretvoriti u ovaj oblik.

```
@RELATION haberman

@ATTRIBUTE age          {1,2,3}
@ATTRIBUTE prescription         {1,2}
@ATTRIBUTE astigmatic           {1,2}
@ATTRIBUTE tears                {1,2}
@ATTRIBUTE class        {1,2,3}

@DATA

1,1,1,1,3
1,1,1,2,2
1,1,2,1,3
1,1,2,2,1
1,2,1,1,3
1,2,1,2,2
1,2,2,1,3
1,2,2,2,1
2,1,1,1,3
```

**Slika 1. Primjer .arff datoteke**

**Tablica 1. Sažetak iz skupa podataka Iris**

| Duljina čašičnog listića | Širina čašičnog listića | Duljina latice | Širina latice | Klasa |
|---|---|---|---|---|
| 5.1 | 3.5 | 1.4 | 0.2 | Iris setosa |
| 4.9 | 3.0 | 1.4 | 0.2 | Iris setosa |
| 7.0 | 3.2 | 4.7 | 1.4 | Iris versicolor |
| 6.4 | 3.2 | 4.5 | 1.5 | Iris versicolor |
| 6.3 | 3.3 | 6.0 | 2.5 | Iris virginica |
| 5.8 | 2.7 | 5.1 | 1.9 | Iris virginica |

**Tablica 2. Sažetak iz skupa podataka Lenses**

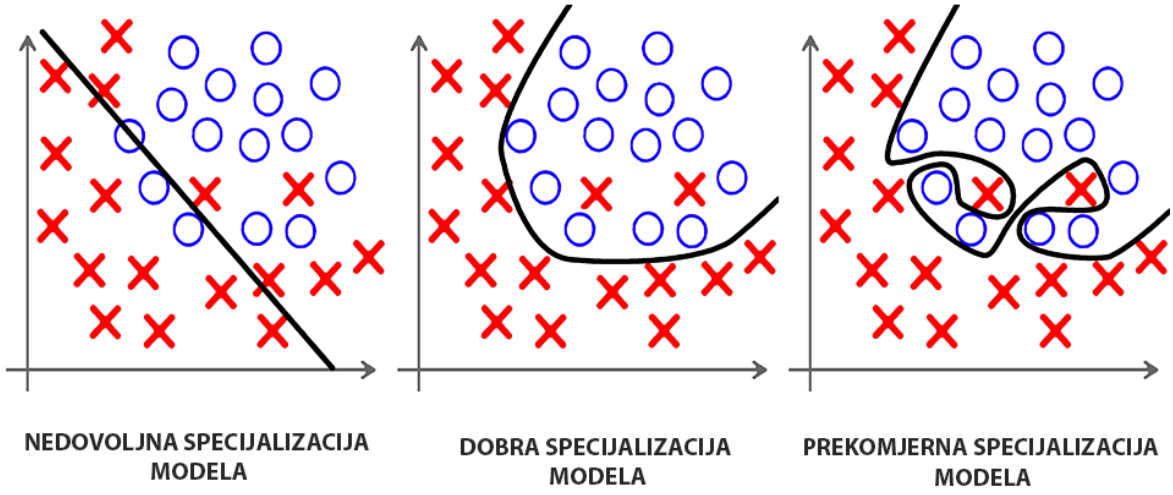
| Dob | Poremećaj vida | Astigmatičnost | Proizvodnja suza | Tip leća |
|---|---|---|---|---|
| Mlad | Kratkovidan | Ne | Reducirana | Bez leća |
| Mlad | Kratkovidan | Ne | Normalna | Meke leće |
| Pred-prezbiopna | Dalekovidan | Da | Reducirana | Bez leća |
| Pred-prezbiopna | Kratkovidan | Da | Normalna | Tvrde leće |

# 3. Klasifikacija

Klasifikacija je jedan od važnijih problema kojima se bave strojno učenje i statistika. Ideja je da se temeljem informacija iz skupa podataka za učenje, gdje za svaki uzorak znamo klasu, novim uzorcima predvidimo klasu.

Objekt u memoriji koji uči o podacima ili ih klasificira naziva se klasifikator. Postoje brojni algoritmi koji izgrađuju klasifikatore. Uspješnost klasifikatora izražava se u postotku točnih klasifikacija nad testnim skupom podataka. Taj skup podataka odvojen je od skupa za učenje. Često je u praktičnoj primjeni uspješnost klasifikatora subjektivna, jer ovisi o tome koliko rezultati imaju smisla čovjeku. Primjerice, ako koristimo dva klasifikatora koji koriste klasifikacijska pravila, klasifikatore A i B, koji kao rezultate daju skup od 5, odnosno 15 pravila uz preciznost 95%, odnosno 99%, izabrat ćemo klasifikator A. Iako klasifikator A ima manju preciznost, vraća puno manje pravila od klasifikatora B, što nama ima veću praktičnu važnost.

Važan pojam kod klasifikacije je i specijalizacija modela. Ako je specijalizacija modela prekomjerna (*eng. Overfitting*), daje se prevelika važnost slučajnim varijacijama vrijednosti podataka. Takvi modeli tipično imaju veliku točnost predviđanja na skupu uzoraka za učenje, a značajno nižu na skupu podataka za testiranje. Drugi ekstrem je nedovoljna specijalizacija modela (*eng. Underfitting*), gdje je pristranost izrazito velika i točnost predviđanja je niska u oba skupa podataka. Dobra razina specijalizacije modela nalazi se između ova dva slučaja i ovisi o podacima za učenje kao i o samom algoritmu za klasifikaciju. Točnost predviđanja za dobar model visoka je i u podacima za učenje, kao i u podacima za testiranje.




**Slika 2. Različite razine specijalizacije modela**

# 4. Klasifikacijska pravila

Problem učenja klasifikacijskih pravila može se opisati na sljedeći način: „Temeljem skupa podataka za učenje, pronađi skup klasifikacijskih pravila koja se mogu koristiti za predikciju klase novih uzoraka“. Pravila koja se uče su sljedećeg oblika:

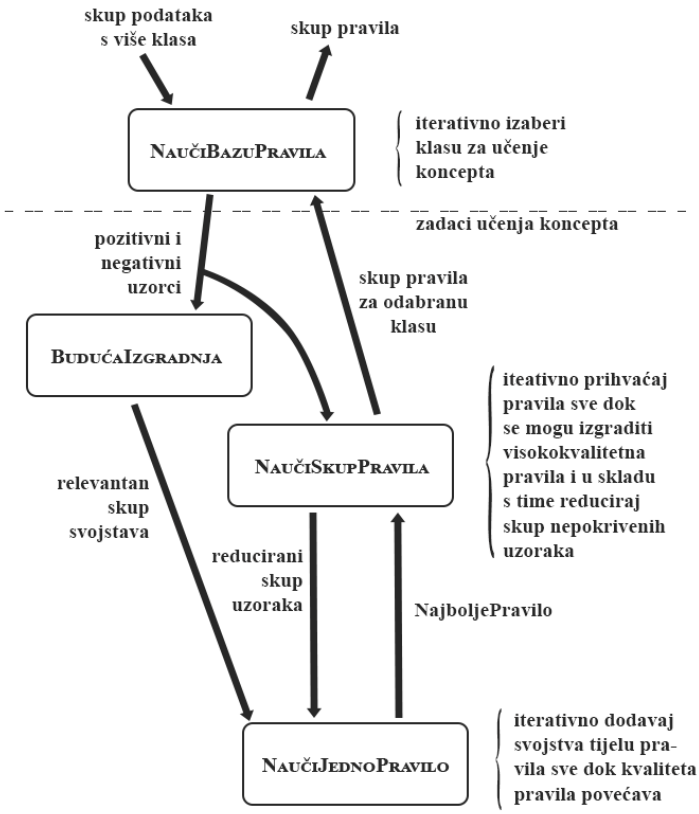
AKO $f_1$ I $f_2$ I $f_3$ ONDA Klasa = K,

gdje svojstvo $f_k$ predstavlja test koji uzorak mora zadovoljiti kako bi pripadao klasi K. Ako su atributi diskretni, test je oblika ATRIBUT = VRIJEDNOST, dok se kod kontinuiranih atributa koriste oblici ATRIBUT > VRIJEDNOST i ATRIBUT <= VRIJEDNOST.

Pravilo se sastoji od dva dijela: antecedenta i konsekvensa. Antecedent je uvjetni dio pravila (dio između AKO i ONDA) i sastoji se od konjunkcije svojstava $f_k$. Drugi naziv za antecedent je tijelo pravila. Konsekvens ili glava pravila je zaključak izvučen temeljem antecedenta i kod klasifikacijskih pravila označava klasu kojoj uzorak pripada.

## 4.1. Učenje klasifikacijskih pravila

Proces učenja klasifikacijskih pravila sastoji se od 3 slijedne faze: izgradnje svojstava, izgradnje pravila i izgradnje hipoteze. Posljednje dvije faze mogu se izmjenjivati nekoliko puta.




**Slika 3: Faze učenja pravila**

Prilikom izgradnje svojstava, skup podataka za učenje se pretvara u skup parova `ATRIBUT=VRIJEDNOST`, tj. `ATRIBUT > VRIJEDNOST` ili `ATRIBUT<= VRIJEDNOST`.

Nakon što smo odredili skup svojstava, slijedi izgradnja pravila. Prvi korak prilikom izgradnje pravila je odabir klase, tj. glave pravila. Nakon toga heuristički izgrađujemo tijelo pravila. Nakon što smo izgradili pravila preostaje nam samo da ta pravila posložimo u hipotezu. Učenje pravila može se opisati kao problem pretrage stanja. Da bismo to napravili moramo definirati sljedeće pojmove:

- Prostor pretraživanja
- Strategija pretraživanja
- Funkcija kvalitete

Prostor pretraživanja predstavlja skup svih pravila koje je moguće dobiti kombinirajući sva moguća svojstva i sve moguće klase. Veličina prostora

pretraživanja direktno ovisi o atributima i već se za jednostavne primjere pokazuje da je potrebno koristiti određene strategije pretraživanja.

Prije nego što razmotrimo strategije pretraživanja treba definirati pojmove generalizacije i specijalizacije, a da bismo to napravili trebamo prvo definirati funkciju `POKRIVEN(p, S)` koja vraća skup svih uzoraka unutar skupa `S` koji zadovoljavaju pravilo `p`.

Pravilo $p_1$ je općenitije (generalnije) od pravila $p_2$ ako i samo ako pravila $p_1$ i $p_2$ imaju istu glavu i `POKRIVEN(p1, S)` $\subseteq$ `POKRIVEN(p2, S)`. Suprotno od općenitosti je specifičnost.

Svaki prostor pretraživanja ima univerzalno pravilo koje pokriva sve uzorke, pravilo kojemu je tijelo `ISTINA`. Algoritmi izgradnje pravila koriste ovu činjenicu i daljnjom specijalizacijom i generalizacijom izgrađuju pravilo. Ovakav pristup naziva se pristup odozgo prema dolje, jer kreće od najopćenitijeg pravila i kao rezultat daje specifičnije pravilo. Postoji i obrnuti pristup, tzv. pristup odozdo prema gore, koji kreće od najspecifičnijeg pravila i kao rezultat daje općenitije pravilo. Primjer algoritma koji koristi pristup odozgo prema dolje je algoritam na slici 4.

funkcija NaučiJednoPravilo($k_i$, $P_i$, $N_i$)
*Ulaz:*
$k_i$: vrijednost klase
$P_i$: skup pozitivnih uzoraka za klasu $k_i$
$N_i$: skup negativnih uzoraka za klasu $k_i$
S: skup svojstava
*Algoritam:*
p := ( $k_i \leftarrow$ B), gdje je B $\leftarrow \emptyset$
ponavljaj
sagradi poboljšanja $\rho(p) \leftarrow \{ p' \mid p' = (c_i \leftarrow B \wedge s)\}$ za svako $s \in S$
procijeni sve $p' \in \rho(p)$ po kriteriju kvalitete
p := najbolje poboljšanje p' u $\rho(p)$
dok p ne zadovolji prag kvalitete
ili ne pokriva niti jedan uzorak iz $N_i$
*Izlaz:*
naučeno pravilo p

**Slika 4: Algoritam izgradnje pravila pristupom od odozgo prema dolje [6]**

Algoritam gradi pravilo tako da u svakom koraku doda novo svojstvo tijelu pravila. Bira se svojstvo koje će maksimizirati proizvoljnu funkciju kvalitete. Ovaj korak se ponavlja dok nismo postigli željenu razinu specifičnosti. Ovakva metoda naziva se metodom uspona na vrh, jer u svakome koraku tražimo lokalni maksimum.

Da bismo definirali funkciju kvalitete, prvo trebamo uvesti neke pojmove. Već smo definirali funkciju `POKRIVEN(p, S)` koja vraća podatke iz podskupa skupa `S` koji zadovoljavaju pravilo `p`. Taj skup nazvat ćemo skupom pokrivenih uzoraka. Ostale uzorke, tj. razliku između skupa `S` i rezultata funkcije `POKRIVEN(p, S)` nazivamo skupom nepokrivenih uzoraka. Uzorke možemo podijeliti i na pozitivne i negativne. Pozitivni uzorci su svi uzorci koji pripadaju klasi koja se nalazi u glavi pravila, dok su negativni svi ostali. Ako ova 4 pojma napišemo pomoću matrice zbunjenosti (*eng. confusion matrix*), dobit ćemo 4 nova pojma: Istinito pozitivne, lažno pozitivne, istinito negativne i lažno negativne.

**Tablica 3: Tablica zbunjenosti**

| **Stvarna vrijednost** | **Predviđen pozitivan** | **Predviđen negativni** |
|---|---|---|
| Pozitivni (P) | Istinito pozitivni (IP) | Lažno negativni (LN) |
| Negativni (N) | Lažno pozitivni (LP) | Istinito negativni (IN) |

Istinito pozitivni uzorci su uzorci koji su uistinu pozitivni i klasificirani su kao pozitivni, za razliku od lažno pozitivnih koji su krivo klasificirani kao pozitivni. U statistici ova pogreška se naziva pogreškom prve vrste. Lažno negativni uzorci su uzorci koje je klasifikator odbacio, iako ih je trebao prihvatiti, a u statistici to odgovara pogrešci drugog reda. Za istinito negativne, točno je predviđeno da ne pripadaju određenoj klasi.

Koristeći ove pojmove možemo definirati različite funkcije kvalitete. Funkcija bi trebala istovremeno raditi sljedeće:

- Maksimizirati broj istinito pozitivnih uzoraka
- Minimizirati broj lažno pozitivnih uzoraka

Postoje različite funkcije kvalitete, primjerice funkcija `PRECIZNOST(p)` računa preciznost pravila, tj. relativnu frekvenciju istinito pozitivnih uzoraka u odnosu na sve pozitivne uzorke:

$$\texttt{PRECIZNOST(p)} = \frac{IP}{IP+LP}$$

Druge heurističke funkcije kvalitete spomenut ćemo kasnije u radu kada budemo promatrali neke implementacije algoritama.

Izgradnja hipoteze svodi se na prihvaćanje ili odbijanje novih pravila, ovisno o implementacijskim značajkama algoritma. Nakon što se pravilo prihvati ili odbije, algoritam može završiti s učenjem klasifikacijskih pravila ili može ponovo ući u fazu izgradnje pravila.

# 5. RIPPER

Trenutačno jedan od najpoznatijih i najuspješnijih algoritama za učenje klasifikacijskih pravila je RIPPER [4], akronim za *Repeated Incremental Pruning to Produce Error Reduction*. Kako bi bolje opisali ovaj algoritam, prvo moramo proučiti algoritam IREP [5], koji je svojevrsna preteča RIPPER-u.

## 5.1. IREP

IREP se sastoji od dvije glavne faze: faze rasta i faze obrezivanja. Prvo se podaci podijele u dva skupa u proizvoljnom omjeru: skupa za rast i skup za obrezivanje. Taj omjer je uglavnom 2:1, iako nije strogo definiran. U prvoj fazi gradimo pravilo sve dok ne postane savršeno, tj. dok nema lažnih pozitivnih uzoraka koje pokriva. Pri tome kao funkciju kvalitete koristimo funkciju INFORMACIJSKA_DOBIT($p_1$, $p_2$). Funkciju definiramo na sljedeći način:

$$\text{INFORMACIJSKA\_DOBIT}(pr_{novo}, pr_{staro}) = p_{novo} * \left[\log\left(\frac{p_{novo}}{t_{novo}}\right) - \log\left(\frac{P_{staro}}{T_{staro}}\right)\right],$$

gdje je $pr_{novo}$ pravilo $pr_{staro}$ nakon što smo dodali jedno svojstvo u tijelo pravila. Oznake $p_{novo}$ i $t_{novo}$, označuju broj istinito pozitivnih, odnosno pozitivnih uzoraka koje pokriva $pr_{novo}$, dok se omjer $\frac{P}{T}$ odnosi na omjer broja pokrivenih uzoraka u odnosu na sve uzorke pravila $p_{staro}$. Ovakva heuristika stavlja naglasak na pokrivenost što većeg broja pozitivnih uzoraka, bez obzira na preciznost pravila.

Nakon što smo izgradili savršeno pravilo, ulazimo u fazu obrezivanja. Prvo se izračuna vrijednost pravila, zatim se izbriše zadnje svojstvo iz tijela pravila i ponovo se izračuna vrijednost pravila. Ovaj proces se ponavlja sve dok vrijednost pravila raste. Postoje različiti načini na koje se može računati vrijednost pravila. U RIPPER-u prilikom faze koja je zapravo modificirani IREP, funkcija VRIJEDNOST(p) ima sljedeću definiciju:

$$\text{VRIJEDNOST}(p) = \frac{IP+1}{IP+LN+2},$$

gdje IP predstavlja broj istinito pozitivnih uzoraka, a LN broj lažno negativnih uzoraka.

Ove dvije faze treba proći za svaku moguću klasu nad skupom podataka, pri tome nakon svake iteracije maknuti pokrivene uzorke iz skupa podataka. Klase treba poslagati tako da se prvo uče pravila za klase koje su najmanje zastupljene, dok je posljednja najzastupljenija klasa. Problem ovog algoritma je prilično jednostavan uvjet zaustavljanja, koji za rezultat daje lošija pravila. Dobar uvjet zaustavljanja temelji se na principu najmanje duljine opisa, pojma koji nije posve trivijalan, a koristi ga algoritam RIPPER.

## 5.2. Faze algoritma RIPPER

Za razliku od IREP-a, algoritam RIPPER vraća skup pravila za svaku klasu. Skup pravila izgrađuje se kroz niz povezanih faza. Postoje 4 glavne faze algoritma: gradnja, optimizacija, brisanje i čišćenje. Cijeli pseudokod prikazan je na slici 5.

```
Inicijaliziraj E za skup uzoraka
Za svaku klasu K, od najmanje prema najvećoj
    GRADNJA:
        Podijeli E u skupove za rast i obrezivanje u omjeri 2:1
        Ponavljaj dok (a) nema više nepokrivenih uzoraka klase K; ili (b) duljina opisa (DO)
            skupa pravila i uzoraka je 64 bita veća od najmanje trenutno pronađene DO;
            ili (c) stopa pogreške je veća od 50%
        faza RASTA: Pohlepno dodaj uvjete tijelu pravila dok pravilo nije 100% precizno
            testiranjem svake moguće vrijednosti svakog atributa i izborom uvjeta s
            najvećom informacijskom dobiti D
        faza OBREZIVANJA: Obrezuj uvjete redom od posljednjeg sve dok vrijednost
            pravila raste
    OPTIMIZACIJA:
        IZGRADI VARIJANTE:
        Za svako pravilo P za klasu K,
            Podijeli E u nove skupove za rast i obrezivanje
            Iz skupa za obrezivanje ukloni sve uzorke pokrivenim drugim pravilima za K
            Koristeći faze RASTA i OBREZIVANJA izgradi dva nova pravila koristeći nove
                skupove:
                    P1 je novo pravilo, napravljeno od nule
                    P2 se gradi pohlepnim dodavanjem antecedenta pravilu P
            IZABERI PREDSTAVNIKA:
            Zamijeni pravilo P onime koje ima najmanju DO
    BRISANJE:
        Ako postoje nepokriveni primjeri klase K, vrati se na fazu GRADNJE
    ČIŠČENJE:
        Izračunaj DO cijelog skupa pravila redom za svako pravilo izostavljeno; izbriši sva
            pravila koja povećavaju DO
        Ukloni sve uzorke koji su pokriveni upravo izgrađenim pravilima
Nastavi
```

**Slika 5: Pseudokod algoritma RIPPER [4]**

Prva faza je zapravo izmijenjeni IREP algoritam. Faze rasta i obrezivanja opisane su u prošlome poglavlju. Faza gradnje ponavlja se sve dok jedan od sljedeća tri uvjeta ne bude zadovoljen:

1. Nema više nepokrivenih uzoraka klase K
2. Najmanja duljina opisa (NDO) skupa pravila je 64 bita veća od trenutno najmanje NDO
3. Stopa pogreške je veća od 50%

U svakoj iteraciji ove faze dodaje se novo pravilo u skup pravila. Pravila 1 i 3 su poprilično jednostavna, dok je pravilo 2 manje trivijalno i bazira se na teoriji informacije.

Princip najmanje duljine opisa je stav da je za skup podataka najbolja ona teorija koja minimizira veličinu teorije zbrojenu s količinom informacije potrebne da se opišu iznimke unutar teorije. Matematički ovo se može zapisati na sljedeći način:

$$NDO = DULJINA[IZNIMKE|TEORIJA] + DULJINA[TEORIJA]$$

Implementacijski, ovaj princip je ostvaren pomoću više metoda prikazanih na sljedećim isječcima koda (slike 6-8).

```
public static double getMDL(RuleSet ruleset, List<DataRow> list,
Map<String,Set<String>> attributes, String currentClass, String className) {
            if(ruleset.getRules().isEmpty()) {
                  Rule rule = new Rule(currentClass, className);
                  return ruleExceptionDL(list, rule);
            }
            double mdl = 0;
            for(Rule rule : ruleset.getRules()) {
                  mdl+=(ruleExceptionDL(list, rule)+ruleTheoryDL(list,
attributes, rule));
            }
            return mdl;
      }
```

**Slika 6: metoda `getMDL`**

Metoda `getMDL` računa najmanju duljinu opisa za argument `ruleset`. Pri tome koristi dvije ključne metode: `ruleExceptionDL` i `ruleTheoryDL`. Metoda `ruleExceptionDL` računa broj bitova potrebnih da se opišu iznimke koristeći sljedeću formulu:

$$\texttt{DUJINA\_IZNIMKI} = \log_2\binom{IP+LP}{LP} + \log_2\binom{IN+LN}{LN},$$

gdje IP predstavlja broj istinito pozitivnih uzoraka, a LP broj lažno pozitivnih uzoraka. Broj istinito i lažno negativnih označavamo sa IN, odnosno LN.

Metoda `ruleTheoryDL` računa broj bitova potrebnih da se opiše sama teorija. U našem slučaju to je duljina klasifikacijskih pravila. Duljina teorije računa se po sljedećoj formuli:

$$\texttt{DULJINA\_TEORIJE} = \frac{1}{2} * (k * \log_2\frac{1}{p} + (n-k) * \log_2\frac{1}{1-p} + kbitova),$$

gdje *k* predstavlja broj pravila u skupu pravila, *n* broj svih vrijednosti koje atributi mogu postići, *p* omjer *k/n*. Oznaka *kbitova* predstavlja broj bitova potrebnih da se broj prirodni broj *k* zapiše u binarnom obliku.

```
private static double ruleExceptionDL(List<DataRow> list, Rule rule) {
            int tp = Classification.getTruePositives(list, rule).size();
            int fp = Classification.getFalsePositives(list, rule).size();
            int tn = Classification.getTrueNegatives(list, rule).size();
            int fn = Classification.getFalseNegatives(list, rule).size();
            int P = Classification.getPositives(list, rule).size();
            int N = Classification.getNegatives(list, rule).size();


            return FastMath.log(2,
CombinatoricsUtils.binomialCoefficientDouble(tp+fp, fp))+
                        FastMath.log(2,
CombinatoricsUtils.binomialCoefficientDouble(tn+fn, fn));
      }
```

**Slika 7: Metoda `ruleExceptionDL`**

```
private static double ruleTheoryDL(List<DataRow> list, Map<String,Set<String>>
attributes, Rule rule) {
            double k = rule.getRuleSet().size();
            if(k==0)
                  return 0;
            double kbits = numberOfBits((int) k);
            double n = 0;
            for(String key : attributes.keySet()) {
                  n+=attributes.get(key).size();
            }
            double p = k/n;

            return 0.5*(k*FastMath.log(2, 1/p)+(n-k)*FastMath.log(2, 1/(1-
p))+kbits);
      }
```

**Slika 8: Metoda `ruleTheoryDL`**

Nakon što je jedan od tri uvjeta zaustavljanja ispunjen, RIPPER prelazi u fazu optimizacije. Prilikom faze optimizacije za svako do sada generirano pravilo, napravimo dva nova pravila: pravila $p_1$ i $p_2$. Pravilo $p_1$ je potpuno novo pravilo, dok pravilo $p_2$ napravimo tako da na postojeće pravilo pohlepno dodajemo svojstva tijelu pravila. Zatim slijedi faza obrezivanja. Nova pravila $p_1$ i $p_2$ uspoređujemo sa starim pravilom koristeći novu metriku: preciznost. Preciznost se definira kao omjer

zbroja apsolutnih frekvencija istinito pozitivnih i negativnih uzoraka i zbroja apsolutnih frekvencija pozitivnih i negativnih uzoraka, odnosno ukupnog broja uzoraka:

$$\texttt{PRECIZNOST(p)} = \frac{IP+IN}{P+N}$$

Pravilo koje ima najveću preciznost se zadržava, a druga dva se odbacuju. Sljedeća faza je faza brisanja, koja se sastoji od provjere jednog uvjeta. Ako postoje uzorci klase K koji nisu pokriveni generiranim pravilima za tu klasu, algoritam ponovo ulazi u fazu izgradnje. Nakon što je i ovaj uvjet zadovoljen, algoritam ulazi u posljednju fazu – fazu čišćenja. U toj fazi računa se duljina opisa cijelog skupa pravila, tj. teorije. Nakon toga redom se uklanjaju pravila i ponovo se računa duljina opisa. Sva pravila koja povećavaju duljinu opisa se brišu. Kao rezultat ove faze dobivamo skup pravila za klasu K. Prije nego što počnemo učiti pravila za sljedeću klasu, moramo izbrisati sve uzorke klase K iz skupa podataka za učenje.

## 5.3 Numeričke vrijednosti

Algoritam RIPPER radi isključivo s kategoričkim vrijednostima. Da bi ovaj algoritam funkcionirao nad numeričkim atributima, potrebno je atribute diskretizirati. Primijenjena je sljedeća jednostavna metoda diskretizacije:

- Izračunaj medijan vrijednosti atributa
- Vrijednosti manje od medijana pretvori u oblik `<medijan`
- Vrijednosti veće ili jednake medijanu pretvori u oblik `>=medijan`

## 5.4 Implementacija algoritma RIPPER

Algoritam RIPPER realiziran je razredom `RIPPER.java`. Skup pravila možemo naučiti pozivajući statičku metodu `getRules` koja kao argumente prima skup podataka za učenje i naziv klase.

```
public static List<RuleSet> getRules(DataSet dataSet, String className)
```

**Slika 9: Statička metoda `getRules`**

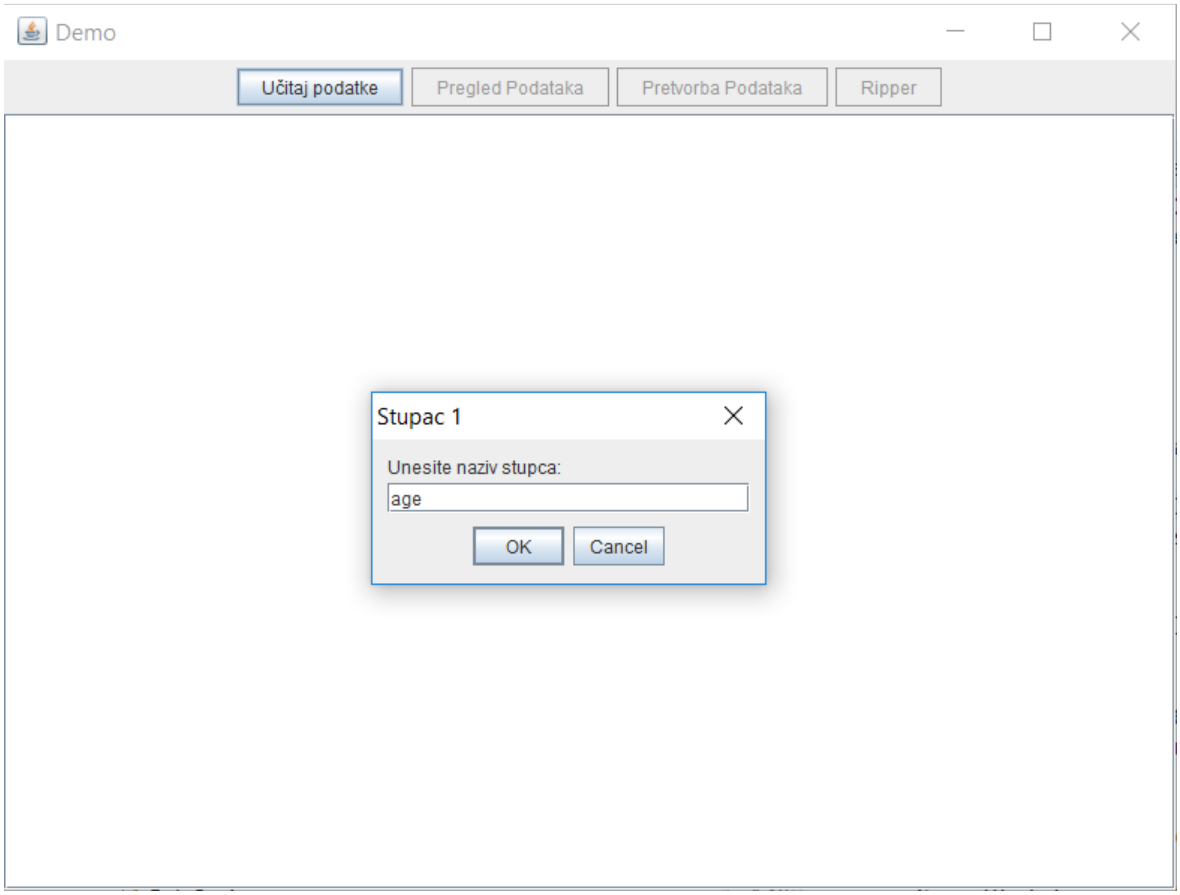
Razred `RIPPER.java` koristi brojne pomoćne metode i razrede. Najosnovniji razred `Rule.java` realizira klasifikacijsko pravilo. Kako RIPPER vraća skup

pravila potrebna je i klasa koja se sastoji od skupa pravila i metoda koje se odnose na njih. To je ostvareno u razredu `RuleSet.java`. Neke od zanimljivijih metoda ovoga razreda pokazane su u prethodnome potpoglavlju prilikom objašnjavanja principa najmanje duljine opisa. Metrike su realizirane metodama razreda `Evaluation.java`. Ulazni skup podataka realiziran je razredom `DataSet.java` koji kao člansku varijablu ima listu redova tablice, koji su realizirani razredom `DataRow.java`.

## 5.5. Grafičko sučelje

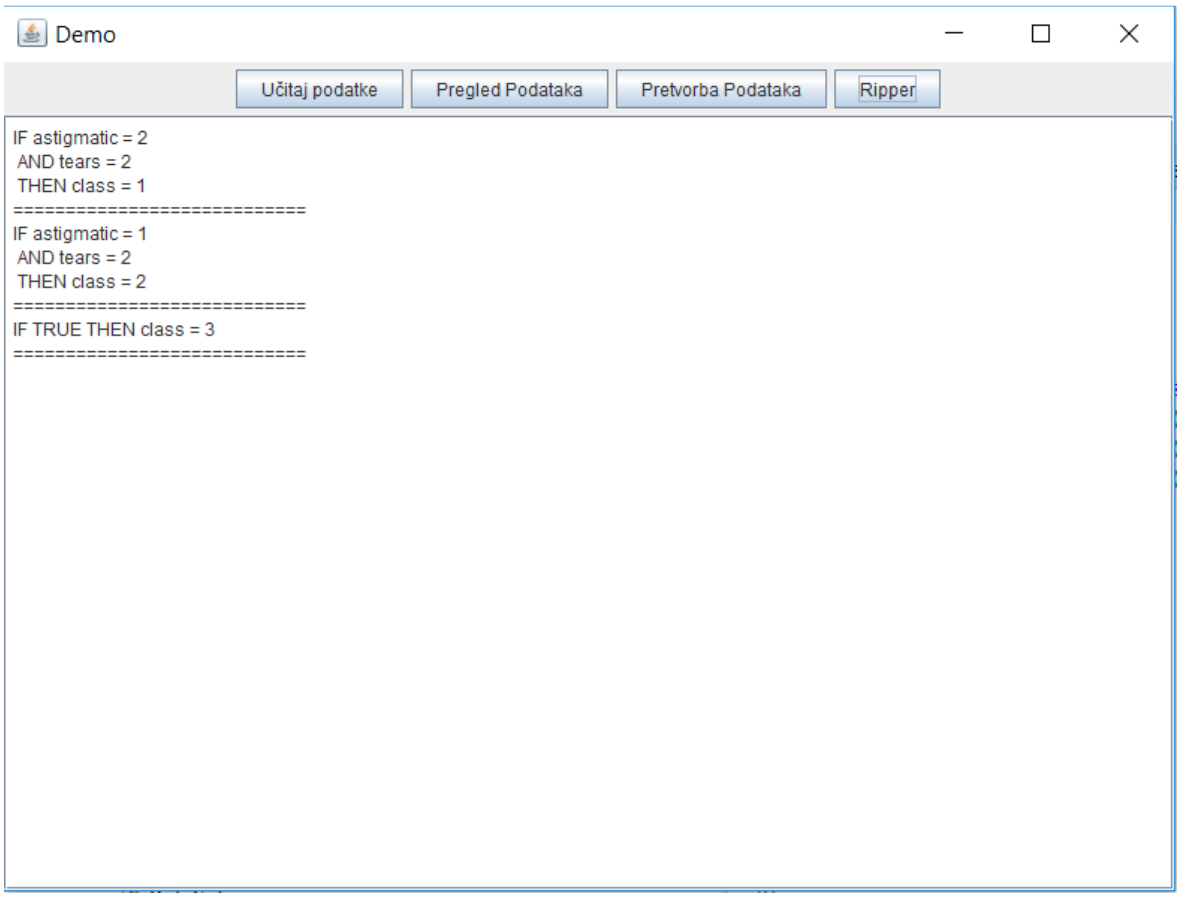
U sklopu ovog završnog rada napravljeno je i jednostavno grafičko sučelje za učenje klasifikacijskih pravila. Sučelje nam omogućuje unos podataka, pretvorbu podataka, jednostavni pregled podataka i učenje klasifikacijskih pravila.

Podaci se unose klikom na gumb „Učitaj Podatke“. Zatim se od korisnika traži da unese nazive stupaca, tj. nazive svih atributa. Aplikacija pretpostavlja da zadnji redak predstavlja klasu i samo tako formatirani podaci će se prihvatiti. Također podaci moraju biti odvojeni zarezom (.csv datoteka). Nakon učitavanja podataka, ostali gumbi postanu omogućeni.



**Slika 10: Grafičko sučelje – učitavanje podataka**

Klikom na gumb „Pregled Podataka“ prikazuje se prvih 10 redaka tablice. Gumb „Pretvorba Podataka“ služi nam za pretvaranje numeričkih atributa u oblik koji je pogodan algoritmu RIPPER. Posljednji gumb služi za učenje skupa pravila temeljem učitanog skupa podataka za učenje. Nakon što je algoritam završio s učenjem pravila, rezultati se ispisuju na sučelje. Treba napomenuti da ovo sučelje služi isključivo za demonstracijske svrhe i ne predstavlja pravi način korištenja razreda `RIPPER.java`.




**Slika 11: Grafičko sučelje – rezultat algoritma RIPPER**

# 6. Usporedba rezultata

Algoritam RIPPER implementiran je u brojnim programskim jezicima i paketima. Razred `JRip.java` unutar paketa `Weka.jar`, nastao na sveučilištu u Waikatu iz Novog Zelanda, služit će nam za usporedbu uspješnosti implementacije algoritma RIPPER.

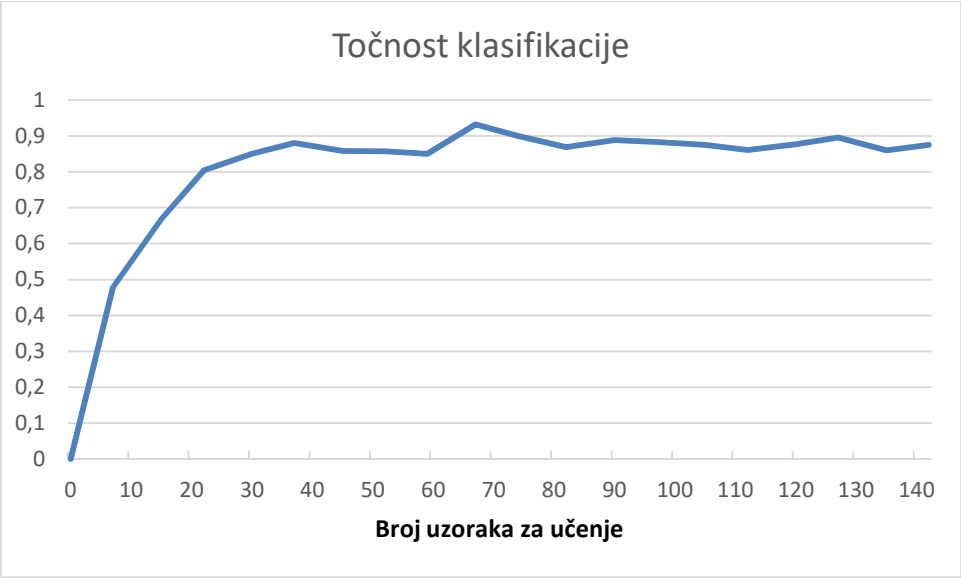
Našu implementaciju algoritma RIPPER prvo smo ispitivali tako da smo postupno povećavali skup za učenje. Nakon toga koristili smo postupak unakrsne validacije od 10 preklopa. Uspješnost klasifikacijskih pravila odredili smo koristeći razred `Analysis.java` koji skup podataka dijeli na skup za učenje i skup za testiranje, potom uči skup pravila i provjerava njihovu točnost. Za usporedbu smo koristili sljedeće skupove podataka: *Haberman*, *Lenses* i *Iris*.

## 6.1 Skup podataka *Iris*

Skup podataka Iris jedan je od najpopularnijih skupova u strojnome učenje i sadrži podatke o 3 različite vrste cvijeta Iris: *Setosa*, *Virginic*a i *Versicolor*. Atributi su sljedeći:

- Duljina čašičnog listića – numerički
- Širina čašičnog listića – numerički
- Duljina latice – numerički
- Širina latice – numerički
- Razred – kategorički

Naš algoritam prosječno je ispravno klasificirao 84% slučajeva, dok je implementacija u biblioteci `Weka.jar` bila ispravna u 90% slučajeva. Ispravnost klasifikacije smo odredili tako da smo postupno povećavali skup podataka za učenje i na kraju izračunali srednju vrijednost mjerenja. Uzorci su birani slučajnim odabirom. Prilikom analize drugih skupova podataka, koristit ćemo metodu unakrsne validacije od 10 preklopa. Treba napomenuti da je točnost algoritma znatno manja za prvih par mjerenja jer je korišteno malo uzoraka. Srednja vrijednost bez prva tri mjerenja daje prosječnu ispravnost algoritma od 88%.




**Slika 12: Rezultati analize skupa podataka *Iris***

## 6.2. Skup podataka *Haberman*

Skup podataka *Haberman* prikazuje smrtnost 306 pacijenata operiranih zbog raka dojke u sveučilišnoj bolnici u Chicagu. Podaci su prikupljani od 1958. do 1970. godine. Atributi su sljedeći:

- Dob za vrijeme operacije – numerički
- Godina operacije – numerički
- Broj detektiranih aksilarnih čvorova – numerički
- Ishod operacije – kategorički

Posljednji argument zapravo predstavlja klasu i moguća su dva ishoda: pacijent je umro unutar 5 godina nakon operacije i pacijent je ostao živ nakon 5 ili više godina nakon operacije. Rezultati se vide na grafu na Slici 13. Točnost klasifikacije uglavnom se kreće oko 90%. Prosječna točnost, ne računajući slučaj kada je bilo 0 podataka za učenje je 88%. Skup podataka za učenje bira se slučajnim izborom iz skupa svih uzoraka. Koristeći metodu unakrsne validacije od 10 preklopa dobivamo prosječnu preciznost od 74%, a koristeći biblioteku `Weka.jar`, za isti skup podataka točno je klasificirano 77% uzoraka.


**Slika 13: Rezultati analize skupa podataka *Haberman***

## 6.3 Skup podataka *Lenses*

Skup podataka *Lenses* pokazuje 24 pacijenta s problemima vida. Atributi su sljedeći:

- Dob – kategorički
- Poremećaj vida – kategorički
- Astigmatičnost – kategorički
- Proizvodnja suza – kategorički
- Tip leća – kategorički

Posljednji atribut predstavlja klasu. Analizom dobili smo prosječnu točnost od 71%, uz pripadajući graf prikazan na Slici 14. Niska vrijednost prosječne točnosti rezultat je činjenice da je podataka za učenje u prvih 10 slučajeva bilo premalo i da bi te slučajeve trebalo zanemariti ako želimo dobiti pravu sliku o kvaliteti algoritma. Ako to učinimo točnost algoritma je 86% što je puno bolje. Koristeći metodu unakrsne validacije od 10 preklopa dobivamo točnost algoritma od 80%, dok za isti skup Weka.jar generira skup pravila uz točnost od 92%.


**Slika 14: Rezultati analize skupa podatak *Lenses***

# 7. Zaključak

Klasifikacija predstavlja jednu od ključnih grana u strojnome učenju. Klasifikacijska pravila samo su jedna od brojnih metoda koji rješavaju problem klasifikacije. Prednost klasifikacijskih pravila je činjenica da vraća rezultat koji je jasan čovjeku. Za razliku od drugih klasifikatora gdje se uspješnost klasifikacije mjeri preciznošću, klasifikacijska pravila unose i subjektivnu procjenu uspješnosti. Naime bolji su klasifikatori koji vraćaju manji skup pravila, jer takvi imaju veću praktičnu primjenu.

Trenutno jedan od najpopularnijih algoritama za učenje klasifikacijskih pravila je algoritam RIPPER. Za razliku od nekih jednostavnijih algoritama, RIPPER vraća skup pravila za svaku klasu. RIPPER se pokazao kao prilično uspješan na različitim skupovima podataka.

Osim što smo objasnili kako funkcionira algoritam RIPPER, implementirali smo ga u programskom jeziku Java. Naša implementacija se pokazala malo lošijom od one unutar biblioteke `Weka.jar`, no i dalje je preciznost prilično visoka. To možemo objasniti činjenicom da `Weka.jar` koristi složenije metode diskretizacije. Također nedostaci naše implementacije su nemogućnost obrade podataka sa nedostajućim vrijednostima i zaglavljivanje algoritma u beskonačnoj petlji prilikom korištenja velikih skupova podataka. Algoritam možemo poboljšati primjenom jedne od boljih metoda diskretizacije, primjerice metodom diskretizacije temeljene na entropiji. Prilikom testiranja nismo imali nedostajuće vrijednosti, ali ukoliko želimo bolji algoritam treba obratiti pozornost i na te slučajeve. Problem zaglavljivanja u beskonačnoj petlji možemo riješiti uporabom razred `BigDecimal.java`.

Algoritme smo testirali nad skupovima podataka preuzetih sa stranica Sveučilišta u Irvineu u Kaliforniji. Preuzeti podaci dolazili su u različitim oblicima, pa smo ih morali pretvoriti u zajednički oblik. Također preuzete podatke morali smo pretvoriti u .arff kako bi mogli ispitati algoritam implementiran unutar biblioteke `Weka.jar`.

Napravljeno je i jednostavno grafičko sučelje, koje omogućuje unos, prikaz i pretvorbu podataka. Također kroz sučelje se može pokrenuti algoritam RIPPER, koji na sučelju prikazuje naučena pravila.

Strojno učenje relativno je novo znanstveno područje. Primjerice algoritam RIPPER nastao je 1995. godine i zapravo je bio poboljšanje prijašnjeg algoritma IREP. Upravo nam to daje nadu da su nova otkrića na ovome području moguća i da je realno očekivati ih nekoj bližoj budućnosti.

# 8. Literatura


1. Hühn J; Hüllermeier: "Furia: An Algorithm For Unordered Fuzzy Rule Induction", Data Mining and Knowledge Discovery Vol. 19, No. 3, (2009), 293-319
2. Cendrowska J: "PRISM: An algorithm for inducing modular rules", International Journal of Man-Machine Studies Vol. 27, No. 4 (1987), 349-370
3. Dain O; Cunningham R. K; Boyer S: "IREP++: a Faster Rule Learning Algortihm", Fourth SIAM International Conference on Data Mining, 2003
4. Cohen W. W: "Fast Effective Rule Induction", Twelfth International Conference on Machine Learning, 1995
5. Witten, I. H; Frank E; Hall M. A: "Data Mining: Practical machine learning tools and techniques", Morgan Kaufmann, Burlington, USA, 2011
6. Fürnkranz, J; Gamberger D; Lavrač N: "Foundation of Rule Learning", Springer-Verlag, Berlin, Germany, 2012
7. Hansen M. H; Yu B: "Model Selection and the Principle of Minimum Description Length", Journal of the American Statistical Association Vol. 96, No. 454 (2001), 746-774

## Postupci za učenje klasifikacijskih pravila

Klasifikacijska pravila služe za klasifikaciju uzoraka, pri čemu model klasifikacije ostaje jasan čovjeku. U ovome radu objašnjena je klasifikacija, kao i klasifikacijska pravila. Obrađen je i algoritam klasifikacije RIPPER, teoretski ali i implementacijski. Implementirani algoritam uspoređen je s postojećim algoritmom dostupnim u biblioteci `Weka.jar`. Također, napravljeno je jednostavno grafičko sučelje koje omogućuje unos i obradu podataka.

## Ključne riječi:

Klasifikacija, klasifikacijska pravila, RIPPER, skup podataka

## Classification Rules Learning Methods

Classification rules are used to classify instances, while classification model remains clear to human. In this paper, classification and classification rules are explained. Algorithm RIPPER was first explained theoretically, then it was implemented. Our version of RIPPER was compared to existing implementation from library `Weka.jar`. A simple graphical user interface for data importing and processing was also created.

## Key words:

Classification, Classification rules, RIPPER, data set