SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4330

Rješavanje klasifikacijskih problema evolucijom stabala odluke

Vinko Kolobara

Zagreb, lipanj 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 9. ožujka 2016.

ZAVRŠNI ZADATAK br. 4330

Pristupnik: **Vinko Kolobara (0036475679)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Rješavanje klasifikacijskih problema evolucijom stabala odluke**

Opis zadatka:

Opisati problem klasifikacije i postojeće algoritme rješavanja, uz naglasak na algoritme stabla odluke. Istražiti mogućnosti primjene genetskog programiranja u problemima klasifikacije. Ostvariti programski sustav za evoluciju klasifikatora u okviru postojećeg okruženja za evolucijsko računanje. Posebnu pažnju posvetiti algoritmima klasifikacije u obliku stabla odluke. Usporediti učinkovitost ostvarenog sustava s obzirom na postojeće algoritme na dostupnim ispitnim primjerima. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Zadatak uručen pristupniku: 18. ožujka 2016.
Rok za predaju rada: 17. lipnja 2016.

Mentor:

Izv. prof. dr. sc. Domagoj Jakobović

Predsjednik odbora za završni rad modula:

Prof. dr. sc. Siniša Srbljić

Djelovođa:

Doc. dr. sc. Tomislav Hrkać

SADRŽAJ

1. Uvod

Kao što znamo, učenje je prikupljanje znanja ili vještina kroz promatranje, iskustvo. Još od nastanka računala, zanimljiva ideja je bila omogućiti i računalu da razmišlja i uči na sličan način kao i mi. Iz toga se razvilo područje umjetne inteligencije, a specifičnije i strojnog učenja. Posebno zanimljivi problemi strojnog učenja su problemi klasifikacije koji su u ovom radu rješavani pomoću genetskog programiranja i stabala odluke. Za provjeru uspješnosti takve metode korišteni su neki popularni skupovi podataka[4].

U ovom radu su opisane vrste strojnog učenja, problemi koji se pojavljuju u strojnom učenju te kako ih pokušati riješiti unakrsnom provjerom. Nakon toga je detaljnije opisana klasifikacija kao podvrsta nadziranog učenja te načini rješavanja. Nadalje slijedi opis stabala odluke, genetskog programiranja i primjene genetskog programiranja na stabla odluke te metode evaluacije takvog modela. Na kraju rada je opisana programska implementacija i rezultati provođenih eksperimenata.

2. Strojno učenje

2.1. Definicija

Strojno učenje možemo definirati kao područje koje omogućava računalima učenje bez eksplicitnog programiranja[10] i to pružajući računalu samo primjere kako bi se neki zadatak trebao rješavati. To omogućuje računalu učenje slično ljudskom, temeljeno na iskustvu i dostupnim informacijama. Recimo da trebamo odrediti je li neki *mail spam*. Računalu na raspolaganju stoji sadržaj *maila*, ali i prethodni *mailovi* koji su klasificirani kao *spam* ili *ne-spam*. Nakon postupka učenja na već poznatim *mailovima*, računalo bi trebalo moći i za svaki novi primjer ispravno odrediti je li *spam* ili ne.

Strojnim učenjem je moguće rješavati širok raspon problema. Može se reći da program uči iz iskustva E u odnosu na zadatak T i mjeru sposobnosti P, ako se njegova sposobnost na zadatak T, mjerena s P, poboljšava s iskustvom E[6]. Za neki problem koji mislimo rješavati strojnim učenjem, moramo definirati zadatak problema, mjeru sposobnosti koja se maksimizira, i iskustvo. Tako bi već navedeni primjer detektiranja *spama* mogli definirati kao:

- zadatak T: određivanje je li neki *mail spam*
- mjera sposobnosti P: postotak točno određenih *mailova*
- iskustvo E: prethodno poznati primjeri i učenje na njima

2.2. Podatci za učenje

Kada rješavamo neki problem strojnim učenjem, nalazimo se u okolini gdje imamo pristup različitim informacijama. Možemo biti u nekakvoj sobi i promatrati što se sve tu nalazi, čuti zvukove, možemo pri prodaji kuće vidjeti kolike su cijene drugih kuća i prilagoditi i svoju cijenu. Kao što vidimo, informacija je mnogo i u postupku učenja se trebaju nekako iskoristiti.

U problemima strojnog učenja imamo skup za učenje (engl. *training set*) koji se

sastoji od ulaznih podataka $X_1, ..., X_n$ i može, ali ne mora sadržavati i odgovarajuće izlazne podatke $Y_1, ..., Y_n$ (izuzetak je podržano učenje koje je objašnjeno kasnije) gdje je X_i vektor, a Y_i uglavnom neka vrijednost. Vektor X_i obično zovemo značajke (engl. *features*), dok je Y_i oznaka (engl. *labels*).

Svaki član vektora značajki X_i predstavlja neku značajku problema kojeg rješavamo. Za primjer *spama*, to bi mogle biti: duljina poruke, gramatičke pogreške, postojanje ključnih riječi koje sugeriraju na *spam*....

Svaki od izlaznih podataka Y_i pripada nekoj vrijednosti koja može biti kontinuirana ili iz nekog diskretnog skupa. Za primjer *spama*, moguće vrijednosti za Y_i bi bile $\{spam, ne - spam\}$.

Cilj učenja je dobiti model koji se zna snaći i na skupu za učenje, ali i na novim, dosad neviđenim, primjerima.

2.3. Vrste učenja

Strojno učenje možemo podijeliti na tri velike cjeline:

- nadzirano učenje (engl. *supervised learning*)
- nenadzirano učenje (engl. *unsupervised learning*)
- podržano učenje (engl. *reinforcement learning*)

2.3.1. Nadzirano učenje

Nadzirano učenje se koristi kada imamo neki skup za učenje koji se sastoji i od ulaznih podataka $X_1, ..., X_n$ i od odgovarajućih izlaznih podataka $Y_1, ..., Y_n$. Na tom skupu za učenje provodimo algoritam učenja. Rezultat je model sposoban točno odrediti vrijednosti na skupu za učenje, ali i za neke nove, potpuno nepoznate podatke. Koristi se u prepoznavanju uzoraka, rukopisa, govora, detekciji *spama*...

2.3.2. Podržano učenje

Podržano učenje je nešto drugačije od druge dvije vrste. Ovdje imamo neku okolinu u kojoj se program nalazi. Nema nikakve otprije poznate podatke, nema skupa za učenje i cilj mu je maksimizirati nekakvu nagradu u toj okolini. Učenje se provodi tako što svaka akcija programa uzrokuje nekakav odgovor od okoline koji može biti pozitivan (nešto dobro radi), negativan (nešto je loše uradio) ili neutralan (nije ni dobro ni loše). Temeljem tih podražaja iz okoline uči se ponašati u toj okolini i pokušava

maksimizirati pozitivne podražaje. Jedan od primjera je učenje autonomnog vozača gdje temeljem akcija (ubrzanje, usporavanje, skretanje) iz okoline (staze na kojoj se nalazi) dobiva pozitivan (ako napreduje ispravno do cilja), negativan (ako ide unazad, skrenuo izvan staze) ili neutralan podražaj.

2.3.3. Nenadzirano učenje

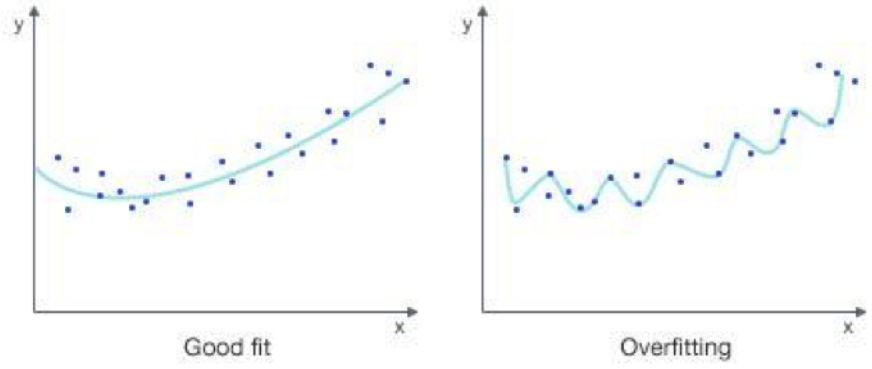
Nenadzirano učenje se koristi kada imamo skup za učenje koji se sastoji samo od ulaznih podataka $X_1, ..., X_n$, dok izlazne podatke ne znamo. Provodi se algoritam nenadziranog učenja na tom skupu i cilj mu je pronaći sličnosti među podatcima i ovisno o tome ih grupirati u različite skupine. Na koncu bi model naučen nenadziranim učenjem trebao moći neke nove ulazne podatke grupirati ispravno. Također se može koristiti u prepoznavanju uzoraka, uz razliku što ovdje ne zna točno što koji podatak predstavlja već ih samo grupira po sličnosti.

2.4. Problemi strojnog učenja

Osnovni problemi koji se pojavljuju kod strojnog učenja su prekomjerna generalizacija (engl. *overgeneralization*) i prekomjerna specijalizacija (engl. *overfitting*).

Prekomjerna specijalizacija se odnosi na pogrešnu interpretaciju skupa za učenje što uzrokuje pretjeranu "opsjednutost" određenim slučajnim značajkama koje u općenitoj situaciji nemaju previše smisla i takav model se loše ponaša za nove vrijednosti.

Prekomjerna generalizacija se odnosi na stvaranje preopćenitog modela koji bi čak i neke baš specifične vrijednosti deklarirao kao općenitu.



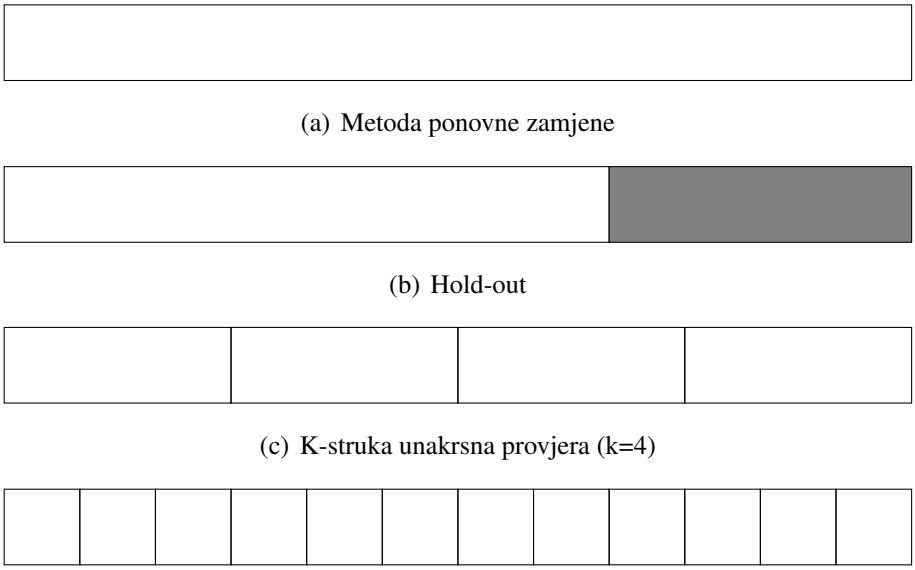
Slika 2.1: Primjer prekomjerne specijalizacije

Jedan način rješavanja tih problema je unakrsna provjera (engl. *crossvalidation*).

2.4.1. Unakrsna provjera

Unakrsna provjera služi za usporedbu i evaluiranje algoritama za učenje. Radi se tako što podatke za učenje dijeli na skup za učenje (engl. *training set*) i skup za provjeru (engl. *validation set*). Osnovne vrste unakrsne provjere su:

- Metoda ponovne zamjene (engl. *resubstitution*)
- Hold-out
- K-struka unakrsna provjera (engl. *K-fold*)
- Leave-One-Out



(a) Metoda ponovne zamjene

(b) Hold-out

(c) K-struka unakrsna provjera (k=4)

(d) Leave-One-Out (n = k = 12)

Slika 2.2: Prikaz metoda unakrsne provjere

Metoda ponovne zamjene

Metoda ponovne zamjene se sastoji od učenja na cijelom skupu za učenje nakon čega se opet na cijelom skupu vrši provjera. Često dolazi do prekomjerne specijalizacije, što znači da ova metoda nije baš najsretnije rješenje.

Hold-out

Ova metoda razdvaja izvorni skup za učenje na novi skup za učenje i skup za provjeru. Onda, tijekom izvođenja algoritma, uči se samo na novom skupu za učenje, a skup za provjeru se koristi kao mjera generalizacije algoritma. Loša stvar pri korištenju ove

metode je velika ovisnost o podjeli izvornog skupa za učenje. Ako ta podjela ne bude dobro napravljena, algoritam za učenje neće pokazati dobre rezultate.

K-struka unakrsna provjera

Ideja ove metode je podjela izvornog skupa za učenje na k različitih podskupova otprilike jednake veličine. Onda se u svakoj od k iteracija za skup za provjeru odabere jedan od tih podskupova, a ostalih $k - 1$ služe za učenje. Konačna mjera se uzima kao prosjek tih k iteracija.

Leave-One-Out

Ovo je posebni slučaj k-struke unakrsne provjere gdje je k jednak broju n podataka u skupu za učenje. Tako se u svakoj od n iteracija algoritam uči na $n - 1$ podataka, a provjerava na 1 izdvojenom podatku. Uglavnom se koristi kod malog broja podataka za učenje.

3. Klasifikacija

3.1. Definicija

Klasifikacija općenito ima dva značenja. Može se odnositi na postupak u kojem iz nekih podataka trebamo pronaći određene sličnosti i iz toga izvući razrede za podatke. Druga vrsta je kada znamo koji su mogući razredi u koje možemo svrstati podatke i cilj nam je ulazne podatke svrstati u odgovarajuće razrede. U strojnom učenju se prva vrsta uglavnom zove grupiranje (engl. *clustering*) i odnosi se na nenadzirano učenje dok se druga vrsta zove klasifikacija i podvrsta je nadziranog učenja. U daljnjem tekstu s riječi klasifikacija govorit će se o drugoj vrsti.

Formalno, klasifikaciju možemo definirati na sljedeći način: neka je X skup svih mogućih ulaznih podataka za određeni problem. Neka je Y skup svih mogućih razreda $Y = \{y_1, y_2, ...y_n\}$. Neka je D skup podataka za učenje. Korištenjem algoritma za učenje cilj je naučiti klasifikator h na skupu za učenje D ispravno dodijeliti razred iz Y svim podatcima iz X:

$$h : X \to Y$$

3.2. Algoritmi za klasifikaciju

Postoji velik broj algoritama koji se koriste u klasifikaciji podataka, od kojih su najčešće korišteni:

- Vjerojatnosni algoritmi (engl. *Probabilistic*)
- Algoritmi zasnovani na pravilima (engl. *Rule-Based*)
- Učenje zasnovano na slučajevima (engl. *Instance-based learning*)
- SVM (Support vector machine)
- Neuronske mreže
- Stabla odluke (engl. *Decision trees*)

Vjerojatnosni algoritmi

Vjerojatnosni algoritmi koriste statističko zaključivanje za pronalažanje najboljeg razreda koji odgovara određenom primjeru. Ova vrsta algoritma svim razredima dodjeljuje vjerojatnost za svaki pojedini primjer, a konačno dodijeljeni razred je onaj kojem pripada najveća vjerojatnost za neki primjer.

Slika 3.1: Primjer prepoznavanja znamenki

Na slici 3.1 možemo vidjeti primjer rukom crtanog broja. Za ovakav problem su razredi brojevi od 0 do 9. Na tablici 3.1 možemo vidjeti jedan primjer dodijeljenih vjerojatnosti za svaki pojedini razred. Razred dodijeljen tom primjeru bit će onaj s najvećom vjerojatnosti, tj. 8.

Tablica 3.1: Vjerojatnosti za pojedine razrede

razred i	0	1	2	3	4	5	6	7	8	9
p_i	0.09	0.04	0.20	0.31	0.05	0.15	0.25	0.07	0.94	0.27

Algoritmi zasnovani na pravilima

Kao što i ime govori, algoritmi zasnovani na pravilima, koriste određena pravila za određivanje razreda. Jedan problem koji se često pojavljuje su konflikti među pojedinim pravilima pa je potreban i način razrješavanja konflikata. Obično se koristi redoslijed unosa pravila, pravilo prije napisano ima veći prioritet.

```
R1: if broj_bodova_labos<15 then prolaz=false
R2: if broj_bodova_kontinuirano>50 then prolaz=true
```

Recimo da imamo iznad navedena pravila. Ako bi student imao

$$broj\_bodova\_kontinuirano = 65$$

$$broj\_bodova\_labos = 14.5$$

aktiviralo bi se prvo pravilo i nažalost ne bi položio predmet.

Učenje zasnovano na slučajevima

Učenje zasnovano na slučajevima se temelji na uspoređivanju neviđenog primjera sa skupom za učenje kojeg već ima u memoriji. Otud potiče i drugi naziv - Memory-based learning.

Primjer takvog algoritma je k najbližih susjeda. Za svaki ispitni slučaj, nađe njegovih k najbližih susjeda (mogu biti najbliži po udaljenosti ili nekom drugom kriteriju), i ispitnom slučaju će biti dodijeljen najčešće pojavljivani razred u odabranim susjedima.

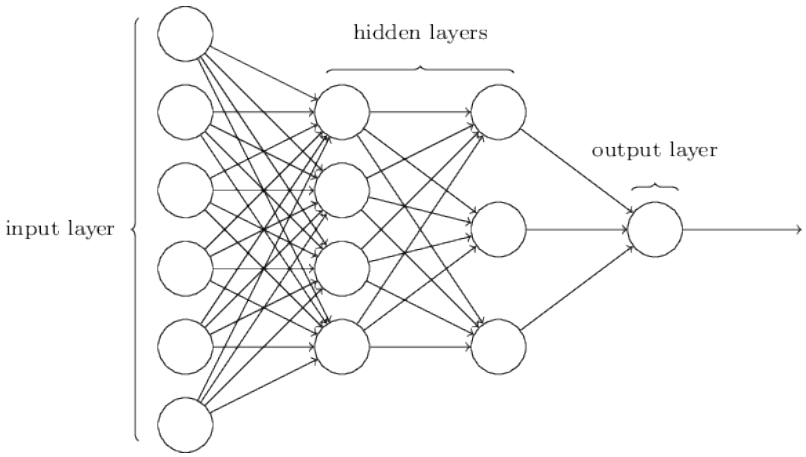
Support vector machine

SVM se uglavnom koristi u binarnim klasifikacijskim problemima. Cilj je razdvojiti dvije klase s jednom linearnom kombinacijom značajki. Zbog takvog pristupa, važno je tu podjelu napraviti pažljivo.

Problem ovih algoritama je velika sporost, ali zato su vrlo efikasni i precizni u rješavanju mnogih problema.

Neuronske mreže

Inspiracija neuronskim mrežama je ljudski mozak i njegov način funkcioniranja. Neuronske mreže se sastoje od međusobno povezanih neurona. Dijele se na ulazni sloj, skrivene slojeve i izlazni sloj. U ulazni sloj ulaze ulazni podatci, tj. informacije iz okoline (značajke), a izlazni sloj vraća odgovarajuću vrijednost razreda. Skriveni slojevi služe za dodatna računanja.



Slika 3.2: Umjetna neuronska mreža

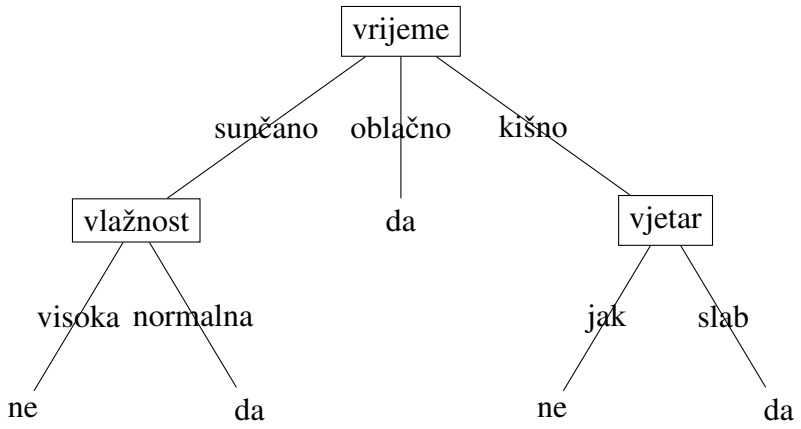
Funkcionira tako što svaka veza ima određenu težinu w_i koja se množi za vrijednošću ishodišnog čvora veze, i suma svih tih vrijednosti čini vrijednost sljedećeg čvora.

Uobičajeno se suma tih vrijednosti još provuče kroz neku funkciju kako bi dobili prihvatljivije vrijednosti.

4. Stabla odluke

Stabla odluke i učenje stablima odluke je jedan od najčešće korištenih algoritama za zaključivanje. Koristi se od medicinske dijagnostike pa sve do provjere kreditne sposobnosti.

4.1. Prikaz stabla odluke



Slika 4.1: Stablo odluke za problem igranja tenisa

Stablo odluke se sastoji od čvorova, grana i listova. Svaki čvor predstavlja jednu značajku rješavanog problema, u slučaju slike 4.1 to bi bili vrijeme, vlažnost i vjetar. Grane koje idu iz čvorova predstavljaju svaku moguću vrijednost koju taj čvor može poprimiti i u ovisnosti o vrijednosti tog atributa za određeni ispitni slučaj, putujemo kroz stablo određenom granom. Na kraju se nalaze listovi koji ujedno predstavljaju i razrede na koje možemo klasificirati određene ulazne podatke. Npr. u slučaju da je vrijeme = kišno, vlažnost = visoka, vjetar = slab; izvela bi se krajnja desna grana i došli bi do krajnjeg desnog lista, tj. razreda Da.

4.2. Izgradnja stabla odluke

Kao što se moglo vidjeti, nakon izgradnje stabla odluke, nije teško odrediti kojem razredu pripadaju određeni ulazni podatci. Pravi problem se nalazi u izgradnji stabla odluke, tj. u algoritmima za stabla odluke. Najčešće korišteni su:

- ID3
- C4.5

ID3

Algoritam ID3 stvara stabla odluke od vrha prema dnu. Korištenjem statističkih metoda, utvrđuje koja značajka bi trebala postati korijen stabla. Nakon toga, stvara novu granu za svaku od mogućih vrijednosti značajke, i za svaki novonastali čvor ponovno određuje koju značajku najbolje postaviti na to mjesto, i tako dok ne iscrpi sve značajke. Ovaj algoritam je pohlepan, moguće je ne pronaći optimalno rješenje (koje svakom ulaznom podatku dodijeli ispravan razred).

C4.5

C4.5 je proširenje algoritma ID3 koje rješava određene probleme koje je ID3 imao:

- Nemogućnost korištenja kontinuiranih značajki
- Različite težine pojedinih značajki, neka značajka može biti vrijednija za određeni problem od ostalih
- Podrezavanje stabla
- Nepostojanje određene vrijednosti u skupu za učenje

Koraci su slični kao i kod ID3 algoritma, uz popravljanje već navedenih problema.

4.3. Primjena genetskog programiranja na stabla odluke

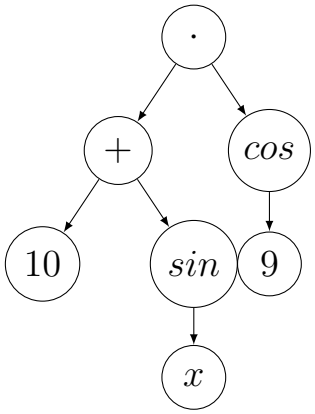
4.3.1. Genetsko programiranje

Genetsko programiranje je evolucijski algoritam koji stvara programe stablaste strukture koji rješavaju određeni problem. Nisu mu potrebna predznanja o problemu, ili znanstvenim područjima vezanim za taj problem.

Kao algoritam inspiriran prirodom, genetsko programiranje je nasumičan proces koji neće uvijek pronaći dobro rješenje. Ali ipak, u velikom broju područja i problema je efikasan i često se koristi.

Prikaz algoritma genetskog programiranja

Programi koji nastaju genetskim programiranjem uglavnom se prikazuju kao stabla čiji su čvorovi funkcije koje se mogu izvesti, a listovi vrijednosti ili varijable.



Slika 4.2: Primjer GP stabla, izvodi funkciju $(10 + sin(x)) \cdot cos(9)$

Program kao na slici 4.2 izvodi izračun $(10 + sin(x)) \cdot cos(9)$. Funkcije su $*$, $+$, sin, cos, neke od njih primaju 2 argumenta $(\cdot, +)$, a neke 1 (cos, sin). Listovi su vrijednosti $10, 9$ i varijabla x.

Izvođenje ovakvog programa je jednostavno. Krenuvši od korijenskog čvora, svaku funkciju izvodimo, a vrijednosti listova predajemo funkcijama. Tako bi tijek izvođenja programa 4.2 bio sljedeći:

Izvedi korijenski čvor $\cdot$, odredi vrijednost djece
- Izvedi čvor $+$, odredi vrijednost djece
  - Izvedi čvor 10, vrati vrijednost
  - Izvedi čvor sin, odredi vrijednost djeteta
    - Izvedi čvor x, vrati vrijednost
  - Vrijednost čvora sin je $sin(x)$
- Vrijednost čvora $+$ je $10 + sin(x)$
- Izvedi čvor cos, odredi vrijednost djeteta
  - Izvedi čvor 9, vrati vrijednost
- Vrijednost čvora cos je $cos(9)$

Vrijednost korjenskog čvora je $(10 + sin(x)) \cdot cos(9)$

Algoritam genetskog programiranja

Algorithm 1 Genetsko programiranje

populacija := slučajno generiraj populaciju programa koji rješavaju problem
repeat
  Izvedi svaki program i dodijeli mu sposobnost (engl. *Fitness*)
  Odaberi roditelje iz populacije koji će sudjelovati u reprodukciji
  Stvori dijete roditelja korištenjem genetskih operatora
until Nije pronađeno rješenje ili neki drugi uvjet zaustavljanja (npr. maksimalni broj generacija
return Najbolje rješenje

U algoritmu 1 možemo primijetiti neke nejasnoće. Što uzeti za mjeru sposobnosti programa, kako odabrati roditelje, kako stvoriti dijete, kako stvoriti početnu populaciju? Ta pitanja su vrlo važna u genetskom programiranju i potrebno je dobro odgovoriti na njih kako bi se algoritam ponašao očekivano.

Stvaranje početne populacije

Uobičajene metode stvaranja početne populacije su *full*, *grow* i *ramped-half-and-half*.

Grow metoda stvara nasumično stablo čija dubina može biti bilo koji broj manji od maksimalne dubine m. Počevši od korjenskog čvora, za svaki čvor odabire nasumično je li funkcija ili završni znak. Ako je funkcija, dodijeli mu nasumično jednu od mogućih funkcija, ako je završni znak dodijeli mu nasumično jedan od završnih znakova.

Full metoda stvara nasumično stablo čija je dubina jednaka maksimalnoj dubini m. Počevši od korjenskog čvora, svaki čvor koji je na dubini manjoj od m je funkcijski čvor i dodjeljuje mu se nasumično odabrana funkcija, a čvorovi na dubini m su završni znakovi.

Ramped-half-and-half metoda kombinira grow i full metode te stvara otprilike pola populacije grow metodom, a pola populacije full metodom.

Genetski operatori

Genetski operatori su ključ pravilnog izvođenja algoritma genetskog programiranja. Mogući operatori su:

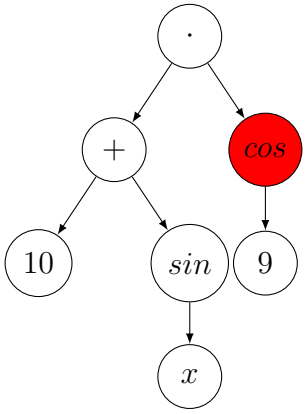
- Selekcija (engl. *Selection*),

– Križanje (engl. *Crossover*),

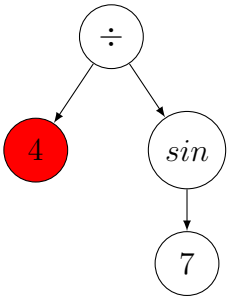
– Mutacija (engl. *Mutation*)

Selekcija se odnosi na odabir roditelja iz populacije. Taj odabir se radi pomoću prethodno utvrđene mjere sposobnosti (engl. *Fitness*) pojedine jedinke. Postoje različite metode selekcije, od kojih su najčešće korištene turnirska selekcija, gdje se pravi turnir od n jedinki i najbolja se bira za roditelja; proporcionalna selekcija, gdje se svakoj jedinki dodijeli vjerojatnost odabira temeljem mjere sposobnosti, i onda se slučajno odabere jedna od njih (jedinke s većom vjerojatnosti će vjerojatnije biti odabrane); elitistička selekcija, gdje se 1 ili više najboljih rješenja iz trenutne populacije izravno prebaci u sljedeću generaciju.

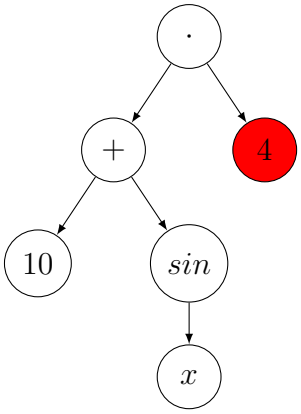
Križanje se odnosi na kombinaciju dvije jedinke i stvaranje nove jedinke, djeteta iz njih. Kao i u prirodi, kombinacijom dvije dobre jedinke, moguće je dobiti novu, bolju jedinku, što je i inspiracija za ovaj operator. Križanje se uglavnom izvodi tako da se odabere jedan ili više čvorova svakog roditelja i međusobno se zamjene, tako da se dobije novo stablo.



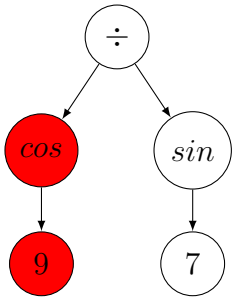
(a) Roditelj 1



(b) Roditelj 2

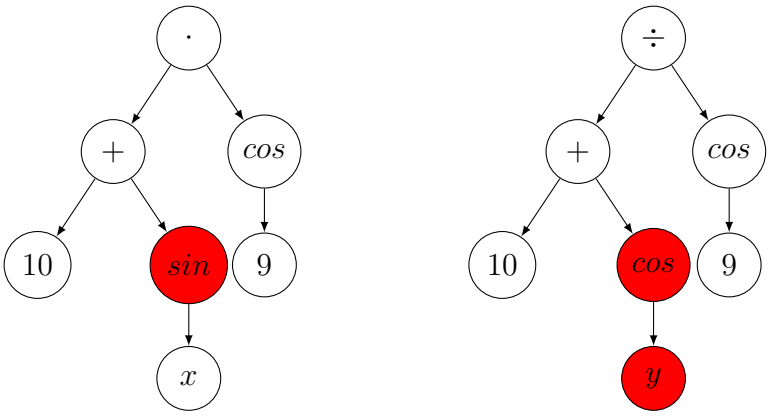


(c) Dijete 1



(d) Dijete 2

Mutacija se odnosi na mijenjanje neke jedinke. Također, prirodom inspirirana metoda, mutacijom i u prirodi se mogu dobiti zanimljive stvari. Postoji mnogo različitih vrsta mutacija: zamjena čvorova, gdje se odabere jedan čvor u stablu i zamijeni se sa nekom drugom funkcijom ako je na tom mjestu bila funkcija, ili drugim završnim znakom ako je na tom mjestu bio završni znak; mutiranje podstabla, gdje se odabere jedan čvor u stablu i stvori se novo podstablo na tom mjestu...



(e) Jedinka (f) Mutirana jedinka

Slika 4.3: Primjer mutacije

4.3.2. GP i stabla odluke

Genetsko programiranje radi sa stablima i to radi dobro. Stoga se pojavljuje jednostavna ideja povezivanja genetskog programiranja i stabala odluke. To se može ostvariti bez prevelikih promjena s obzirom na to da obje metode koriste isti zapis rješenja.

Postoje različite ideje ostvarivanja te veze. Jedna od najrasprostranjenijih je pomoću algoritama stabala odluke, stvoriti stablo odluke ili cijelu populaciju takvih stabala i to predati algoritmu genetskog programiranja kao početnu populaciju[8]. Ovakva metoda kombinira prednosti i algoritama stabala odluke i genetskog programiranja.

Ali možemo i zanemariti potpuno algoritme stabala odluke, i uraditi to samo sa genetskim programiranjem. U tom slučaju potrebno je definirati dodatne funkcije koje se mogu s tim nositi. Te funkcije bi morale moći raditi i s diskretnim i s kontinuiranim značajkama. Završni znakovi takvog stabla bi bili mogući razredi kojima neki ulazni podatak može pripadati. Još jedna dodatna promjena koja se uvodi je prekid izvođenja jedinke kada se dođe do završnog znaka, jer smo tada dobili odgovor na pitanje kojem razredu podatak pripada. Takav slučaj zanemaruje prednosti algoritama odluke, čak

ništa od toga ni ne koristi, nego pokušava samo genetskim operatorima razviti prihvatljivo rješenje.

4.3.3. Evaluacija klasifikatora

Za evaluaciju klasifikatora uglavnom se koristi matrica zbunjenosti (engl. *confusion matrix*). U binarnoj klasifikaciji znamo da kao vrijednost razreda možemo dobiti samo dvije vrijednosti, pozitivnu i negativnu. Tada se matrica zbunjenosti kreira tako da redovi označavaju pozitivni u negativnu vrijednost, a stupci klasifikatorovo predviđanje.

Tablica 4.1: Prikaz matrice zbunjenosti

	predviđena pozitivna	predviđena negativna
pozitivna	TP	FN
negativna	FP	TN

Matrica zbunjenosti za binarnu klasifikaciju prikazana je u tablici 4.1. Oznaka TP označava da je predviđena pozitivna vrijednost kada je bila i očekivana (engl. *True positive*). FN označava da je predviđena negativna vrijednost kada je bila očekivana pozitivna (engl. *False negative*). FP označava da je predviđena pozitivna vrijednost kada je očekivana negativna (engl. *False positive*). TN označava da je predviđena negativna vrijednost kada je bila i očekivana (engl. *True negative*).

Te nam oznake pomažu u izračunu mjere sposobnosti klasifikatora, a najčešće korištene su:

- točnost (engl. *accuracy*)
- pogreška (engl. *error*)
- osjetljivost (engl. *sensitivity*)
- specifičnost (engl. *specificity*)
- preciznost (engl. *precision*)
- omjer dijagnostičkih vjerojatnosti (engl. *Diagnostic odds ratio*) (DOR)
- f1 ocjena (engl. *f1 score*)

Točnost je možda najjednostavnija mjera. Označava koliki je postotak točno predviđenih podataka. Računa se kao

$$\frac{TP + TN}{TP + TN + FP + FN}$$

Pogreška je suprotnost točnosti, i označava koliki je postotak pogrešno predviđenih podataka. Računa se kao

$$\frac{FP + FN}{TP + TN + FP + FN}$$

ili kao

$$1 - accuracy$$

Osjetljivost govori koliki je postotak točno predviđenih istinitih vrijednosti od svih očekivanih istinitih vrijednosti. Računa se kao

$$\frac{TP}{TP + FN}$$

Ova mjera je dobra ako nam je važno točno predvititi što više istinitih vrijednosti. **Specifičnost** govori koliki je postotak točno predviđenih negativnih vrijednosti od svih očekivanih negativnih vrijednosti. Računa se kao

$$\frac{TN}{TN + FP}$$

Preciznost govori koliko je vjerojatno da je pozitivno predviđen rezultat zaista pozitivan. Računa se kao

$$\frac{TP}{TP + FP}$$

DOR govori koji je omjer vjerojatnosti točno predviđenih pozitivnih vrijednosti ako su i očekivane i predviđenih pozitivnih vrijednosti ako su očekivane negativne. Računa se kao:

$$\frac{\frac{TP}{FP}}{\frac{FN}{TN}}$$

F1 mjera je kombinacija preciznosti i osjetljivosti. Računa se kao

$$\frac{2 \cdot precision \cdot sensitivity}{precision + sensitivity}$$

Ovisno o problemu koji rješavamo, različite mjere nam bolje odgovaraju na pitanje što ustvari želimo od klasifikatora. U slučaju višerazredne klasifikacije, gdje nisu samo 2 razreda (pozitivni i negativni), također je moguće napraviti matricu zbunjenosti, i koristiti iste mjere uz neke izmjene.

Uobičajeno je napraviti matrice zbunjenosti za svaki razred posebno, izračunati mjeru sposobnosti tog razreda i na kraju kao konačnu mjeru uzeti prosjek mjera svih razreda

$$totalFit = \frac{\sum_{i=1}^{n} fit_i}{n}$$

Recimo da imamo zadatak klasificirati 300 živih bića u 3 carstva (gljive, biljke, ljudi). Mogući rezultati su prikazani na 4.2. Stupac predstavlja ispravan razred, dok

Tablica 4.2: Prikaz matrice zbunjenosti za problem više razreda

	gljiva	biljka	čovjek
gljiva	79	19	2
biljka	25	70	5
čovjek	1	1	98

retci predstavljaju razred određen klasifikatorom. Određena ćelija predstavlja broj tako određenih vrijednosti.

Za svaki razred odredimo posebno vrijednosti TP, TN, FP, FN. Za gljive bi te vrijednosti bile: $TP = 79$, $TN = (70 + 5) + (1 + 98)$, $FP = 25 + 1$, $FN = 19 + 2$. Neka koristimo pogrešku kao mjeru sposobnosti. Izračunamo $fit_{gljiva} = error(TP, TN, FP, FN) = \frac{26+21}{300} = 0.15667$. To uradimo za svaki razred i dobijemo $fit_{biljka} = 0.16667$, $fit_{covjek} = 0.03$. Konačno, ukupna pogreška je prosjek tih vrijednosti i dobijemo $\frac{0.15667+0.16667+0.03}{3} = 0.11778$. Analogno tome se računa i za svaku od ostalih mjera.

5. Programsko ostvarenje

5.1. ECF

ECF (Evolutionary Computation Framework) je programski okvir za jezik C++ koji služi za primjenu evolucijskog računanja. Sadrži razne algoritme, vrste reprezentacije genotipa, a omogućuje i paralelno izvođenje algoritama.

Radi tako što postoji konfiguracijska datoteka u koju se u zapišu potrebni parametri za izvođenje algoritma. Uz predefinirane parametre koje sadrži ECF, moguće je dodati i vlastite i tako proširiti ECF implementacijom potrebnih funkcija.

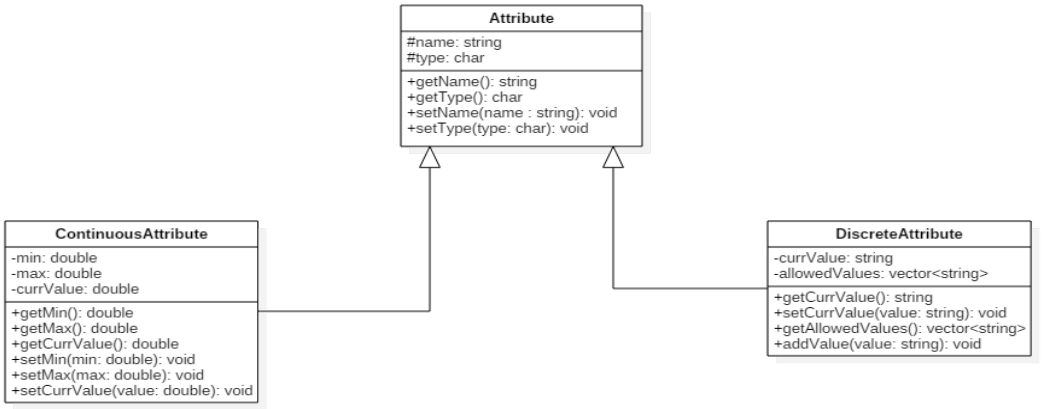
Iskorišteni su dijelovi koji se odnose na genetsko programiranje kako bi se napravio program koji pomoću jednostavnog genetskog programiranja primjenjenog na stabla odluke rješava problem klasifikacije.

5.2. Ulazni podatci

Format u kojem se očekuju ulazni podatci, tj. podatci za učenje i testiranje je CSV, tj. zarezom odvojene vrijednosti gdje zadnja vrijednost predstavlja razred. Putanja do datoteke sa skupom za učenje se zadaje parametrom "inputfile", a putanja do datoteke sa skupom za ispitivanje parametrom "testfile". Ako parametar "testfile" nije prisutan, uzima 30% skupa za učenje za ispitivanje, a preostalih 70% ostaje za učenje.

5.3. Značajke

Razred **Attribute** je razred koji predstavlja općenitu značajku. Stoga sadrži samo naziv značajke i tip. Taj razred nasljeđuju **ContinuousAttribute** koji predstavlja kontinuiranu značajku kojoj je potrebno minimalna vrijednost, maksimalna vrijednost i trenutna vrijednost, te **DiscreteAttribute** koji predstavlja diskretnu značajku kojoj su potrebne sve dozvoljene vrijednosti koje smije sadržavati te trenutna vrijednost. Ovi



Slika 5.1: Dijagram razreda za značajke

razredi služe za prikaz pojedinih značajki nekog problema koji se rješava.

Definicija značajki se vrši u posebnoj datoteci koja se također predaje kao parametar "attfile" konfiguracijskoj datoteci. Oblik datoteke sa značajkama je:

```
naziv_značajke:tip_značajke=[vrijednost1, vrijednost2]
```

Dakle, potrebno je definirati naziv značajke, tip značajke, koji može biti C za kontinuirane i D za diskretne, i vrijednosti značajke. Na vrijednosti kod kontinuiranih značajki se misli na minimalnu i maksimalnu moguću vrijednost koju ta značajka može poprimiti. Ako se ostavi prazno ([]), za raspon se uzima raspon $[-500000, 500000]$. Vrijednosti kod diskretnih značajki su diskretne vrijednosti koje ta značajka može poprimiti. Ovdje se to polje ne smije ostaviti prazno.

5.4. Funkcije stabla

Implementirano je šest različitih funkcija koje se mogu koristiti u stablu odluke, od kojih su dvije za diskretne značajke, a preostale četiri za značajke. To su:

- IfIsDiscreteValue
- WhichDiscreteValue
- IfIsLessThan
- IfIsGreaterThan
- IfIsInRange
- LinearCombination

IfIsDiscreteValue uspoređuje trenutnu vrijednost neke diskretne značajke s nekom prethodno slučajno odabranom vrijednosti (koja je iz skupa dozvoljenih vrijednosti te značajke). Ako su jednake, izvodi se lijevo podstablo, inače se izvodi desno.

WhichDiscreteValue uspoređuje trenutnu vrijednost neke diskretne značajke sa svim dozvoljenim vrijednostima tog atributa. Sadrži n podstabala gdje je n broj dozvoljenih vrijednosti neke značajke, a izvodi se ono podstablo za koje vrijedi da je trenutna vrijednost jednaka vrijednosti u toj grani.

IfIsLessThan uspoređuje trenutnu vrijednost neke kontinuirane značajke sa nekom slučajno odabranom vrijednosti y (koja je u rasponu $[min, max]$ te značajke). Ako je trenutna vrijednost manja, izvodi se lijevo podstablo, inače se izvodi desno.

IfIsGreaterThan radi isto što i prethodna, uz razliku što provjerava je li trenutna vrijednost veća od odabrane vrijednosti x. Ako jest, izvodi se lijevo, inače desno podstablo.

IfIsInRange provjerava je li trenutna vrijednost u nekom rasponu $[x, y]$ gdje su x i y iz raspona $[min, max]$ te značajke. Ako jest, izvodi se lijevo podstablo, inače se izvodi desno.

LinearCombination je linearna kombinacija nekoliko kontinuiranih značajki. Možemo je zapisati kao $ax + by + cz \leq d$ gdje su a, b, c, d konstante, a x, y, z značajke. Ako je nejednakost zadovoljena, izvodi se lijevo podstablo, inače se izvodi desno.

5.5. Genetski operatori

Programski okvir ECF sadrži neke implementacije genetskih operatora od kojih je većina iskoristiva. Budući da su dodane nove funkcije za koje u programskom okviru ECF nisu baš omogućeni operatori mutacije (osim mutacije stvaranjem novog podstabla), bilo je potrebno implementirati i poseban operator mutacije.

5.5.1. Dodatni operator mutacije

Operator u slučaju funkcija s realnim značajkama (osim LinearCombination koji je nešto kompliciraniji), mutira vrijednosti x i/ili y tako što uzima u obzir Gauss-ovu raspodjelu uzetu na 10% raspona $[min, max]$ kontinuirane značajke.

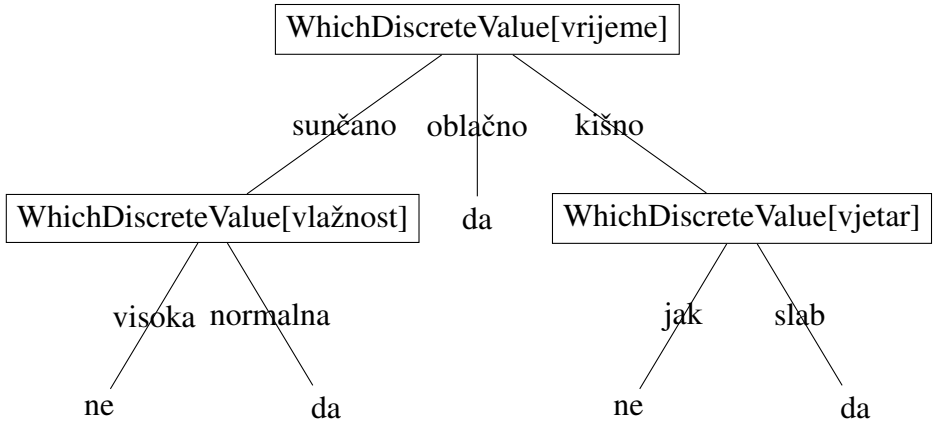
Kod LinearCombination, operator mutira vrijednosti a, b, c, d u skladu sa pripadajućim značajkama i to tako što konstante a, b, c također mutira upotrebnom Gauss-ove raspodjele uzete na 10% raspona kontinuirane značajke, a konstantu d mutira tako što

uzima 10% raspon minimalne i maksimalne moguće vrijednosti lijeve strane nejednakosti.

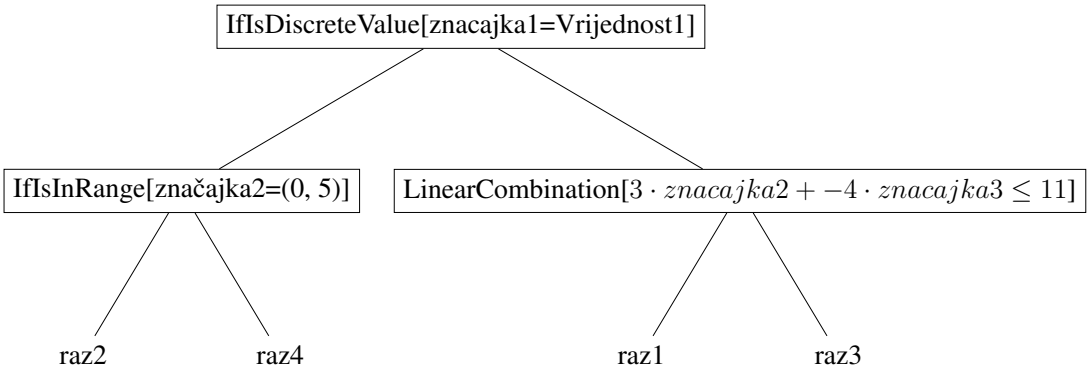
U slučaju funkcija s diskretnim značajkama, samo se slučajno odabere neka druga (ili ista) vrijednost iz skupa dozvoljenih vrijednosti te značajke.

Moguće ga je, kao i svaki drugi operator mutacije u ECF-u, odabrati ili isključiti u konfiguracijskoj datoteci s parametrom "mut.dectree" gdje je vrijednost vjerojatnost da će baš taj operator mutacije biti izabran.

Na sljedećim slikama su prikazani neki od mogućih zapisa stabla odluke pomoću korištenih funkcija.



Slika 5.2: Prikaz već viđenog stabla odluke za problem igranja tenisa pomoću navedenih funkcija



Slika 5.3: Prikaz stabla odluke sa nekim od navedenih funkcija

5.6. Mjere sposobnosti

Korištene mjere sposobnosti su već ranije objašnjene pa ne treba ulaziti u detalje, implementirane mjere su temeljene na matrici zbunjenosti, i to:

– točnost (engl. *accuracy*)

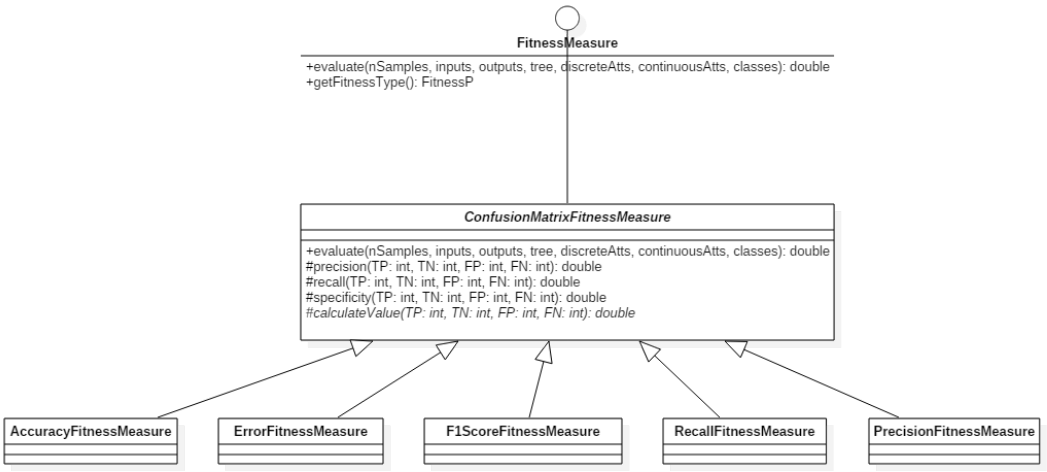
– pogreška (engl. *error*)

– osjetljivost (engl. *sensitivity / recall*)

– preciznost (engl. *precision*)

– f1 ocjena (engl. *f1 score*)

Moguće ih je odabrati dodatnim parametrom "fitnessmeasure" u konfiguracijskoj datoteci.



Slika 5.4: Dijagram razreda za mjere sposobnosti

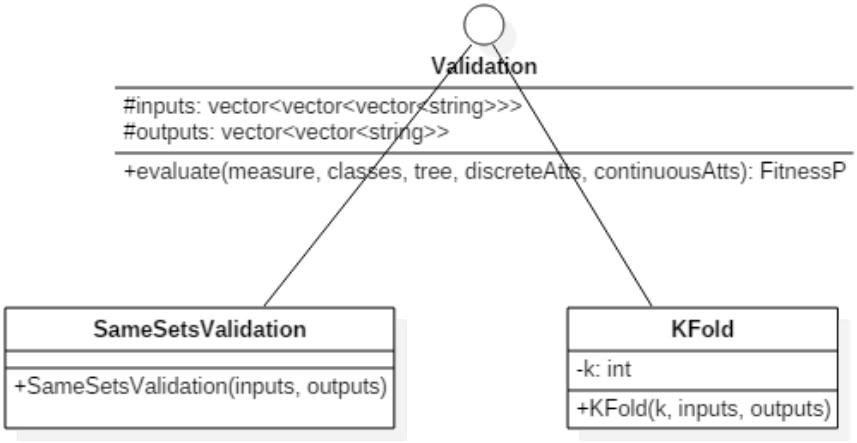
5.7. Unakrsna provjera

Korištene metode unakrsne provjere su isto tako objašnjene ranije u radu. Dodana je također i SameSetsValidation metoda koja ne vrši unakrsnu provjeru. Od ranije navedenih implementirane su:

– k-struka unakrsna provjera (engl. *K-fold*)

– Leave-One-Out

Moguće ih je odabrati parametrom "validation". U slučaju korištenja k-struke unakrsne provjere, moguće je iskoristiti i dodatni parametar "kfoldsize" koji označava broj stvorenih podskupova (uobičajena vrijednost je 2).



Slika 5.5: Dijagram razreda za validaciju

5.8. Kratki pregled parametara

U sljedećim tablicama bit će objašnjeni svi korišteni parametri. Detaljniji način upotrebe je objašnjen na [1].

Na tablici 5.1 su prikazani svi korišteni parametri ECF-a. Postoji i veliki broj drugih parametara koji za potrebe ovog rada nisu pretjerano važni.

Tablica 5.1: Prikaz korištenih parametara ECF-a

naziv parametra	objašnjenje
maxdepth	maksimalna dubina stabla
mindepth	minimalna dubina stabla
functionset	skup funkcija
terminalset	skup završnih znakova
population.size	veličina populacije
mutation.indprob	vjerojatnost mutacije

Na tablici 5.2 su prikazani svi korisnički definirani parametri, njihova objašnjenja te dozvoljene vrijednosti (ako postoje).

Tablica 5.2: Prikaz korisnički definiranih parametara

naziv parametra	objašnjenje	dozvoljene vrijednosti
attributefile	putanja do datoteke s definicijom atributa	-
inputfile	putanja do datoteke sa skupom za učenje	-
testfile	putanja do datoteke sa skupom za ispitivanje	-
resultsfile	putanja do datoteke u koju će se ispisati rezultati	-
fitnessmeasure	oznaka mjere sposobnosti	accuracy, error, f1score, precision, recall
validation	oznaka korištene validacije	kfold, loo
kfoldsize	broj k u k-strukoj provjeri	-
mut.dectree	vjerojatnost odabrane mutacije	-

6. Rezultati mjerenja

Da bi se provjerila efikasnost navedenog algoritma, izveden je na nekoliko skupova za učenje. Izabrani su skupovi različitog broja značajki, različitog broja razreda, od najjednostavnijih primjera do složenijih.

Za čvorove stabla su korištene sve navedene funkcije koje odgovaraju određenom problemu i njegovim značajkama. Tako će se za problem kojem su sve značajke kontinuirane koristiti samo funkcije koje rade s realnim brojevima, a ako postoje različiti tipovi značajki koriste se sve funkcije.

Prvo je potrebno odrediti povoljan uvjet zaustavljanja. Za uvjet zaustavljanja je odabran broj evaluacija mjere sposobnosti tako što je algoritam pokrenut na jednom skupu za učenje i tražio se broj pri kojem će doći do potpune ili djelomične stagnacije.

Nakon toga, potrebno je i odrediti najpovoljnije parametre algoritma. Parametri koji se određuju su veličina populacije i vjerojatnost mutiranja pojedine jedinke. Također, algoritam je pokrenut na jednom skupu sa prethodno utvrđenim uvjetom zaustavljana i odabrani su parametri koji su rezultirali najboljom mjerom sposobnosti.

Većina korištenih skupova je preuzeta sa [4]. To su:

- vozila (engl. *car*)
- iris
- puzlatka (engl. *abalone*)
- poker

Iznimka je skup tenis koji je preuzet iz [6].

6.1. Skupovi podataka

6.1.1. Vozila

Skup podataka na kojem su provjeravani parametri i određivan uvjet zaustavljanja je skup vozila. To je skup koji za značajke ima cijenu vozila, održavanost, broj vrata,

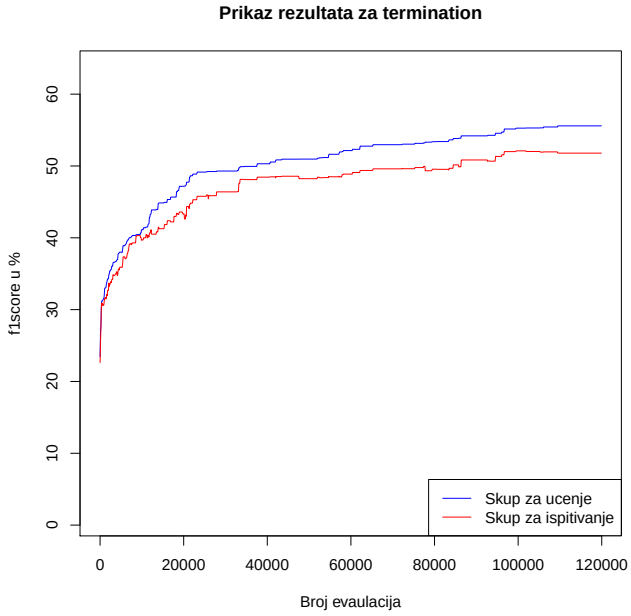
broj osoba, veličina prtljažnika i sigurnost. Cilj je odrediti je li navedeno vozilo prihvatljivo za kupnju. Tako su razredi *unacc*, *acc*, *good*, *vgood* i predstavljaju redom neprihvatljivo, prihvatljivo, dobro, vrlo dobro.

Tablica 6.1: Prikaz značajki i razreda za skup vozila

značajke	moguće vrijednosti	objašnjenje
buying	vhigh, high, med, low	cijena vozila
maint	vhigh, high, med, low	održavanost vozila
doors	2, 3, 4, 5more	broj vrata
persons	2, 4, more	broj mjesta
lug	small, med, big	veličina prtljažnika
safety	low, med, high	sigurnost vozila

razredi
unacc
acc
vgood
good

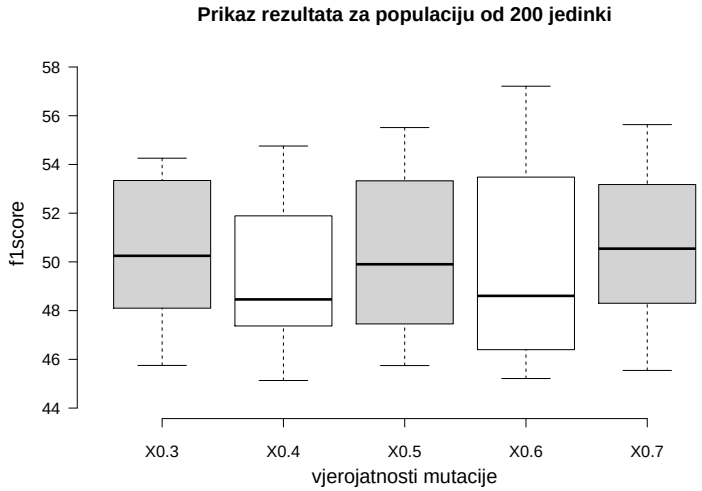
Određivanje uvjeta zaustavljanja je prikazano na slici 6.1. Pokrenuto je 10 puta i može se vidjeti da do duže stagnacije dolazi od 60000 do 100000 evaluacija pa je za uvjet zaustavljanja odabran broj evaluacija od 65000.



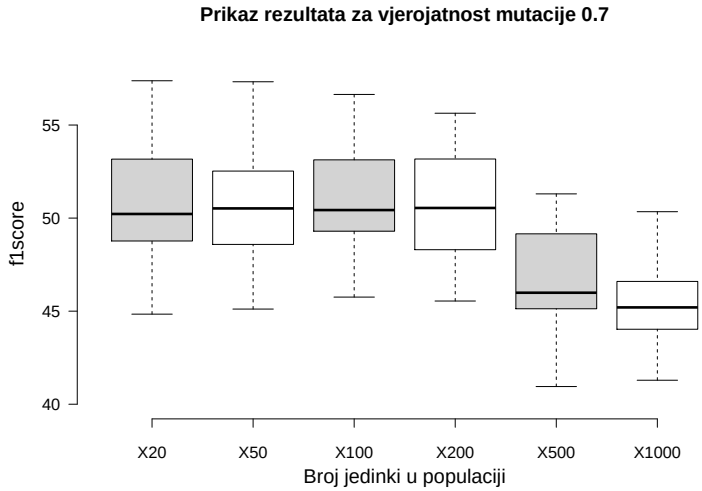
Slika 6.1: Kretanje prosjeka najboljih jedinki 10 pokretanja

Na sljedećim slikama je prikazan utjecaj parametara na ponašanje algoritma za dotični problem. Za svaku kombinaciju parametara pokrenut je 10 puta. Može se primijetiti da vjerojatnost mutacije nema prevelik utjecaj na parametre, razlike se javljaju u 1-2 %. I za veličinu populacije kao parametar može se primijetiti da nema velik utje-

caj, osim u slučaju većih populacija koje broje više od 500 jedinki gdje dobivamo lošija rješenja. Najbolje se pokazala kombinacija parametara gdje je veličina populacije 200 i vjerojatnost mutacije 0.7 pa su ti parametri odabrani za daljnja mjerenja.



Slika 6.2: Prikaz dobivenih rezultata za fiksiranu populaciju i različite vjerojatnosti mutacije



Slika 6.3: Prikaz dobivenih rezultata za fiksiranu vjerojatnost mutacije i različite veličine populacije

6.1.2. Tenis

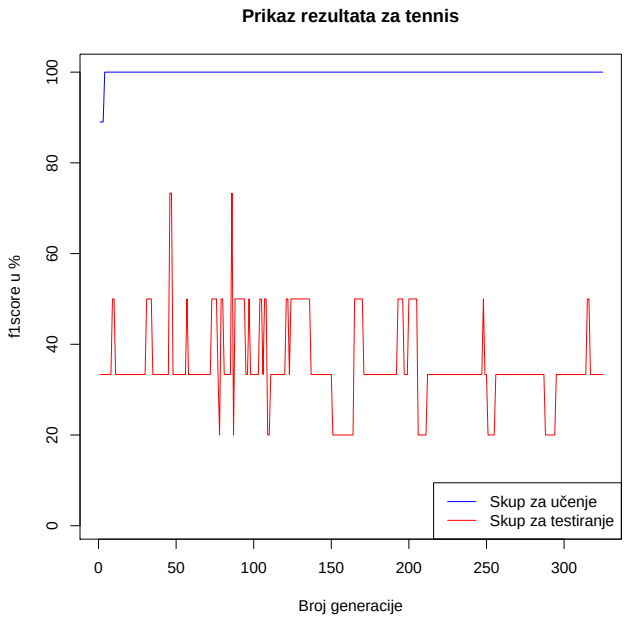
Ovo je osnovni korišteni skup za učenje koji je ujedno i služio za provjeru ispravnosti algoritma. Cilj je odrediti jesu li vremenski uvjeti povoljni za igranje tenisa. Razredi su yes i no, tj. jesu li uvjeti povoljni ili ne.

Tablica 6.2: Prikaz značajki i razreda za skup tenis

značajke	moguće vrijednosti	objašnjenje
outlook	sunny, overcast, rainy	vrijeme
temp	hot, mild, cool	temperatura
hum	high, normal	vlažnost
wind	false, true	vjetrovitost

razredi
yes
no

Na slici 6.4 su prikazani rezultati mjerenja. Mjera sposobnosti na skupu za učenje se uglavnom kreće oko 100%-ne učinkovitosti, i kvalitetno rješenje se vrlo rano dobije. Pojavljuje se problem na skupu za ispitivanje gdje učinkovitost nije najbolja, ali to leži u malom broju podataka za učenje i ispitivanje (14 različitih primjera).



Slika 6.4: Kretanje mjere sposobnosti po generacijama

6.1.3. Iris

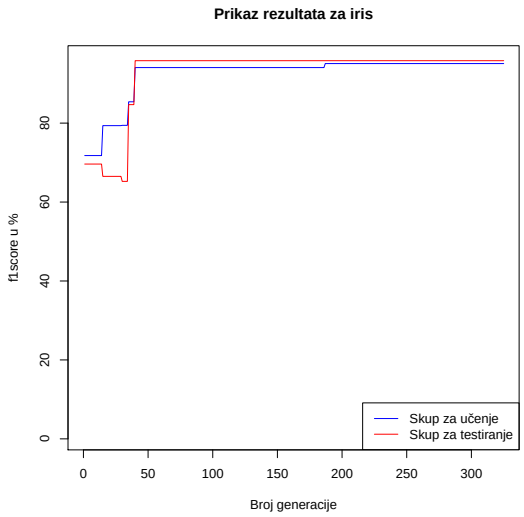
Iris skup podataka je jedan od najčešće korištenih skupova u strojnom učenju. Cilj je prema duljinama petiljki i resi iris cvijeta odrediti kojoj podvrsti pripadaju.

Tablica 6.3: Prikaz značajki i razreda za skup iris

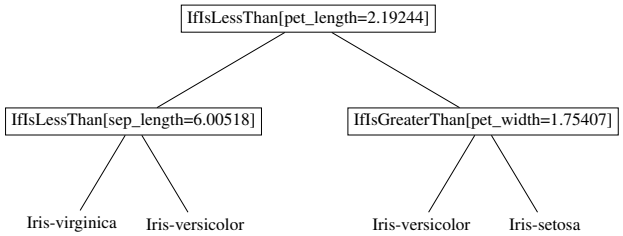
značajke	moguće vrijednosti	objašnjenje
sepal length	$[4.3, 7.9]$	duljina rese
sepal width	$[2.0, 4.4]$	širina rese
petal length	$[1.0, 6.9]$	duljina petiljke
petal width	$[0.1, 2.5]$	širina petiljke

razredi
iris-versicolor
iris-virginica
iris-setosa

Na slici 6.5 možemo vidjeti dobivene rezultate na ovom skupu. Najbolja mjera sposobnosti za ovaj razred je uglavnom iznad 99%, i potrebno je jako malo generacija da bi se došlo do takvog rješenja.



Slika 6.5: Kretanje mjere sposobnosti po generacijama



Slika 6.6: Prikaz jednog mogućeg rješenja ($f1score = 98.17$)

6.1.4. Puzlatka

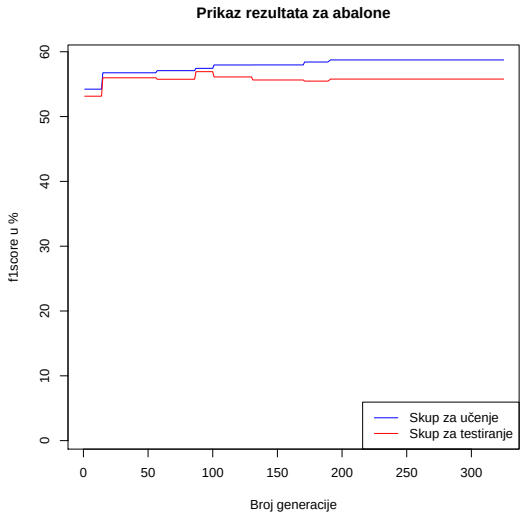
Puzlatka skup podataka za cilj ima odrediti starost puzlatke po nekim fizičkim mjerenjima kao što su duljina, spol, visina ... Izvorni skup podataka ima 29 neravnomjerno raspoređenih razreda koji predstavljaju broj prstenova na puzlatki koji služe za određivanje starosti($broj_{prstenova} + 1.5$) pa je za potrebe ovog rada prepravljen tako da se podijeli na samo 3 razreda broja prstenova: 1-8, 9-10, 11+.

Tablica 6.4: Prikaz značajki i razreda za skup puzlatka

značajke	moguće vrijednosti	objašnjenje
sex	M, F, I	spol
length	$[0.075, 0.815]$	duljina
diameter	$[0.055, 0.650]$	promjer
height	$[0, 1.13]$	visina
whole weight	$[0.002, 2.826]$	ukupna težina
shucked weight	$[0.001, 1.488]$	težina bez školjke
viscera weight	$[0.001, 0.760]$	težina iznutrica
shell weight	$[0.002, 1.005]$	težina školjke nakon sušenja

razredi
1-8
9-10
11+

S ovakvom preraspodjelom dolazimo do rezultata prikazanih na slici 6.7. Može se primijetiti da, za razliku od prethodnih skupova podataka, dolazi do nešto lošije mjere sposobnosti, vjerojatno uzrokovane većim skupom podataka, i možda ne najboljom podjelom na nove razrede.



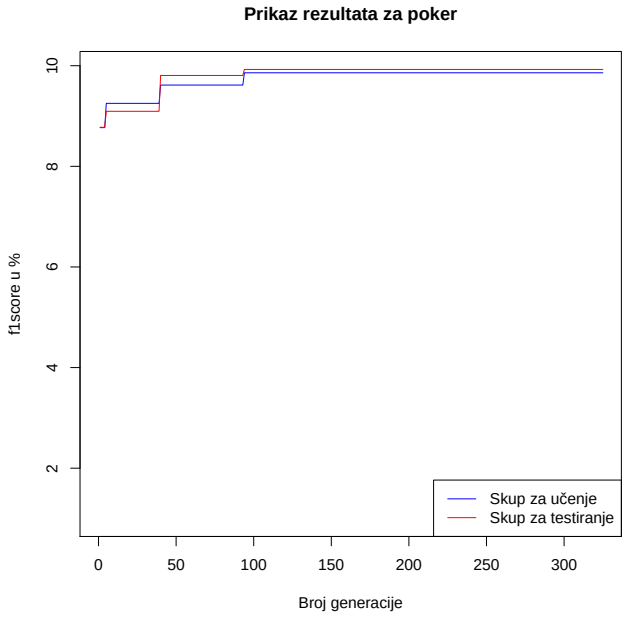
Slika 6.7: Kretanje mjere sposobnosti po generacijama

6.1.5. Poker

Poker skup podataka za cilj ima odrediti koju jačinu pokeraške ruke imamo ako su nam poznate vrijednosti i boje karata u ruci.

Postoji 10 značajki od kojih svaki par predstavlja jačinu (numeriranu od 1 - 13) i boju (numeriranu od 1 - 4) određene karte u ruci (numeriranih od 1-. Postoji 10 različitih razreda, od kojih svaki predstavlja moguću pokerašku ruku.

Na slici 6.8 su prikazani rezultati za ovaj skup podataka. Može se primijetiti da ovdje algoritam nije baš učinkovit, a razlozi vjerojatno leže u samoj prirodi problema. Npr. ako trebamo klasificirati par, potrebno je vidjeti da imamo dvije karte čija je jačina ista (na bilo kojoj poziciji), te da su preostale 3 sve različite. Za funkcije koje su definirane u ovom radu to nije nimalo lak posao.



Slika 6.8: Kretanje mjere sposobnosti po generacijama

7. Zaključak

Iz ovog rada možemo primijetiti da je primjena genetskog programiranja na stabla odluke moguća i da daje pristojne rezultate. Za skupove podataka koji imaju manji broj značajki ili razreda, ponaša se čak i odlično, dok se teže snalazi sa mnogo razreda.

Rezultati dobiveni eskperimentima pokazuju da je ovakva implementacija primjene genetskog programiranja na stabla odluke itekako primjenjiva, a pogotovo se dobro snalazi na problemima s manje razreda. Teži problemi su oni sa više razreda i kompliciranijom definicijom problema (kao poker) na kojem ne nalazi iskoristiva rješenja.

LITERATURA

[1] Ecf programski okvir. http://ecf.zemris.fer.hr.

[2] William J. Clancey. Classification problem solving. U *Proceedings of the Fourth AAAI Conference on Artificial Intelligence*, AAAI'84, stranice 49–55. AAAI Press, 1984.

[3] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-11170-5.

[4] M. Lichman. UCI machine learning repository, 2013. URL http://archive.ics.uci.edu/ml.

[5] Donald Michie, D. J. Spiegelhalter, C. C. Taylor, i John Campbell, urednici. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, Upper Saddle River, NJ, USA, 1994. ISBN 0-13-106360-X.

[6] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 izdanju, 1997. ISBN 0070428077, 9780070428072.

[7] Riccardo Poli, William B. Langdon, i Nicholas Freitag McPhee. *A field guide to genetic programming*. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk, 2008. URL http://www.gp-field-guide.org.uk. (With contributions by J. R. Koza).

[8] Tuve Löfström Rikard König, Ulf Johansson i Lars Niklasson. Improving gp classification performance by injection of decision trees. stranice 2942 – 2949, Barcelona, Spain, 7 2010. WCCI 2010 IEEE World Congress on Computational Intelligence.

[9] Stuart J. Russel i Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Pearson Education Ltd., 2009.

[10] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM JOURNAL OF RESEARCH AND DEVELOPMENT*, stranice 71–105, 1959.

Rješavanje klasifikacijskih problema evolucijom stabala odluke

Sažetak

Strojno učenje obuhvaća moćne metode koje se bave analizom podataka i učenjem računala na način sličan ljudskom. U ovome radu se primjenjuje genetsko programiranje na stabla odluke kako bi se riješili klasifikacijski problemi. I genetsko programiranje i stabla odluke su efikasni algoritmi u rješavanju tih problema, i kombinacijom se mogu dobiti još bolja svojstva. Za implementaciju je korišten ECF i implementiran je algoritam koji koristi jednostavno genetsko programiranje i isproban je na nekoliko skupova podataka na kojima su rezultati uglavnom dobri.

Ključne riječi: Strojno učenje, klasifikacija, stabla odluke, genetsko programiranje

Classification using evolution of decision trees

Abstract

Machine learning consists of powerful methods which are dealing with data analysis and teaching the computer in a similar way as a human. In this thesis genetic programming is applied on decision trees to solve classification problems. Genetic programming and decision trees are efficient algorithms in solving those problems, and combining them even better properties are possible. ECF is used for implementation and the implemented algorithm uses simple genetic programming which tested on several data sets, shows some mostly good results.

Keywords: Machine learning, classification, decision trees, genetic programming