SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 213

PRIMJENA GENETSKOG PROGRAMIRANJA U STROJNOM UČENJU

Ivana Stokić

Zagreb, lipanj 2011.

Sadržaj

Zahvaljujem se mentoru doc. dr. sc. Domagoju Jakoboviću na savjetima i pomoći pri izradi ovog diplomskog rada. Zahvaljujem i dr. sc. Igoru Grudeniću na pomoći u obradi podataka potrebnih za praktični dio ovog rada.

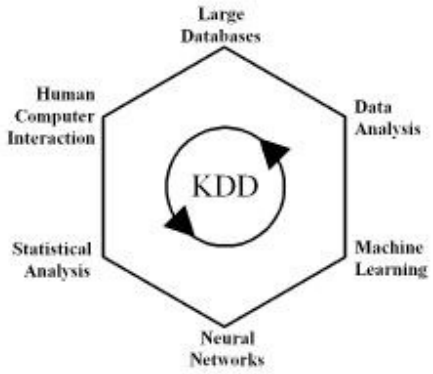
1. Uvod

Dubinska analiza podataka postala je u zadnje vrijeme vrlo popularna. U doba kada se često rukuje s jako velikim skupovima podataka morale su se razviti i tehnike koje će iz velikih količina podataka i različitih informacija izvući korisno znanje. Otkrivanje znanja iz velikih skupova podataka potrebno je u gotovo svim djelatnostima, u medicini, forenzici, kriminalistici, računalnoj sigurnosti, ekonomiji i mnogim drugima. Od svih problema dubinske analize podataka u ovom će radu najviše riječi biti o klasifikaciji i kako se genetsko programiranje kao jedna vrlo moćna tehnika može iskoristiti za klasifikaciju podataka. Opis klasifikacije podataka iz skupa *German credit data* [54] dan je kao uvodni primjer, a glavni skupovi podataka koji su se koristili u izradi ovog rada su skupovi podataka o korisničkim poslovima u računalnim grozdovima iz arhive paralelnog opterećenja grozdova.

U drugom poglavlju ukratko je opisana dubinska analiza podataka te dan pregled osnovnih pojmova vezanih za klasifikaciju podataka i ocjenjivanje rada klasifikatora. U trećem poglavlju ukratko su opisani način rada i vrste genetskog programiranja općenito, a poslije su, u četvrtom poglavlju, detaljnije predstavljeni različiti načini korištenja genetskog programiranja kao klasifikatora. U petom poglavlju opisane su najvažnije metode evaluacije klasifikacijskog modela. U šestom poglavlju detaljnije su opisani koraci i operatori genetskog programiranja, navedeni su i opisani različiti načini stvaranja rješenja, tj. stabala. Predstavljen je program kao praktični dio ovog rada i upute za njegovo korištenje. U sedmom poglavlju nalaze se tablice i grafovi dobiveni provođenjem niza eksperimenata, praćenjem kretanja pogreške klasifikacije mijenjanjem različitih genetskih parametara ili parametara metoda evaluacije. Na kraju je prikazana usporedba dobivenih rezultata s rezultatima (pronađenima u literaturi) drugih klasifikacijskih algoritama na istim skupovima podataka.

2. Dubinska analiza podataka

Dubinska analiza podataka (engl. *data mining*) je netrivijalan proces pronalaženja vrijednih informacija u velikim skupovima podataka te dobivanja zanimljivog, dosad nepoznatog, točnog, korisnog i razumljivog znanja iz tih informacija. Motivacija za otkrivanje znanja slijedi iz rečenice: "*We are drowning in data, but starving for knowledge*" [29]. Jedno od područja dubinske analize podataka je otkrivanje znanja u skupovima podataka (engl. *knowledge discovery in data sets, KDD*), a ono povezuje baze podataka, vizualizaciju podataka, strojno učenje (koje predlaže načine stvaranja računalnih sustava koji automatski poboljšavaju svojstva kroz iskustvo) i različite statističke metode, kako je prikazano na slici 2.1 [28].



Slika 2.1 Sastavni dijelovi procesa otkrivanja znanja u skupovima podataka

Dubinska analiza podataka koristi se u analizi i upravljanju rizikom, u ekonomiji, medicini i mnogim drugim djelatnostima. Često se koristi u domeni računalne sigurnosti - za lakše otkrivanje uljeza (engl. *intrusion detection*), zatim se u zadnje vrijeme vrlo često spominju i klasifikacija teksta (engl. *Text Mining*) i analiza važnih informacija na Internetu (engl. *Web Mining*). Cilj dubinske analize podataka je automatizirati proces traženja znanja u podacima, jer bi njegovo provođenje bez računala predugo trajalo. Definiraju se dvije vrste modela: prediktivni i deskriptivni. Prediktivni model predviđa vrijednosti nekih atributa u budućnosti na temelju već viđenih podataka (zato je važna točnost znanja). Deskriptivni model sadrži znanje o podacima. To bi znanje trebalo biti razumljivo za ljude, jer će ga ljudi interpretirati i poslije koristiti i zato se vrlo često predstavlja u obliku ako – onda pravila. Ako su neki uvjeti zadovoljeni, tada je potrebno poduzeti neku akciju, tj. predvidjeti neku vrijednost u budućnosti. Algoritmi dubinske analize podataka trebali bi biti robusni, jer podaci u bazama podataka često nedostaju ili su nepotpuni. Dubinska analiza podataka zahtijeva dobro poznavanje podataka i domene problema i prednost je što o podacima nije potrebno ništa pretpostavljati.

Postoji niz različitih zadataka dubinske analize, neki od njih su:

- klasifikacija (engl. *classification*) - previđanje vrijednosti nekog atributa u budućnosti na temelju analize već viđenih podataka;
- regresija (engl. *regression*) – predviđanje kontinuirane vrijednosti;
- gupiranje (engl. *clustering*) – oznake grupa nisu unaprijed poznate; cilj je pronaći grupe u podacima;
- vizualizacija (engl. *visualization*) – grafičko prikazivanje podataka i otkrivenog znanja kako bi bilo razumljivije i preglednije;
- pronalaženje promjena u podacima (engl. *deviation detection*);
- pronalaženje asocijacija (engl. *association rules*) – pronalaženje veza među podacima; jedna od razlika između klasifikacijskih i asocijacijskih pravila je u tome što asocijacijska pravila mogu imati više različitih atributa na desnoj strani pravila i što se bilo koji atribut (pa i onaj ciljni) može nalaziti ili na lijevoj ili na desnoj strani pravila, dok se kod klasifikacijskih pravila ciljni atribut nalazi uvijek na desnoj strani.

Općenito se u procesu otkrivanja znanja podaci trebaju očistiti, u smislu ispravljanja uočenih pogrešaka ili nekonzistentnosti u podacima ili popunjavanju nedostajućih vrijednosti; zatim se, ako je potrebno, mogu integrirati, tj. objediniti iz različitih izvora; može se obavljati transformacija podataka kako bi postali pogodniji za primjenu određenog algoritma, npr. diskretizacija pojedinih atributa koji imaju numeričke vrijednosti ako algoritam koji će analizirati podatke ne može raditi s takvim vrijednostima, a često se obavlja i odabir jednog podskupa iz skupa svih atributa – odabiru se oni atributi koji su važni za provođenje određenog zadatka [35] [49], što su tzv. diskriminatorne značajke.

U nastavku će detaljnije biti opisana klasifikacija koja je od svih zadataka dubinske analize podataka za ovaj rad najbitnija.

2.1 Klasifikacija podataka

2.1.1 Definicija klasifikacije

Klasifikacija podataka najčešće je rješavan problem na području inteligentne analize podataka, što je drugi naziv za dubinsku analizu podataka (inteligentna analiza zato što se za njenu realizaciju koriste postupci umjetne inteligencije i strojnog učenja). Dva su osnovna cilja klasifikacije: određivanje nepoznatih razreda (klasa, kategorija, engl. *class*) ili grupa u podacima ili, ako su oznake razreda te pripadnost nekih uzoraka pojedinim razredima unaprijed poznati, cilj je svrstati nove, još neviđene podatke u neki od već unaprijed poznatih razreda. Razrede može unaprijed odrediti učitelj (engl. *supervisor*), oni mogu biti rezultat predikcije iz poznatih atributa (kao što je npr. određivanje hoće li kamate rasti ili neće) ili mogu biti funkcije dostupnih atributa, tj. mogu biti izvedeni iz atributa. Ako su oznake razreda poznate, tada se radi o učenju pod nadzorom (engl. *supervised learning*), a ako razredi / grupe nisu unaprijed poznati, tada se radi o učenju bez nadzora (engl. *unsupervised learning*). Kako su pri izradi praktičnog dijela ovog rada korišteni skupovi podataka s već poznatom pripadnošću razredima i oznake svih razreda su poznate, ovdje se radi o učenju pod nadzorom.

Najjednostavniji je slučaj kad se radi o svrstavanju u samo jedan razred. Ako se podaci trebaju klasificirati u više od jednog razreda, tada se radi o višeklasnoj klasifikaciji (engl. *multiclass classification*). Najpoznatija je binarna klasifikacija, gdje postoje samo dva razreda. Općenito postoji K razreda, gdje se pripadnost pojedinog uzorka x^i može definirati K – dimenzionalnim vektorom $y^{(i)} = \left[y_1^{(i)}, y_2^{(i)}, \dots, y_K^{(i)}\right]^\tau$, a $y_j^{(i)}$ se određuje prema formuli (2.1).

$$y_j^{(i)} = \begin{cases} 1, & ako\ x^i \in C_j \\ 0, & inače \end{cases} \tag{2.1}$$

Npr. $y^{(1)} = [0\ 1\ 0\ 0]$ bi značilo da uzorak $x^{(1)}$ pripada razredu s oznakom C_2.

Jedan uzorak može istovremeno pripadati većem broju razreda, što se naziva klasifikacijom s višestrukim oznakama ili klasifikacijom jedan na više (engl. *multilabel classification*), no ona se rjeđe koristi. Višeklasna klasifikacija obično se svodi na više binarnih klasifikacija, npr. ako postoji N razreda, tada se problem može svesti na N binarnih klasifikacija, gdje će se uvijek odijeliti pripadnost jednom razredu, a pripadnost ostalim razredima uzimat će se kao pripadnost drugom razredu u binarnoj klasifikaciji, što znači da će se binarna klasifikacija provesti N puta, za svaki od postojećih razreda.

Pojedini podatak opisan je nizom diskriminatornih značajki (engl. *features*) koje se nazivaju atributima. Atribut može biti numerički (može imati cjelobrojnu ili kontinuiranu vrijednost) ili nominalni. Nominalni atributi mogu se pretvoriti u numeričke tako da se svakoj nominalnoj vrijednosti pridijeli određena numerička vrijednost. Atribut ne može biti slobodan tekst niti smije uključivati nabrajanje više pojmova [22]. O tim dvama tipovima atributa ovisit će i način klasifikacije samih podataka. Skup podataka za učenje sastoji se od uzoraka, primjera u obliku atribut - vrijednost i za svaki uzorak poznat je razred kojem taj uzorak pripada. Obično su vrijednosti svih atributa nekog uzorka poznate, no to ne mora uvijek biti slučaj. Vrijednost atributa za neki uzorak može biti i nepoznata (engl. *missing value*) i s takvim se uzorcima treba posebno postupati (npr. izbaciti ih iz postupka analize).

Kako bi se pojedini uzorci mogli klasificirati, potrebno je izgraditi odgovarajući klasifikator, a on se izgrađuje učenjem. Klasifikatoru se tijekom učenja daju podaci s već poznatim pripadnostima pojedinom razredu, kako bi on u budućnosti mogao ispravno generalizirati. Ako klasifikator dobro generalizira, tada će on ispravno klasificirati još neviđeni podatak. Cilj klasifikacije je izgraditi klasifikator koji će dobro generalizirati. U svrhu učenja i ispitivanja (testiranja) početni se skup podataka može podijeliti na različite načine na skup za učenje i skup za testiranje (ponekad i u skup za validaciju), a o tome će više riječi biti u petom poglavlju ovog rada, gdje se opisuju različiti načini evaluacije modela.

Formalna definicija klasifikacije glasi: neka postoji skup za učenje koji se sastoji od N primjera $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, gdje su $x_1, x_2, \dots, x_N$ vrijednosti uzoraka, a $y_1, y_2, \dots, y_N$ oznake razreda kojima ti uzorci pripadaju. Klasifikacija je izgradnja klasifikatora $h : X \rightarrow Y$, koji uzorku x_i iz skupa uzoraka X pridjeljuje oznaku razreda (klase) y_i iz skupa svih mogućih razreda Y za dani skup podataka.

Klasifikacijska pravila najčešće dolaze u obliku ako – onda pravila. Ako su neki uvjeti zadovoljeni, tada se može predvidjeti određena oznaka razreda koja se inače nalazi na desnoj strani pravila. Lijeva strana pravila je uvjet (stanje, premisa, antecedens, engl. *condition*), a desna strana je akcija (posljedica, zaključak, konsekvens, engl. *action*) [9].

Postoje različiti algoritmi na području strojnog učenja za klasifikaciju podataka koji se razlikuju po brzini rada, uspješnosti klasifikacije (što najčešće ovisi i o podacima na kojim se gradi i ispituje klasifikator) itd.

2.1.2 Najznačajniji algoritmi za rješavanje problema klasifikacije

Postoji puno različitih klasifikacijskih algoritama koji se razlikuju po jednostavnosti primjene, točnosti (engl. *accuracy*) ili razumljivosti. U nastavku će biti spomenuti

samo najznačajniji. Ovisno o domeni problema, ponekad je bitna brzina algoritma pa će se, u nekim slučajevima, više cijeniti da je klasifikator brz, nego da mu je točnost jednaka 95%. Ili će se možda više zahtijevati točnost od brzine i jednostavnosti i sl. Algoritmi za klasifikaciju dijele se na one koji pripadaju učenju pod nadzorom i one koji pripadaju učenju bez nadzora. Najznačajniji algoritmi iz prve skupine su: višestruka diskriminantna analiza (Fisherova linearna diskriminanta), zatim SVM (engl. *Support Vector Machine*), generalizirane decizijske funkcije, potencijalne funkcije, naivni Bayesov klasifikator, Bayesove mreže (engl. *Bayesian Networks*), stabla odluke (ID3, C4.5 itd.), slučajne šume, višeslojne neuronske mreže... A najznačajniji predstavnici iz skupine učenja bez nadzora su algoritam k najbližih susjeda (engl. *k – nearest neighbours*) te algoritam k srednjih vrijednosti (engl. *K – means*).

2.1.3 Ocjenjivanje rada klasifikatora

U sljedećim su potpoglavljima opisana dva često korištena načina ocjenjivanja rada klasifikatora. To su tablica zabune (engl. *confusion matrix*) i ROC graf.

2.1.3.1 Tablica zabune

U tablici zabune [6] nalaze se informacije o stvarnom stanju i o predviđenom stanju koje je nastalo klasifikacijom. U njoj se nalazi broj pogrešno i broj ispravno klasificiranih uzoraka te se iz nje mogu izračunati različiti parametri, kao što su npr. točnost (engl. *accuracy*), što je udio točno klasificiranih primjera u skupu svih primjera, zatim preciznost (engl. *precision*), što je udio točno klasificiranih u skupu pozitivno klasificiranih, zatim odziv (engl. *recall*) – udio točno klasificiranih u skupu svih pozitivnih primjera, specifičnost (engl. *specificity*), što je udio točno klasificiranih u skupu svih negativnih primjera te F – mjera (F_β). U tablici 2.1 prikazani su sastavni dijelovi tablice zabune.

TP je broj točno pozitivnih (engl. *True – Positive*), a TN broj točno negativnih (engl. True Negative) i to su brojevi ispravno klasificiranih. Neispravno klasificirani su FP – broj netočno pozitivnih (engl. *False Negative*) i FN – broj netočno negativnih (engl. False Negative). Naravno, ovo je primjer samo za problem binarne klasifikacije, a za problem višeklasne klasifikacije gleda se zbroj ispravno, tj. neispravno klasificiranih po pojedinim razredima.

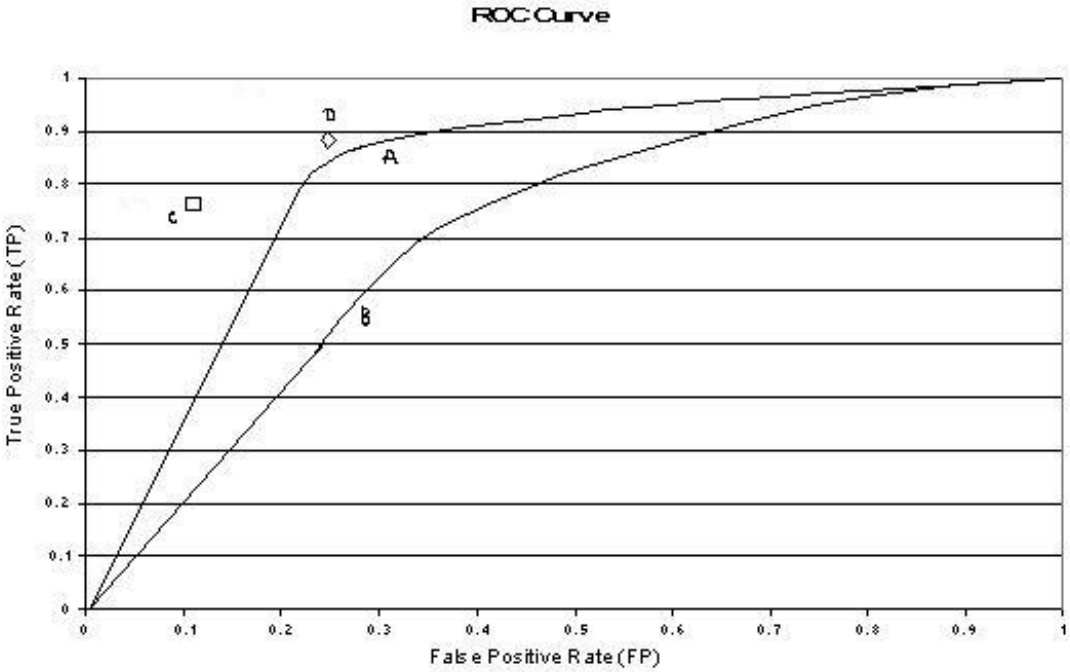
Tablica 2.1 Tablica zabune

		stvarno stanje	
predviđeno stanje		pozitivno	negativno
	pozitivno	TP	FP
	negativno	FN	TN

2.1.3.2 ROC graf

ROC (engl. *Receiver Operating Characteristic*) graf je graf koji na osi apscisa ima udio netočno pozitivnih (FP), a na osi ordinata udio točno pozitivnih (TP) [19]. Svaka točka tog grafa predstavljena je parom (FP, TP). Ako se radi o neparametarskom klasifikatoru, tada je klasifikator predstavljen samo jednom točkom. Točka (0, 1) je savršen klasifikator, jer sve uzorke klasificira ispravno. Točka (1, 0) je loš klasifikator, jer niti jedan uzorak ne klasificira ispravno (udio netočno pozitivnih je 100%). Točka (0, 0) predstavlja klasifikator koji bi sve uzorke klasificirao kao negativne, a točka (1, 1) klasifikator koji bi sve uzorke klasificirao kao pozitivne. Ako se radi o klasifikatoru koji ima i parametar kojim može utjecati na povećanje ili smanjenje broja točno - , tj. netočno – pozitivnih, mijenjanje tog parametra rezultira nizom (FP, TP) parametara koji čine ROC graf. Ovaj graf sadrži sve informacije koje bi sadržavala i tablica zabune, jer je FN komplement od TP, a TN komplement od FP.

Na slici 2.2 [6] prikazan je ROC graf, s A i B su označene dvije krivulje, a s C i D dvije točke na grafu (koje predstavljaju neparametarski klasifikator).



Slika 2.2 Primjer ROC krivulje

3. Genetsko programiranje

Genetsko programiranje (engl. *genetic programming*) je optimizacijska tehnika koja pripada skupini evolucijskih algoritama [27] (engl. *evolutionary algorithms*) za automatsko rješavanje različitih NP teških problema. To je stohastička i o domeni neovisna (engl. *domain – independent*) metoda. Predložio ga je 1992. god. John R. Koza. Rješenje je predstavljeno računalnim programom (engl. *computer program*) koji može imati različite oblike i biti različitih veličina, ovisno o tome o kojoj se vrsti genetskog programiranja radi. Genetsko programiranje je bolja metoda od većine drugih heurističkih metoda, jer prilikom pretraživanja prostora ne pretražuje samo svoje prve susjede, već križanjem jedinki (koje je glavni operator) ono pretražuje različite udaljene dijelove prostora. U sljedećem su potpoglavlju ukratko opisane najvažnije vrste.

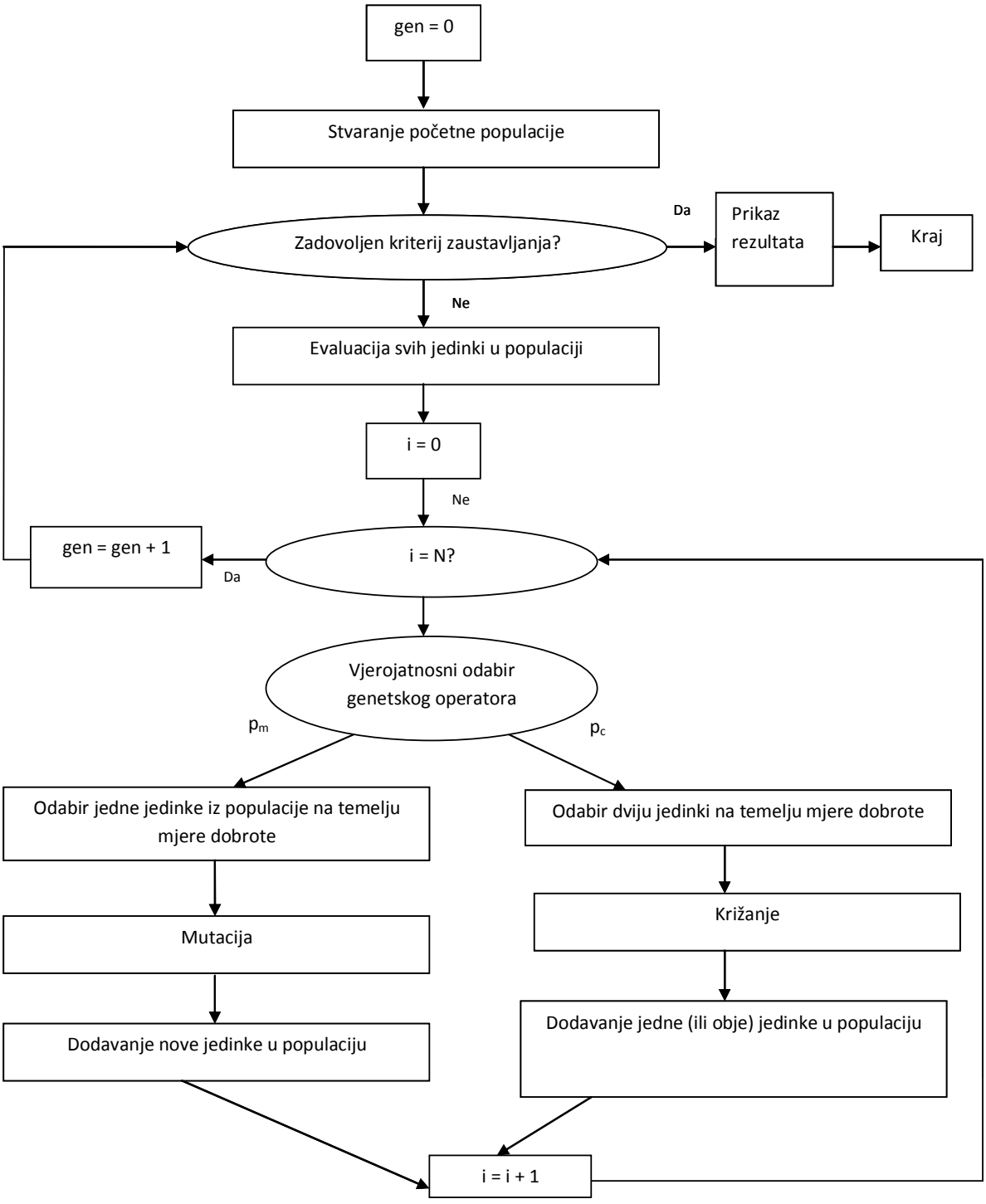
3.1 Razlike u odnosu na genetski algoritam

Genetski se algoritam, također kao jedan od prirodom inspiriranih algoritama, često i uspješno koristi kod rješavanja običnim algoritmima nerješivih NP teških problema. Razlika između genetskog algoritma i genetskog programiranja je prvo u predstavljanju rješenja. Kod genetskog algoritma rješenje, tj. jedna jedinka u populaciji rješenja nepromjenjive je duljine tijekom cijelog procesa. To rješenje može biti niz znakova, može biti niz binarnih znamenki, cijelih ili realnih brojeva i sl. što ovisi o samom problemu. Kod genetskog programiranja rješenje je prikazano računalnim programom; opet ovisno o problemu i kako je najprikladnije riješiti problem, tako će biti prikazana i jedinka u populaciji programa. U tom se slučaju kroz niz generacija razvijaju računalni programi te se odabire najbolji. Program može, kako je uobičajeno, biti prikazan u Lispu ili nekom njemu sličnom jeziku, u obliku stabla, u linearnom obliku kao niz instrukcija strojnog jezika i sl. Glavna je značajka da se tijekom cijelog procesa duljine pojedinih rješenja mijenjaju (npr. kod stabala se mijenjaju veličina i dubina stabla). U terminologiji povezanoj s genetskim programiranjem vrlo se često spominje tzv. *bloat*, što je zapravo nekontrolirani rast duljine rješenja, tj. jedinke u populaciji i zato se mora uvesti način ograničavanja duljine rješenja. Npr., ako se radi o stablima može se odrediti najveći dopušteni broj čvorova nekog stabla ili se npr. kod niza instrukcija može definirati najveći mogući broj instrukcija u programu.

Na slici 3.1 prikazani su osnovni koraci koji se primjenjuju kod genetskog programiranja. Obično se, ako se radi o genetskom algoritmu nakon križanja obavlja mutacija, dok se kod genetskog programiranja može primijeniti i samo jedan od ovih operatora bez primjene drugoga.

S N je označena veličina populacije. Svakoj jedinki u populaciji pridjeljuje se mjera dobrote (engl. *fitness*). Odabire se određen broj jedinki i na njima se provode genetski operatori križanje s vjerojatnošću p_c i mutacija s vjerojatnošću p_m.

U sljedećim potpoglavljima detaljnije su opisani različiti načini predstavljanja rješenja.



Slika 3.1 Graf tijeka genetskog programiranja

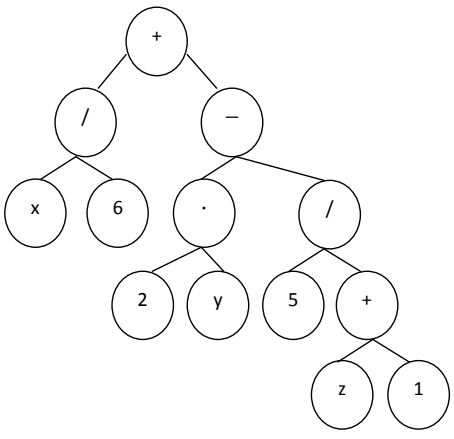
3.2 Vrste genetskog programiranja

Genetsko programiranje se dijeli na nekoliko vrsta, ovisno o načinu prikaza računalnog programa kao rješenja zadanog problema. To su stablasti GP (engl. *tree – based GP*) [41], linearni GP (engl. *linear GP*) [41], GP temeljen na grafovima (engl. *graph – based GP*) [41], celularni GP (engl. *cellular GP*) [26], GP temeljen na gramatici (engl. *grammar – based GP*) [26] te *constrained syntax GP* [26]. Postoji još i kartezijsko genetsko programiranje (engl. *Cartesian GP*) [41], gdje je rješenje prikazano u obliku linearnog kromosoma koji sadrži npr. cijele brojeve. U ovom radu, koji kao temeljni problem uzima problem klasifikacije i način na koji genetsko programiranje taj problem može riješiti, naglasak je stavljen na GP temeljen na stablima.

3.2.1 Genetsko programiranje temeljeno na stablima

Kod stablastog genetskog programiranja program je predstavljen stablom. Stablo se, kao nelinearna struktura, vrlo često koristi kao prikaz rješenja; njime se mogu jednostavno prikazati različiti matematički, logički izrazi i sl. Na slici 3.2 prikazan je (u obliku stabla) aritmetički izraz iz formule (3.1).

$$\frac{x}{6} + \left(2 \cdot y - \frac{5}{z+1}\right) \tag{3.1}$$



Slika 3.2 Prikaz stabla

Izrazi se sastoje od:

- skupa akcija, varijabli, konstanti - završnih znakova (engl. *terminal set*);
- skupa funkcijskih znakova.

U primjeru na slici završni znakovi su konstante (6, 2, 5 i 1) i varijable (x, y i z). Funkcijski znakovi su zbrajanje (+), oduzimanje (-), množenje (·) i dijeljenje (/). Općenito se mogu koristiti različiti završni i funkcijski znakovi (npr. operacije logaritmiranja, trigonometrijske funkcije, logičke...) i one ovise o problemu koji se rješava.

Skupovi završnih (koji se uvijek nalaze u listovima, u dolje navedenim pravilima označeno oznakom T) i funkcijskih znakova (u dolje navedenim pravilima taj je skup označen oznakom F) zajedno čine skup primitiva (engl. *primitives*) za GP. Za primitive vrijede sljedeća pravila:

1. Svaki $t \in T$ je ispravan izraz
2. $f(e_1,\ e_2, \dots, e_n)$ je ispravan izraz ako je $f \in F,\ \arg(f) = n,$ a $e_1,\ e_2, \dots, e_n$ su ispravni izrazi
3. Ne postoje drugi oblici ispravnih izraza

$\mathrm{Arg}(f)$ je broj argumenata funkcije f. Funkcije mogu imati i druge funkcije kao argumente.

Stabla ne moraju biti samo binarna (s najmanje jednim, a najviše dvoje djece), već mogu imati i više djece. Izrazi se najčešće predstavljaju u prefiks notaciji (engl. *prefix, polish notation*) pa se u tom slučaju npr. gornji izraz zapisuje kao:

$$\Big(+(/\ x\ 6)\Big(-(\cdot\ 2\ y)\big(/5\ (+\ z\ 1)\big)\Big)\Big) \tag{3.2}$$

Također, funkcije mogu imati promjenjiv, varijabilan broj argumenata ili mogu imati fiksan broj argumenata pa se prilikom definiranja izraza mogu izostaviti zagrade. Kakav će se prikaz koristiti ovisi o vrsti problema koji se rješava, a o tome pak ovisi ostvarenje genetskih operatora, jer su neki pogodniji za jedan, a neki za drugi oblik rješenja.

3.2.2 Constrained syntax GP

Kod ove vrste rješenja su prikazana stablima koja nemaju samo dva, već imaju više djece, npr. složenije *if* operacije s više od dva uvjeta. Na genetske operatore kod ove vrste moraju se dodati ograničenja kako bi, nakon njihove primjene, novonastala rješenja bila ispravna.

3.2.3 Linearno genetsko programiranje

Kod linearnog genetskog programiranja program je predstavljen nizom instrukcija, kako je prikazano na slici 3.3.

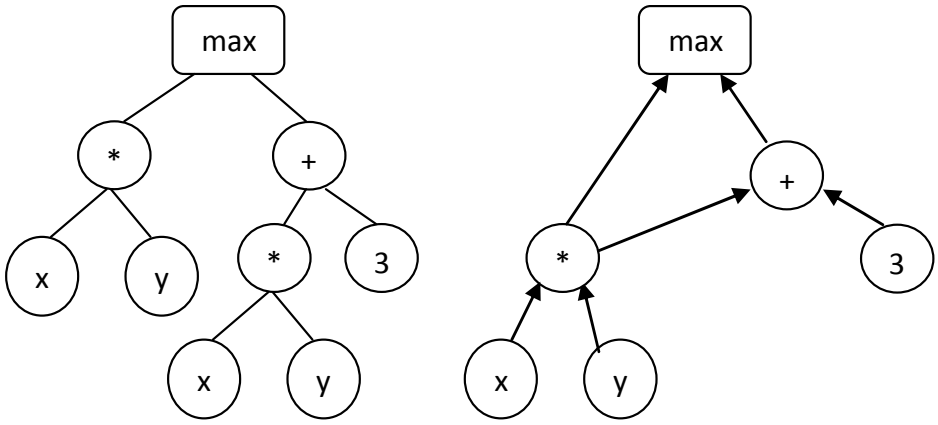
instrukcija 1	instrukcija 2	instrukcija 3	...	instrukcija N

Slika 3.3 Jedinka program

Programi unutar populacije mogu biti različitih duljina. Izvršavanje programa linearnog GP – a može biti brže te nije potrebno imati dodatan interpreter koji će izvršavati programe (kao što je slučaj kod onog temeljenog na stablima). Kod ovih programa obično se ulazni podaci čitaju iz jednog registra te se rezultat operacije sprema u neki drugi registar. Ako nema grananja ili petlji, tada se instrukcije izvršavaju jedna po jedna, dok to nije slučaj kod programa predstavljenih stablima. Postoje operatori križanja i mutacije posebno razvijeni za ovu vrstu GP – a.

3.2.4 Genetsko programiranje temeljeno na grafovima

Ovo je posebna vrsta genetskog programiranja, najpoznatiji je paralelni GP (engl. *parallel distributed GP*). Ova je vrsta prikladna za evoluciju paralelnih programa. Jedinke su ovdje predstavljene grafovima. Na slici 3.4 [41] prikazane su dvije jedinke (max, +, * su funkcijski, a x, y, i 3 su završni znakovi). Na lijevoj strani slike 3.4 prikazana je jedinka u obliku stabla (stablo je graf), a na desnom dijelu nalazi se graf na kojem su spojeni pojedini dijelovi stabla, kako bi se izbjegla ponovna evaluacija istih dijelova.



Slika 3.4 Jedinka u obliku stabla (lijevo) i jedinka u obliku grafa (desno)

3.2.5 Celularno genetsko programiranje

U celularnom ili indirektnom kodiranju rješenja, stabla predstavljaju programe - rješenja koja se koriste za stvaranje složenijih struktura, kao što su složeniji grafovi, neuronske mreže ili Petrijeve mreže. Postoji i posebno kodiranje, tzv. *edge encoding* kojim se predstavljaju jednostavni planarni grafovi.

3.2.6 Genetsko programiranje temeljeno na gramatici

Kod ove vrste definiran je skup pravila za stvaranje populacije rješenja.

3.3 Koraci genetskog programiranja

Koraci genetskog programiranja su jednostavni, kako je prikazano na slici 3.1. Na početku je nekim načinom (najčešće slučajno) potrebno stvoriti početnu populaciju od dostupnih primitiva, tj. završnih i funkcijskih znakova. Mogućem rješenju predstavljenog programom potrebno je pridijeliti mjeru dobrote (engl. *fitness*). Nakon toga potrebno je odabrati pojedine jedinke koje će koristiti genetski operatori nekom metodom selekcije. Genetski operatori su križanje i mutacija; oni se mogu izvršiti jedan iza drugoga ili se može izvršiti samo jedan od njih, a drugi uopće ne.

Križanje ima parametar p_c koji može predstavljati udio populacije koji će sudjelovati u križanju ili vjerojatnost odabira križanja. Mutacija ima parametar p_m koji, također, predstavlja udio populacije koji će sudjelovati u mutaciji ili vjerojatnost odabira mutacije. Kod oba operatora stablo dijete nastalo križanjem ili novo stablo nastalo mutacijom mogu imati veći broj čvorova i veću dubinu od stabala roditelja od kojih su nastali.

Ovisno o mjeri dobrote pojedinih jedinki, određuju se najbolje jedinke trenutne populacije. Najbolja jedinka može biti jedinka s najvećom ili najmanjom mjerom dobrote, što također ovisi o problemu koji se rješava.

Cijeli se proces odvija unutar zadanog broja generacija dok se ne postigne kriterij zaustavljanja. To može biti dostignut unaprijed određen broj generacija, zatim dobiveno zadovoljavajuće rješenje ili ispravno klasificirani svi ili unaprijed određen broj uzoraka, ako se radi o problemu klasifikacije i sl. Nakon toga se prikazuje najbolji rezultat.

Selekcija jedinki te genetski operatori detaljnije su opisani u sedmom poglavlju.

3.4 Problem prebrzog rasta jedinki

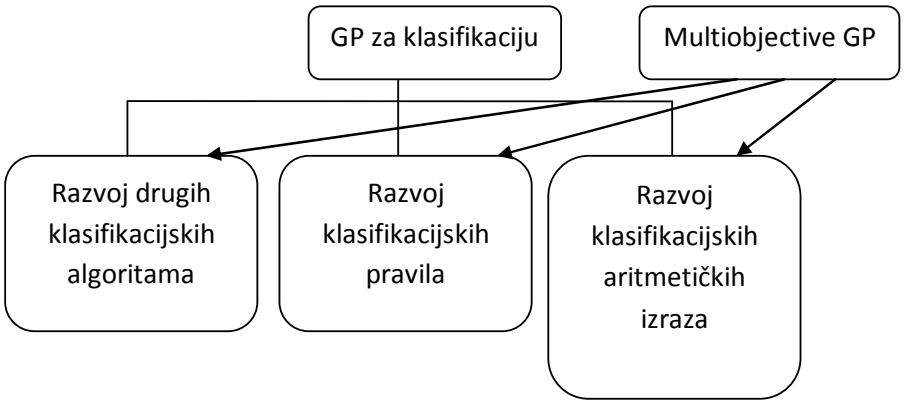
Problem prebrzog rasta jedinki (engl. *bloat*) čest je problem koji kod genetskog programiranja nastaje tijekom procesa rekombinacije i mutacije rješenja te se predlažu različita poboljšanja kako bi se ovaj problem umanjio ili čak spriječio. Npr. kada je jedinka prikazana u obliku stabla, problem se očituje kao nekontrolirani porast veličine stabla, tj. broja čvorova i dubine stabla, a bez nekog poboljšanja mjere dobrote.

Kako bi se problem spriječio, mogu se uvesti mjere kažnjavanja prevelikih jedinki (npr. smanjivanje ili povećanje mjere dobrote takvih jedinki za faktor koji ovisi o broju čvorova ili dubini stabla). Moguće je i unaprijed odrediti najveći dopušteni broj čvorova, dubine stabla ili broj instrukcija (ako se radi o linearnom genetskom programiranju). Također, u rad genetskih operatora mogu se uvesti neka ograničenja kako ne bi nastajale prevelike jedinke, a jedno od njih je podrezivanje stabla (engl. *tree pruning*) ako njegova veličina prekorači dopuštene vrijednosti. Postoji niz problema koji mogu nastati zbog jako velikih jedinki, neki od njih su dulje trajanje evaluacije većih jedinki i nerazumljivost i nečitljivost rješenja za čovjeka. U literaturi je predloženo nekoliko tehnika za rješavanje problema porasta jedinki tijekom evolucijskog procesa, kao što su ograničavanje veličine stabla na 17 [2] ili tzv. *No Limits* pristup gdje se uopće ne ograničava veličina rješenja, zatim *Dyn Depth* pristup [47], gdje se najveća dopuštena veličina na početku ograničava na neku manju vrijednost, a poslije se, po potrebi, povećava za dobivanje najboljeg rješenja, zatim tehnika *operator equalisation* koja kontrolira distribuciju veličina rješenja u populaciji tako što prihvaća svaku novonastalu jedinku s određenom vjerojatnošću ovisno o njenoj veličini [46] i mnoge druge.

4. Primjena genetskog programiranja u strojnom učenju

4.1 Genetsko programiranje primijenjeno na problem klasifikacije podataka

U ovom poglavlju bit će detaljnije objašnjena primjena različitih vrsta GP - a i dan pregled literature primjene stablastog genetskog programiranja za klasifikaciju podataka. Skup funkcijskih i završnih znakova unaprijed je određen. Stablasti GP često se koristi kod klasifikacije različitih vrsta podataka i može se podijeliti u tri različite kategorije, kako je prikazano na slici 4.1 [26]. Prvoj kategoriji pripada razvoj različitih drugih klasifikacijskih algoritama, kao što su stabla odluke, neuronske mreže i sl. Drugoj metodi pripada razvoj pravila za klasifikaciju (engl. *classification rules*) ili logičkih izraza s logičkim operatorima, a trećoj razvoj aritmetičkih izraza ili funkcija.



Slika 4.1 Podjela genetskog programiranja za klasifikaciju

Multiobjective GP je GP koji se koristi za stvaranje drugih klasifikatora svih triju prethodno spomenutih tipova (zato on, na slici 4.1, strjelicama pokazuje prema svim ostalim tipovima GP – a za klasifikaciju).

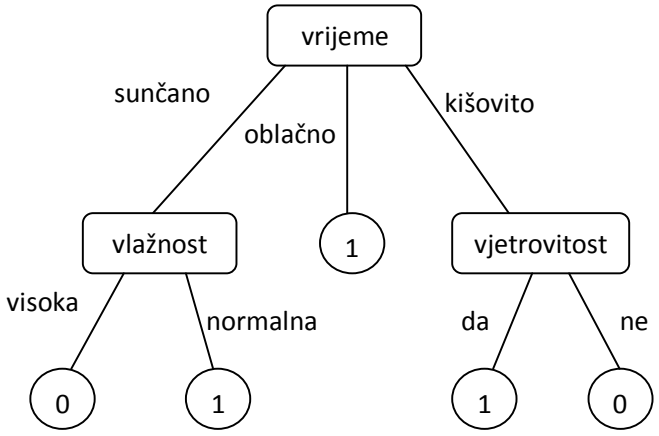
4.1.1 Razvoj klasifikacijskih algoritama

Kada se genetsko programiranje koristi za razvoj drugih klasifikacijskih algoritama postoji unaprijed niz pravila uz čiju se pomoć stvaraju moguća rješenja. Drugi klasifikacijski algoritmi koji se mogu razviti uz pomoć GP – a su najčešće stabla odluke (engl. *decision trees*), zatim stabla odluke temeljena na neizrazitoj logici (engl. *fuzzy decision trees*), neuronske mreže i sl. Definiraju se i posebni genetski operatori (križanje i mutacija) kako bi novonastala rješenja bila ispravna.

4.1.1.1 Razvoj stabala odluke

Genetsko programiranje se često koristi za stvaranje stabala odluke koja će klasificirati podatke. John Koza je prvi predstavio ovaj način rješavanja problema klasifikacije [31]. U svom radu uzima jednostavan skup podataka koji se sastoji od četiri atributa (vrijeme, temperatura, vlažnost, vjetrovitost), svaki od tih atributa može poprimiti nekoliko različitih vrijednosti; temperatura može poprimiti vrijednosti visoka, srednja, niska; vlažnost može biti visoka ili niska; vjetrovitost poprima vrijednosti da ili ne (u smislu je li vjetrovito ili nije), a vrijeme poprima vrijednosti sunčano, oblačno, kišovito. Svaki uzorak se svrstava u jedan od dva moguća razreda (pozitivno ili negativno). Populacija rješenja, tj. programa sastoji se od simboličkih (S – izraza) u LISP programskom jeziku. Kako bi se stabla odluke mogla stvoriti genetskim programiranjem, svaki je atribut pretvoren u funkciju. Npr. atribut *vrijeme* pretvoren je u funkciju koja, ako je vrijeme oblačno, vraća svoj drugi argument, drugo podstablo. Skup funkcijskih znakova je {TEMP, HUM, OUT, WIND}, prema skraćenim nazivima engleskih riječi *temperature, humidity, outlook, windy*. Svaka od tih fukcija ima redom 3, 2, 3 i 2 argumenta. Skup završnih znakova sastoji se samo od dviju vrijednosti 0 i 1, jer postoje samo dva razreda. Ova su stabla slična onima koje koristi algoritam ID3.

Primjer stabla kako ga je predstavio *J. Koza* u svom radu nalazi se na slici 4.2.

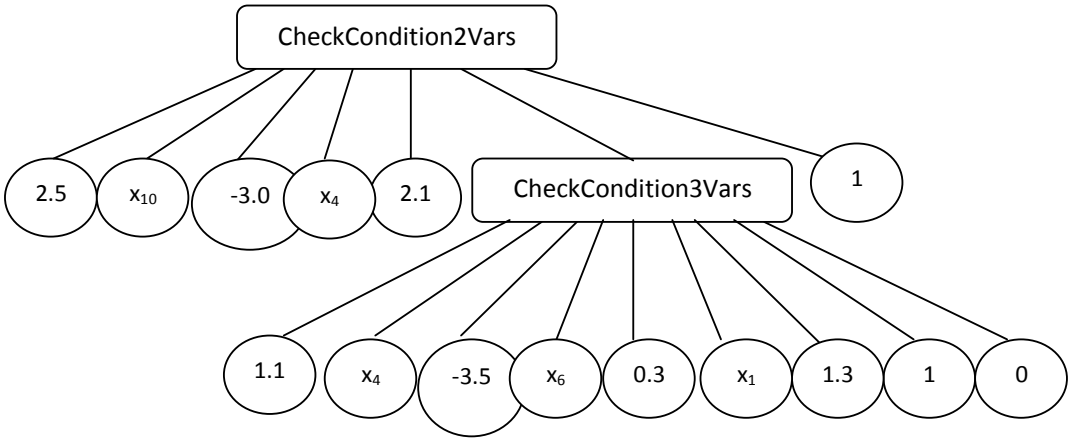


Slika 4.2 Primjer stabla

U listovima se nalaze oznake razreda. Pored grana stabala prikazane su neke vrijednosti pojedinih atributa iz ostalih čvorova stabla. U nastavku se nalazi pripadni S – izraz gore prikazanog stabla. WIND 1 1 predstavlja samo jedan atom, jer, bez obzira na vrijednost atributa WIND (vjetrovitost), odluka će uvijek biti 1 pa je umjesto WIND 1 1 u stablu prikazan samo list s oznakom 1.

(OUT (HUM 0 1) (WIND 1 1) (WIND 1 0))

U [34] predstavljen je razvoj stabala odluke sličnih onima u algoritmu C4.5 uz pomoć metode *Gene Expression Programming*. Marmelstein [36] i Bojarczuk [2] su koristili predefinirana sintaksna pravila za stvaranje stabala odluke i standardne genetske operatore za njihov daljnji razvoj. Folino [16] je koristio hibridno genetsko programiranje zajedno sa simuliranim kaljenjem za razvoj stabala odluke. Bot [5] je koristio linearna stabla odluke. Prikaz jednog takvog stabla nalazi se na slici 4.3.

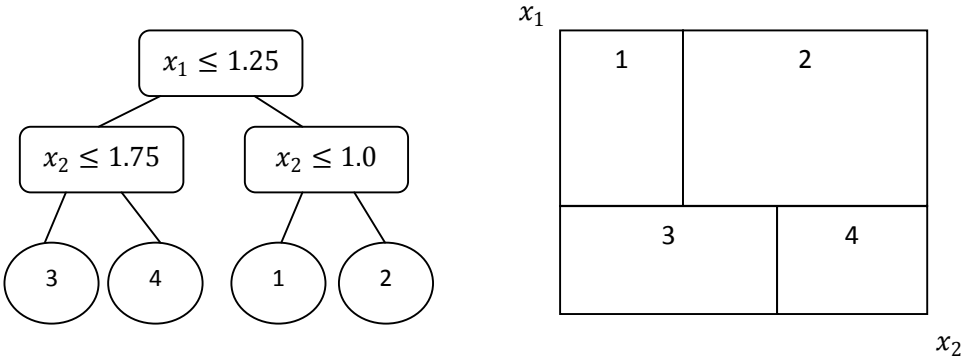


Slika 4.3 Primjer linearnog stabla odluke

Ovo je stablo različito od uobičajenih stabala odluke, jer čvorovi mogu imati više od dvoje djece. Kao što se vidi na slici 4.3, prvi čvor (*CheckCondition2Vars*) ima sedmero čvorova djece, od kojih se šest sastoji od završnih znakova, a jedan ima funkcijski znak (*CheckCondition3Vars*), koji pak ima devetero čvorova djece. Ovakva, tzv. linearna stabla odluke sadrže čvorove koji se sastoje od linearnih kombinacija, tj. od aritmetičkih izraza s više varijabli. Prvi čvorovi djeca čvora roditelja su konstante i varijable, a zadnji čvorovi djeca predstavljaju oznake razreda. Tako, čvorovi djeca korijena gornjeg stabla predstavljaju aritmetički izraz $2.5 \cdot x_{10} - 3 \cdot x_4 \leq 2.1$, a čvorovi djeca čvora *CheckCondition3Vars* predstavljaju izraz $1.1 \cdot x_4 - 3.5 \cdot x_6 + 0.3 \cdot x_1 \leq 1.3$. Stablo se interpretira tako da se gleda je li izraz $2.5 \cdot x_{10} - 3 \cdot x_4$ stvarno manji ili jednak od vrijednosti 2.1, ako jest, tada je odluka sljedeći čvor *CheckCondition3Vars*, u suprotnom je odluka na razredu 1. Nadalje, ako je izraz $1.1 \cdot x_4 - 3.5 \cdot x_6 + 0.3 \cdot x_1$ stvarno manji ili jednak od 1.3, tada je odluka na razredu 1, inače na razredu 0. U čvorovima se može nalaziti bilo kakav aritmetički izraz. Skup završnih znakova ovdje se sastoji od skupa varijabli, kojih ima koliko i atributa u skupu podataka. Ako je ukupan broj atributa jednak n, varijable tada imaju oznake od x_0 do x_{n-1}. Prilikom evaluacije pojedinog čvora, za pojedinu varijablu uzima se odgovarajuća vrijednost iz skupa podataka. Završne znakove, kao što je vidljivo iz primjera stabla, čine i konstante koje se nalaze u nekom intervalu, u ovom slučaju u intervalu $[-10, 10]$ te oznake razreda (ako je broj razreda k), tada se oznake razreda kreću od 0 do k - 1.

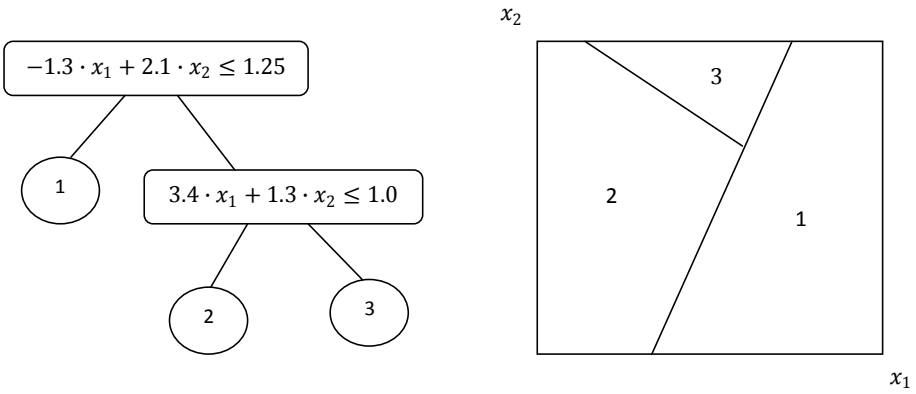
Funkcijski znakovi su *CheckCondition1Var, CheckCondition2Vars* i *CheckCondition3Vars* koji evaluiraju pojedini aritmetički izraz s jednom, dvije ili tri varijable.

Ova su linearna stabla odluke poznatija pod imenom *oblique decision trees* i različita su od tzv. *axis – parallel decision trees*, od češće upotrebljavanih stabala odluke. Na sljedećim slikama prikazana je razlika između ovih dviju vrsta stabala odluke. Linearna stabla odluke u čvorovima imaju izraze koji mogu imati jednu, ali i više od jedne varijable, dok nelinearna u čvorovima uvijek imaju izraze sa samo jednom varijablom. Hiperravnine koje kod linearnih stabala odluke, u prostoru atributa, odjeljuju pojedine razrede nisu paralelne s osima, već mogu prema osima imati bilo kakvu orijentaciju, dok ova druga vrsta stvara podjelu u prostoru atributa hiperravninama koje su paralelne s osima. Na slici 4.4 [5] prikazano je nelinearno stablo odluke i podjela u dvodimenzionalnom prostoru atributa koje ono stvara.



Slika 4.4 Nelinearno stablo odluke i odgovarajuća podjela 2D prostora atributa

Na slici 4.5 [5] prikazano je linearno stablo odluke i podjela u dvodimenzionalnom prostoru atributa koje ono stvara.



Slika 4.5 Linearno stablo odluke i odgovarajuća podjela 2D prostora atributa

U čvorovima ovakvih stabala općenito se nalazi izraz oblika $\sum_i c_i \cdot x_i \leq prag$ te se, ako je izraz istinit, vraća vrijednost desne grane, a u protivnom se vraća vrijednost lijeve grane. Genetsko programiranje može se koristiti za razvoj ovakvih stabala, pri čemu se koriste standardni genetski operatori.

Zatim se u literaturi spominje i tzv. *buildingblock* pristup, kod kojeg se stabla odluke, tijekom evolucije, grade od jednostavnijih prema složenijima [15]. *Eggermont* predstavlja atome za stabla odluke (engl. *atomic representation*) [11]; ovaj će pristup detaljnije biti objašnjen u sedmom poglavlju i ostvaren u praktičnom dijelu ovog rada. U [53] je objašnjeno korištenje genetskog programiranja temeljenog na gramatici i predložena je kontekstno neovisna gramatika (engl. *context free grammar*) za razvoj stabala odluke, neuronskih mreža bez povratnih veza, sustava temeljenih na neizrazitoj logici i Petrijevih mreža uz pomoć posebne vrste kodiranja rješenja. Korištenje gramatike za stvaranje rješenja smanjuje prostor potrage samo na ispravna rješenja, dok se, kad se unaprijed određena gramatička pravila ne koriste, moraju dodavati različita druga ograničenja na operatore kako bi se pojavljivala i u evoluciji sudjelovala samo potpuno ispravna rješenja. Predstavljen je i samostalni GP rješavač (engl. *Autonomous GP Solver*) [39] koji može sam odrediti zna li riješiti dani problem ili ne te se primjenjuje na probleme klasifikacije, kao i na probleme regresije. Za rješavanje klasifikacijskih problema razvijanjem drugih klasifikacijskih algoritama predstavljeno je i nekoliko inačica paralelnog genetskog programiranja.

4.1.1.2 Postupno prilagođavanje težina

Postupno prilagođavanje težina (engl. *Stepwise Adaptation of Weights, SAW*) [12] je još jedna metoda koja nije vezana isključivo za genetsko programiranje, nego bi se mogla koristiti i u drugim evolucijskim algoritmima. Na slici 4.6 detaljnije je prikazan način rada ove metode.

```
On – line osvježavanje težina (dobrote f) {
        postavi početne vrijednosti težinama;
        dok nije zadovoljen kriterij  zaustavljanja {
                za TP idućih evaluacija
                        neka GP radi s trenutnom dobrotom f;
                redefiniraj težine i ponovo izračunaj dobrotu jedinki;
        }
}
```

Slika 4.6 Način rada metode postupnog prilagođavanja težina

Kod ove metode GP tijekom rada sam prilagođava težine, tj. dobrotu jedinki dok se ne dobije neko zadovoljavajuće rješenje. Obično se za T_P uzimaju vrijednosti oko 200 do 1000. Na početku se svim težinama zadaje ista početna vrijednost, tipično 1. Nakon svakog perioda od T_P evaluacija, težinama uzoraka koje je najbolje do sad pronađeno stablo pogrešno klasificiralo dodaje se vrijednost $\Delta\omega$ te se, u skladu time ponovo računa i mjera dobrote. Dobrota se računa formulom (4.1).

$$f(x) = \sum_{r \in D} \omega_r \cdot error(x, r) \tag{4.1}$$

D je skup svih podataka, ω_r je težina uzorka r, a $error(x,r)$ mjeri pogrešku klasifikacije, tj. $error(x,r)$ je jednak 1, ako je uzorak r pogrešno klasificiran, a 0 inače.

4.1.2 Razvoj logičkih klasifikacijskih pravila

Genetsko se programiranje može koristiti i za stvaranje i razvoj logičkih klasifikacijskih pravila koja su obično u obliku ako – onda pravila. Stabla odluke mogu se prikazati uz pomoć niza ako – onda pravila, gdje je svaki put od korijena do pojedinog lista predstavljen jednim takvim izrazom. Ako – onda pravila mogu se izvući i izravno iz skupa podataka. Računalni program kao jedinka, tj. moguće rješenje predstavljeno stablom je zapravo jedno pravilo koje se sastoji od završnih i unaprijed definiranih logičkih funkcija. Završni i funkcijski znakovi (atributi) nalaze se u uvjetu, a oznaka razreda (ciljnog atributa) nalazi se u akciji pravila.

U [17], *Freitas* je predstavio radni okvir za korištenje genetskog programiranja u problemima strojnog učenja. *SQL* upitom prikazanim na slici 4.7 određuje se dobrota jedinke, a ona je predstavljena strukturom koja se naziva *Tuple Set Descriptor (TSD)*. Dobrota jedinke određuje se izvršavanjem SQL upita. Prednost ove metode je što nema redundancije, zatim skalabilnost, prenosivost, privatnost podataka i mogućnost paralelnog izvršavanja na *SQL* poslužiteljima.

```
Select      Goal – Attribute, Count(*)
From        Mine - Relation
Where       Tuple – Set – Descriptor
Group By    Goal - Attribute
```

Slika 4.7 SQL upit za izračun dobrote jedinke

Pretpostavlja se da se svi podaci koje treba klasificirati nalaze u relacijskoj bazi podataka, u *Mine – Relation*. Gore naveden *SQL* upit može se napisati i u obliku pravila:

ako (*Tuple Set Descriptor*) tada (Razred = C_i)

TSD je izraz koji se sastoji od atributa, operatora usporedbe $>, <, \geq, \leq, =, \neq$ te logičkih operatora AND, OR, NOT.

Nakon izvršavanja SQL upita sa slike 4.7 dobivaju se vrijednosti iz tablice 4.1. Ta je tablica veličine 2xn, gdje je n broj vrijednosti ciljnog atributa (eng. *Goal – Attribute*) ili broj razreda.

Tablica 4.1 Tablica dobivena nakon izvršavanja SQL upita

	$G_1, G_2, ..., G_n$	ukupno
zadovoljava TSD	$C_{11}, C_{12}, ..., C_{1n}$	C_{1+}
ne zadovoljava TSD	$C_{21}, C_{22}, ..., C_{2n}$	C_{2+}
ukupno	$C_{+1}, ..., C_{+n}$	C_{++}

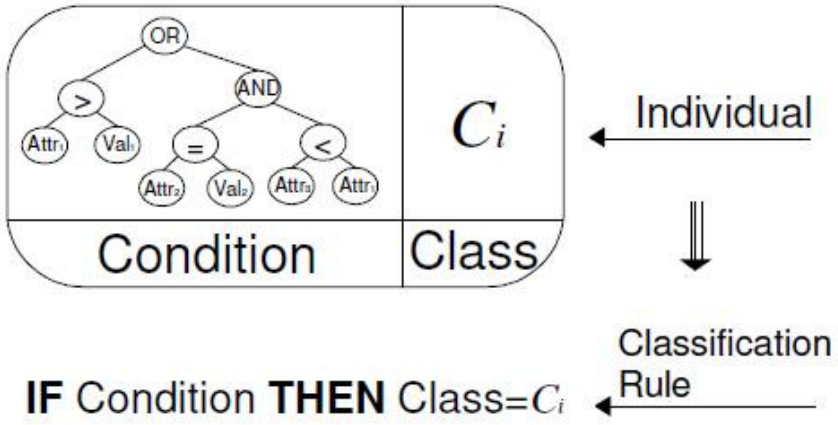
U prvom redu tablice nalaze se vrijednosti oblika C_{1j}, gdje je j = 1, ..., n i svaka od tih vrijednosti prikazuje broj n – torki koje zadovoljavaju TSD i imaju oznaku razreda G_j. U drugom retku tablice prikazane su vrijednosti oblika C_{2j}, gdje je j = 1, ..., n i svaka

od tih vrijednosti prikazuje broj n – torki koje ne zadovoljavaju TSD, a imaju oznaku razreda G_j. C_{1+}, C_{2+} je zbroj vrijednosti po pojedinom retku, a s C_{++} označen je ukupan broj n – torki. Odabire se oznaka G_j onog razreda koji u prvom retku tablice ima najveću vrijednost $C_{max} = max_{j=1}^{n} C_{1j}$.

Slično, u [51] je predstavljeno paralelno, celularno genetsko programiranje za rješavanje problema binarne klasifikacije. Klasifikacijsko pravilo glasi:

ako *uvjet* tada *ciljni atribut = C*, gdje je C oznaka razreda

Uvjet u lijevom dijelu pravila predstavljen je stablom – jedinkom celularnog GP – a, a prethodno navedeni *SQL* upit i ovdje se koristio za evaluaciju pojedine jedinke. Jedinka GP – a i pripadajuće klasifikacijsko pravilo prikazani su na slici 4.8 [51]. Konačno rješenje je stablo koje ispravno klasificira najveći broj uzoraka iz zadanog skupa podataka.



Slika 4.8 Jedinka celularnog genetskog programiranja i pripadajuće ako – onda pravilo

Ovaj celularni GP za svaku jedinku definira susjedstvo (npr. *von Neumann*, ako jedinka ima četiri susjeda ili *Moore* ako ih ima osam) i stavlja ih u jednu višedimenzionalnu mrežu, gdje svaka jedinka ima jednak broj susjeda - četiri ili osam. Paralelizacija ovakvog pristupa može se izvršiti tako da se svakoj ćeliji u mreži pridijeli jedan procesor. Svaki procesor komunicira samo sa svojim susjedima, a vrijednosti koje se pritom razmjenjuju su obično vrijednost dobrote, a rijetko stablo, tj. jedinka. Kako se jedinka evaluira uz pomoć izvršavanja *SQL* upita (tako joj se pridjeljuje dobrota), kad nekoj ćeliji zatreba vrijednost dobrote, pripadni procesor tada može komunicirati s *SQL* poslužiteljem na kojem će se izvršiti upit i tako dobiti potrebna vrijednost dobrote.

Eggermont [11] je predstavio stabla odluke koja se mogu preoblikovati u niz ako – onda pravila koja su, u najjednostavnijem slučaju, oblika *atribut – operator - vrijednost*. Operator je ovdje logička funkcija (te zato pripada ovoj kategoriji razvoja logičkih pravila). Metoda je primjenjiva na atribute i numeričkih i nominalnih vrijednosti.

Wong [56] je koristio GP za razvoj logičkih pravila temeljenih na induktivnom logičkom programiranju. Za definiranje pravila korištena je ovdje i kontekstno neovisna gramatika. Sustav se zove *Logic GENetic PROgramming system (LOGENPRO)*.

Huang [24] je razvio GP koji radi u dva stadija (engl.*Two – stage genetic programming, S2GP*). U prvom se stvaraju ako – onda pravila, a u drugom diskriminirajuće funkcije za primjere iz skupa podataka koji nisu pokriveni prvim dijelom. Ova je metoda bolje rješavala različite tzv. *credit classification* probleme (najčešće pitanje dodijeliti kredit ili ne) od mnogo starijih i poznatijih klasifikacijskih algoritama kao što su *CART* ili *C4.5*. Zatim se GP koristio za stvaranje pravila temeljenih na neizrazitoj logici [37] ili pravila u obliku disjunktivne normalne forme [4] [37].

Lin [35] je predložio višeslojni GP (engl. *layered genetic programming*), kod kojega različiti slojevi odgovaraju različitim populacijama te se tako odvijaju izlučivanje značajki i klasifikacija. Prethodno navedene metode uglavnom se koriste na skupovima podataka koji sadrže samo nominalne atribute. Ako atributi imaju i numeričke vrijednosti, tada se unaprijed moraju uvesti neka ograničenja pa je ta vrsta genetskog programiranja poznata po nazivu *constrained – syntax GP*. Provodi se i pretvaranje podataka u diskretne i korištenje *boolean* funkcija (engl. *booleanization of data*) [11].

4.1.3 Razvoj klasifikacijskih aritmetičkih izraza

Aritmetički izrazi koriste atribute kao varijable te se upotrebljavaju za klasifikaciju podataka čije su vrijednosti numeričke - cjelobrojne ili realne. Zaključak (desni dio) nekog pravila ovdje je predstavljen jednom realnom vrijednošću. U složenijem slučaju, kad se radi o primjeni genetskog programiranja na višekasnu klasifikaciju, problem se može riješiti binarnom dekompozicijom [48]. Kod ove tehnike razvijen je klasifikator za svaki razred, gdje se svi ostali razredi privremeno gledaju kao jedan (nepoželjan) razred. Svi klasifikatori sudjeluju u stvaranju jednog, konačnog klasifikatora. Odluka se temelji na izlazima svih klasifikatora, a onaj koji ima pozitivan ili najveći izlaz moći će donijeti konačnu odluku, bit će pobjednik. *Durga* [38] je predložio kromosom (vektor) koji se sastoji od klasifikatora za sve razrede i za razvoj mu je potrebno samo jedno pokretanje GP - a. *Mendes* [37] je pak predložio razvoj

dviju populacija istovremeno; prva sadrži pravila temeljena na neizrazitoj logici, tzv. *fuzzy* pravila, a druga sadrži tzv. *membership* funkcije.

4.2 Prednosti i nedostaci primjene genetskog programiranja za izgradnju klasifikatora

U nastavku su ukratko opisane prednosti i nedostaci primjene genetskog programiranja na problem klasifikacije podataka.

4.2.1 Prednosti

Jedna od prednosti korištenja genetskog programiranja je njihova robusnost i mogućnost rada sa zašumljenim podacima. Također, GP izvodi globalno pretraživanje prostora, dok većina ostalih algoritama strojnog učenja koji se često koriste za klasifikaciju podataka izvode pohlepno (engl. *greedy*) pretraživanje prostora. Zbog globalnog pretraživanja prostora veća je vjerojatnost pronalaženja ispravnog rješenja, tj. globalnog optimuma (a kod klasifikacije je obično važna točnost), dok ostali algoritmi, poput neuronskih mreža gotovo uvijek pronalaze samo lokalni optimum i pri njemu ostaju. Za razliku od jedinki nepromjenjive, fiksne duljine koje se koriste kod genetskih algoritama, GP može koristiti različite funkcije, varijable, konstante za konstrukciju svojih rješenja koja su uvijek promjenjive duljine. Klasifikator koji koristi GP je puno razumljiviji od klasifikatora koji koriste npr. neuronske mreže. Također, GP može raditi s malim brojem podataka i nije potrebno koristiti baš sve dostupne podatke za klasifikaciju, a uglavnom i radi s izvornim podacima (rijetko su potrebni pretprocesiranje i transformacija podataka, što dokazuju inačice GP – a koje mogu raditi i s numeričkim i nominalnim atributima i sl.) .

4.2.2 Nedostaci

Jedan od glavnih nedostataka korištenja GP – a je česta potreba za evaluacijom jedinki tijekom rada. Ako je veličina populacije m, a broj generacija n, tada će ukupno biti potrebno $m \cdot n$ evaluacija (izračuna i osvježavanja vrijednosti dobrote). Također, čest problem GP – a je tzv. *bloat*. Tijekom rada jedinke rastu, a time se najčešće ne povećava točnost klasifikatora (jednostavniji klasifikatori obično imaju veću točnost klasifikacije na skupu za ispitivanje). Velika rješenja su i manje razumljiva i čitljiva te ih je teže interpretirati.

5. Metode evaluacije modela

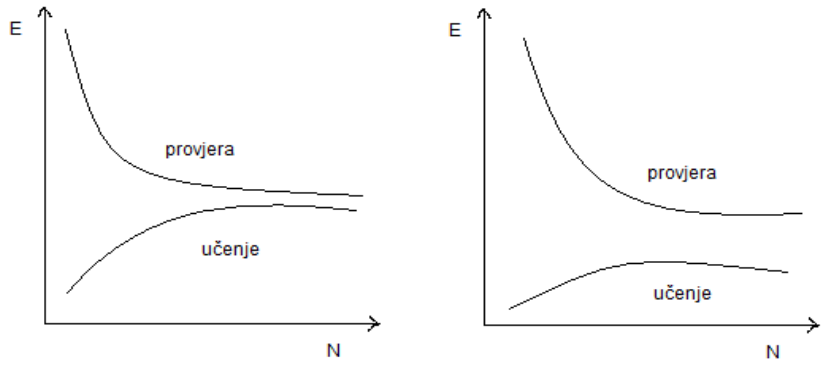
Ciljevi statističkih metoda evaluacije modela su odabir modela i procjena uspješnosti njegovog rada. Uspješnost se određuje procjenom (vjerojatnosti) stvarne pogreške modela. Model je najćešće regresijski model (onaj koji se koristi za određivanje kontinuiranih funkcija na temelju dostupnih podataka) ili klasifikacijski model (klasifikator čija je zadaća opisana u drugom poglavlju). U idućim potpoglavljima podrazumijeva se klasifikacijski model. Kod evaluacije se procjenjuje vjerojatnost neispravne klasifikacije $P(y \neq \hat{y})$, $\hat{y}$ je ovdje oznaka razreda kojeg je procijenio model za uzorak y, a formulom (5.1) određuje se pogreška klasifikacije.

$$PE = \frac{FP+FN}{N} \tag{5.1}$$

N je ukupan broj primjera. PE je procjena pogreške (engl. *prediction error, missclassification rate, classification error*). FP je broj pogrešno klasificiranih uzoraka kao pozitivnih (engl. *false positives*), a FN je broj pogrešno klasificiranih kao negativnih (engl. *false negatives*).

Dostupan skup podataka dijeli se na skup za učenje (engl. *training set*) i skup za ispitivanje (engl. *test set*) ako se želi odrediti pogreška već postojećeg, odabranog modela. Ako se želi odabrati klasifikacijski model, tada se cijeli dostupan skup podataka, uz prethodno navedene skupove, dijeli i na skup za provjeru (engl. *validation set*). Ovaj skup koristi se za fino podešavanje parametara modela. Pogreška generalizacije (to je pogreška koja se određuje na temelju skupa podataka različitog od onog koji je korišten za učenje) određuje se na skupu za ispitivanje (ovdje se više ne vrši podešavanje parametara). Poželjno je da sva tri skupa budu međusobno disjunktna [7]. Empirijska pogreška nije dobra procjena stvarne pogreške, jer se određuje na skupu nad kojim je klasifikator učio (tada je model previše specijaliziran). Isto tako, poželjno je da skup za učenje ne bude premali, ali niti prevelik. Ako je skup za učenje prevelik, tada je teže pronaći model koji bi ispravno klasificirao sve primjere za učenje. U tom slučaju pogreška na skupu za učenje će rasti, a takvo ponašanje nije poželjno. Kod modela koji imaju visoku pristranost u nekom će se trenutku empirijska i generalizacijska pogreška izjednačiti, što znači da je model prejednostavan da bi bio dobar klasifikator za dani skup podataka, a ako je pak pogreška na skupu za provjeru veća od pogreške na skupu za učenje, tada model ne generalizira dobro, tj. ne klasificira ispravno još neviđene primjere. Na slici 5.1 [8] prikazana je ovisnost empirijske pogreške i pogreške generalizacije o broju primjera za učenje za podnaučeni i za prenaučeni model. N je ukupan broj primjera

za učenje, E je pogreška. Važno je da se u skupu primjera za učenje nalazi većina informacija o svojstvima razreda kojima uzorci pripadaju.



Slika 5.1 Ovisnost empirijske i pogreške generalizacije o broju uzoraka za učenje N za podnaučeni (lijevo) i prenaučeni (desno) model

Postoje jednostruke i višestruke metode evaluacije modela. Višestruke metode daju bolju procjenu stvarne pogreške modela. U tablici 5.1 prikazan je način podjele skupa kod jednostrukih i višestrukih metoda. U tablici je *n* ukupan broj primjera, *j* je veličina skupa za učenje (koji se može mijenjati od 1 do *n*), a *B* je ukupan broj podjela [19].

Tablica 5.1 Način podjele skupa podataka kod jednostrukih i višestrukih metoda

	jednostruka metoda	**višestruka metoda**
broj primjera za učenje	j	j
broj primjera za ispitivanje	n – j	n - j
broj iteracija	1	B << n

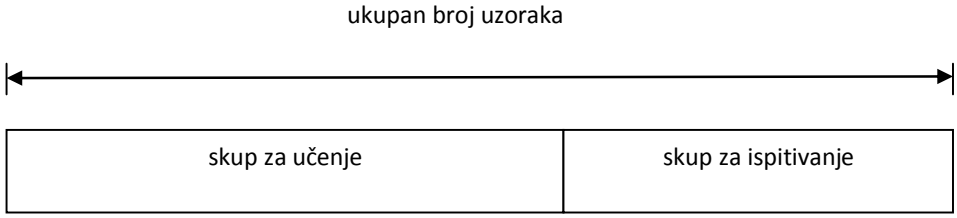
U nastavku su nabrojane i detaljnije opisane pojedine tehnike evaluacije modela, a neke od njih su ostvarene i u praktičnom dijelu ovog rada.

5.1 Jednostruke metode

Kod jednostrukih metoda dostupan se skup podataka samo jednom dijeli na skup za učenje i skup za ispitivanje. Problem ovih metoda je upravo samo jedna podjela cijelog skupa podataka, tj. određivanje skupa primjera za učenje, jer je moguće dobiti nekarakterističan skup za učenje nad kojim nije moguće dobro naučiti hipotezu koja će klasificirati podatke. Najpoznatija jednostruka metoda je tzv. *Holdout* metoda.

5.1.1 Holdout metoda

Kod ove metode skup podataka odvaja se (engl. *held – out*) na skup za učenje i skup za ispitivanje. Prednost ove metode je jednostavnost i mala računalna složenost. Kako bi procjena pogreške bila što bolja, potrebno je pažljivo podijeliti skup na dva nova skupa, jer će rezultat ovisiti o tome koji su primjeri u skupu za učenje, a pogotovo koji su u skupu za ispitivanje. Na slici 5.2 prikazan je jedan mogući način podjele podataka. Ova je metoda poznata i kao jednostavna unakrsna provjera (engl. *simple cross – validation*). Podjela skupa te učenje i ispitivanje bi se, radi poboljšanja rezultata, mogli ponoviti u više iteracija, međutim, uobičajena je samo jedna iteracija.



Slika 5.2 Jedan način podjele skupa kod Holdout metode

5.2 Višestruke metode

Kod višestrukih metoda (tzv. višestruke t&t – *train and test* metode) model se uči nad svakim primjerom iz skupa podataka. Skup podataka dijeli se više puta na niz slučajno odabranih primjera (engl. *random subsampling*), a ukupna pogreška je srednja pogreška, tj. srednja vrijednost pogrešaka svake od odabranih podjela skupa podataka. Najpoznatija višestruka metoda je unakrsna provjera (engl. *cross – validation*).

5.2.1 Metoda ponovne zamjene

Kod metode ponovne zamjene (engl. *resubstitution method*) isti se skup primjera koristi i za učenje i za ispitivanje [44] i to je jednostavna metoda. Ova metoda koristi sve dostupne uzorke, ali joj je najveći problem prenaučenost. Model će pokazivati dobre rezultate na skupu za učenje, ali će rezultati na skupu za ispitivanje biti jako loši (model ne generalizira dobro).

5.2.2 Unakrsna provjera

Unakrsna provjera odvija se u više iteracija. Glavni se skup dijeli na više dijelova, a u svakoj se iteraciji drugi dio skupa koristi za ispitivanje. Za svaku podjelu ponovo se izvršavaju učenje nad skupom za učenje i ispitvanje nad skupom za ispitivanje te se nad potonjim skupom određuje pogreška za tu određenu podjelu. Konačna pogreška je srednja vrijednost pogrešaka pojedinih podjela. Metoda unakrsne provjere dijeli se na k – struku unakrsnu provjeru (engl. *k – fold cross – validation*) i na *LOO* metodu (engl. *Leave – one – out*). Unakrsnu validaciju prvi je put predložio *Kurtz* (1948. god., *simple cross – validation*), proširio ju je Mosier (1951. god., *double cross – validation*) te nekoliko desetljeća kasnije još proširili Krus i Fuller (1982. god., *multicross – validation*).

5.2.2.1 K – struka unakrsna provjera

Glavni skup podataka dijeli se na k dijelova u svakoj od k iteracija, pri čemu se jedan dio koristi za ispitivanje, a preostalih k – 1 koristi se za učenje. Na slici 5.3 prikazan je način rada k – struke križne validacije [23]. Neka je ukupan broj podjela k jednak 4. Sivom bojom označeni su skupovi za ispitivanje te je vidljivo kako se kod svake iteracije uzima za ispitivanje onaj dio skupa koji se u prethodnoj iteraciji još nije koristio za ispitivanje.

Veličine pojedinih dijelova skupa su kod svake podjele otpilike uvijek iste (engl. *roughly – equal – sized*). Prednost ove metode je što se, na kraju, svaki primjer jednom nađe i u skupu za učenje i u skupu za ispitivanje pa, za razliku od prethodno navedene *Holdout* metode rezultat ne ovisi o tome kako će skup biti podijeljen. Za broj podjela k najčešće se uzima 10. Ako je broj podjela jako velik, tada će model biti nepristran, varijanca modela bit će velika, ali će i računanje biti dugotrajno (velik broj iteracija, svaki put se ponovo pokreće isti proces). Ako je pak broj podjela premali, tada je prednost što će izvršavanje trajati kraće i varijanca modela će biti manja, ali je nedostatak što će pristranost biti velika.

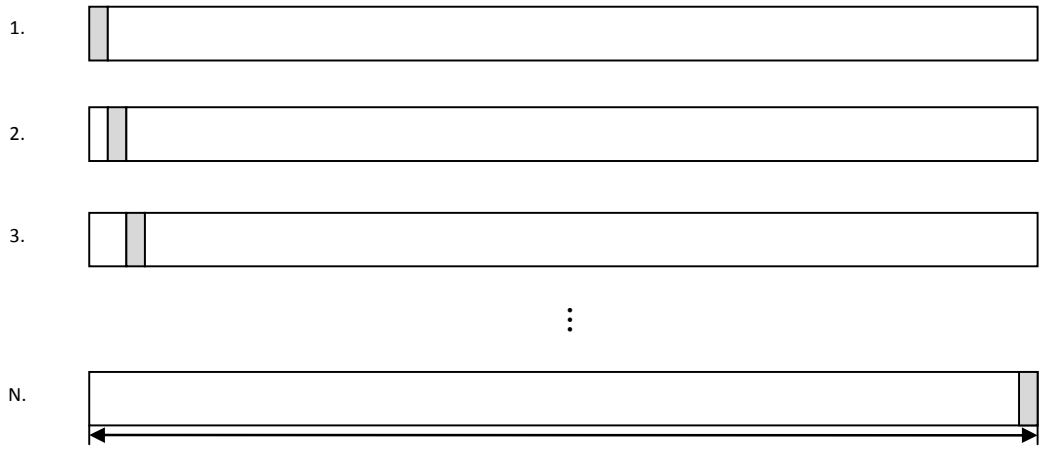
Slika 5.3 Podjela skupa podataka kod k – struke unakrsne provjere

Podatke je prije podjele potrebno bolje organizirati. Najbolje je ako se u svakoj grupi nalazi otprilike jednako uzoraka – predstavnika svakog od razreda. Ova se metoda također može više puta ponoviti (engl. *repeated k – fold cross – validation*), s tim da bi se redoslijed uzoraka prije svakog ponavljanja trebao promijeniti (npr. slučajno uzimanje uzoraka u pojedinu grupu, pri čemu treba paziti da budu zastupljeni predstavnici svakog razreda). U literaturi se često spominje i tzv. 5 x 2 – struka unakrsna validacija (pet puta se ponavlja dvostruka validacija s izmijenjenim uzorcima u pojedinoj grupi), ali je ipak 10 – struka validacija još uvijek najćešće korištena metoda.

5.2.2.2 Leave – one – out

Ova metoda poseban je slučaj k – struke križne validacije. Za parametar k uzima se ukupan broj primjera iz skupa podataka, što znači da se vrši N podjela (ako je N ukupan broj primjera). Za svaku podjelu (iteraciju) nad N – 1 primjera klasifikator ponovo uči, a nad preostalim primjerom (u svakoj iteraciji različit) vrši se ispitivanje i određivanje pogreške. Ukupna pogreška određuje se na isti način kako je opisano u prethodnom potpoglavlju. Na slici 5.4 prikazana je podjela ukupnog broj primjera kroz k = N iteracija [45]. Sivom bojom označeni su skupovi za ispitivanje.

Nedostatak ove metode je prevelika računska složenost ako se radi o jako velikom broju uzoraka u skupu podataka, zato se ona najčešće koristi kad postoji manji skup uzoraka.



Slika 5.4 Podjela skupa podataka kod LOOCV metode

U tablici je prikazana razlika između *LOO* metode i 10-struke unakrsne provjere [19]. N je broj svih primjera (uzoraka).

Tablica 5.2 Razlika između LOO metode i 10 – struke unakrsne validacije

	Leave – one - out	**10 – struka unakrsna validacija**
broj uzoraka za učenje	N – 1	90%
broj uzoraka za ispitivanje	1	10%
broj iteracija	N	10

Računanje ukupne pogreške kod gore objašnjenih metoda navedena je u formuli (5.2). E je ukupna pogreška, a E_i je pogreška nakon svake od k podjela skupa.

$$E = \frac{1}{K}\sum_{i=1}^{K} E_i \tag{5.2}$$

5.2.3 Bootstrapping

Bootstrapping metoda je posebno pogodna za procjenjivanje pogreške modela kod problema klasifikacije kada je dostupan samo mali broj uzoraka, tj. kada je dostupan mali skup podataka. Pod statistički malim skupom podataka smatra se skup podataka kod kojeg postoji 30 ili manje primjera [19]. Postoji nekoliko *bootstrapping* metoda, npr. *e0* ili *0.632 bootstrap*.

5.2.3.1 Metoda e0

Procjena pogreške modela na skupu za ispitivanje zove se e0 procjena te je po tome ova vrsta *boostrppinga* i dobila ime. Kod ove metode skup uzoraka za učenje je jednake veličine kao i ukupan broj uzoraka. Iz skupa svih primjera uzimaju se primjeri dok broj primjera u skupu za učenje ne bude jednak N, ako je N ukupan broj svih dostupnih primjera. Razlika od prethodnih metoda je u tome što se pojedini primjeri mogu ponavljati (replicirati), što znači da nije bitno da su svi primjeri u skupu primjera za učenje jedinstveni. A oni primjeri koji ne budu u skupu za učenje predstavljaju skup za ispitivanje. Prosječan broj međusobno različitih primjera u skupu za učenje je 0.632 ukupnog broja primjera, dok je broj preostalih međusobno različitih primjera u skupu za ispitivanje jednak 0.368 ukupnog broja primjera.

5.2.3.2 0.632 bootstrap

0.632 boostrap ili kraće 0.632B je još jedna vrsta *bootstrappinga*. To je linearna kombinacija $0.368 \cdot trerr + 0.632 \cdot e0$, gdje je $trerr$ pogreška mjerena na oba skupa, i na skupu za učenje i na skupu za ispitivanje.

Izvršavanje ovih metoda ponavlja se nekoliko puta, obično oko 100 ili 200 puta. *Bootstrapping* je računalno još zahtjevniji od *LOO* metode, ali obično bolje procjenjuje pogrešku. Za umjereno velike skupove podataka, *e0* daje pesimističniju procjenu pogreške od stvarne pogreške, ali daje dobru procjenu ako je stvarna pogreška modela relativno visoka. *0.632 bootstrap* je za relativno veliki broj primjera preoptimistična procjena, ali za manji skup podataka i kad je stvarna pogreška modela mala daje dobre rezultate.

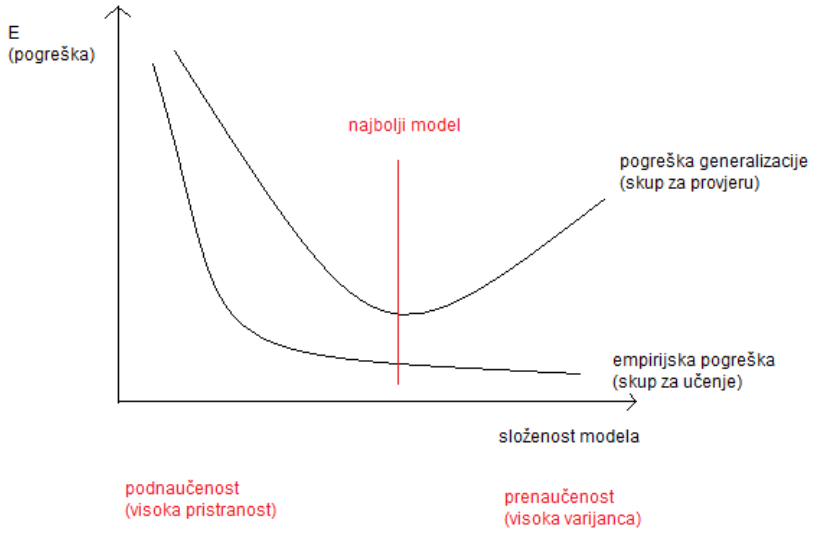
U tablici 5.3 prikazane su značajke *boostrap* tehnike procjene pogreške klasifikacijskog modela [19]. S n je označen ukupan broj primjera.

Tablica 5.3 Značajke bootstrapping metode

	bootstrapping
broj primjera u skupu za učenje	n (j različitih)
broj primjera u skupu za ispitivanje	n - j
broj iteracija	100 - 200

Kod svih gore nabrojanih metoda preporuča se da veličina skupa za učenje po klasi bude tri do pet puta veća od broja značajki uzoraka, tj. od dimenzionalnosti vektora značajki [44].

Općenito je potrebno pažljivo odabrati klasifikacijski model. Cilj hipoteze modela nije ispravno klasificirati podatke nad kojima je klasifikator učio, jer će većina tih uzoraka ionako biti dobro klasificirana, a nije niti rijetkost da je uspješnost klasifikacije uzoraka iz skupa za učenje 100%. Pogreška modela će tada biti puno manja od stvarne pogreške. Cilj hipoteze modela jest ispravno klasificirati podatke koje klasifikator još nije vidio. Prema načelu Occamove britve (engl. *Occam's razor*) potrebno je odabrati jednostavan, ali ne prejednostavan model. Jednostavni modeli bolje generaliziraju, imaju manju računalnu složenost, lakše uče, jer imaju manji broj parametara koje treba optimirati i jednostavan model lakše je tumačiti i iz njega izdvojiti znanje [8]. Niti prenaučenost (engl. *overfitting*) niti podnaučenost (engl. *underfitting*) nisu poželjne. Kod prenaučenosti (složeni) model će pretpostavljati i više no što se u stvarnim podacima nalazi, a kod podnaučenosti (previše jednostavan) model neće dobro klasificirati niti primjere za učenje pa je jasno kako tada neće dobro klasificirati niti primjere koje još nije vidio. Prenaučeni modeli imaju veliku varijancu (a malu pristranost), što znači da male promjene u uzorcima za učenje uzrokuju velike promjene u hipotezi modela. Podnaučeni modeli imaju veliku pristranost (ali malu varijancu), što znači da se izlazi klasifikatora jako razlikuju od izlaza kakvi bi stvarno trebali biti. Na slici 5.5 prikazana je ovisnost empirijske i pogreške generalizacije o složenosti modela [8].



Slika 5.5 Ovisnost empirijske pogreške i pogreške generalizacije o složenosti modela

Ako se koriste sva tri skupa, tada algoritam izgleda ovako [23]:

1. Podijeli dostupan skup podataka na skupove za učenje, provjeru i ispitivanje

2. Odaberi arhitekturu i parametre koje je potrebno naučiti
3. Uči model nad skupom za učenje
4. Evaluiraj model nad skupom za provjeru
5. Ponovi korake od 2 do 4 mijenjajući arhitekturu i parametre
6. Odaberi najbolji model i uči ga nad podacima za učenje i provjeru zajedno
7. Ispitaj odabrani model nad skupom za ispitivanje

Ako se koriste metode za unakrsnu validaciju ili *bootstrapping*, tada je korake 3 i 4 potrebno ponoviti k puta.

U tablici 5.4 prikazane su prednosti i nedostaci pojedinih detaljnije opisanih metoda evaluacije (*Holdout*, metoda ponovne zamjene, k – struka unakrsna provjera, *Leave – one out (LOO)*, iterativna k – struka unakrsna provjera). Neke od tih metoda su računalno zahtjevnije, a neke daju bolju procjenu pogreške modela.

Tablica 5.4 Prednosti i nedostaci nekih metoda evaluacije

metoda validacije	**prednosti**	**nedostaci**
Holdout	nezavisno učenje i ispitivanje	ograničeni podaci za učenje i testiranje; velika varijanca
ponovna zamjena	jednostavna	prenaučenost
k – struka unakrsna	svaki uzorak i u skupu za učenje i u skupu za ispitivanje	ovisnost procjene greške o parametru k (k broj podjela, broj grupa), dugotrajna za veliki k
LOO	nepristranost	vrlo velika varijanca; velika računalna složenost
iterativna k – struka unakrsna	svaki uzorak i u skupu za učenje i u skupu za ispitivanje	ponavljanje uzoraka; dugotrajno izvođenje za veliki skup uzoraka
bootstrapping	bolja procjena pogreške od *LOO*	računalna zahtjevnost; dugotrajno izvođenje

6. Skupovi podataka

Postoji nekoliko različitih repozitorija u kojima se nalaze različiti skupovi podataka koji se mogu koristiti za izgradnju klasifikatora. Podaci su najraznovrsniji, od podataka vezanih za medicinu, računala, Internet, ekonomiju pa do podataka s područja zoologije, botanike itd. Najpoznatiji repozitorij je *UCI Machine Learning Repository* [54], gdje se nalaze različiti skupovi podataka, kao npr. *Australian credit dataset, Vehicle dataset, Heart disease, German credit data, Iris dataset*...

U nastavku su opisani podaci koji su se koristili prilikom izrade praktičnog dijela ovog rada.

6.1 German credit data

Ovaj skup podataka ima ukupno tisuću primjera za učenje. Atributa ima dvadeset, a razreda su samo dva. O osobama su sakupljene različite informacije (svi uzorci imaju vrijednosti svih atributa, što znači da nema vrijednosti koje nedostaju) i, ovisno o sakupljenim informacijama, tj. vrijednostima pojedinih atributa, za osobu se određuje je li ili nije kreditno sposobna. Ovaj se skup podataka često koristi te postoje poznati podaci uspješnosti klasifikacije za različite algortime klasifikacije. Skup podataka dolazi u dva oblika, u jednom su atributi samo numerički, a u jednom atributi imaju i numeričke i nominalne vrijednosti. Skup podataka koji ima samo numeričke atribute dodatno je stvoren kako bi taj skup podataka bio pogodan za algoritme koji rade samo s numeričkim vrijednostima, pri čemu su nominalni atributi posebno kodirani i tako pretvoreni u numeričke. U praktičnom dijelu rada korištena je inačica koja ima i numeričke i nominalne atribute. U nastavku su navedena značenja pojedinih atributa, a dodatne informacije o mogućim vrijednostima za pojedini atribut mogu se naći u dokumentaciji koja se nalazi uz svaku datoteku sa skupom podataka, gdje su pobliže objašnjeni sami atributi, vrijednosti te dodaci značajni za taj skup podataka. Numerički atributi su: trajanje kredita u mjesecima (atribut dva), iznos kredita (atribut pet), iznos kamate (atribut osam), dužina boravka u trenutnom mjestu stanovanja (atribut jedanaest), dob (atribut trinaest), broj već postojećih kredita osobe u nekoj banci (atribut šesnaest) te broj jamaca (atribut osamnaest). Nominalni atrobuti su redom: trenutno stanje računa u banci, kreditna prošlost, razlog uzimanja kredita, stanje ušteđevine, duljina trenutnog radnog odnosa, osobni status i spol, dužnik / jamac još kome, nekretnine koje osoba posjeduje, budući planovi, stambeno stanje, posao, posjeduje li osoba telefon i je li osoba strani radnik.

Numerički atributi svi imaju cjelobrojne vrijednosti, a nominalni različite vrijednosti posebnog značenja ovisno o tome o kojem se atributu radi, npr. A11, A45. U tablici 6.1 nalazi se opis ovog skupa.

Tablica 6.1 Osnovne informacije o skupu podataka German credit data

naziv skupa	broj uzoraka	broj atributa	broj razreda
German Credit	1000	20 (7 numeričkih, 13 nominalnih)	2

Od ukupno 1000 uzoraka, 700 pripada razredu s oznakom jedan, što znači da je 700 osoba, od 1000 ispitanih, kreditno sposobno, dok ih 300 nije.

6.1.1 Matrica troška

Značajno za ovaj skup je postojanje tzv. matrice troška (engl. *cost matrix*) koja je prikazana u tablici 6.2.

Klasifikator koji se želi izgraditi trebao bi minimizirati trošak, tj. štetu nastalu pogrešnom klasifikacijom. Trošak pogrešne klasifikacije (engl. *missclassification cost*) je kazna za pogrešnu klasifikaciju nekog uzorka. Svaka matrica troška ili matrica grešaka ima n^2 članova, gdje je n broj razreda. Na dijagonali se nalazi broj ispravno klasificiranih uzoraka, a elementi izvan glavne dijagonale predstavljaju broj različitih tipova pogrešne klasifikacije uzoraka. Ova matrica, kao i matrica zabune, sadrži sve kombinacije stvarnih i procijenjenih stanja.

Tablica 6.2 Tablica troška za skup podataka German credit data

		procijenjeno stanje	
		kreditno sposoban	kreditno nesposoban
stvarno stanje	kreditno sposoban	0	1
	kreditno nesposoban	5	0

Ukupan trošak prikazan je jednadžbom (6.1).

$$Trošak = \sum_{i=1} \sum_{j=1} E_{ij} C_{ij} \tag{6.1}$$

E_{ij} je broj grešaka tipa ij, a C_{ij} je trošak vezan uz taj tip greške.

Iz matrice troška u tablici 6.2 slijedi kako je lošije nekoga ocijeniti kao kreditno sposobnog, ako je on zapravo kreditno nesposoban nego suprotno, ocijeniti ga kao kreditno nesposobnog, ako je zapravo kreditno sposoban. Ako se neki uzorak pogrešno klasificira, klasifikator će biti kažnjen. Tipu greške kad je uzorak klasificiran kao kreditno sposoban, iako je kreditno nesposoban (netočno pozitivan, *FP*) pridijeljen je trošak 5, a tipu greške kad je uzorak ocijenjen kao kreditno nesposoban, iako je kreditno sposoban (netočno negativan, *FN*) pridijeljen je trošak 1. Vrijednost troška može biti i negativna.

6.2 Skup podataka u arhivi paralelnog opterećenja grozdova

Drugi korišteni skup podataka je skup podataka o korisničkim poslovima uzet iz arhive paralelnog opterećenja grozdova (engl. *Parallel Workload Archive, PWA*) [40]. Podaci u ovom skupu su, za razliku od podataka u prethodno navedenom kod kojeg nije uključeno vrijeme, poredani prema vremenu postavljanja poslova (engl. *Submit Time*), to su tzv. *time – series* podaci. U toj se arhivi nalaze podaci u obliku dnevnika i modela (koji su izvedeni iz dnevnika) sakupljenih s različitih računalnih grozdova u svijetu. Kako bi se pojednostavnilo korištenje dnevnika (engl. *log*) i modela, stvoren je jednostavan format, tzv. *standard workload format (swf)* i datoteke u kojima se nalaze prikupljeni podaci o korisničkim poslovima imaju ekstenziju *.swf*.

Bitni podaci o svakom poslu prikazani su u jednom retku takve datoteke. Svaka datoteka ima unaprijed određen broj stupaca (koji predstavljaju atribute), a vrijednosti u stupcima su cjelobrojne ili realne odvojene jednom ili više praznina. Atributi koji su nevažni za neki dnevnik ili model imaju vrijednost -1 i oni će se prilikom obrade smatrati nedostajućim vrijednostima.

Na slici 6.1 može se vidjeti primjer iz datoteke (dnevnika) *CTC-SP2-1996-2.1-cln.swf*, na slici se može vidjeti kako se na početku datoteke *swf* formata nalaze komentari koji počinju znakom točka – zarez, a koji predstavljaju različite dodatne informacije o sakupljenim podacima, kao što su npr. naziv računalnog grozda, najveći broj poslova i uzoraka (koji ne mora biti jednak, jer se podaci o jednom poslu mogu protezati kroz više redova – ovo je označeno posebnim vrijednostima kod oznake statusa posla), zatim vremenski period prikupljanja podataka i sl. Ovdje se nalaze i imena osoba koje su zaslužne za stvaranje i održavanje arhive, zatim podaci o broju procesora računala, količini memorije, najvećem dopuštenom vremenu izvršavanja poslova itd.

Nazivi svih datoteka slijede određenu konvenciju i sastoje se od dva dijela. U prvom dijelu naznačeno je o kojem se skupu podataka radi (u obliku *<site> - <machine> - <year>*), a u drugom dijelu o kojoj se inačici datoteke radi. Npr. naziv jedne od datoteka je *CTC-SP2-1996-2.1-cln.swf.* Prvi dio označava ime skupa, a to je *CTC-*

SP2-1996. Ostatak, *2.1.-cln*, znači da je to druga inačica (te da su i na njoj naknadno izvršene manje promjene, što je označeno brojem *1*), a *cln* znači da se radi o pročišćenoj (engl. *cleaned*) inačici (koja se od originalnih razlikuje po tome što su izbačeni neki poslovi; o kojim se poslovima radi detaljnije je navedeno na početku datoteke, u komentarima). Za klasifikaciju podataka u praktičnom dijelu ovog rada korištene su pročišćene inačice datoteka (dnevnika), kako je predloženo u *PWA*.

```
; Version: 2.2
; Computer: Linux cluster (Seth)
; Installation: HPC2N - High Performance Computing Center North, Sweden
; Acknowledge: Ake Sandgren
; Information: http://www.hpc2n.umu.se/support/userguide/Seth/batchsystem.html
; Conversion: Sarah Gingichashvili and Dror Feitelson (feit@cs.huji.ac.il) 03 Jan 2008
; MaxJobs: 527371
; MaxRecords: 527371
; Preemption: No
; UnixStartTime: 1027839845
; TimeZone: 3600
; TimeZoneString: Europe/Stockholm
; StartTime: Sun Jul 28 09:04:05 CEST 2002
; EndTime:   Mon Jan 16 20:31:24 CET 2006
; MaxNodes: 240
; MaxProcs: 240
; Note: uses MAUI scheduler
; MaxQueues: 1
; Queue:  1 [fque:1]
; MaxPartitions: 2
; Partition:  1 DEFAULT
; Partition:  2 ALL
;
    1        0 308434  31405   30  31405    -1   30  37980    -1  1   1   1  -1 -1  1 -1 -1
    2        8 308426  31039   30  31039    -1   30  37980    -1  1   1   1  -1 -1  1 -1 -1
    3       12 308483  30978   30  30978    -1   30  37980    -1  1   1   1  -1 -1  1 -1 -1
    4   179350  99808 143967    2 142977    -1    2 144000    -1  1   3   1  -1 -1  1 -1 -1
    5   193020  84357  73167   20  73168    -1   20  74400    -1  1   2   1  -1 -1  1 -1 -1
    6   193032  84345  73167   20  73168    -1   20  74400    -1  1   2   1  -1 -1  1 -1 -1
    7   289471     58  56427   20  56428    -1   20  56400    -1  1   2   1  -1 -1  1 -1 -1
    8   289669     54  56293   20  56356    -1   20  56400    -1  1   2   1  -1 -1  1 -1 -1
    9   305017     54  43210   20  43211    -1   20  45600    -1  1   2   1  -1 -1  1 -1 -1
   10   310085  29510   2326   40   2326    -1   40  43200    -1  1   4   1  -1 -1  1 -1 -1
   11   342072     32  43306   32  43307    -1   32  43200    -1  1   4   1  -1 -1  1 -1 -1
   12   342343      5    553   40 552.21    -1   40  43200    -1  1   4   1  -1 -1  1 -1 -1
   13   342921    102  10027   40  10027    -1   40  43200    -1  1   4   1  -1 -1  1 -1 -1
   14   347226     15  33156    8  33157    -1    8  86400    -1  1   5   1  -1 -1  1 -1 -1
   15   347593     15  86462    8  86464    -1    8  86400    -1  1   5   1  -1 -1  1 -1 -1
   16   347726      5     -1    8 240379    -1    8 345600    -1 -1   5   1  -1 -1  1 -1 -1
   17   347844      9   2201    8   2201    -1    8 432000    -1  1   5   1  -1 -1  1 -1 -1
   18   347927     48     -1    8 240135    -1    8 432000    -1 -1   5   1  -1 -1  1 -1 -1
   19   347985     51     -1    8 240074    -1    8 432000    -1 -1   5   1  -1 -1  1 -1 -1
```

Slika 6.1 Primjer dijela datoteke CTC-SP2-1996-2.1-cln.swf

6.2.1 Značajke korisničkih poslova

U svakoj se datoteci nalazi osamnaest značajki, tj. atributa. To su: *Job Number*, što je broj posla, počinje od jedan nadalje. Zatim *Submit Time*, vrijeme početka obavljanja prvog posla (u sekundama). Nakon toga slijede *Wait Time* i *Run Time*, što su redom vrijeme između postavljanja posla i početka njegovog izvršavanje te samo vrijeme izvršavanja. Zatim *Number of Allocated Processors* i *Average CPU Time Used*. Slijede *Used memory* (u kB), *Requested Number of Processors* (koliko je procesora korisnik zatražio), zatim *Requested Time*, čije je točno značenje određeno u komentarima na početku datoteke (može biti zatraženo vrijeme izvršavanja posla ili srednje vrijeme po procesoru). Slijede *Requested Memory*, *Status*, *User ID* i *Group*

ID, *Executable (Application) Number* (cijeli broj od jedan do ukupnog broja različitih aplikacija), *Queue Number* (cijeli broj od jedan do ukupnog broja redova u sustavu), *Partition Number* (cijeli broj od jedan do ukupnog broja particija u sustavu), *Preceding Job Number* (broj prethodnog posla; posao može započeti s radom tek je kad njemu prethodni završio s radom) i *Think Time from Preceding Job* (vrijeme u sekundama koje bi trebalo proći između završetka rada prethodnog i početka rada novog posla).

Neki atributi mogu imati, za sve poslove, vrijednost -1, što znači da taj atribut nije bitan za taj određeni dnevnik i ti atributi se, prilikom analize, ne uzimaju u obzir. Poslovi mogu imati i vrijeme izvršavanja jednako -1 i takvi ne ulaze u analizu.

6.2.2 Atributi važni za klasifikaciju

Atributi koji su se koristili za klasifikaciju su: *Submit Time, Requested Number of Processors, Requested Time, Requested Memory, Status, User ID, Group ID, Executable (Application) Number, Queue Number* i *Partition Number* (indeksi ovih atributa su, kako su poredani stupci u datoteci, redom 2, 8, 9, 10, 11, 12, 13, 14, 15, 16). Ostali atributi nisu uzeti u obzir, jer neki ne opisuju dovoljno značajke poslova ili nisu poznati u trenutku dolaska posla u sustav [20]. Atributi koji za dani skup ne sadrže niti jednu drugu vrijednosti osim -1 ne pridonose klasifikaciji i oni su zanemareni. U tablici 6.3 su u zadnjem stupcu navedeni indeksi samo valjanih atributa (onih koji su za dani skup podataka bitni, osim atributa *Status*) koji bi se trebali koristili kod izgradnje klasifikatora. Taj je podatak naveden za svaki grozd posebno, uz ukupan broj uzoraka te broj valjanih uzoraka (isključeni su uzorci koji za jedan od valjanih atributa imaju vrijednost -1, jer bi takvi uzorci mogli stvarati probleme prilikom izgradnje klasifikatora). Kod svakog uzorka atributi se pojavljuju u istom redoslijedu.

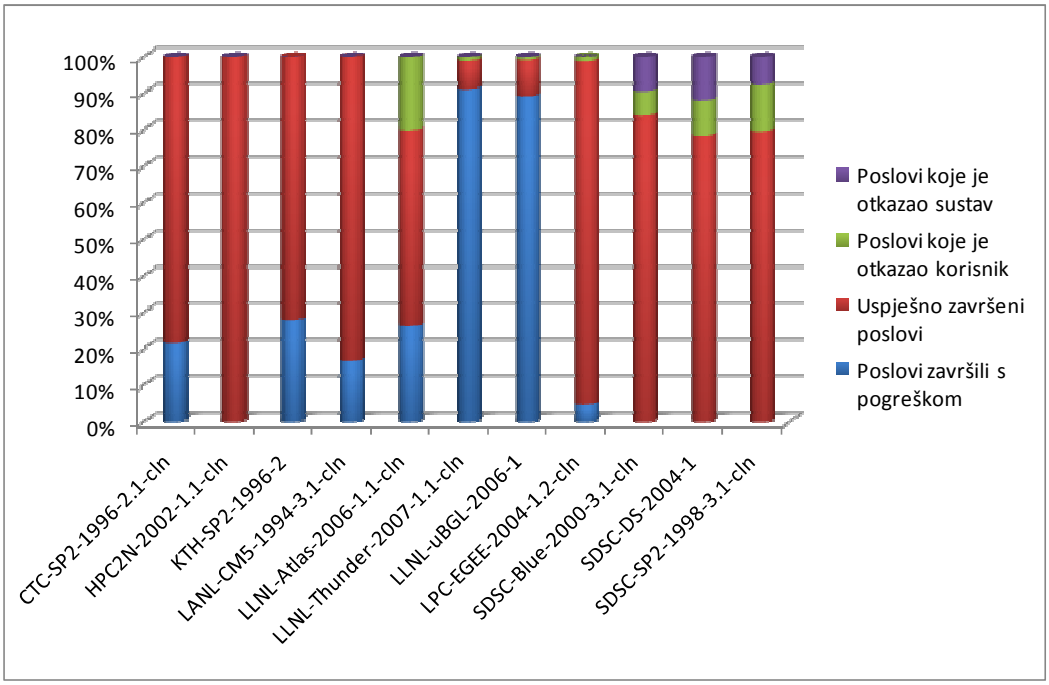
Tablica 6.3 Broj valjanih atributa i uzoraka za neke datoteke

naziv datoteke	ukupno uzoraka	broj valjanih uzoraka	indeksi valjanih atributa
CTC-SP2-1996-2.1-cln	79302	77216	2, 9, 12, 14, 15
HPC2N-2002-1.1-cln	527371	202876	2, 8, 9, 12, 13
KTH-SP2-1996-2	28490	28489	2, 8, 9, 12, 13
LANL-CM5-1994-3.1-cln	201387	110731	2, 8, 9, 10, 12, 13, 14, 15
LLNL-Atlas-2006-1.1-cln	60332	32215	2, 8, 9, 12, 14, 16

LLNL-Thunder-2007-1.1-	128662	38560	2, 8, 9, 12, 14, 16
LLNL-uBGL-2006-1	112611	9128	2, 8, 9, 12, 14
LPC-EGEE-2004-1.2-cln	234889	220695	2, 8, 9, 12, 15, 16
SDSC-Blue-2000-3.1-cln	250440	243314	2, 8, 9, 12, 15
SDSC-DS-2004-1	96089	29197	2, 8, 9, 12,13, 14, 15, 16
SDSC-SP2-1998-3.1-cln	73496	59659	2, 8, 9, 12, 13, 14, 15

6.2.3 Status korisničkih poslova

Cilj praktičnog dijela rada bio je predvidjeti status još neviđenih poslova. Postoji nekoliko tipova statusa (u ovom slučaju je to klasa ili kategorija u koju se određeni posao sa svojim značajkama treba svrstati). Posao može uspješno završiti (tada mu je dan status 1). Ako posao ima status 0, tada je on završio s pogreškom (neispravno je funkcionirao ili je prekinut od strane operacijskog sustava zbog nelegalnog korištenja memorijskog prostora). Posao može imati i status 5, što znači da je prekinut. Posao može prekinuti sustav, tj. raspoređivač poslova ili korisnik i te dvije vrste statusa su razdvojene. Ako neki posao ima oznaku statusa 5, tada se gleda vrijeme izvršavanja posla i je li ono veće od zatraženog vremena. U većini datoteka, zatraženo vrijeme je zapravo atribut *Requested Time.* Ako je vrijeme izvršavanja posla (*Run Time*) prekoračilo zatraženo vrijeme, a status posla jednak je 5, pretpostavlja se da je posao prekinuo sustav te se on tada svrstava u kategoriju poslova prekinutih od strane sustava. U protivnom, tj. ako se posao nije izvršavao dulje od zatraženog, tada se svrstava u kategoriju poslova prekinutih od strane korisnika.



Slika 6.2 Grafikon udjela u broju poslova za poslove različitih statusa

Na slici 6.2 su, za različite datoteke, prikazani udjeli poslova različitih statusa u ukupnom broju poslova, a u tablici 6.4 su u postotcima navedeni pojedini udjeli.

Navedeno je jedanaest datoteka, a analiza je izvršena nad pročišćenim inačicama tamo gdje je to bilo moguće.

Tablica 6.4 Udjeli u broju poslova za poslove različitih statusa

naziv datoteke	završili s pogreškom	uspješni	otkazao korisnik	otkazao sustav
CTC-SP2-1996-2.1-cln	21.59	78.41	0.00	0.00
HPC2N-2002-1.1-cln	0.00	100.00	0.00	0.00
KTH-SP2-1996-2	27.90	72.10	0.00	0.00
LANL-CM5-1994-3.1-cln	16.69	83.31	0.00	0.00
LLNL-Atlas-2006-1.1-cln	26.17	53.87	19.93	0.03
LLNL-Thunder-2007-1.1-cln	91.21	7.84	0.93	0.01

LLNL-uBGL-2006-1	89.23	10.06	0.66	0.04
LPC-EGEE-2004-1.2-cln	4.75	94.17	1.08	0.00
SDSC-Blue-2000-3.1-cln	0.00	84.19	6.52	9.28
SDSC-DS-2004-1	0.00	78.52	9.64	11.84
SDSC-SP2-1998-3.1-cln	0.00	79.78	12.66	7.56

U samo tri slučaja (kako je vidljivo iz naslova datoteka u tablici 6.4 ili grafikonu na slici 6.2, to su datoteke: *KTH-SP2-1996-2, LLNL-uBGL-2006-1 i SDSC-DS-2004-1*) korištene su originalne inačice, jer pročišćene nisu dostupne u *PWA*.

Kod samo četiri grozda računala ne pojavljuju se poslovi s pogreškom, kao i poslovi koje je otkazao korisnik, a kod pet grozdova računala ne pojavljuju se poslovi koje je otkazao sustav. Poslovi koje je otkazao sustav kreću se od 0.01% do čak 11.84%.

Poslovi koje je otkazao korisnik sudjeluju u ukupnom broju poslova od 0.93% do 19.93%; udio poslova koji su završili s pogreškom kreće se od 4.75% do 91.21%, a uspješno završenih poslova od 7.84 do čak 100% kod jednog grozda (poslovi u datoteci *HPC2N-2002-1.1-cln*).

Kod ove analize nisu se uzimali u obzir poslovi kojima je vrijeme izvršavanja (*Run Time*) ili zatraženo vrijeme (*Requested Time*) nepoznato (jednako -1), a ostali poslovi su (ako im i nedostaje vrijednost bilo kojeg drugog atributa), uključeni, iako se za klasifikaciju nisu koristili.

7. Programsko ostvarenje

U ovom poglavlju detaljnije je opisana teorijska pozadina, tj. što se iz teorije koristilo pri izradi praktičnog dijela ovog rada (izgradnja GP klasifikatora) te nakon toga slijedi pregled grafičkog korisničkog sučelja i upute za korištenje.

7.1 Teorijska pozadina

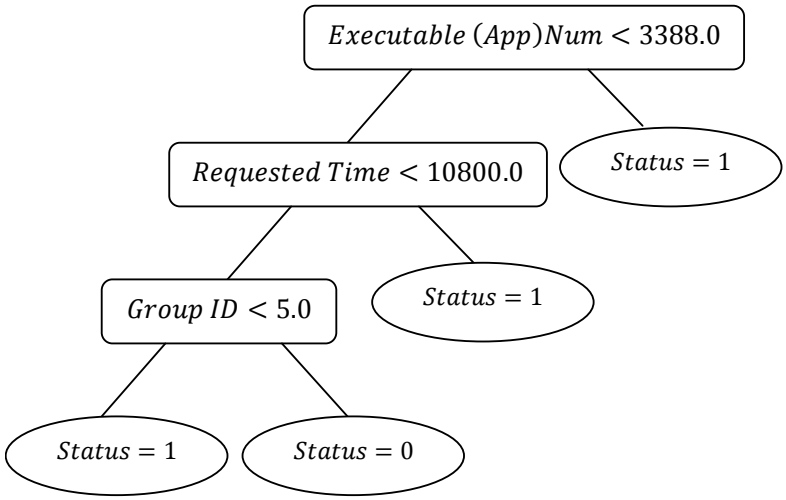
Za izgradnju klasifikatora korisničkih poslova koristilo se stablasto genetsko programiranje s jedinkom strukture slične stablu odluke (engl. *top – down atomic representation*) [11] [13] [14].

7.1.1 Stablasto genetsko programiranje

Ovdje su detaljnije opisani koraci stablastog genetskog programiranja korištenog u praktičnom dijelu rada.

7.1.1.1 Jedinka stablaste strukture

Jedinka je predstavljena potpunim binarnim stablom. U čvorovima pojedinog stabla nalaze se atomi [11] oblika *atribut operator vrijednost*. Kako je cilj klasifikacije predvidjeti status posla kojemu se status još ne zna, atribut *Status* je razred u koji se posao mora svrstati pa se u skupu funkcijskih znakova nalaze svi atributi s odgovarajućim vrijednostima, osim atributa Status. Skup završnih znakova, atoma sastoji se od samo dva atoma, a to su *Status = 0* i *Status = 1*. Iako se razlikuju četiri statusa poslova, slučaj je pojednostavljen na binarnu klasifikaciju. Klasifikator se uči na poslovima koji su završili uspješno (*Status = 1*) i onima koji su završili s pogreškom (*Status = 0*). Poslovi koje je otkazao korisnik mogu se zanemariti jer takvi poslovi ne koriste u velikoj mjeri dostupna sredstva, a poslovi koje je otkazao sustav nemaju poznato stvarno vrijeme izvođenja i nemoguće je procijeniti koji bi od tih poslova završili uspješno, a koji ne bi [20]. Ovisno o tome koji se operator u trojci *atribut operator vrijednost* u čvoru stabla nalazi, iako to nije jedina razlika, razlikuju se četiri vrste GP – a: jednostavan GP [11], GP temeljen na informacijskoj dobiti (engl. *Gain GP*) [11], GP temeljen na omjeru dobitka (engl. *Gain – ratio GP*) [11] te GP temeljen na grupiranju (engl. *Clustering GP*) [11]. Veličina stabla (engl. *tree size*) je broj čvorova stabla, a dubina (engl. *tree depth*) je najveća dubina u stablu. Na slici je prikazano stablo nastalo jednostavnim GP - om dubine tri, a veličine sedam. Čvorovi stabla su različito prikazani, kako bi se odvojili funkcijski i završni atomi. Završni atomi nalaze se samo u listovima.



Slika 7.1 Primjer stabla nastalog jednostavnim genetskim programiranjem

Osim ovakvog prikaza stabla, moguć je ispis stabla i u prefiks (tzv. poljskoj) notaciji. Primjer stabla na slici 7.1 u prefiks notaciji:

$(Executable\ (App)Num < 3388.0\ \ (Requested\ Time < 10800.0\ \ (Group\ ID < 5.0$

$Status = 1\ \ Status = 0))$

7.1.1.2 Funkcija dobrote

Funkcija dobrote sastoji se od dviju vrijednosti (engl. *multiobjective fitness*). Prva je pogreška klasifikacije koju jedinka ima na skupu za učenje, a druga je veličina jedinke, tj. broj čvorova u stablu. Formulom (7.1) računa se prva vrijednost dobrote.

$$fitness_1(x) = \frac{\sum_{r \in U} f(x,r)}{|U|} \tag{7.1}$$

S U je označen skup za učenje, $|U|$ je kardinalitet tog skupa, a $f(x,r)$ se izračunava prema formuli (7.2).

$$f(x,r) = \begin{cases} 1, & ako\ je\ stablo\ x\ pogrešno\ klasificiralo\ uzorak\ r \\ 0, & inače \end{cases} \tag{7.2}$$

Brojnik kod $fitness_1(x)$ je zapravo zbroj broja uzoraka koji su pogrešno klasificirani (onih koji su netočno pozitivni i netočno negativni). Za svako stablo u populaciji izračunava se pogreška klasifikacije na cijelom skupu za učenje.

Bolja jedinka ima manju pogrešku klasifikacije na skupu za učenje i ta se pogreška prva promatra, a ako dvije jedinke imaju jednaku pogrešku klasifikacije na skupu za učenje, tada se gleda druga vrijednost funkcije dobrote - veličina stabla. Prilikom selekcije prednost uvijek imaju manja stabla.

7.1.1.3 Stvaranje početne populacije

Za stvaranje početne populacije (inicijalizaciju populacije) koristi se najpoznatija, tzv. *ramped half – and half* metoda [41]. Ova metoda se sastoji od dviju drugih metoda, a to su *grow* metoda i *full* metoda. Populacija se dijeli u D – 1 grupa; svaka grupa koristi drugu dubinu (vrijednosti se kreću od 2 do D). Najveća dubina slučajno se određuje iz intervala [2, 6]. Pola svake grupe stvara se *grow*, a pola *full* metodom (kako i samo ime govori), kako bi se u populaciju unijela stabla različitih veličina i dubina. Obje metode, i *grow,* i *full*, osjetljive su na veličinu skupova funkcijskih i završnih znakova. Ako je veličina skupa završnih znakova veća, tada će *grow* metoda stvarati stabla s malo čvorova, a ako je veličina skupa funkcijskih znakova veća, tada će se ona ponašati slično *full* metodi [41].

7.1.1.3.1 *Grow* metoda

Kod ove se metode slučajno uzimaju atomi iz skupa primitiva (i iz skupa funkcijskih i iz skupa završnih atoma) te se tako stvaraju čvorovi, sve dok dubina novonastalog stabla ne bude jednaka željenoj dubini stabla. Kada dubina novonastalog stabla bude jednaka željenoj dubini, mogu se za čvorove uzeti samo atomi iz skupa završnih znakova (atoma). Korijen stabla je na dubini 0. Na slici 7.2 nalazi se pseudokod ove metode.

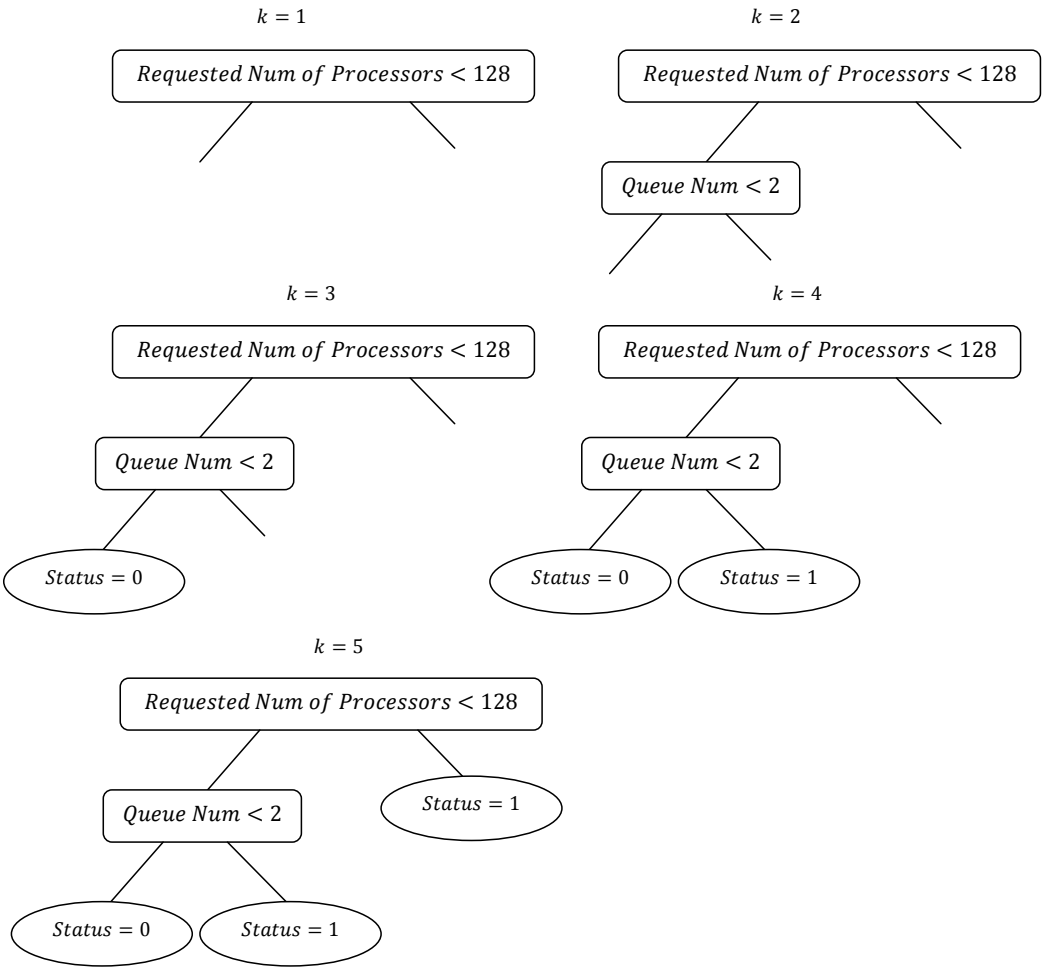
```
Cvor grow (dubina) {
        if (trenutnaDubina < maxDubina) {
                noviCvor := random (F ∪ Z);
                for (i = 1; i < brojDjece; i++)
                        child[i] = grow (trenutnaDubina + 1);
        }
        else
                noviCvor = random (Z);
return noviCvor;
}
```

Slika 7.2 Pseudokod grow metode

Metoda je rekurzivna; F je skup funkcijskih, a Z skup završnih atoma.

Na slici 7.3 prikazan je način rada ove metode za stvaranje stabla željene dubine dva po koracima (k je broj koraka). Slika 7.1 prikazuje jedno stablo nastalo *grow*

metodom. Prema primjeru iz slike 7.1 vidi se kako ovom metodom, za razliku od *full* metode, nastaju stabla različitih veličina.



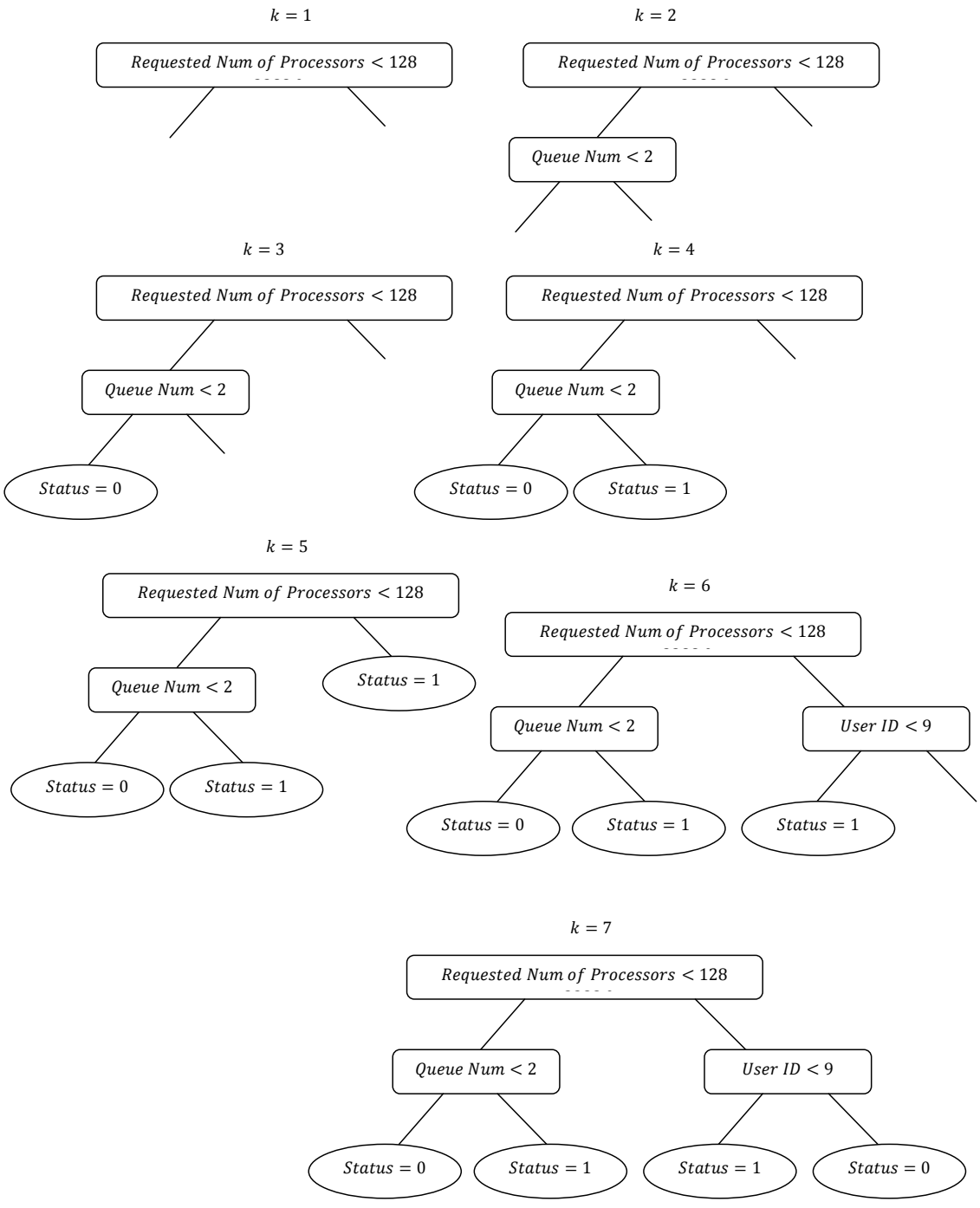
Slika 7.3 Grafički prikaz stvaranja stabla grow metodom

7.1.1.3.2 *Full* metoda

Ova metoda stvara stabla kojima su svi listovi na istoj dubini. Sve dok nije postignuta željena dubina, stvarat će se čvorovi s funkcijskim atomima, a nakon što ona bude postignuta, mogu se u čvorove dodati samo završni atomi. Na slici 7.5 grafički je prikazano stvaranje stabla željene dubine dva, a na slici 7.4 nalazi se pseudokod metode.

```
Cvor full (dubina) {
        if (trenutnaDubina < maxDubina) {
                noviCvor := random (F);
                for (i = 1; i < brojDjece; i++)
                        child[i] = full (trenutnaDubina + 1);
        else
                noviCvor = random (Z);
        }
return noviCvor;
}
```

Slika 7.4 Pseudokod full metode



Slika 7.5 Grafički prikaz stvaranja stabla full metodom

Postoje još neke metode stvaranja početne populacije koje se nisu koristile pri izradi praktičnog dijela rada, ali su ovdje ukratko navedene i objašnjene. Postoji uniformno stvaranje početne populacije, gdje se stabla ne stvaraju slučajno, kako je slučaj kod *ramped half – and – half* metode, već se pazi na važnost pojedinih funkcijskih

znakova pa se oni važniji nalaze bliže korijenu stabla, a oni manje bitni, ako jesu u stablu, nalaze se što dalje od korijena. Drugi način je dodjeljivanje dobre jedinke (engl. *seeding the initial population*) početnoj populaciji. Stablo može osmisliti čovjek ili ono može biti rezultat nekog već prije pokretanog GP programa i takvo je stablo bolje od nekog slučajno stvorenog.

7.1.1.4 Selekcija

Odabrana selekcija je k – turnirska selekcija. Iz postojeće populacije slučajno se odabere k jedinki. Od tih k jedinki odabere se najbolja koja postaje roditelj za križanje ili ona koja će sudjelovati u mutaciji. Ako se radi o križanju, selekcija će se izvesti dva puta, jer su za križanje potrebne dvije jedinke – roditelji.

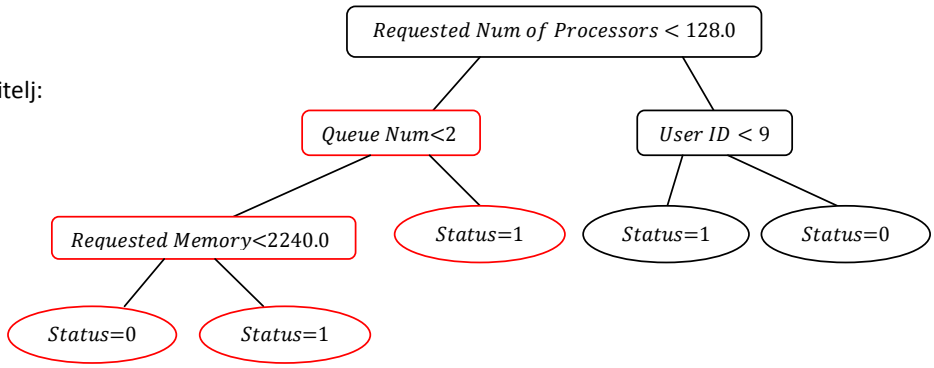
Postoje i druge vrste selekcije, npr. eliminacijska turnirska, kod koje se slučajno odabire k jedinki te se najlošija od njih eliminira iz trenutne populacije ili dobro poznata proporcionalna selekcija (engl. *roulette – wheel selection*), gdje je vjerojatnost odabira jedinke razmjerna njenoj dobroti.

7.1.1.5 Križanje

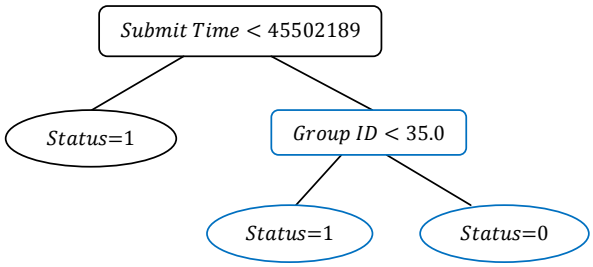
Križanje se izvodi s određenom vjerojatnošću – vjerojatnošću križanja od 90% ili više. Izabrano križanje je križanje kojim kopije dviju odabranih jedinki razmjenjuju slučajno odabrana podstabla (engl. *subtree crossover*). Na slici 7.6 prikazan je ovakav način križanja. Crvenom o plavom bojom označena su podstabla koja će sudjelovati u razmjeni.

Postoje još neke vrste križanja, iako je programski ostvarena samo prethodno objašnjena vrsta. Postoji uniformno križanje kod kojeg se određuje hoće li se u dijete kopirati čvor prvog ili drugog roditelja. Zatim križanje kod kojeg nastaju manja djeca, jer se želi ograničiti stvaranje prevelikih stabala (engl. *size – fair crossover*) i još mnoge druge vrste, ovisno o problemu koji se rješava.

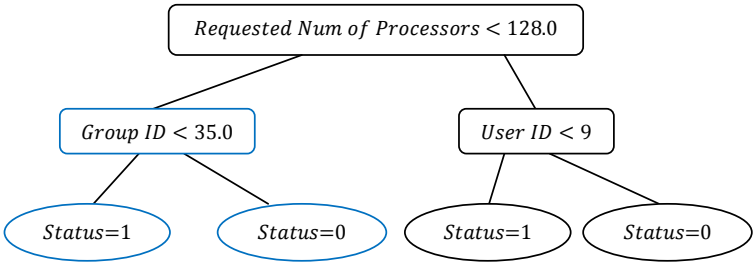
Prvi roditelj:



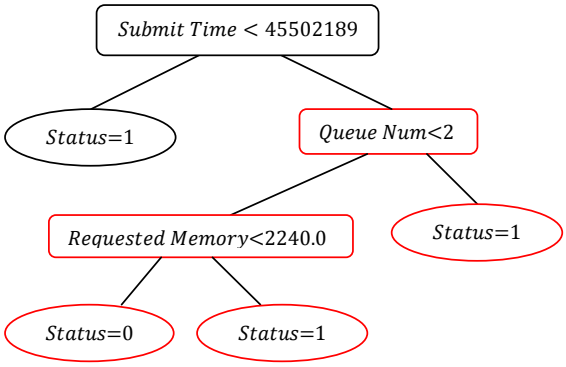
Drugi roditelj:



Prvo dijete:



Drugo dijete:

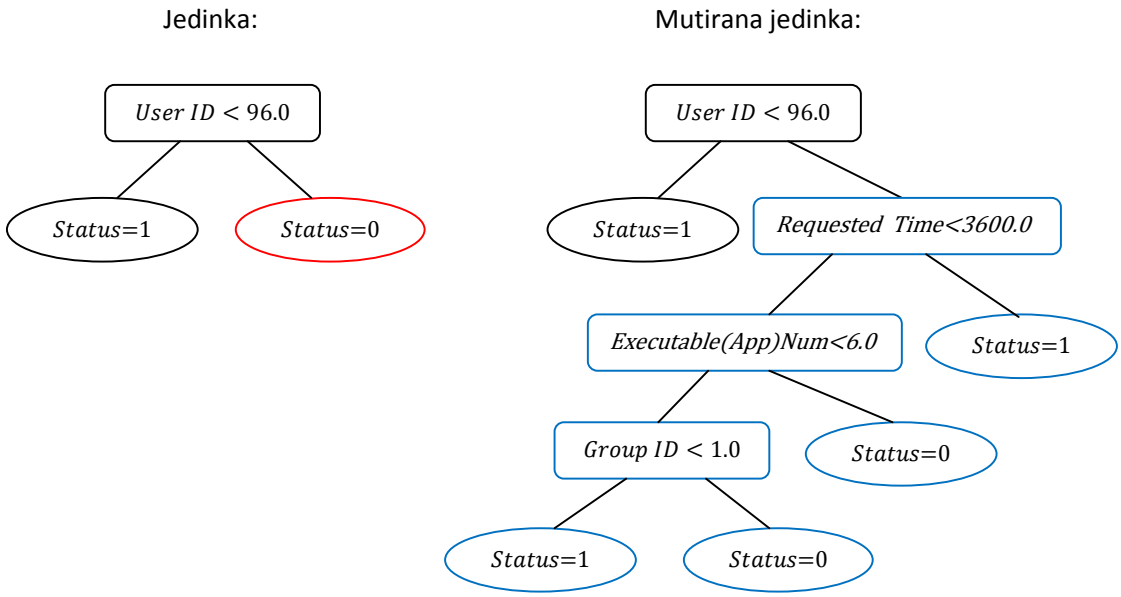


Slika 7.6 Grafički prikaz križanja razmjenom slučajno odabranih podstabala

7.1.1.6 Mutacija

Mutacija se izvodi s određenom vjerojatnošću – vjerojatnošću mutacije koja je najčešće jako mala, oko 1%. Mutacijom se slučajno odabire neki čvor u stablu te se podstablo kojem je taj čvor korijen zamjenjuje s novim podstablom koje je stvoreno jednom od metoda, ili *full*, ili *grow* metodom. Na slici 7.7 prikazana je mutacija. Slučajno je odabran jedan čvor u stablu (označen crvenom bojom), ovdje se baš radi o listu te je on zamijenjen novostvorenim stablom stvorenim *grow* metodom veličine 6 (označeno plavom bojom na slici). Mutacija se može shvatiti i kao križanje jedinke sa slučajno stvorenim stablom.

Postoje još neke vrste mutacije, iako je programski ostvarena samo prethodno navedena.



Slika 7.7 Grafički prikaz mutacije dodavanjem novog podstabla

Postoji mutacija koja mijenja svaki čvor u stablu s određenom vjerojatnošću (engl. *point mutation*). Promjena čvora znači odabir nekog drugog znaka iz skupa funkcijskih ili nekog drugog znaka iz skupa završnih znakova ako se radi o listu i zamjena novog znaka sa starim. Zatim postoji mutacija kod koje se slučajno odabrani čvor u stablu zamjenjuje slučajno nastalim stablom, ali je dano je ograničenje na veličinu stabla koje će se stvoriti – veličina mu je iz intervala $\left[\frac{1}{2}l, \frac{3}{2}l\right]$, gdje je l veličina stabla koje se treba mutirati, a nerijetko veličina novostvorenog stabla može biti jednaka stablu koje sudjeluje u mutaciji (engl. *size – fair subtree mutation*). Mutacija koja slučajno odabire podstablo jedinke koja sudjeluje u mutaciji i zamjenjuje ga sa slučajno odabranim čvorom te iste jedinke još je jedna vrsta (engl. *hoist mutation*).

Njen cilj, kao i cilj mutacije koja slučajno odabrani čvor zamjenjuje završnim čvorom (engl. *shrink mutation*), je smanjiti veličinu mutirane jedinke.

7.1.1.7 Rekombinacija

Ako *p* - zbroj vjerojatnosti mutacije i križanja nije jednak 100%, tada se s vjerojatnošću *1 – p* odvija rekombinacija. Rekombinacija turnirskom selekcijom odabire jednu jedinku te je kopira u novu populaciju.

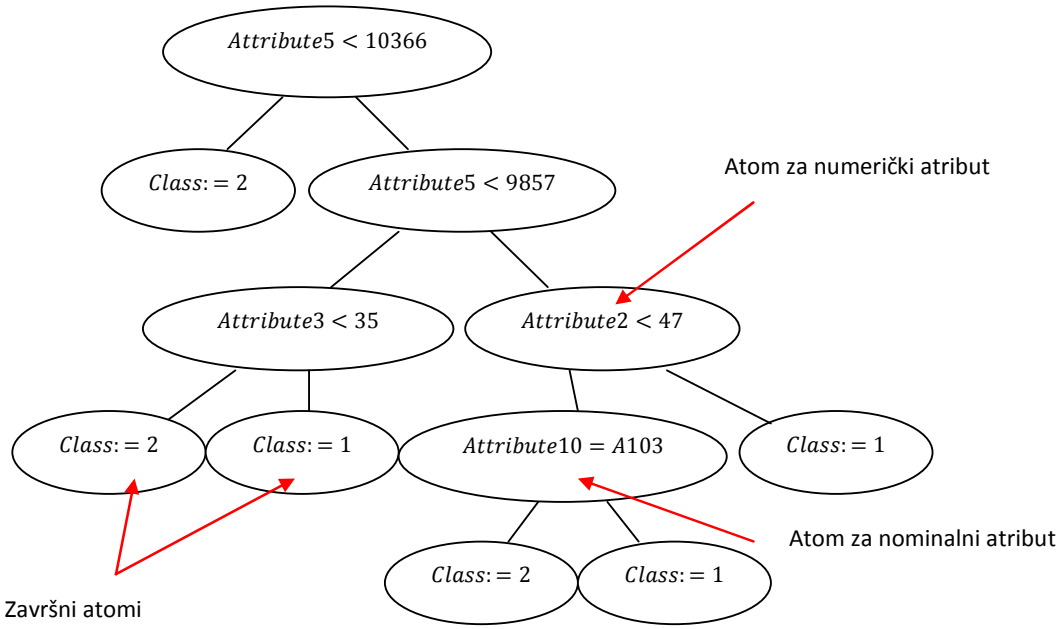
Križanje se puno češće izvodi od mutacije. Jedna najbolja jedinka prenosi se u sljedeću generaciju bez djelovanja genetskih operatora te je tako uveden i elitizam.

7.2 Različiti načini predstavljanja rješenja

Ovdje su opisane programski ostvarene vrste genetskog programiranja iz [11] za *German credit set* te skupove podataka iz arhive paralelnog opterećenja grozdova. Razlika između jednostavnog GP i ostalih vrsta je u tome što jednostavni GP za izgradnju stabala koristi sve atribute i za svaki atribut sve moguće vrijednosti atributa, dok ostali na temelju dodatnih izračuna biraju samo neke vrijednosti za koje se smatra kako će pridonijeti poboljšanju rezultata klasifikacije. Veličina prostora pretraživanja, kao i veličina skupa funkcijskih znakova puno su manje kod ostalih metoda nego kod jednostavnog GP – a.

7.2.1 Jednostavni GP

Ovdje je rješenje predstavljeno stablom u kojem se nalaze atomi. Atomi su oblika $atribut_i < vrijednost$, ako je $atribut_i$ numerički ili oblika $atribut_i = vrijednost$ ako se radi o nominalnom atributu. Za stvaranje ovakvih atributa koriste se sve moguće kombinacije (atribut, vrijednost) iz skupa podataka. Npr., kod skupa podataka *German Credit*, postoje i nominalni i numerički atributi i, pri izgradnji stabala, koristit će se oba tipa atoma, dok se kod skupa podataka iz *PWA* koriste samo atomi oblika $atribut_i < vrijednost$, jer se radi samo o numeričkim atributima. Primjer stabla za taj skup nalazi se na slici 7.1. Primjer stabla za skup *German Credit* nalazi se u nastavku, na slici 7.8. Na slici se u čvorovima stabla vide obje vrste atoma (za numeričke i nominalne atribute) i atomi oblika *Class := 1* ili *Class := 2* za listove.



Slika 7.8 Prikaz jednog mogućeg rješenja za skup podataka German credit data

Veličina prostora pretraživanja ovdje ovisi o najvećoj dopuštenoj veličini stabala, o broju funkcijskih i završnih atoma od kojih se može sagraditi stablo odluke.

U sljedećim potpoglavljima opisat će se način određivanja različitih atoma za izgradnju GP stabala odluke te objasniti i prikazati kako je time smanjen prostor pretraživanja u odnosu na njegovu veličinu kod korištenja jednostavnog GP. GP temeljen na informacijskoj dobiti i onaj temeljen na omjeru dobitka koriste informacijsku dobit iomjer dobitka (preuzete iz algoritama ID3, tj. algoritma C4.5) kako bi podijelili domenu vrijednosti atributa u grupe i time smanjili broj funkcijskih atoma uz čiju se pomoć grade stabla. Treći, GP temeljen na grupiranju, koristi jedan od poznatih algoritama za grupiranje kako bi se smanjio prostor pretraživanja.

7.2.2 GP stabla izgrađena uz pomoć informacijske dobiti

Računanjem informacijske dobiti prema algoritmu ID3 (engl. *Induction of Decision Trees*) određuje se kako dobro pojedini atribut odjeljuje uzorke za učenje u skladu s ciljnom klasifikacijom. Ona se računa prema formuli (7.3) [11].

$$informacijska\ dobit[X|S] = entropija(S) - količina\ informacije[X|S] \qquad (7.3)$$

Informacijska dobit računa se kao razlika između entropije danog skupa S i količine informacije koja se dobije podjelom skupa S prema određenom testu X.

Entropija skupa S računa se prema formuli (7.4).

$$entropija(S) = -\sum_{i=1}^{C} p_i \cdot log_2\, p_i \tag{7.4}$$

Udio uzoraka u cijelom skupu S koji pripadaju razredu i (C je ukupan broj razreda) je p_i.

Količina informacije dobivena podjelom skupa S prema testu X računa se prema formuli (7.5).

$$količina\ informacije[X|S] = \sum_{i=1}^{N} \frac{|S_i^X|}{|S|} \cdot entropija(S_i^X) \tag{7.5}$$

Količina informacije dobije se računanjem entropije dijela skupa koji predstavlja jednu podjelu prema testu X (N je ukupan broj podjela skupa S). S_i^X je i – ta podjela skupa S prema testu X, gdje je $S_i^X \subseteq S$. $|S_i^X|$ je kardinalitet tog podskupa, a $|S|$ je kardinalitet skupa S.

Slijedi primjer računanja informacijske dobiti na jednostavnom primjeru iz tablice 7.1. Pretpostavlja se da se u skupu S nalazi samo jedan atribut A te da postoje samo dva razreda s oznakama 0 i 1.

Tablica 7.1 Jednostavan skup podataka

atribut A	oznaka razreda
10	0
20	0
30	1
40	0
50	0
60	1

Atomi koji se mogu izvući iz gornjeg skupa podataka su oblika $A < vrijednost\ praga$, a to su $A < 10$, $A < 20$, $A < 30$, $A < 40$, $A < 50$, $A < 60$ (to će biti i pojedini testovi X_i po kojima se dijeli skup podataka) te završni atomi $Razred = 0$ i $Razred = 1$. Prvo se računa entropija skupa S (kardinalitet skupa: $|S| = 6$). Broj uzoraka koji pripadaju razredu s oznakom 1 je dva, a onih koji pripadaju razredu 0 je pet. Prema tome, udio uzoraka koji pripadaju razredu 1 je $\frac{2}{6} = \frac{1}{3}$.

$$entropija(S) = -\sum_{i=1}^{2} p_i \cdot log_2 p_i = -\frac{2}{3} \cdot log_2 \frac{2}{3} - \frac{1}{3} log_2 \frac{1}{3} = 0.9183$$

Nakon toga se računa količina informacije koja se dobije podjelom skupa po pojedinim vrijednostima praga (a to su sve vrijednosti danog atributa koje dijele skup podataka, npr. 10 ovdje nije vrijednost praga, jer test $A < 10$ ne dijeli skup podataka S).

$$količina\ informacije[A < 20|S] = \frac{1}{6} \cdot (-1 \cdot log_2 1) + \frac{5}{6} \cdot \left(-\frac{3}{4} \cdot log_2 \frac{3}{4} - \frac{1}{4} \cdot log_2 \frac{1}{4}\right) \approx 0.6761$$

$$količina\ informacije[A < 30|S] = \frac{2}{6} \cdot \left(-\frac{2}{2} \cdot log_2 \frac{2}{2}\right) + \frac{4}{6} \cdot \left(-\frac{2}{4} \cdot log_2 \frac{2}{4} - \frac{2}{4} \cdot log_2 \frac{2}{4}\right) \approx 0.\dot{6}$$

$$količina\ informacije[A < 40|S] = \frac{3}{6} \cdot \left(-\frac{2}{3} \cdot log_2 \frac{2}{3} - \frac{1}{3} \cdot log_2 \frac{1}{3}\right) + \frac{3}{6} \cdot \left(-\frac{2}{3} \cdot log_2 \frac{2}{3} - \frac{1}{3} \cdot log_2 \frac{1}{3}\right) \approx 0.9183$$

$$količina\ informacije[A < 50|S] = \frac{4}{6} \cdot \left(-\frac{3}{4} \cdot log_2 \frac{3}{4} - \frac{1}{4} \cdot log_2 \frac{1}{4}\right) + \frac{2}{6} \cdot \left(-\frac{1}{2} \cdot log_2 \frac{1}{2} - \frac{1}{2} \cdot log_2 \frac{1}{2}\right) \approx 0.2076$$

$$količina\ informacije[A < 60|S] = \frac{5}{6} \cdot \left(-\frac{4}{5} \cdot log_2 \frac{4}{5} - \frac{1}{5} \cdot log_2 \frac{1}{5}\right) + \frac{1}{6} \cdot (-1 \cdot log_2 1) \approx 0.6016$$

Nakon toga se izračunava informacijska dobit:

$$informacijska\ dobit[A < 20|S] \approx 0.9183 - 0.6761 = 0.2421$$

$$informacijska\ dobit[A < 30|S] \approx 0.9183 - 0.\dot{6} = 0.2516$$

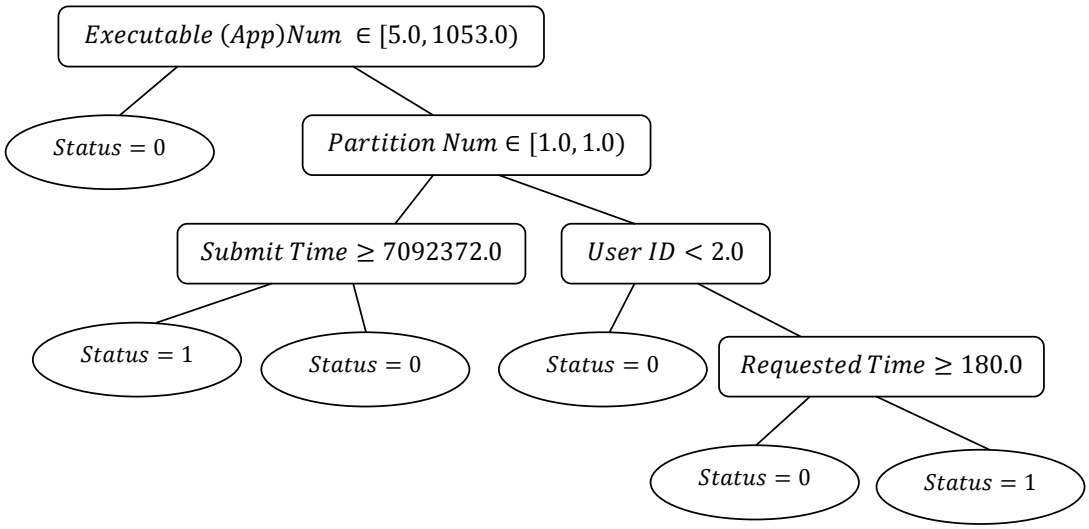
$$informacijska\ dobit[A < 40|S] \approx 0.9183 - 0.9183 = 0$$

$$informacijska\ dobit[A < 50|S] \approx 0.9183 - 0.2076 = 0.7107$$

$$informacijska\ dobit[A < 60|S] \approx 0.9183 - 0.6016 = 0.3167$$

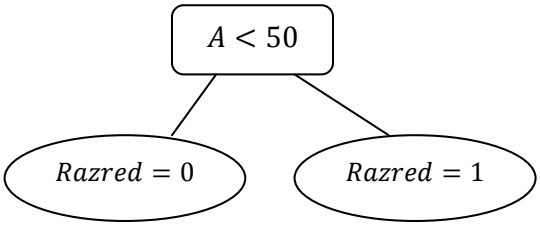
Kako bi se odabrao najbolji test, tj. najbolja podjela prostora, gleda se koja podjela prostora ima najveću vrijednost informacijske dobiti, u ovom primjeru to je podjela prostora na $A < 50$. To znači da će se, pri izgradnji stabala, za atribut A koristiti samo atom $A < 50$. Prvi podskup sadrži četiri uzorka, a drugi dva uzorka.

Na slici 7.9 prikazano je stablo nastalo podjelom domene atributa i računanjem informacijske dobiti svih mogućih kombinacija vrijednosti atributa učitanog skupa.



Slika 7.9 Primjer stabla nastalog GP – om temeljenim na informacijskoj dobiti

U nastavku, na slici 7.10 prikazano je stablo nastalo ovom vrstom GP – a iz primjera u tablici 7.1.



Slika 7.10 Primjer jednostavnog stabla

(Executable (App) Num ∈ [5.0, 1053.0) Status = 0 (Partition Num ∈ [1.0, 1.0) (Submit Time >= 7092372.0 Status = 1 Status = 0) (User ID < 2.0 Status = 1 (Requested Time >= 180.0 Status = 0 Status = 1))))

Slika 7.11 Stablo u prefiks obliku

Prefiks oblik stabla sa slike 7.9 nalazi se na slici 7.11.

7.2.3 GP stabla izgrađena uz pomoć omjera dobitka

Omjer dobitka (engl. *gain – ratio*) koristi informacijsku podijeljenost (engl. *split information*) koja se računa na sličan način kao i entropija za neki skup, jedina razlika je u tome što se po podskupovima ne gleda udio uzoraka po razredima, već samo koliko uzoraka po podskupovima kod svake podjele skupa ima, prema formuli (7.6).

$$informacijska\ podijeljenost[X|S] = -\sum_{i=1}^{N} \frac{|S_i^X|}{|S|} \cdot log_2 \frac{|S_i^X|}{|S|} \quad (7.6)$$

S_i^X je i – ti podskup nakon podjele skupa S po X.

Omjer dobitka računa se po formuli (7.7).

$$omjer\ dobitka[X|S] = \frac{informacijska\ dobit[X|S]}{informacijska\ podijeljenost[X|S]} \quad (7.7)$$

Informacijska dobit računa količinu informacije koja se dobiva podjelom skupa na podskupove, dok omjer dobitka računa udio informacije koja je korisna za klasifikaciju.

Neka je za primjer ponovo dan skup podataka iz tablice 7.1.

Za računanje omjera dobitka potrebno je izračunati informacijsku podijeljenost za svaku od podjela; informacijska dobit je već izračunata u prethodnom koraku.

$$informacijska\ podijeljenost[A < 20|S] = -\frac{1}{6} \cdot log_2 \frac{1}{6} - \frac{5}{6} \cdot log_2 \frac{5}{6} \approx 0.65$$

$$informacijska\ podijeljenost[A < 30|S] = -\frac{2}{6} \cdot log_2 \frac{2}{6} - \frac{4}{6} \cdot log_2 \frac{4}{6} \approx 0.9183$$

$$informacijska\ podijeljenost[A < 40|S] = -\frac{3}{6} \cdot log_2 \frac{3}{6} - \frac{3}{6} \cdot log_2 \frac{3}{6} \approx 1$$

Informacijska podijeljenost za podjelu prostora na $A < 50$ jednaka je informacijskoj podijeljenosti za $A < 30$, a ona za podjelu na $A < 60$ jednaka je informacijskoj podijeljenosti za $A < 20$.

Omjeri dobitka jednaki su:

$$omjer\ dobitka[A < 20|S] = \frac{0.2421}{0.65} \approx 0.3725$$

$$omjer\ dobitka[A < 30|S] = \frac{0.2516}{0.9183} \approx 0.3740$$

$$omjer\ dobitka[A < 40|S] = \frac{0\ b}{1} \approx 0$$

$$omjer\ dobitka[A < 50|S] = \frac{0.7107}{0.9183} \approx 0.7739$$

$$omjer\ dobitka[A < 60|S] = \frac{0.3167}{0.65} \approx 0.4872$$

I ovdje se gleda za koju se podjelu skupa dobije najveći omjer dobitka, a to je za podjelu na $A < 50$.

Domena pojedinog atributa dijeli se na nekoliko podskupova (na najmanje 2 podskupa) te se na osnovi odabranih atributa stvaraju atomi od kojih će se izgraditi stablo.

Općenito, neka u nekom skupu S postoji atribut A_i te neka je s V_i označen skup mogućih vrijednosti pragova $V_i = \{v_1^i, \dots, v_{k-2}^i\}$, gdje je $n \leq k-1$ broj mogućih različitih vrijednosti praga, a k je najveći dopušteni broj podjela skupa. Kako bi se pronašao optimalan skup vrijednosti praga, potrebno je izračunati informacijsku dobit i/ili omjer dobitka za najviše k – 1 vrijednosti pragova, tako da $informacijska\ dobit$ (i/ili $omjer\ dobitka$)$[A_i < v_1^i, A_i \in [v_1^i, v_2^i), \dots, A_i \geq v_n | S]$ za određenu kombinaciju vrijednosti pragova bude veća ili jednaka od vrijednosti dobiti ili omjera za bilo koju drugu kombinaciju pragova.

Atomi koji nastaju ovim načinom su oblika:

- $A < prag_1$,
- $A \in [prag_1,\ prag_2), \dots A \in [prag_{i-1},\ prag_i)$ i
- $A \geq prag_i$.

Npr. za k = 2, za skup podataka u tablici 7.1, tražilo bi se k – 1 vrijednosti praga (znači, samo jedna). Budući da se računom dobilo kako se za atom $A < 50$ dobije najveća vrijednost omjera dobitka, odabrana vrijednsot praga bila bi 50. Za atribut A, atom $A < 50$ jedini bi se koristio kod izgradnje stabla. Za k = 3, tražile bi se dvije vrijednosti praga za svaki atribut. Skup bi se podijelio na tri dijela i za svaku kombinaciju mogućih vrijednosti praga (u ovom slučaju za kombinacije 20 i 30, 20 i 40, 20 i 50, 20 i 60, zatim za 30 i 40, 30 i 50, 30 i 60, za 40 i 50, 40 i 60 te 50 i 60) računala bi se informacijska dobit i/ili omjer dobitka te bi se izabrale one dvije vrijednosti praga za koju su dobit i/ili omjer najveći. Npr. ako bi se dobilo da se za kombinacije 20 i 50 dobiju najveće vrijednosti informacijske dobiti (omjera dobitka), jedini atomi koji bi se koristili za izgradnju stabala bili bi $A < 20, A \in [20, 50)$ i $A \geq 50$. Za k = 4, morale bi se naći tri vrijednosti pragova, npr. ako su to 20, 30 i 50, tada bi se za izgradnju stabala koristili atomi: $A < 20, A \in [20, 30), A \in [30, 50)$ i $A \geq 50$, tako dalje i za ostale brojeve podjela. Kako broj kombinacija koje se moraju ispitati i za koje se moraju računati informacijska dobit i/ili omjer dobitka raste eksponencijalno s povećanjem broja podjela (particija skupa) k, tada će se, kod eksperimenata u ovom radu, broj podjela ograničiti na k = 2 i k = 3. Isto tako, zbog jako velikog broj uzoraka na raspolaganju za pojedine grozdove za ispitivanje se, zato što bi računanje za sve kombinacije jako dugo trajalo, neće uzimati svi uzorci, već najviše njih 1200. Pretpostavlja se da se iz dostupnih skupova podataka može uzeti manji broj uzoraka,

jer to ne bi smjelo smanjiti točnost klasifikacije. U literaturi se spominje kako je 5000 uzoraka u skupu za učenje iskustveni maksimum do kojeg se očekuje poboljšanje kvalitete rezultata [22].

Stablo na slici 7.9 moglo je nastati i ovom vrstom GP - a.

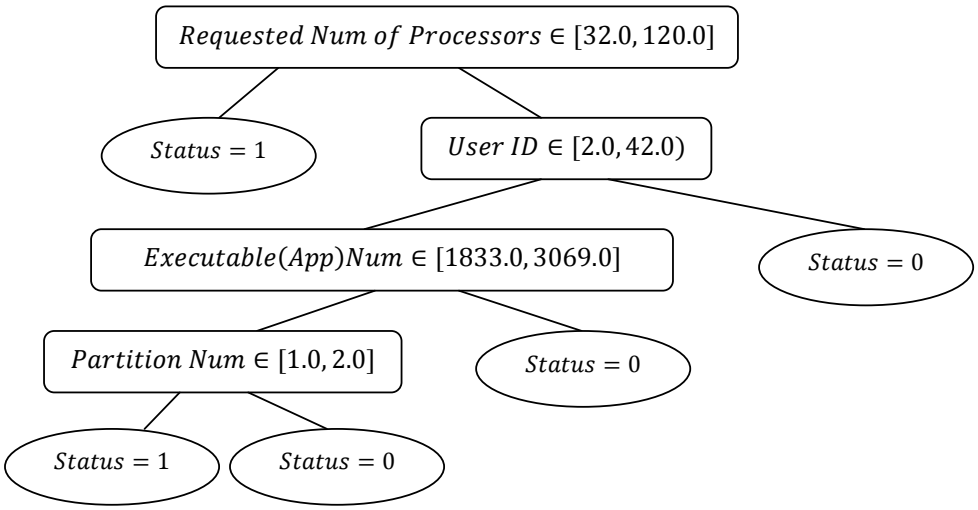
7.2.4 GP stabla izgrađena uz pomoć grupiranja

Za grupiranje vrijednosti (numeričkih) atributa koristi se algoritam K srednjih vrijednosti (engl. *K – means Algorithm*). Ovaj deterministički algoritam vrlo je brz, a jednostavan. Grupiraju se vrijednosti za svaki od atributa (realni brojevi), što znači da se radi o jednodimenzionalnim uzorcima. Na početku se odabere $K \leq N$ središta grupa, gdje je N broj uzoraka. Iako bi se za odabiranje početnih središta grupa (centara grupa) mogle koristiti i neke složenije metode, ovdje se početni centri slučajno odabiru. U k – toj iteraciji ovog algoritma uzorci se razdijele po grupama na način da uzorak x pripada grupi G_i (k) ako i samo ako je $\|x - c_j(k)\| < \|x - c_i(k)\|$, gdje je $\|x - c_j(k)\|$ oznaka za Euklidsku udaljenost uzorka x od centra c_j. Uzorci se stavljaju u onu grupu čijem su središtu najbliži. Nakon toga se izračunavaju nova središta grupa kao srednja vrijednost svih uzoraka u svakoj grupi te se postupak pridjeljivanja uzoraka novim grupama odvija sve dok $c_j(k+1) = c_j(k)$, za sve $j = 1, 2, \dots, K$ tj. sve dok centri u trenutnom i prethodnom koraku nisu isti.

Ako je najmanja vrijednost i – te grupe označena s min_i, a najveća s max_i, tada su atomi koji nastaju ovim načinom oblika $A \in [min_i,\ max_i]$. Npr. za četiri grupe atomi bi izgledali ovako:

- $A \in [min_1,\ max_1]$,
- $A \in [min_2,\ max_2]$,
- $A \in [min_3,\ max_3]$ i
- $A \in [min_4,\ max_4]$.

Na slici 7.12 prikazano je stablo nastalo grupiranjem vrijednosti pojedinih atributa učitanog skupa.



Slika 7.12 Stablo izgrađeno uz pomoć grupiranja algoritmom K srednjih vrijednosti

7.2.5 Metode evaluacije

Od metoda evaluacije programski su ostvarene *Holdout* metoda, metoda k – struke unakrsne provjere, *Leave – one – out* metoda, metoda ponovne zamjene te metoda koja dijeli skup na tri dijela – skup za učenje, skup za provjeru modela (uzima se 20% skupa za učenje, pri čemu se na dijelu koji je odvojen za provjeru klasifikator ne uči) te skup za ispitivanje. Ove metode su detaljnije opisane u petom poglavlju. Kod k – struke unakrsne provjere pojedini uzorci se u skupove svrstavaju slučajnim odabirom. S obzirom da je zadatak ovog rada rješavanje problema klasifikacije, a ne raspoređivanja poslova, pretpostavlja se da je vrijednost svakog atributa svakog posla koji se nalazi u skupu za ispitivanje poznata, osim statusa čija se vrijednosti jedina previđa. Vrijeme se prilikom odvajanja cijelog skupa u skupove za učenje i ispitivanje ne uzima u obzir, tj. poslovi ne moraju biti poredani prema vremenu dospjeća u sustav niti je potrebno paziti na redoslijed njihovog izvršavanja.

7.3 Grafičko korisničko sučelje

U izradi klasifikatora koji za klasifikaciju koristi genetsko programiranje korištena je razvojna okolina Eclipse i programski jezik Java.

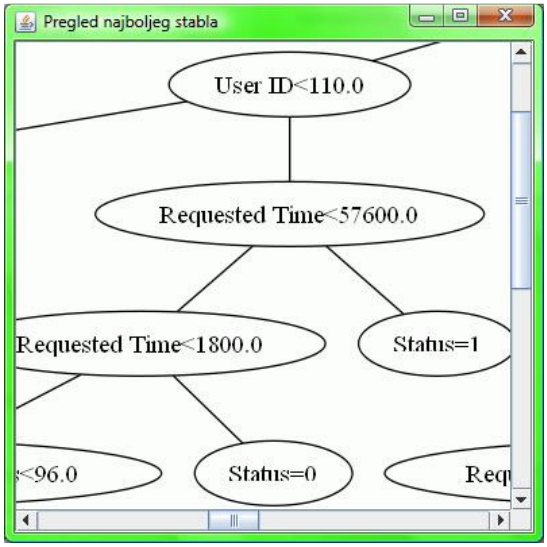
7.3.1 Upute za korištenje

Korisnik nakon klika na gumb *Odabrite i učitajte datoteku...*, kao što se vidi na prvoj slici, može odabrati željenu datoteku s podacima. Ne moraju se učitavati svi podaci, već se može učitati i manji broj uzoraka (u tom je slučaju potrebno u kućicu *Broj*

uzoraka upisati koliko uzoraka se želi učitati, inače je označeno da se svi uzorci učitavaju (odabir *Svi uzorci*)). Podaci se nakon toga učitavaju i na glavnom prozoru se ispod gumba za učitavanje ispisuju osnovni podaci o izabranoj datoteci, kao što su ime datoteke, koliko ima ukupno uzoraka (poslova) itd. Nakon toga, korisnik mora učitati parametre za početak procesa klasifikacije, tj. za pokretanje algoritma. Korisnik može unijeti sljedeće parametre: veličinu populacije, broj generacija, parametar k za k – turnirsku selekciju, zatim vjerojatnost mutacije i križanja te najveću dopuštenu veličinu stabla. U procesu učenja klasifikatora, sva stabla koja budu veća od unesene najveće dopuštene veličine bit će podrezana. Klikom na gumb *Početne vrijednosti* u prostor za parametre upisuju se određene vrijednosti parametara pa se učenje klasifikatora može pokrenuti i s tim vrijednostima (ako korisnik, npr., trenutno nema ideju koje bi vrijednosti unio za koje parametre).

Korisnik može odabrati koju će se vrstu algoritma koristiti, tj. time se određuju svi mogući atomi za izgradnju stabala tijekom procesa evolucije. Korisnik mora odabrati i koju metodu evaluacije želi koristiti te parametar za *Holdout* metodu (postotak uzoraka iz cijelog učitanog skupa koji će se nalaziti u skupu za testiranje) ili broj grupa *k* ako je odabrana k – struka unakrsna provjera. Ako je odabrana bilo koja od preostalih metoda, parametar se ne unosi, jer za njihovo izvođenje nije potreban nikakav parametar. Ako korisnik ne unese neki od parametara, ispisat će mu se upozorenje da nije unio neki od parametara i daljnji rad će biti onemogućen sve dok se ne unesu parametri koji nedostaju. Klikom na gumb *Pokreni* pokreće se učenje (i, nakon toga, ispitivanje) klasifikatora. U donjem lijevom kutu ispisuje se dobrota najbolje jedinke u svakoj generaciji, a u donjem desnom kutu može se pratiti tijek rada. Na prvom grafičkom indikatoru tijeka rada može se pratiti tijek rada određivanja atoma koji će se koristiti za izgradnju stabala, a drugi prikazuje tijek učenja i ispitivanja klasifikatora. Na svakom je ispisan i postotak, tj. koliko je već prošlo od početka rada, a u prostor ispod se ispisuje koja se trenutna generacija izvodi ili o kojoj se trenutno grupi radi (ako je za metodu evaluacije izabrana k – struka unakrsna provjera ili *Leave – one – out* metoda).

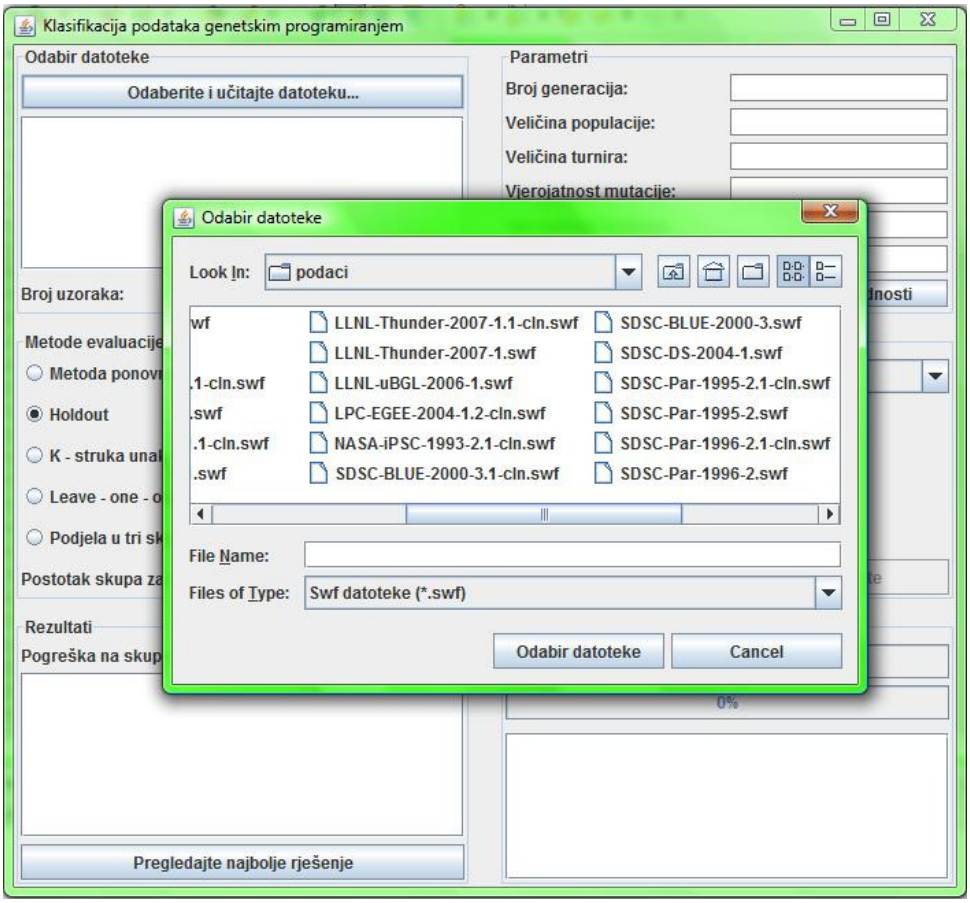
Nakon završetka rada, osim što se može vidjeti kolika je pogreška klasifikacije na skupu za ispitivanje (testiranje), može se kliknuti na gumb *Pregledajte najbolje rješenje*. Time se otvara novom prozoru u kojem je grafički prikazano najbolje stabla. Taj grafički prikaz nastaje uz pomoć alata *GraphViz*. Kako taj alat zahtijeva zapis stabla u tzv. *dot* formatu, ostvareno je i pretvaranje stabla u *dot* format, ono se zapisuje u datoteku *bestTree.dot* te se predaje alatu *GraphViz* koji stablo grafički prikazuje, kako je prikazano na slici 7.13.



Slika 7.13 Prozor za pregled najboljeg stabla

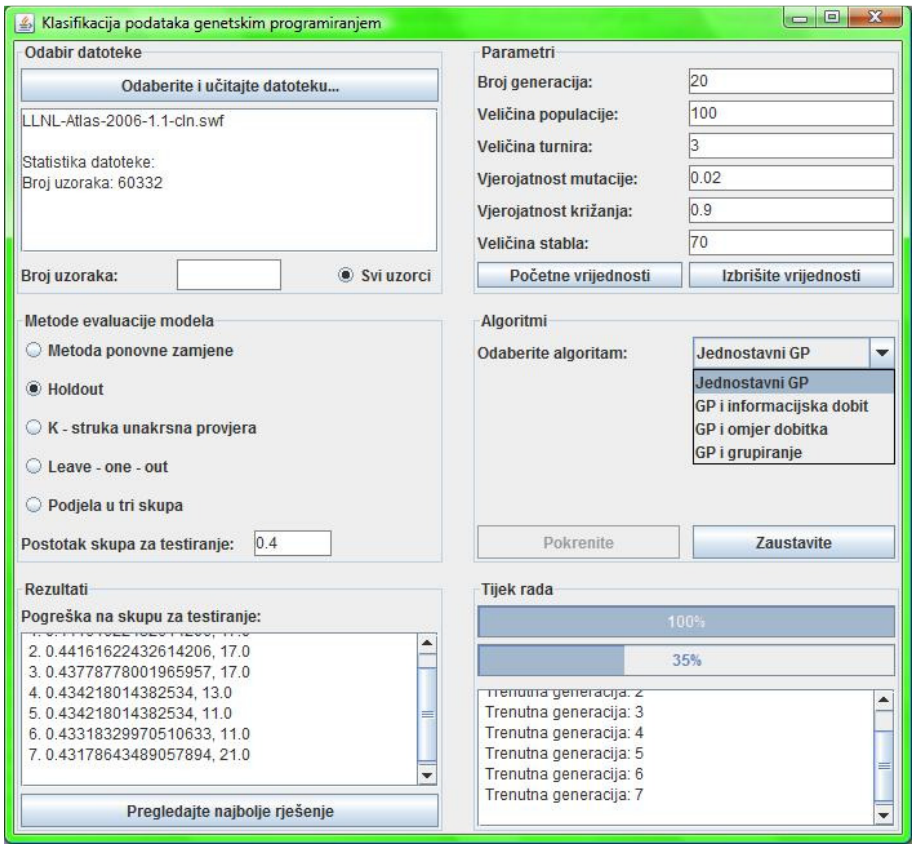
Najbolje stablo u svakoj generaciji ispisuje se u datoteku *bestTrees.txt* u prefiks obliku. Uz stablo se ispisuje i njegova dobrota i pogreška klasifikacije tog stabla na skupu za ispitivanje.

7.3.2 Prikaz grafičkog korisničkog sučelja



Slika 7.14 Prikaz prozora za odabir datoteke

Na slici 7.14 nalazi se prikaz prozora za odabir datoteke, tj. skupa podataka koji će se koristiti za klasifikaciju.



Slika 7.15 Prikaz tijeka rada na grafičkom korisničkom sučelju

Na slici 7.15 prikazano je kako grafičko sučelje izgleda tijekom izgradnje i rada klasifikatora.

8. Mjerenja

Provedeno je nekoliko eksperimenata, a cilj im je bio prikazati kretanje pogreške klasifikacije na skupu za učenje (što je dobrota pojedine jedinke) i na skupu za ispitivanje. Za većinu eksperimenata izvođenje algoritma pokrenuto je trideset puta i na grafovima je prikazana srednja pogreška klasifikacije, kao i najmanja i najveća vrijednost unutar tih tridesetak pokretanja. Najmanje i najveće vrijednosti na slikama prikazane su isprekidanim linijama (za skup za ispitivanje ovakvom linijom: ....., a za skup za učenje ovakvom: _ . _ . _, što nije posebno naznačeno u legendi grafa). Iznad svakog grafa označen je naziv datoteke nad kojom su se vršila mjerenja.

U prvom nizu eksperimenata mijenjali su se parametri genetskog programiranja, veličina populacije i broj generacija. Nakon toga, mijenjali su se parametri korištenih metoda evaluacije. U eksperimentima su se koristile dvije metode evaluacije, *Holdout* metoda i k – struka unakrsna provjera. Sukladno tome, mijenjao se udio uzoraka u skupu za ispitivanje, što je parametar *Holdout* metode i broj grupa (k) za k – struku unakrsnu validaciju. Ostale metode nisu se koristile pri mjerenjima; *LOOCV* metoda nije se koristila jer njeno izvršavanje traje predugo ako se učenje i ispitivanje vrše nad velikim brojem uzoraka, a za jednostavan GP koristila se većina uzoraka (nekoliko desetaka tisuća njih), dok se metoda ponovne zamjene nije koristila zbog svoje neučinkovitosti (nije preporučljivo klasifikator ispitivati nad istim uzorcima nad kojima je i učio). Iako se u većini gore opisanih eksperimenata dostupni skup podataka dijelio samo na podskup za učenje i podskup za ispitivanje, u jednom eksperimentu dodana je i podjela skupa na dodatni podskup koji služi za validaciju, tj. određivanje koliko je neko rješenje dobro na temelju validacijske pogreške koju to rješenje ima.

U tablici 8.1 prikazane su sve programski ostvarene kombinacije pojedinih vrsta GP – a i metoda evaluacije, ali zbog previše kombinacija, u eksperimentima su se koristile samo tri vrste GP – a (jednostavni GP, GP tremeljen na informacijskoj dobiti i GP temeljen na omjeru dobitka) te od metoda evaluacija *Holdout* metoda i k – struka unakrsna validacija.

Tablica 8.1 Prikaz svih kombinacija vrsta GP – a i metoda evaluacije

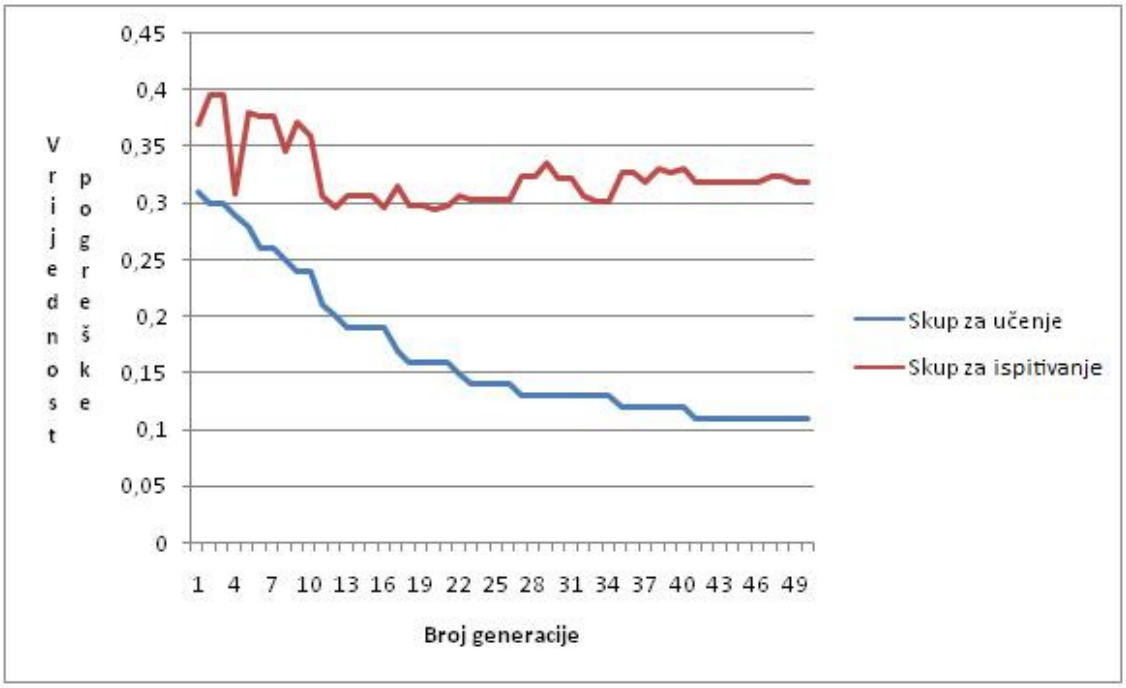
vrsta GP - a	metoda evaluacije
jednostavni GP	*Holdout* metoda k – struka unakrsna provjera *Leave – one – out* metoda metoda ponovne zamjene metoda podjele na tri skupa - (skup za učenje, ispitivanje i provjeru)
GP temeljen na informacijskoj dobiti	
GP temeljen na omjeru dobitka	
GP temeljen na grupiranju	

U većini mjerenja korišten je jednostavni GP, a nekoliko je mjerenja, koristeći samo *Holdout* metodu, imalo za cilj usporediti različite vrste GP – a (GP kod kojeg je za odabir atoma bilo potrebno računati informacijsku dobit i onaj kod kojeg se trebao računati omjer dobitka). S obzirom na problem eksponencijalnog rasta kombinacija koje se kod zadnje spomenutih dviju vrsta moraju računati, broj uzoraka koji se koriste za učenje i ispitivanje smanjen je na 1200. U sljedećem potpoglavlju dan je prikaz grafova koji su dobiveni korištenjem skupa *German credit*, a nakon toga prikazan je niz grafova na dvama skupovima iz arhive paralelnog opterećenja grozdova.

8.1.1 German credit data

Za ovaj skup podataka napravljeno je nekoliko eksperimenata za metode evaluacije *Holdout* i k – struku unakrsnu validaciju. Korišteni algoritam je jednostavni GP.

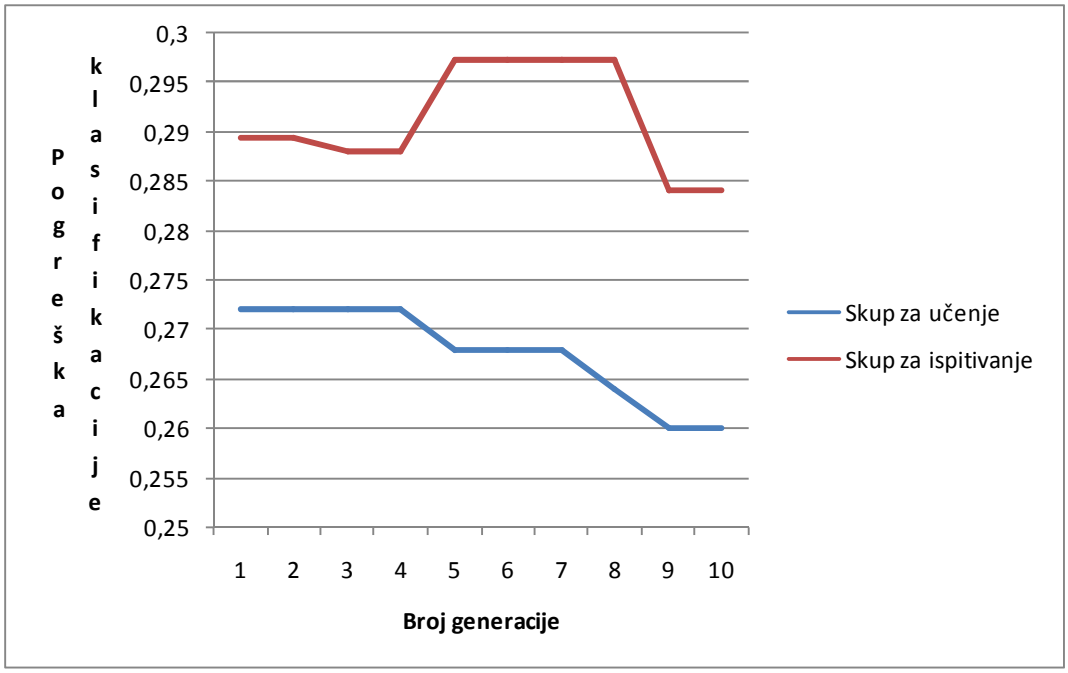
- 10 – struka unakrsna provjera, veličina populacije 500, broj generacija 50



Slika 8.1 Graf kretanja vrijednosti pogreške klasifikacije kroz generacije

Na ovom grafu (na slici 8.1) moguće je vidjeti kako pogreška na skupu za učenje pada, a na skupu za testiranje postiže minimum u 22. generaciji, a nakon toga raste.

- *Holdout* metoda, veličina populacije 1000, broj generacija 10



Slika 8.2 Graf kretanja pogreške klasifikacije kroz generacije

Prema grafu na slici 8.2 koji prikazuje kretanje pogreške na skupovima za učenje i ispitivanje kroz generacije, pogreška na skupu za učenje, također, pada. Pogreška na skupu za ispitivanje postiže minimum u devetoj generaciji. Ta pogreška pada do treće generacije, zatim raste u petoj generaciji, stagnira te ponovo pada u devetoj generaciji.

8.1.2 Skupovi podataka iz arhive paralelnog opterećenja grozdova

Za eksperimente su odabrane dvije datoteke: *LANL-CM5-1994-3.1-cln.swf* i *LLNL-Atlas-2006-1.1-cln.swf* u kojima se nalaze podaci o poslovima dvaju različitih grozdova, prvi prikupljeni u periodu u radoblju između 1994. i 1996., a drugi između 2006. i 2007. god. Prvi ima 201 387 poslova, a drugi manje – 60 332. Oni se razlikuju i po valjanim atributima koji su kao značajke uzoraka korišteni u klasifikaciji. U prvom skupu podataka ima više valjanih atributa (njih osam), dok ih drugi ima nešto manje - šest. Vrijednosti pogrešaka na skupovima za učenje i ispitivanje određivani su uvijek na najboljem stablu zadnje generacije.

8.1.2.1 Jednostavni GP

Sljedeća mjerenja izvedena su korištenjem jednostavnog GP – a.

- Promjena veličine populacije

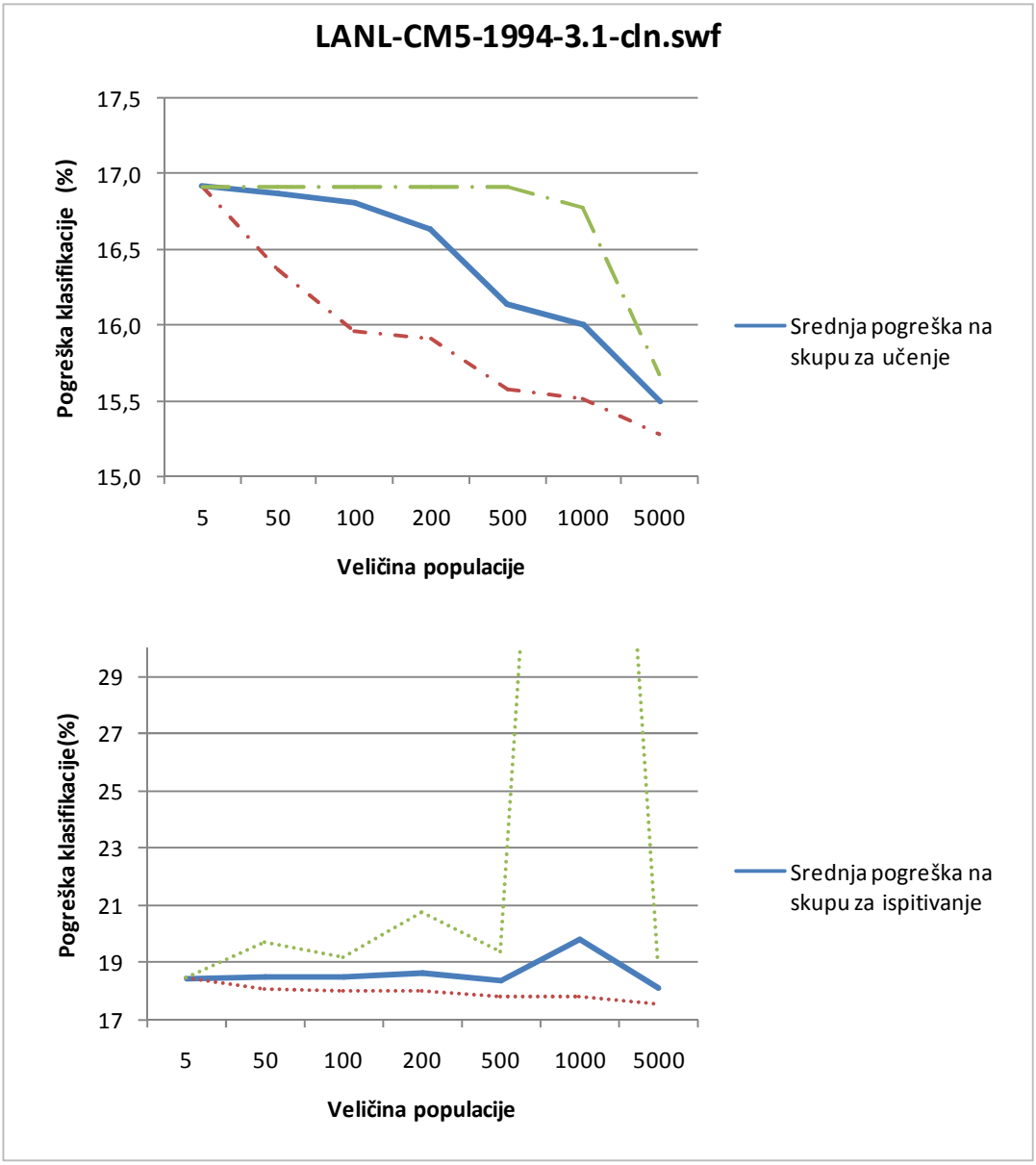
Mjerenja su izvršena za veličine populacije 5, 50, 100, 200, 500, 1000 i 5000. U tablicama 8.2 i 8.3 naznačene su srednja, minimalna i maksimalna pogreška klasifikacije u postotku u odnosu na različite veličine populacije na oba skupa, i na skupu za učenje, i na skupu za ispitivanje.

Za mjerenja čiji su rezultati prikazani u nastavku koristila se *Holdout* metoda – dostupni skup dijelio se na dva dijela, pri čemu je 75% uzoraka odlazilo u skup za testiranje, a 25% njih u skup za učenje. Vjerojatnost mutacije bila je jednaka 1%, a vjerojatnost križanja 90%. Dopuštena veličina stabla ograničena je na 70 čvorova. Parametar k – turnirske selekcije jednak je tri za sve eksperimente.

- *LANL-CM5-1994-3.1-cln.swf*

Tablica 8.2 Prikaz minimalne, srednje i maksimalne klasifikacijske pogreške za različite veličine populacije

	pogreška na skupu za učenje (%)			pogreška na skupu za ispitivanje (%)		
veličina populacije	srednja	minimalna	maksimalna	srednja	minimalna	maksimalna
5	16.9207	16.9207	16.9207	18.4664	18.4664	18.4664
50	16.8751	16.3680	16.9207	18.5021	18.0811	19.7163
100	16.8110	15.9598	16.9207	18.5137	18.0257	19.1624
200	16.6305	15.9129	16.9207	18.6039	18.0257	20.7531
500	16.1419	15.5697	16.9207	18.3932	17.7825	19.3864
1000	15.9978	15.5083	16.7762	19.8280	17.8066	61.3633
5000	15.4970	15.2771	15.6781	18.1235	17.5742	19.1167



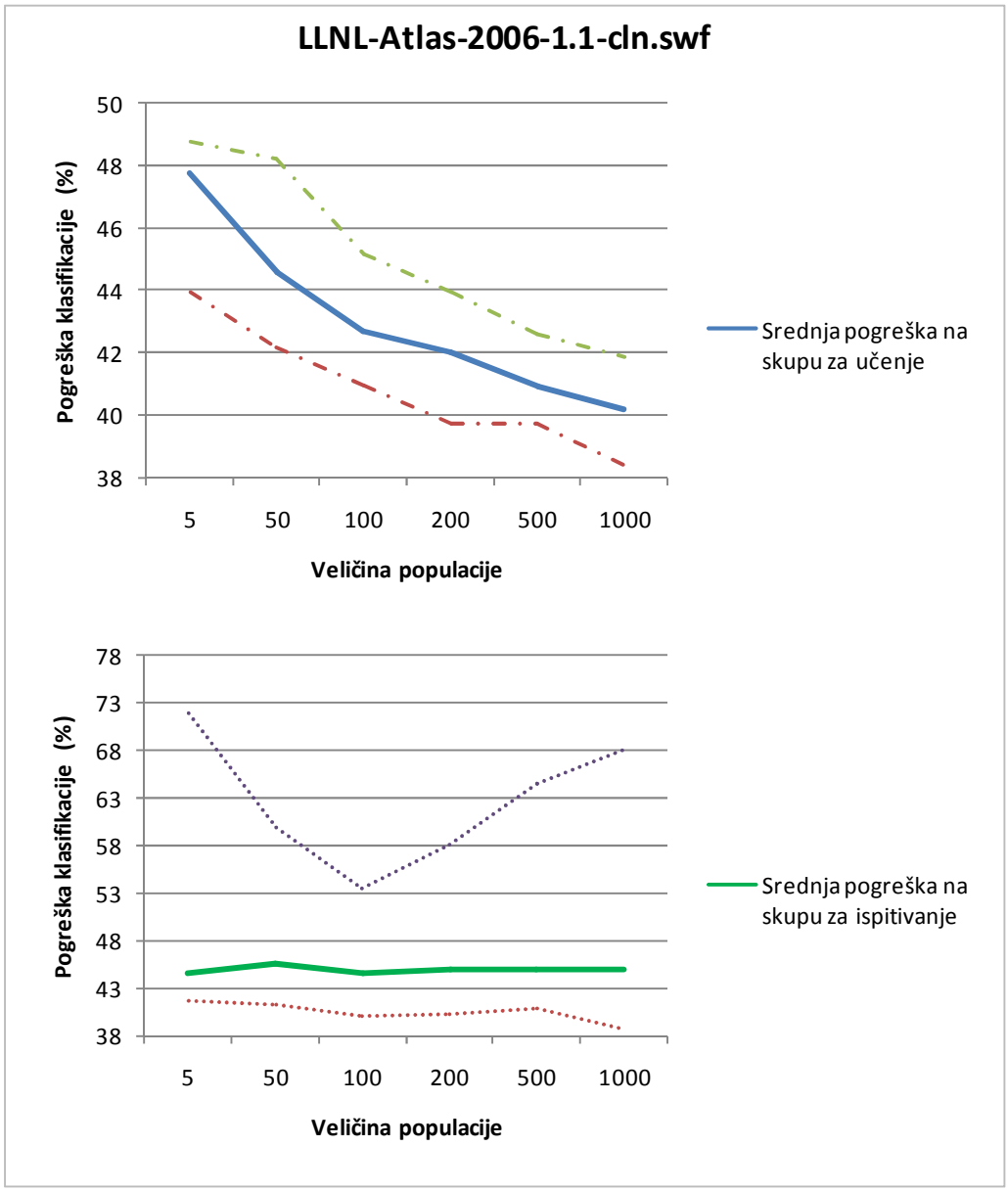
Slika 8.3 Graf kretanja pogreške klasifikacije (%) ovisno o veličini populacije

Na gornjem grafu na slici 8.3 može se vidjeti kako srednja pogreška na skupu za učenje pada od oko 17 do malo iznad 15%. Pogreška na skupu za ispitivanje, što se vidi na donjem grafu na slici 8.3 najprije pada pa neznatno raste (za veličinu populacije 200 i 1000), zatim opet pada. To znači da GP postaje bolji klasifikator za veću populaciju. Najmanja je vrijednost pogreške (oko 18%) ovdje izmjerena za veličinu populacije 5000, no, kako izvršavanje klasifikacije za tako veliku populaciju dugo traje, za daljnja mjerenja nad ovom datotekom koristi se veličina populacije 500.

- *LLNL-Atlas-2006-1.1-cln.swf*

Tablica 8.3 Prikaz minimalne, srednje i maksimalne klasifikacijske pogreške za različite veličine populacije

	pogreška na skupu za učenje (%)			pogreška na skupu za ispitivanje (%)		
veličina populacije	srednja	minimalna	maksimalna	srednja	minimalna	maksimalna
5	47.7752	43.9464	48.7520	44.5383	41.7491	72.0003
50	44.5867	42.1582	48.2429	45.5656	41.3104	59.9230
100	42.6897	40.9413	45.1509	44.6234	40.0977	53.4663
200	42.0514	39.7616	43.9588	45.0714	40.3253	58.1185
500	40.9090	39.7367	42.5928	45.0336	40.8468	64.5462
1000	40.1945	38.3956	41.8974	45.0336	38.6615	68.1843



Slika 8.4 Graf kretanja pogreške klasifikacije (%) ovisno o veličini populacije

Za razliku od prethodnog slučaja, ovdje je pogreška klasifikacije na oba skupa dosta velika (oko 40%). Moguće zato što se kod ovog skupa prilikom klasifikacije koristi samo šest atributa. Pogreška klasifikacije na skupu za učenje, kao što je i očekivano, pada, a pogreška na skupu za ispitivanje opet pada do veličine populacije 50, zatim raste pa opet pada, raste i konačno, za veliku populaciju, stagnira. Kako se najmanja srednja pogreška na skupu za ispitivanje (44.62 %) postiže za veličinu populacije 100, u daljnjim se mjerenjima za ovaj skup koristi veličina populacije 100.

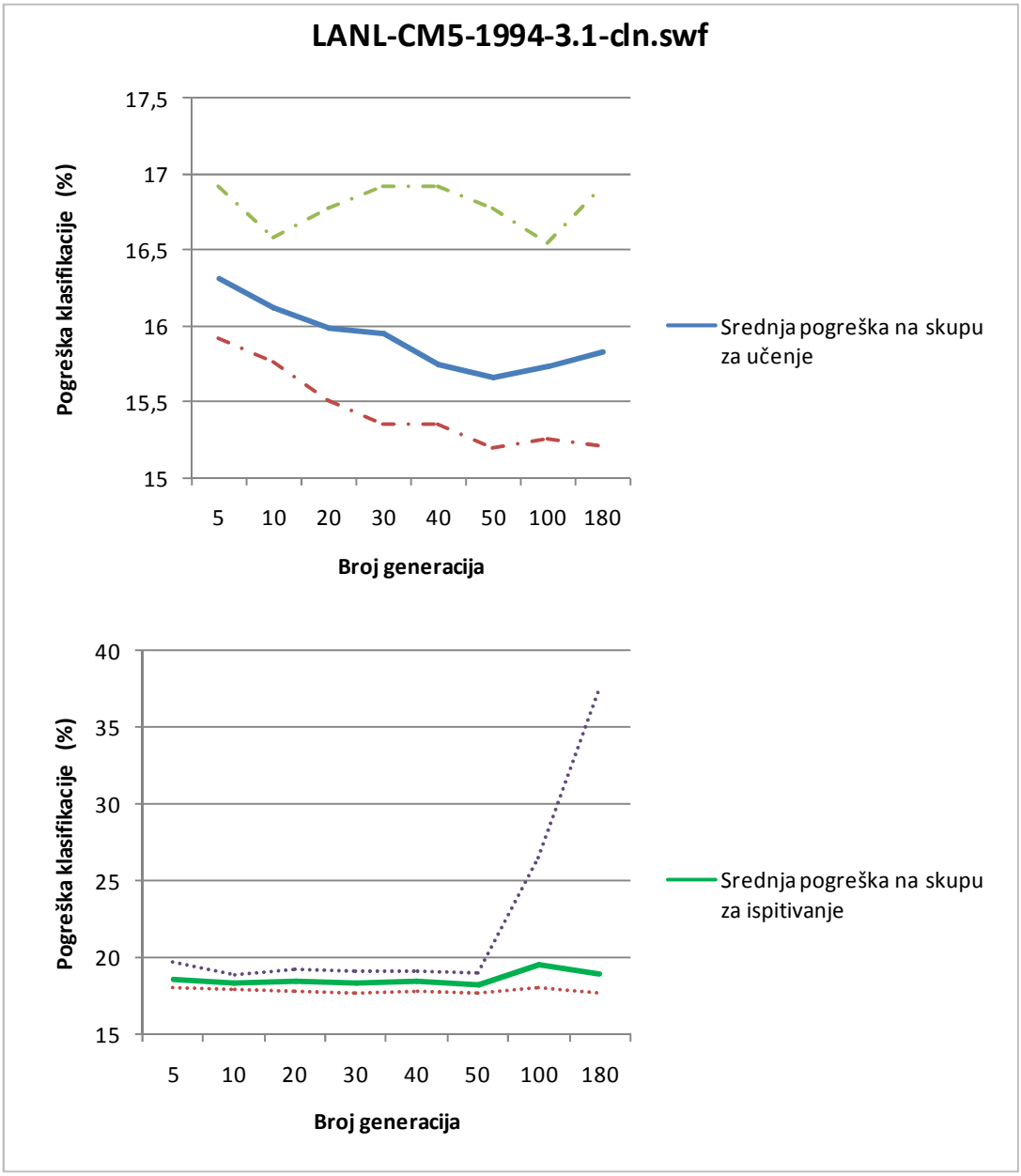
- Promjena broja generacija

Mjerenja su izvršena za broj generacija 5, 10, 20, 30, 40, 50 i 180. U tablicama 8.4 i 8.5 naznačene su srednja, minimalna i maksimalna pogreška klasifikacije u postotku u odnosu na različit broj generacija na oba skupa, i na skupu za učenje, i na skupu za ispitivanje.

- *LANL-CM5-1994-3.1-cln.swf*

Tablica 8.4 Prikaz minimalne, srednje i maksimalne klasifikacijske pogreške za različit broj generacija

	pogreška na skupu za učenje (%)			pogreška na skupu za ispitivanje (%)		
broj generacija	srednja	minimalna	maksimalna	srednja	minimalna	maksimalna
5	16.3173	15.9201	16.9207	18.5458	18.0245	19.6946
10	16.1147	15.7684	16.5848	18.3204	17.9595	18.9047
20	15.9912	15.5083	16.7762	18.3958	17.8066	19.2780
30	15.9528	15.3529	16.9207	18.3083	17.7006	19.0504
40	15.7449	15.3493	16.9207	18.4277	17.8307	19.1624
50	15.6666	15.1976	16.7799	18.2682	17.6609	18.9553
100	15.7333	15.2626	16.5450	19.5297	17.9800	26.5437
180	15.8328	15.2084	16.9171	18.9772	17.6199	37.6818



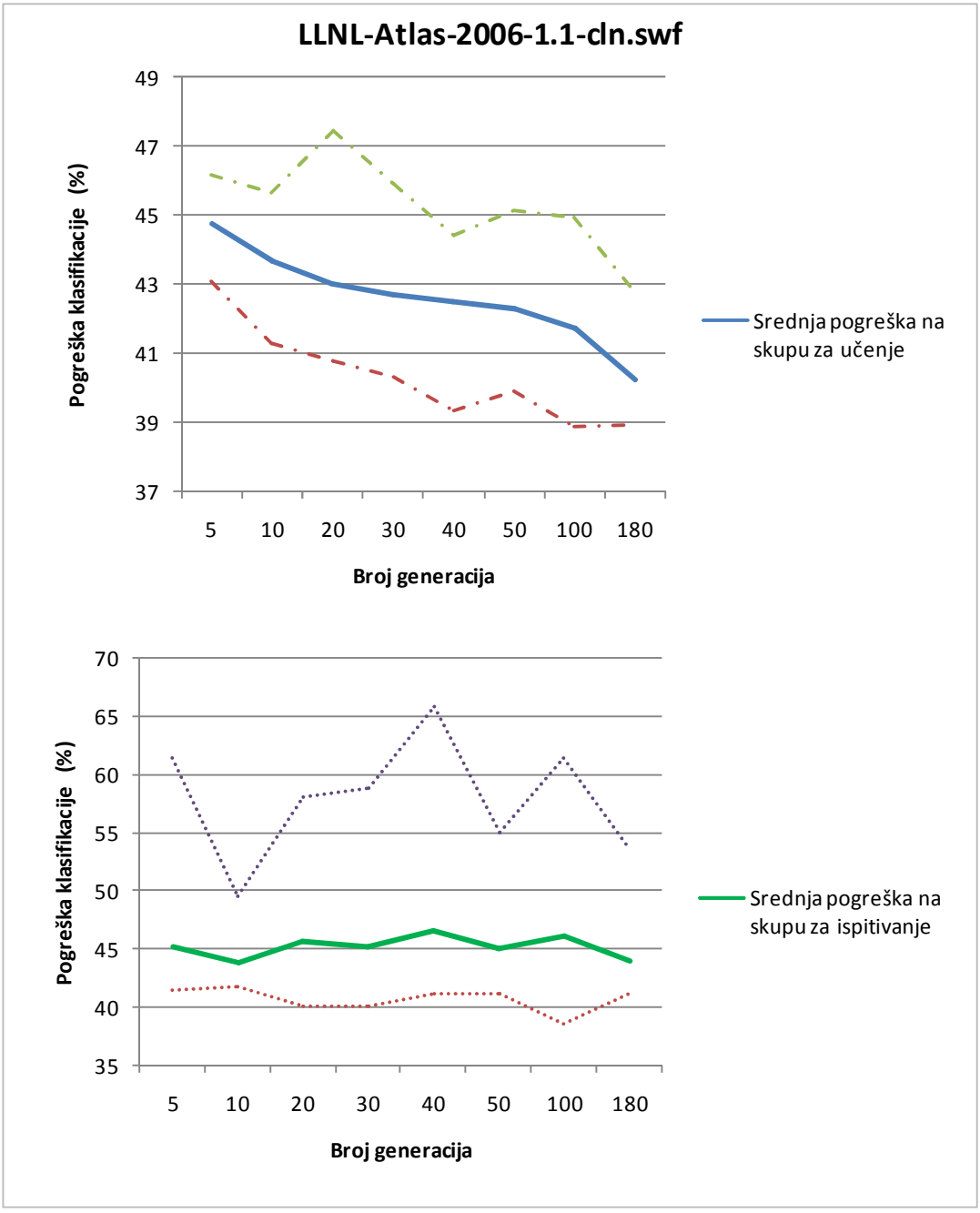
Slika 8.5 Graf kretanja pogreške klasifikacije (%) ovisno o broju generacija

Na grafu na slici 8.5 može se vidjeti kako se najmanja srednja pogreška klasifikacije na skupu za ispitivanje dobije za 50 generacija. Nakon toga ta pogreška počinje rasti i može se reći da model postaje prenaučen, jer se pogreška na skupu za učenje smanjuje, a ona na skupu za ispitivanje nastavlja rasti. Nakon stote generacije pogreška na skupu za učenje počinje rasti (što nije poželjno), a na skupu za ispitivanje počinje se smanjivati. Za daljnja mjerenja korištenjem ovog skupa podataka, kao optimalnu vrijednost, uzima se 50 generacija.

- *LLNL-Atlas-2006-1.1-cln.swf*

Tablica 8.5 Prikaz minimalne, srednje i maksimalne klasifikacijske pogreške za različit broj generacija

	pogreška na skupu za učenje (%)			pogreška na skupu za ispitivanje (%)		
broj generacija	srednja	minimalna	maksimalna	srednja	minimalna	maksimalna
5	44.7490	43.0771	46.1443	45.2404	41.4097	61.4172
10	43.6711	41.2765	45.6724	43.7914	41.6912	49.4557
20	43.0245	40.7798	47.4233	45.6828	40.0977	58.0357
30	42.7182	40.3328	45.9332	45.2109	40.0935	58.8386
40	42.4914	39.3518	44.4182	46.5327	41.2110	65.8665
50	42.2878	39.9230	45.1260	45.0609	41.0827	55.0226
100	41.7327	38.8551	44.9522	46.0769	38.5497	61.4130
180	40.2459	38.9172	42.7294	43.9464	41.0952	53.6526



Slika 8.6 Graf kretanja pogreške klasifikacije (%) ovisno o broju generacija

Na gornjem grafu na slici 8.6 za skup poslova grozda *LLNL-Atlas,* srednja pogreška na skupu za učenje kontinuirano pada, dok, kako se vidi iz donjeg grafa, pogreška na skupu za ispitivanje malo pada, malo raste. Najmanja vrijednost srednje pogreške na

skupu za ispitivanje postiže se već za 10 generacija, dok je najveća vrijednost za 40 generacija (oko 46.5 %).

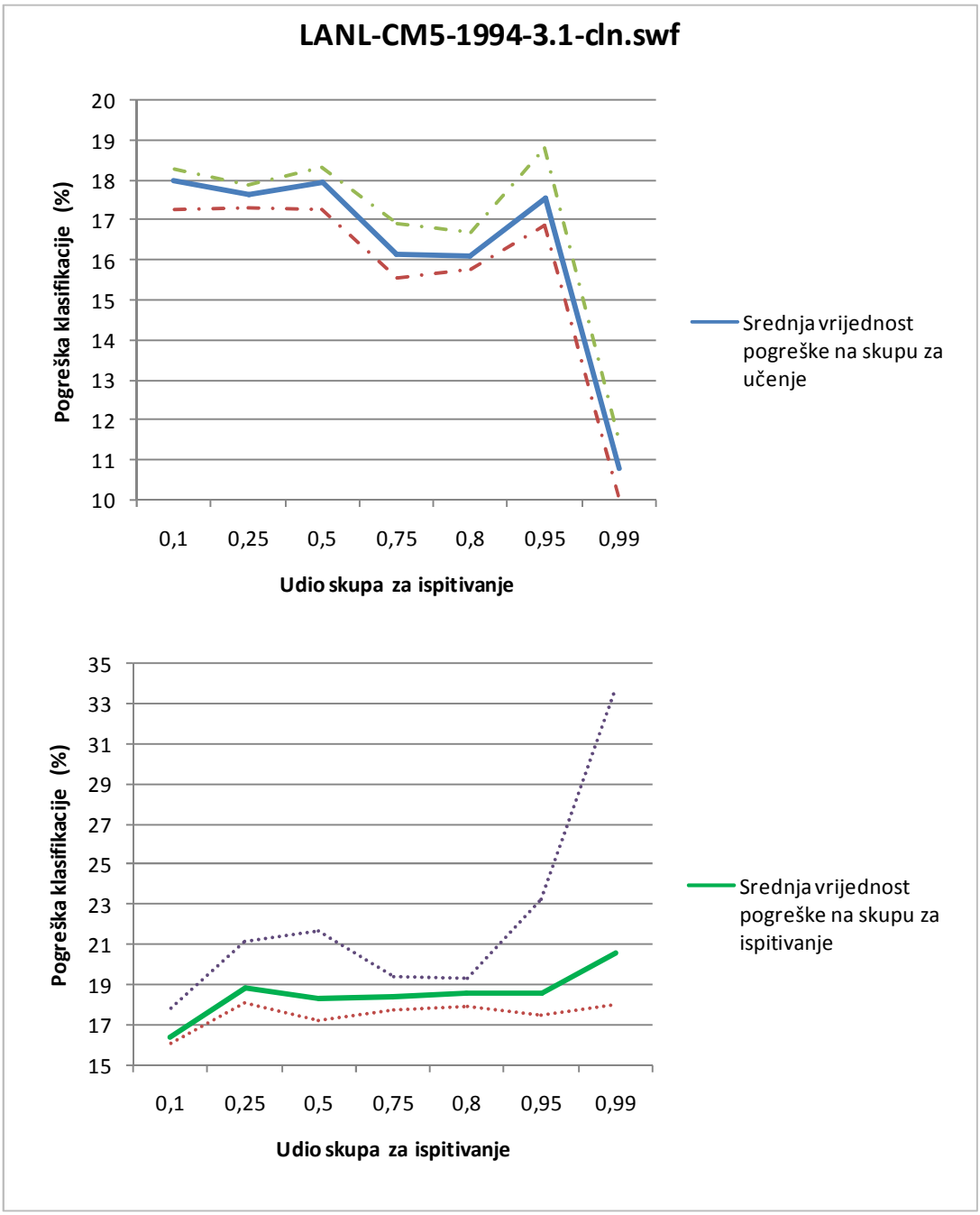
- Promjena veličine skupa za učenje (*Holdout* metoda)

Ovdje su mjerenja izvršena za različite vrijednosti parametra *Holdout* metode. Cilj je bio otkriti ovisnost uspješnosti klasifikacije o veličini skupa za učenje. U tablicama 8.6 i 8.7 naznačene su srednja, minimalna i maksimalna pogreška klasifikacije u odnosu na različite veličine parametra *Holdout* metode na oba skupa, i na skupu za učenje, i na skupu za ispitivanje.

- *LANL-CM5-1994-3.1-cln.swf*

Tablica 8.6 Prikaz minimalne, srednje i maksimalne klasifikacijske pogreške za različite veličine skupa za ispitivanje

	pogreška na skupu za učenje (%)			pogreška na skupu za ispitivanje (%)		
udio skupa za ispitivanje	srednja	minimalna	maksimalna	srednja	minimalna	maksimalna
0.1	17.9853	17.2702	18.2787	16.3813	16.0661	17.8181
0.25	17.6487	17.3045	17.8704	18.8016	18.0551	21.1293
0.5	17.9480	17.2636	18.3401	18.3301	17.1914	21.6743
0.75	16.1419	15.5697	16.9207	18.3932	17.7825	19.3864
0.8	16.1010	15.7545	16.6938	18.5375	17.9445	19.2868
0.95	17.5440	16.8714	18.7861	18.5600	17.5210	23.2798
0.99	10.7709	10.1174	11.5628	20.6159	18.0236	33.6344



Slika 8.7 Graf kretanja pogreške klasifikacije (%) ovisno o veličini skupa za ispitivanje

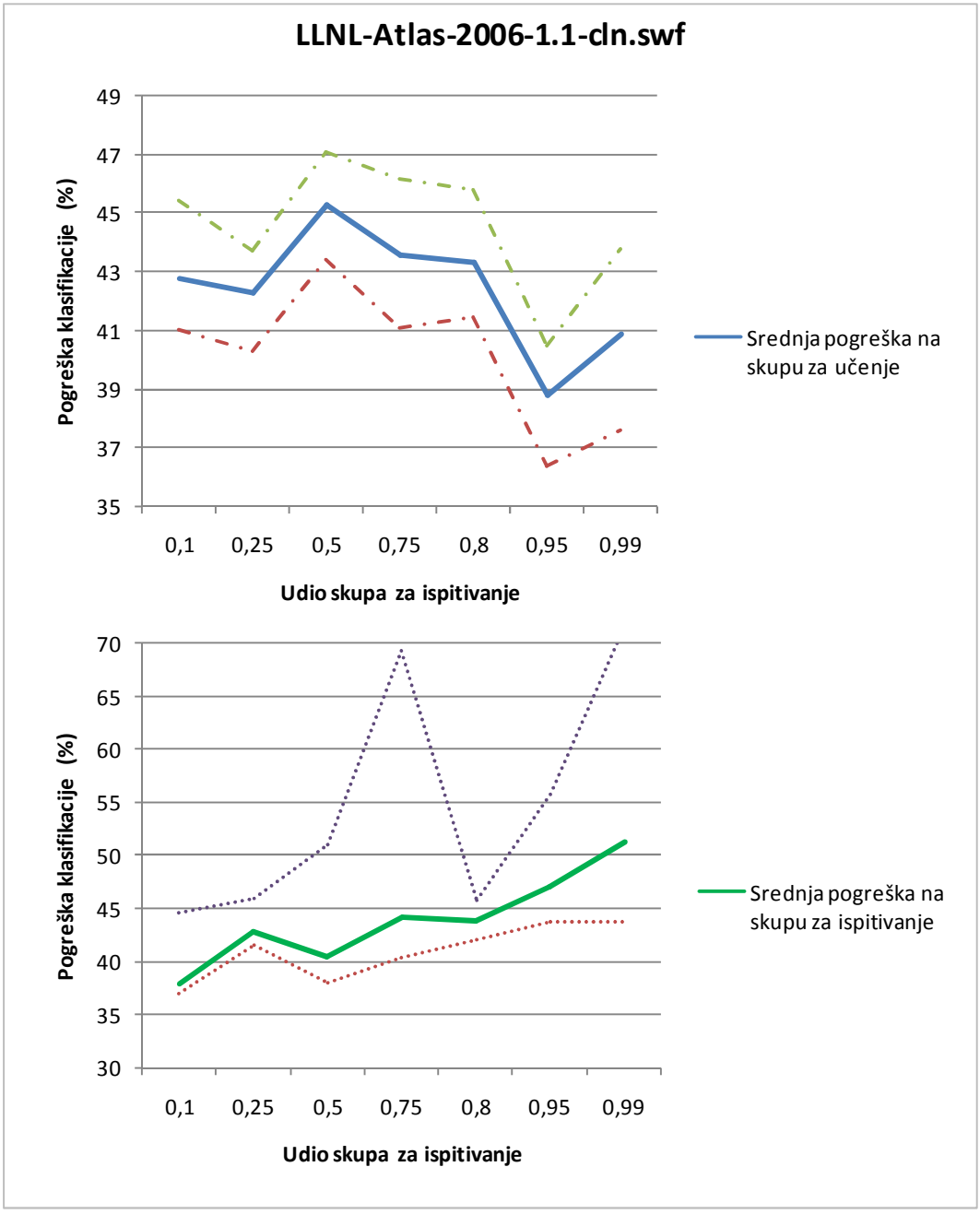
Na grafovima sa slike 8.7 vidi se kretanje najmanje, najveće i srednje pogreške klasifikacije. Kriterij zaustavljanja algoritma bio je 50 generacija i veličina populacije 500. Iz donjeg grafa se može zaključiti kako je pogreška na skupu za ispitivanje manja ako klasifikator za učenje ima na raspolaganju veći broj uzoraka. Ako je broj uzoraka u skupu za učenje jako mali, kao npr. kad je udio uzoraka u skupu za

ispitivanje modela jednako 95 ili 99%, tada je pogreška klasifikacije jako velika. Od ukupno 110 731 uzorka, 1% (oko 1000) uzoraka bi se koristilo za učenje, a to je premalo uzoraka da bi klasifikator mogao naučiti dobro generalizirati. Pogreška na skupu za učenje, kako se vidi na gornjem grafu slike 8.7, je u tom slučaju najmanja (krivulja naglo pada), ali zato je srednja pogreška na skupu za ispitivanje veća nego za druge udjele. Klasifikator će jako dobro klasificirati podatke iz kojih uči (pogreška oko 10%), ali zato neće moći uspješno klasificirati još neviđene podatke (one iz skupa za ispitivanje). Za manju pogrešku na skupu za ispitivanje (a to je cilj) bolje je klasifikatoru dati više uzoraka na kojima će moći učiti.

- *LLNL-Atlas-2006-1.1-cln.swf*

Tablica 8.7 Prikaz minimalne, srednje i maksimalne klasifikacijske pogreške za različite veličine skupa za ispitivanje

	pogreška na skupu za učenje (%)			pogreška na skupu za ispitivanje (%)		
udio skupa za ispitivanje	srednja	minimalna	maksimalna	srednja	minimalna	maksimalna
0.1	42.7781	41.0271	45.4661	37.8495	37.0071	44.5514
0.25	42.2854	40.2922	43.7275	42.7629	41.6367	46.0325
0.5	45.2894	43.4221	47.1037	40.3959	38.0704	51.0461
0.75	43.5415	41.0903	46.1443	44.2046	40.3543	69.3514
0.8	43.3354	41.4312	45.7932	43.8754	42.0379	45.7202
0.95	38.8178	36.3975	40.4969	47.1130	43.8472	55.7019
0.99	40.8592	37.5776	43.7888	51.2757	43.7382	71.5540



Slika 8.8 Graf kretanja pogreške klasifikacije (%) ovisno o veličini skupa za ispitivanje

Na gornjem grafu sa slike 8.8 također se vidi kako je klasifikator bolje klasificirao neviđene podatke ako mu je na raspolaganju bio veći broj uzoraka za učenje, u ovom primjeru, kad je udio uzoraka za ispitivanje između 10 i 75% – tad je pogreška još uvijek relativno mala, a onda počinje naglo rasti. S obzirom da se u ovom skupu

podataka za binarnu klasifikaciju može koristiti 32 215 uzoraka, 99% uzoraka odvojenih za ispitivanje značilo bi da ih se samo oko 300 koristi za učenje, što je premalo da bi klasifikator mogao dobro naučiti i dobro klasificirati još neviđene podatke.

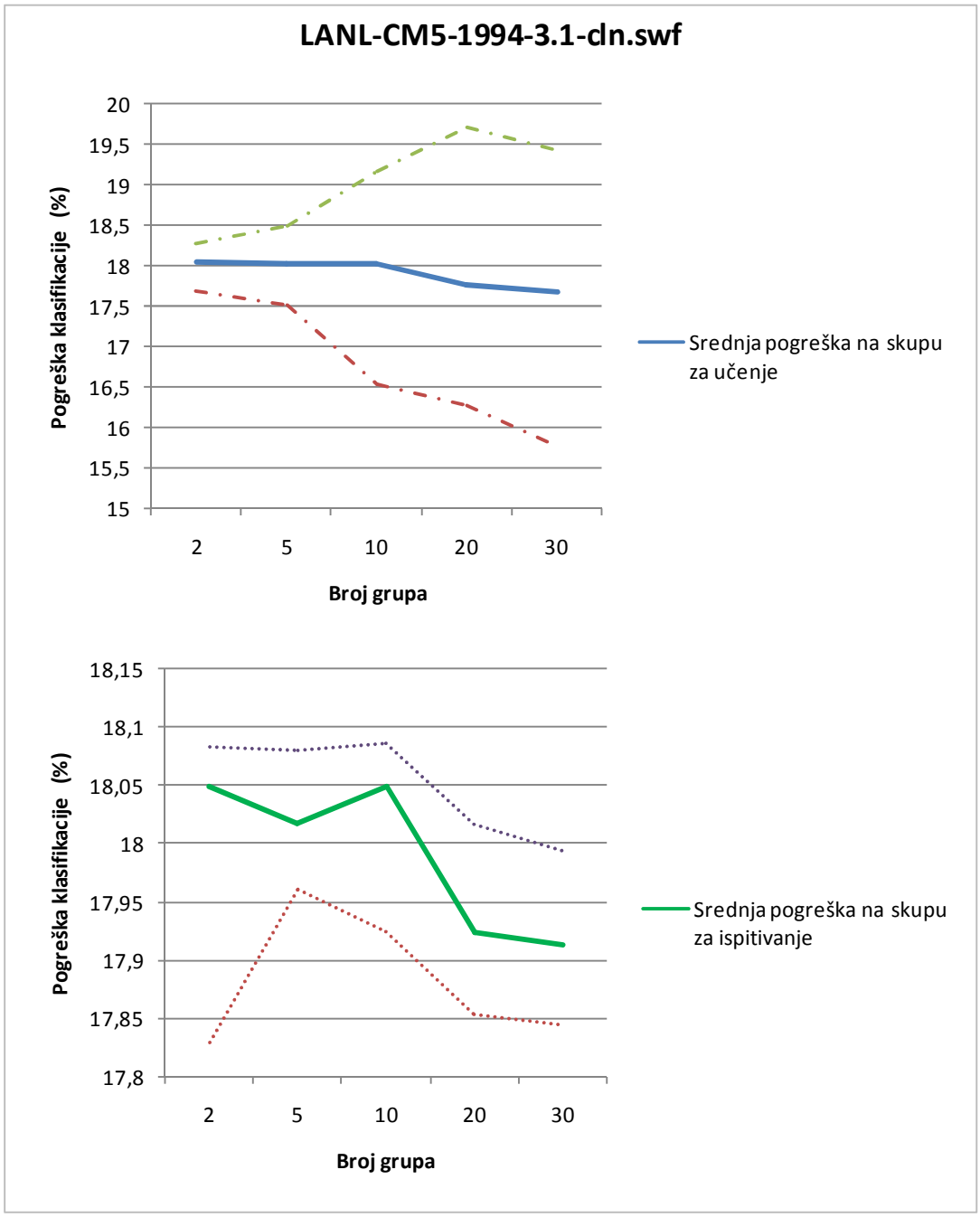
- Promjene po grupama kod k – struke unakrsne validacije

Mjerenja su izvršena za različite veličine parametra k kod k – struke unakrsne validacije. U tablicama 8.8 i 8.9 naznačene su srednja, minimalna i maksimalna pogreška klasifikacije u odnosu na različit broj grupa na oba skupa, i na skupu za učenje, i na skupu za ispitivanje.

- *LANL-CM5-1994-3.1-cln.swf*

Tablica 8.8 Prikaz minimalne, srednje i maksimalne klasifikacijske pogreške za različit parametar k kod k – struke unakrsne validacije

	pogreška na skupu za učenje (%)			pogreška na skupu za ispitivanje (%)		
broj grupa	srednja	minimalna	maksimalna	srednja	minimalna	maksimalna
2	18.0431	17.6953	18.2661	18.0490	17.8288	18.0826
5	18.0202	17.5193	18.4954	18.0177	17.9609	18.0799
10	18.0343	16.5267	19.1728	18.0486	17.9245	18.0868
20	17.7654	16.2753	19.7038	17.9235	17.8531	18.0170
30	17.6875	15.7681	19.4256	17.9134	17.8451	17.9944



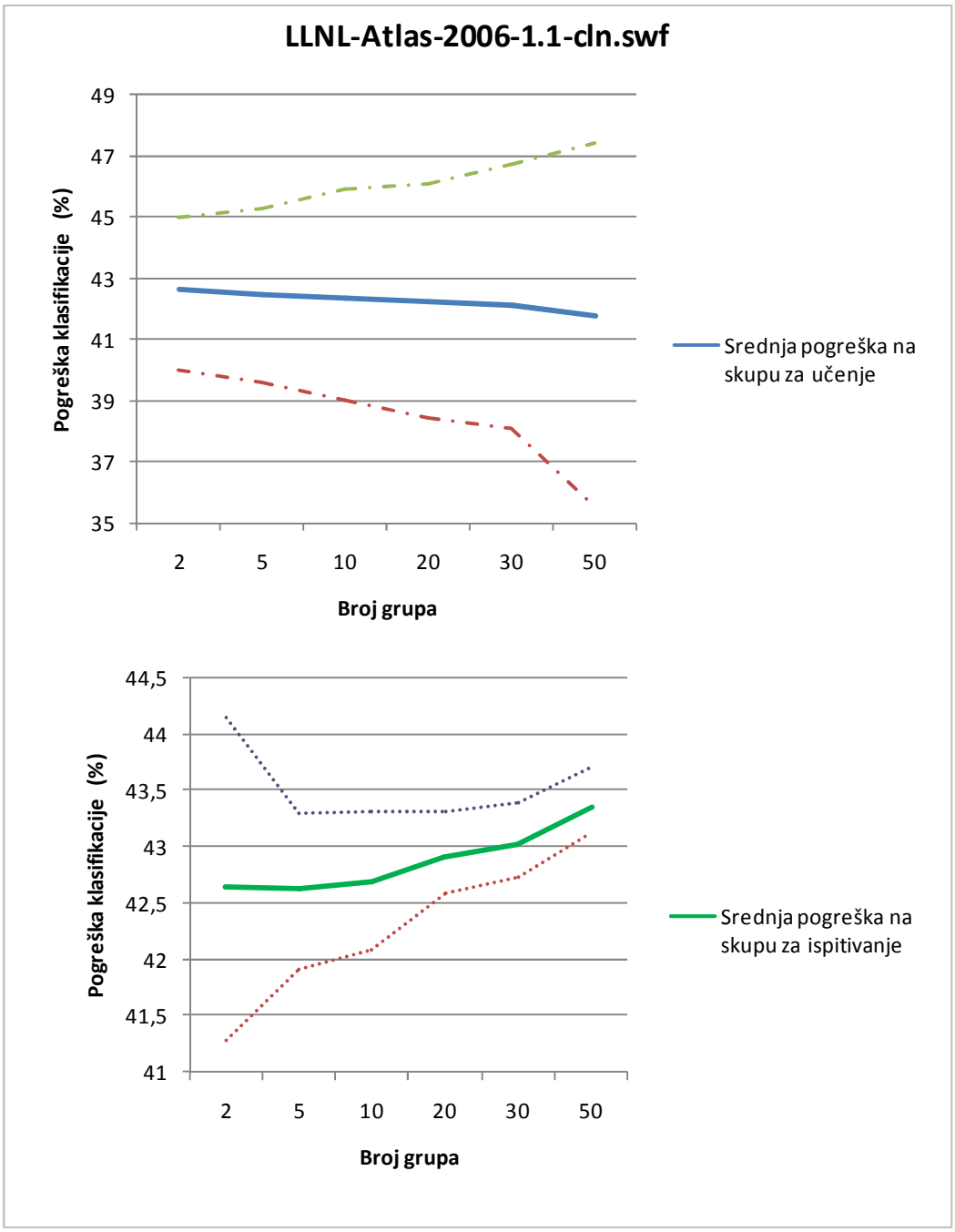
Slika 8.9 Graf kretanja pogreške klasifikacije (%) ovisno o broju grupa

Kako se vidi iz grafova na slici 8.9, pogreška od oko 18% dobije se za broj grupa između 2 i 10. Za broj grupa veći od 10 ova je metoda zahtjevna i dugotrajna pa se ne preporuča korištenje većeg broja grupa, a iz grafa proizlazi i da se za veći broj grupa ne dobivaju značajno bolja rješenja u odnosu na manji broj grupa.

- *LLNL-Atlas-2006-1.1-cln.swf*

Tablica 8.9 Prikaz minimalne, srednje i maksimalne klasifikacijske pogreške za različit parametar k kod k – struke unakrsne validacije

	pogreška na skupu za učenje (%)			pogreška na skupu za ispitivanje (%)		
broj grupa	srednja	minimalna	maksimalna	srednja	minimalna	maksimalna
2	42.6328	40.0298	44.9618	42.6491	41.2851	44.1440
5	42.4822	39.5934	45.2739	42.6330	41.9083	43.3005
10	42.3614	39.0441	45.9032	42.6956	42.0891	43.3090
20	42.2637	38.4233	46.1204	42.9048	42.5864	43.3049
30	42.1247	38.0819	46.7412	43.0238	42.7278	43.3913
50	41.8034	35.5590	47.4419	43.3544	43.1174	43.7006



Slika 8.10 Graf kretanja pogreške klasifikacije (%) ovisno o broju grupa

Na slici 8.10 nalaze se grafovi koji prikazuju kretanje pogreške na skupovima za učenje i ispitivanje u odnosu na broj grupa na koje se početni skup dijeli. Za veći broj grupa srednja pogreška na skupu za ispitivanje malo poraste, dok pogreška na skupu za učenje očekivano pada. I ovdje se bolji rezultati dobiju za manji broj (do 10) grupa.

8.1.2.2 GP temeljen na informacijskoj dobiti

U nastavku su prikazani rezultati eksperimenata u kojima je korišten GP temeljen na informacijskoj dobiti, taj je oblik GP detaljnije objašnjen u potpoglavlju 7.2.2.

Kod ovih mjerenja koriste se sljedeći GP parametri: veličina populacije 500, broj generacija 20, vjerojatnost križanja 0.9, vjerojatnost mutacije 0.01, najveća dopuštena veličina stabla 70.

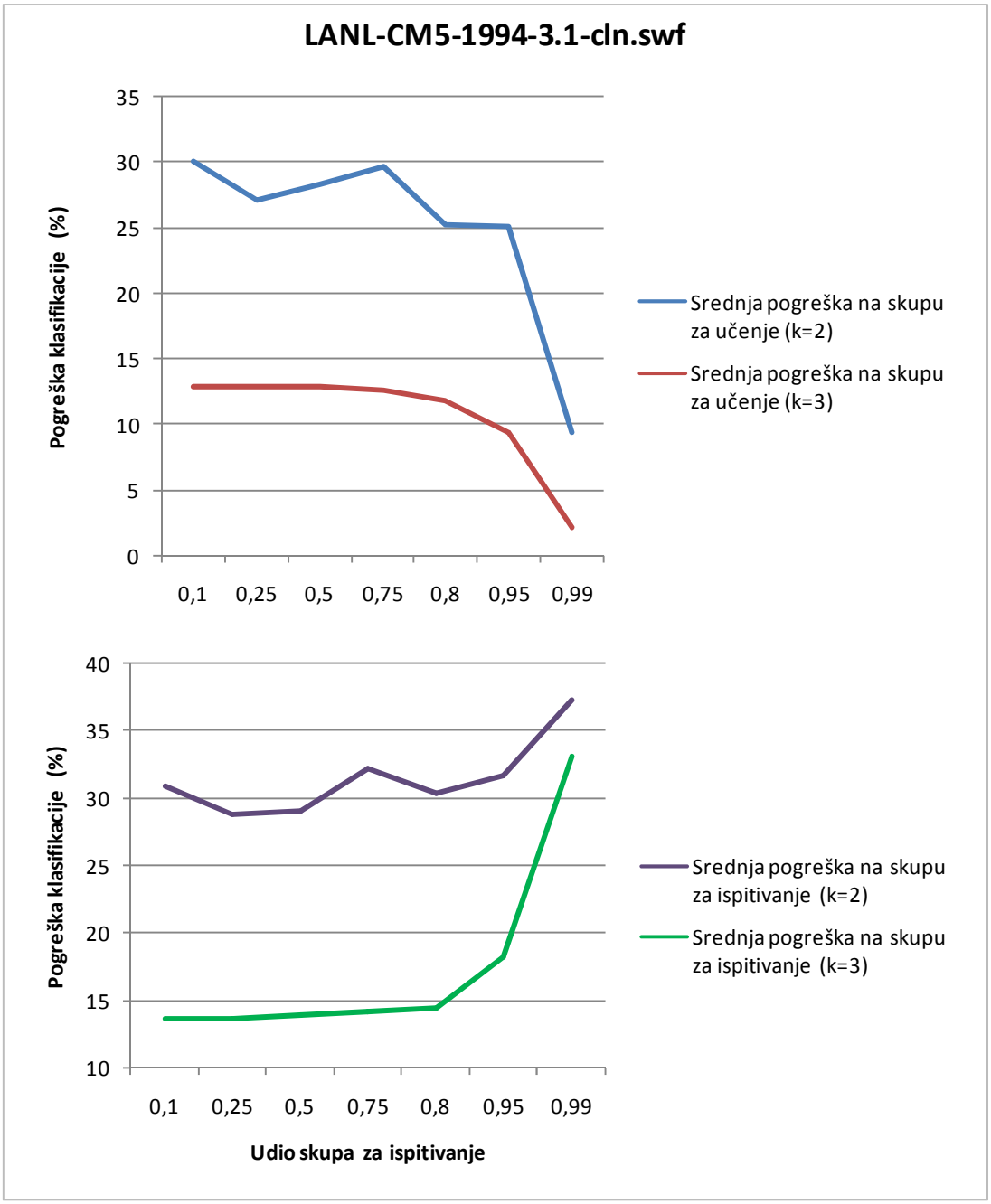
- Promjena parametra *Holdout* metode

Odabrano je ukupno 1200 uzoraka zbog računalne zahtjevnosti metode. Algoritam se pokreće 15 puta te se od tih 15 pokretanja računa srednja vrijednost pogreške.

- *LANL-CM5-1994-3.1-cln.swf*

Tablica 8.10 Prikaz srednje klasifikacijske pogreške za različite veličine skupa za ispitivanje za k = 2 i k = 3

	pogreška na skupu za učenje (%)		pogreška na skupu za ispitivanje (%)	
udio skupa za ispitivanje	srednja (k = 2)	srednja (k = 3)	srednja (k = 2)	srednja (k = 3)
0.1	30.1235	12.9012	30.8333	13.6111
0.25	27.2074	12.9556	28.7778	13.6000
0.5	28.3444	12.9222	29.0000	13.8889
0.75	29.7778	12.7333	32.1556	14.2222
0.8	25.2720	11.9386	30.3611	14.4722
0.95	25.1111	9.4444	31.6550	18.2749
0.99	9.4444	2.2222	37.3232	33.0640



Slika 8.11 Graf kretanja pogreške klasifikacije (%) ovisno o veličini skupa za ispitivanje

Iz vrijednosti iz tablice 8.10 dobiveni su grafovi na slici 8.11. Na grafovima se može vidjeti ponašanje pogreške klasifikacije na skupovima za učenje (gornji graf) i ispitivanje (donji graf) ovisno o tome je li prostor vrijednosti atributa podijeljen na dva ili tri dijela, tj. traži li se, za svaki atribut, samo jedna (k = 2, za podjelu prostora na

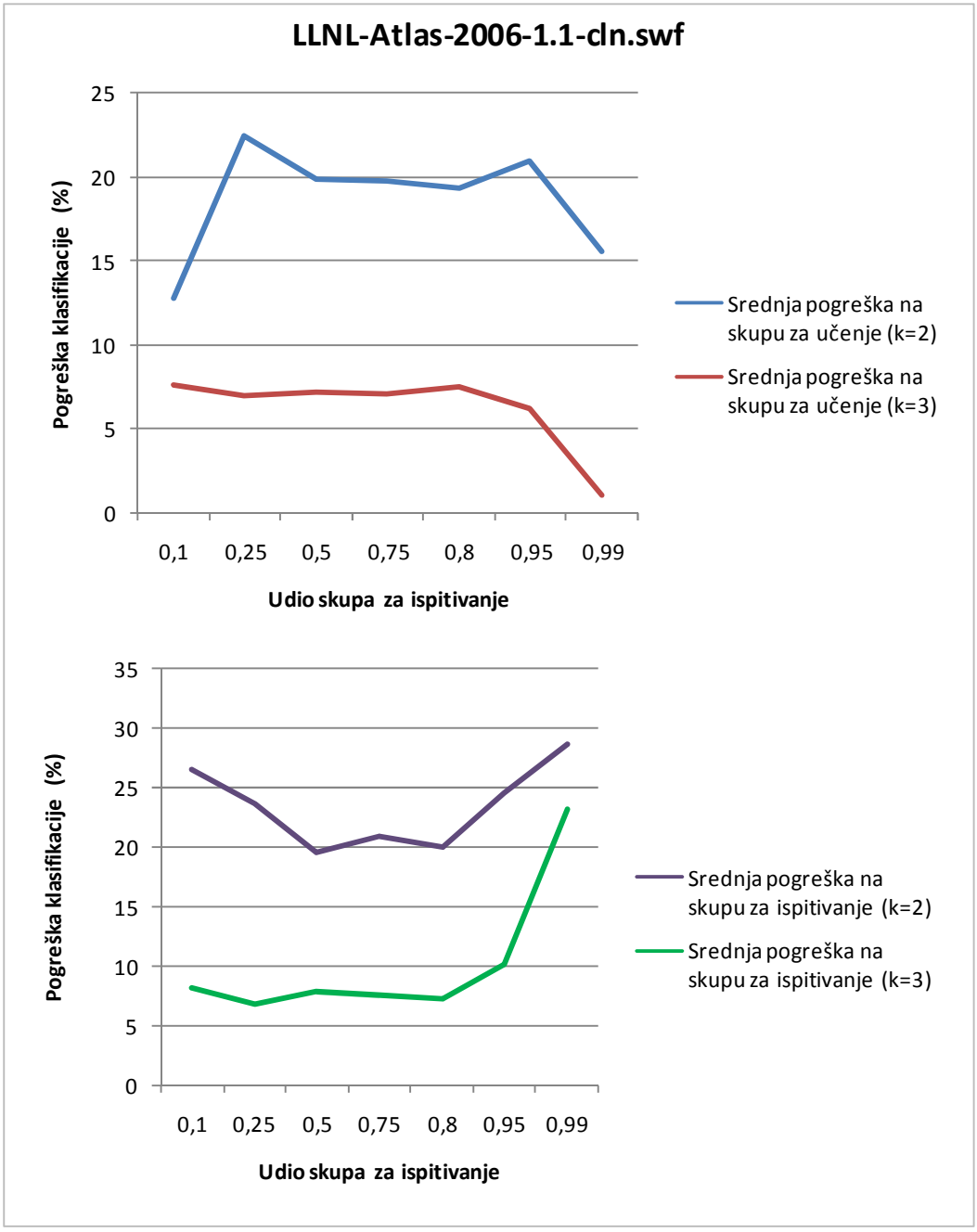
dva dijela) ili dvije (k = 3, za podjelu prostora na tri dijela) vrijednosti praga. Kad se radi o podjeli prostora na samo dva dijela (za svaki atribut posebno), tada se to odvija brže i jednostavnije, jer je puno manje kombinacija za provjeriti prilikom računanja informacijske dobiti, nego kad se radi o podjeli prostora na tri dijela. Iako je manje zahtjevna metoda (kad je k = 2), ona ipak daje znatno lošije rezultate, nego kad se domena atributa dijeli na tri dijela. Naravno, i ovdje se vidi velika razlika u vrijednosti srednje pogreške ako se u skupu za učenje nalazi samo manji broj uzoraka (na gornjem grafu vidljiv je veliki odmak obiju krivulja te se u tablici 8.10 može vidjeti razlika srednjih pogrešaka, npr. oko 9% za parametar 0.99 ili čak 30% za parametar 0.1).

Za k = 3, najbolji rezultat dobije se za kada je u skupu za učenje od 90 do 20% uzoraka. Ako je u tom skupu manje od 20% uzoraka, rezultati klasifikacije nisu dobri, jer pogreška na skupu za ispitivanje počinje rasti, dok se srednja pogreška na skupu za učenje istovremeno smanjuje. Razlika između pogrešaka za k = 2 i k = 3 dosta je velika – oko 15%.

- *LLNL-Atlas-2006-1.1-cln.swf*

Tablica 8.11 Prikaz srednje klasifikacijske pogreške za različite veličine skupa za ispitivanje za k = 2 i k = 3

	pogreška na skupu za učenje (%)		pogreška na skupu za ispitivanje (%)	
udio skupa za ispitivanje	srednja (k = 2)	srednja (k = 3)	srednja (k = 2)	srednja (k = 3)
0.1	12.7778	7.5741	26.5937	8.1667
0.25	22.4741	6.9852	23.6889	6.7778
0.5	19.8667	7.1667	19.6111	7.8222
0.75	19.7556	7.0444	20.9333	7.6000
0.8	19.3584	7.4756	20.0000	7.2083
0.95	21.0000	6.2222	24.5906	10.1404
0.99	15.5556	1.1111	28.7149	23.1762



Slika 8.12 Graf kretanja pogreške klasifikacije (%) ovisno o veličini skupa za ispitivanje

Kao i iz prethodne slike, i na ovoj (slika 8.12) se može vidjeti kako se lošiji rezultati dobiju za k = 2, puno bolji za k = 3. Za k = 2 se na početku dobije veća razlika srednjih pogrešaka na skupu za učenje (gornji graf) i ispitivanje (donji graf). Zatim je

ta pogreška slična za udio uzoraka u skupu za ispitivanje od 25 do 80%, a poslije se pogreška na skupu za ispitivanje opet naglo povećava (jer klasifikator uči na malom broju uzoraka).

8.1.2.3 GP temeljen na omjeru dobitka

U nastavku su prikazani rezultati eksperimenata u kojima se koristi GP temeljen na omjeru dobitka, taj je oblik GP detaljnije objašnjen u potpoglavlju 7.2.3.

Kod ovih mjerenja, kao i kod prethodnih gdje se koristi GP temeljen na informacijskoj dobiti, koriste se GP parametri: veličina populacije 500, broj generacija 20, vjerojatnost križanja 0.9, vjerojatnost mutacije 0.01, najveća dopuštena veličina stabla 70.

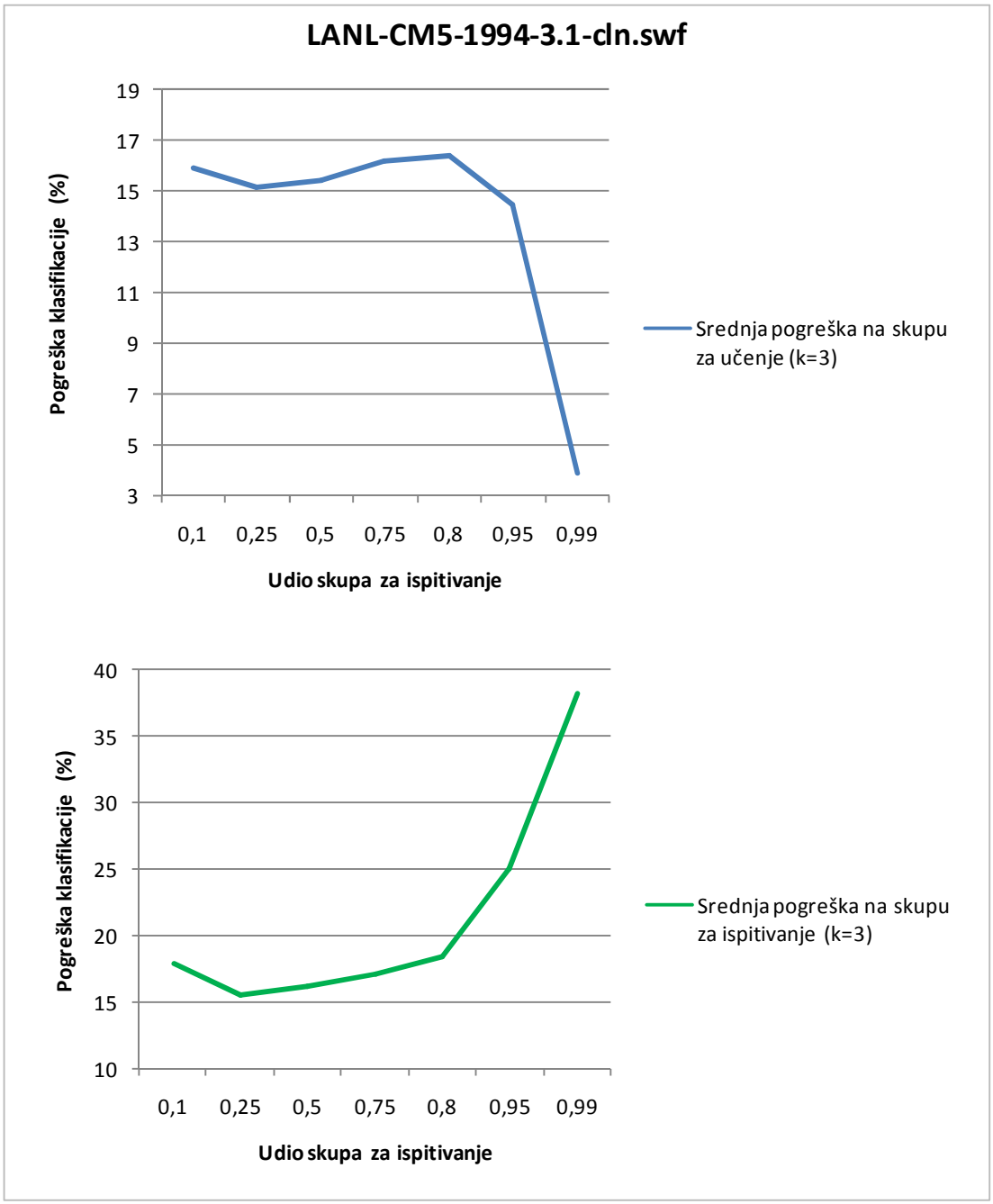
- Promjena parametra *Holdout* metode

Odabrano je ukupno 1200 uzoraka zbog velike računalne zahtjevnosti metode. Algoritam se pokreće 15 puta te se od tih 15 pokretanja računa srednja vrijednost pogreške. Ovdje se domena vrijednosti za svaki atribut dijeli na tri dijela (k = 3). Rezultati su prikazani u tablicama 8.12 i 8.13.

- *LANL-CM5-1994-3.1-cln.swf*

Tablica 8.12 Prikaz srednje klasifikacijske pogreške za različite veličine skupa za ispitivanje za k = 3

udio skupa za ispitivanje	srednja pogreška na skupu za učenje (%)	srednja pogreška na skupu za ispitivanje (%)
0.1	15.9167	17.9167
0.25	15.1481	15.5333
0.5	15.4333	16.2222
0.75	16.2222	17.1407
0.8	16.4296	18.4236
0.95	14.4444	25.0468
0.99	3.8889	38.1762



Slika 8.13 Graf kretanja srednje pogreške klasifikacije (%) ovisno o udjelu uzoraka u skupu za ispitivanje za k = 3

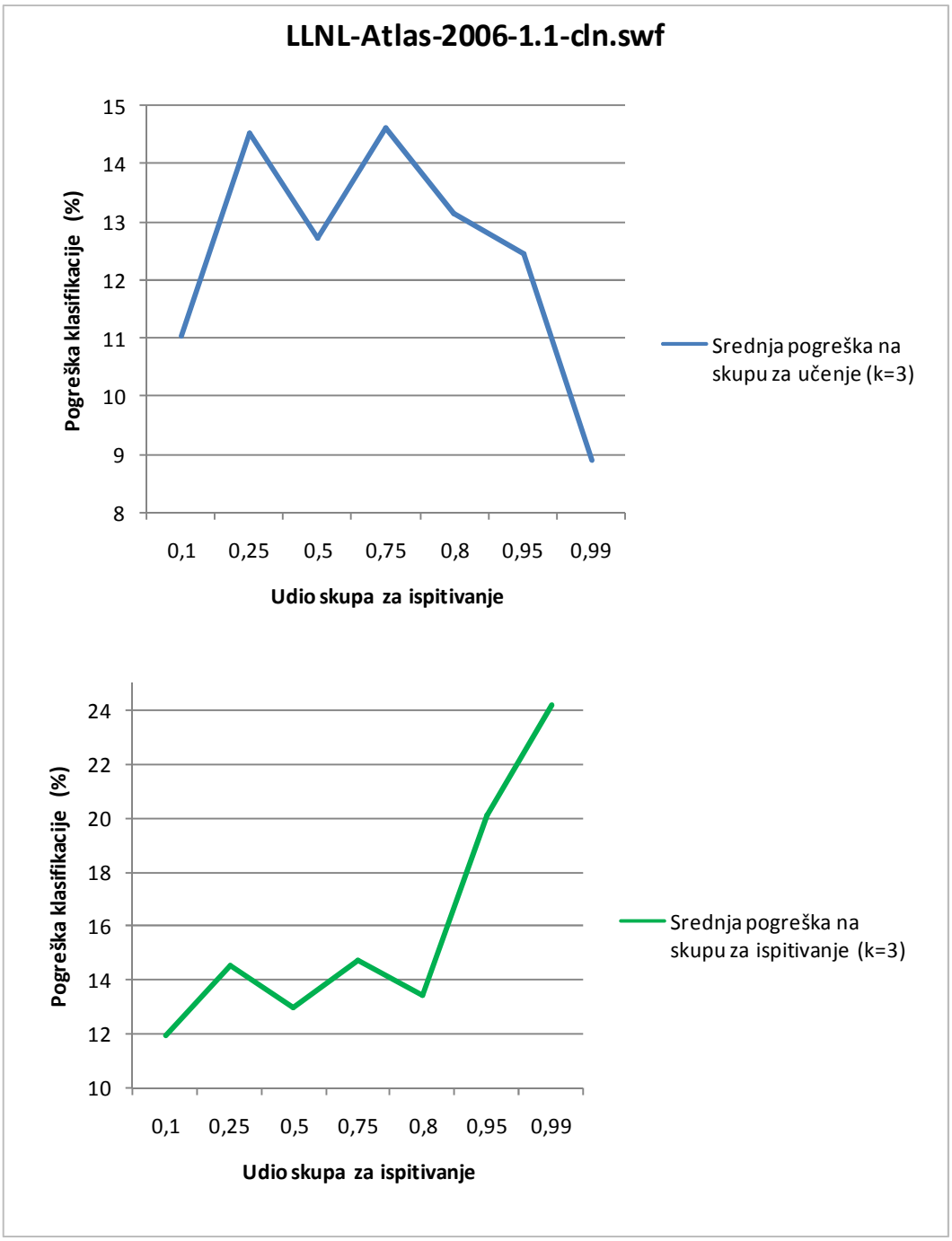
Iz donjeg grafa na slici 8.13 vidi se kako za udio uzoraka u skupu za ispitivanje od 25% srednja vrijednost za oba skupa lagano pada, zatim počinje rasti, dok, za 99%, dostiže jako veliku na skupu za ispitivanje (ili malu vrijednost na skupu za učenje) što je posljedica prenaučenosti. Klasifikator ovdje uči nad određenim uzorcima i većinu

njih naučio je ispravno klasificirati, što se vidi na gornjem grafu (pogreška pada prema 3%), ali puno lošije (s pogreškom više od 10 puta većom) klasificira podatke koje nije vidio (pogreška na skupu za ispitivanje oko 38%).

- *LLNL-Atlas-2006-1.1-cln.swf*

Tablica 8.13 Prikaz srednje klasifikacijske pogreške za različite veličine skupa za ispitivanje za k = 3

udio skupa za ispitivanje	srednja pogreška na skupu za učenje (%)	srednja pogreška na skupu za ispitivanje (%)
0.1	11.0370	11.9167
0.25	14.5259	14.5111
0.5	12.7111	12.9667
0.75	14.6222	14.7407
0.8	13.1381	13.4375
0.95	12.4444	20.1287
0.99	8.8889	24.1863



Slika 8.14 Graf kretanja srednje pogreške klasifikacije (%) ovisno o udjelu uzoraka u skupu za ispitivanje za k = 3

Na slici 8.14 još se ljepše vidi velika razlika u srednjim pogreškama kod oba skupa kada je postotak uzoraka za ispitivanje jako velik (iznad 80%). Najmanja srednja

pogreška na skupu za ispitivanje (donji graf) postiže se za udio uzoraka od 10, 50 i 80%.

U tablici 8.14 nalazi se usporedba rezultata za sve tri vrste GP – a za skup podataka *LANL-CM5-1994-3.1-cln.swf*, a u tablici 8.15 usporedba za skup podataka *LLNL-Atlas-2006-1.1-cln.swf*. Za usporedbu su uzeti rezultati eksperimenata gdje se koristila *Holdout* metoda te sljedeći GP parametri: veličina populacije 500, broj generacija 20, vjerojatnost križanja 0.9, vjerojatnost mutacije 0.01, najveća dopuštena veličina stabla 70. Kod podjele prostora (za vrste GP – a, osim jednostavnog) uzimao se parametar k = 3. Prikazane su vrijednosti pogreške klasifikacije (%) samo na skupu za ispitivanje za tri vrste GP – a za pojedine vrijednosti udjela uzoraka u skupu za ispitivanje. Za stvaranje stabala uzimalo se ukupno 1200 uzoraka (za sve tri vrste GP – a), uzimala se srednja vrijednost pogreške u 30 mjerenja.

- *LANL-CM5-1994-3.1-cln.swf*

Tablica 8.14 Usporedba rezultata za tri vrste GP - a

	srednja pogreška na skupu za ispitivanje (%)		
udio skupa za ispitivanje	jednostavni GP	GP temeljen na informacijskoj dobiti	GP temeljen na omjeru dobitka
0.1	14.2222	13.6111	17.9167
0.25	14.3767	13.6000	15.5333
0.5	14.7222	13.8889	16.2222
0.75	15.1407	14.2222	17.1407
0.8	15.3924	14.4722	18.4236
0.95	17.9942	18.2749	25.0468
0.99	29.7531	33.0640	38.1762

Iz gornje tablice slijedi kako je za skup podataka *LANL-CM5-1994-3.1-cln.swf* pogreška na skupu za ispitivanje korištenjem jednostavnog GP –a veća od pogreške kad se koristio GP temeljen na informacijskoj dobiti, ali je manja od srednje pogreške na skupu za ispitivanje korištenjem GP – a temeljenog na omjeru dobitka.

- *LLNL-Atlas-2006-1.1-cln.swf*

Tablica 8.15 Usporedba rezultata za tri vrste GP - a

	srednja pogreška na skupu za ispitivanje (%)		
udio skupa za ispitivanje	jednostavni GP	GP temeljen na informacijskoj dobiti	GP temeljen na omjeru dobitka
0,1	7,6944	13,6111	17,9167
0,25	7,5222	13,6000	15,5333
0,5	7,4722	13,8889	16,2222
0,75	7,6333	14,2222	17,1407
0,8	8,0764	14,4722	18,4236
0,95	10,2690	18,2749	25,0468
0,99	17,8423	33,0640	38,1762

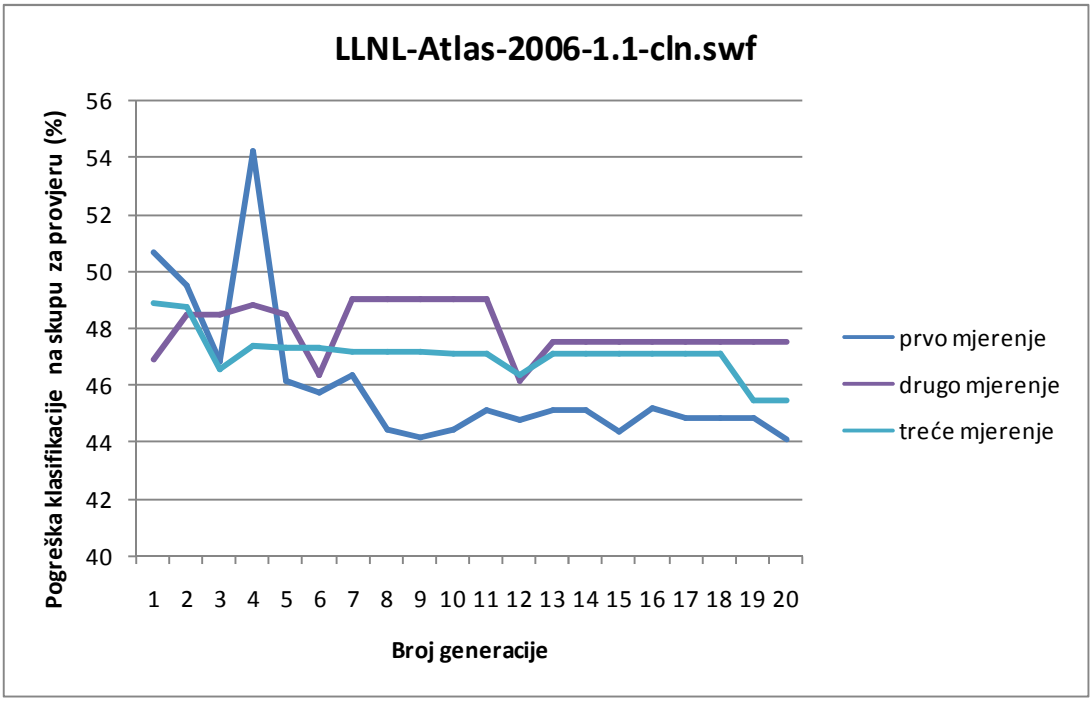
Iz gornje tablice proizlazi kako je pogreška klasifikacije na skupu za ispitivanje za skup podataka *LLNL-Atlas-2006-1.1-cln.swf* korištenjem jednostavnog GP – a puno manja nego kad se koriste ostale dvije vrste GP – a. Jednostavni GP se na ovom skupu podataka pokazao boljim.

8.2 Odabir najboljih rješenja uz pomoć skupa za provjeru

Najbolja rješenja mogu se odabrati i uz pomoć skupa za provjeru (validaciju). Valjani uzorci učitanog skupa podataka dijele se na skup za učenje i skup za testiranje, a dodatno se od odabranog skupa za učenje 20% uzoraka odvaja za provjeru. Uz pomoć uzoraka za provjeru odabire se najbolje rješenje. Računa se pogreška nekog rješenja nad tim skupom - najbolje rješenje će biti ono koje će imati najmanju validacijsku pogrešku. Nakon toga se određuje njegova pogreška i na skupu zaispitivanje. Stablo koje je odabrano na ovaj način dobro će generalizirati, tj. klasficirati još neviđene poslove, a on se koristi i kako bi se izbjegla prenaučenost. U

prethodnim mjerenjima koristla su se samo dva skupa, jedan za učenje i nakon toga se za najbolje stablo ispitivala pogreška na skupu za ispitivanje. Ovdje se za jednostavni GP izvršava učenje nad određenim brojem uzoraka te se odabire desetak najboljih stabala (za svaku generaciju) nad skupom za učenje (pretpostavka je da se oko 50% uzoraka koristi za učenje i 50% uzoraka za ispitivanje). Od 50% uzoraka koji su se našli u skupu za učenje, 20% ih se odabire za provjeru, a na ostalih 30% vrši se učenje. Od deset odabranih najboljih stabala u svakoj generaciji odabire se kao najbolje ono koje ima najmanju validacijsku pogrešku. U nastavku su prikazani rezultati za podatke samo jednog grozda te je dana usporedba vrijednosti pogreške na skupu za ispitivanje s prethodnim mjerenjima kad se za odabir najboljeg stabla nije koristio skup za provjeru, već samo skup za ispitivanje.

U nastavku, na slici 8.15 prikazano je kako se kreće pogreška na skupu za provjeru za skup podataka *LLNL-Atlas-2006-1.1-cln.swf* unutar 20 generacija za prva tri mjerenja korištenjem jednostavnog GP - a. Dogovor je da se učenje prekine kada pogreška na skupu za provjeru počne rasti, na slici se može vidjeti da se to za prva tri mjerenja događa već nakon dvije ili četiri generacije.



Slika 8.15 Kretanje pogreške klasifikacije na skupu za provjeru za prva tri mjerenja kroz generacije

Na gornjoj se slici vidi kako kretanje pogreške na skupu za provjeru nije monotono. Nakon nekoliko generacija i nakon prvog porasta vrijednosti pogreške dogodi se,

kako je vidljivo iz grafa na slici 8.15, i smanjenje pogreške (time i bolja rješenja) i bilo bi korisno nakon prvog povećanja ne prekinuti učenje nego detaljnije proučavati daljnje ponašanje, no to izlazi iz okvira ovog rada.

- *LLNL-Atlas-2006-1.1-cln.swf*

Tablica 8.16 Pogreška klasifikacije na skupovima za učenje i ispitivanje te pogreška na skupu za validaciju za 20 pokretanja

broj	učenje	validacija	ispitivanje	ispitivanje bez provjere
1.	0.432119205	0.514126048	0.449804433	0.45084851
2.	0.4375	0.522198075	0.433848637	0.44857202
3.	0.423427152	0.451723067	0.431675669	0.438017384
4.	0.4375	0.522198075	0.433848637	0.44857202
5.	0.456539735	0.466935734	0.431551499	0.437448262
6.	0.4375	0.474076374	0.478984293	0.44857202
7.	0.433360927	0.447066129	0.408518036	0.414942053
8.	0.447847682	0.488668115	0.497858074	0.496326573
9.	0.419701987	0.499534306	0.432234432	0.443449917
10.	0.43294702	0.497981993	0.434034892	0.444691639
11.	0.4375	0.466314809	0.431489415	0.44857202
12.	0.436672185	0.514126048	0.433848637	0.447226821
13.	0.436465232	0.522198075	0.433848637	0.44857202
14.	0.432326159	0.523129463	0.433848637	0.448727235
15.	0.425289735	0.465693884	0.423107965	0.430204884
16.	0.432740066	0.522198075	0.433848637	0.448727235
17.	0.457574503	0.522508538	0.436890793	0.45115894
18.	0.427152318	0.456380006	0.44744521	0.450175911
19.	0.432740066	0.471592673	0.428695598	0.449244619
20.	0.437086093	0.424712822	0.432917365	0.475475993

Iz priloženog, prema tablici 8.16, vidi se kako je pogreška koja se računa samo na skupu za ispitivanje u većini slučajeva (što je dokazano statističkim testom) veća od one koja je dobivena ako se za odabir rješenja koristi validacijski skup. Pretpostavlja se kako će stablo koje je imalo najmanju pogrešku na skupu za validaciju (pri čemu to stablo ne mora biti najbolje na skupu za učenje) dosta dobro generalizirati, tj. imati malu pogrešku na skupu za ispitivanje. U slučaju kad se za najbolje stablo uzima ono koje je imalo najmanju pogrešku na skupu za učenje, mora se imati u vidu kako tada pogreška izračunata na skupu još neviđenih uzoraka najčešće nije jednaka stvarnoj pogrešci klasifikatora, dok je pogreška stabla odabranog prema pogrešci na validacijskom skupu bliža stvarnoj pogrešci.

U nastavku je priložen statistički test za usporedbu vrijednosti pogreške s i bez korištenja skupa za provjeru (validaciju). Kako vrijednosti pogreške klasifikacije ne slijede normalnu (Gaussovu) razdiobu, za njihovu usporedbu koristit će se neparametarski statistički test koji se obično koristi u takvim slučajevima za usporedbu dviju grupa, a to je Wilcoxonov parni test [25].

Tablica 8.17 Rezultati Wilcoxonovog parnog testa

		N	srednja razina	zbroj svih razina
X - Y	pozitivna razina	2	10,5	21
	negativna razina	18	10,5	189

Kod ovog se testa gleda zbroj pozitivnih i negativnih razina i uzima se manji. Moduli razlike vrijednosti dviju grupa koje se uspoređuju (izuzimaju se vrijednosti kojima je razlika nula) se sortiraju i njima se redom pridjeljuju prirodni brojevi koji predstavljaju razine. Ako postoje apsolutne vrijednosti koje su jednake, tada se izračuna aritmetička sredina njihovih razina te im se pridijeli nova razina – dobivena aritmetička sredina. Nakon toga se posebno zbrajaju razine negativnih i pozitivnih razlika. U ovom slučaju su to 21 i 189 te se odabire manja od tih dviju vrijednosti. Ovdje je $\min(W_+, W_-) = \min(21, 189) = 21$, kao što se vidi iz tablice 8.17. Nakon toga se dobivena vrijednost 21 uspoređuje s kritičnom vrijednošću očitanom iz tablice za Wilcoxonov test (ovisno o ukupnom broju uzoraka, ovdje ih je 20). Za razinu p = 0.05 iz tablice [50] se očitava vrijednost 52. S obzirom da je dobivena vrijednost 21 manja od one očitane iz tablice, iz toga proizlazi kako se nulta hipoteza mora odbaciti u korist alternativne hipoteze. Može se zaključiti kako se dvije grupe vrijednosti

pogrešaka klasifikacije dobivenih ispitivanjem sa i bez skupa za provjeru (validaciju) razlikuju. Pogreška dobivena ispitivanjem bez provjere je u većini slučajeva veća.

8.3 Usporedba s rezultatima drugih klasifikacijskih algoritama

Iz literature [11], [1] i [10] mogu se dobiti vrijednosti klasifikacijske pogreške dobivene različitim klasifikacijskim algoritmima. Usporedbom rezultata dolazi se do zaključka da su vrijednosti pogreške dobivene jednostavnom GP – om slične onima drugih klasifikacijskih algoritama iz literature. U nastavku su za svaki skup podataka prikazane usporedbe eksperimentima u ovom radu dobivenih vrijednosti klasifikacijskih pogrešaka i pogrešaka navedenih u literaturi.

8.3.1 German credit data

Npr., pogreška na skupu za ispitivanje dobivena korištenjem jednostavnog GP kreće se između 30 i 40%, dok je na skupu za učenje između 20 i 30%.

Pogreška klasifikacije dobivena neuronskim mrežama jednaka je 28.3%, pogreška dobivena algoritmom k najbližih susjeda 26.5% [1]. Pogreška dobivena algoritmom C5.0 jednaka je 27.5%. Za algoritam *Ltree srednja* pogreška (korištenjem 10 – struke unakrsne validacije) jednaka je 26.4%, za *OC1* 25.7, a za C4.5 29.1% [11]. Kada su se neuronske mreže koristile zajedno s *Ada Boosting* metodom, pogreška klasifikacije iznosi 25.3%, a korištenjem algoritma C4.5 i *Ada Boosting* metode 26.7% [10]. Rezultati su malo bolji od onih dobivenim izvođenjem jednostavnog GP. Npr., za 10 – struku unakrsnu validaciju klasifikacijska pogreška kretala se od 30 do 40%.

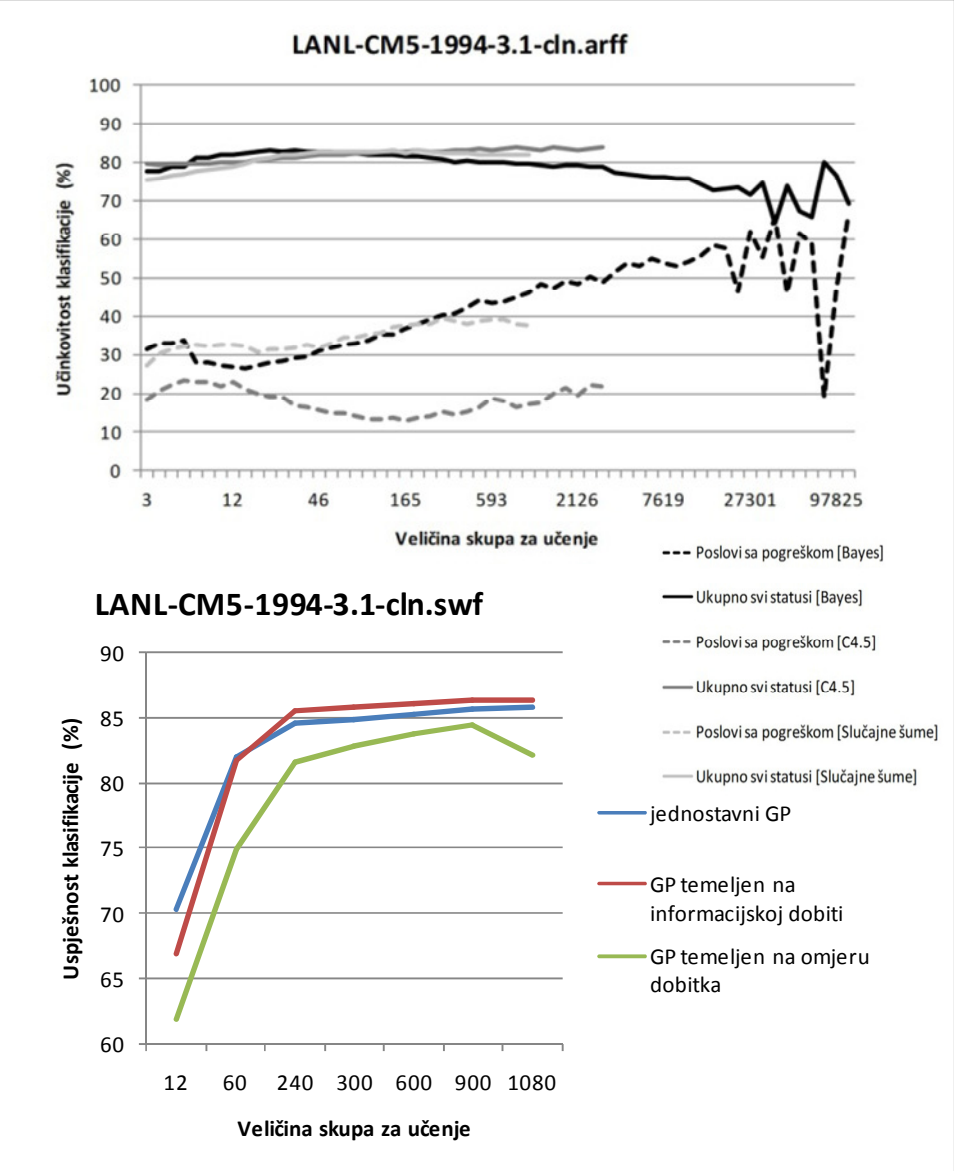
8.3.2 Skupovi podataka iz arhive paralelnog opterećenja grozdova

Ovdje se mogu usporediti rezultati iz [21] s klasifikatorima: Bayesove mreže, stabla odluke i slučajne šume. Kako je u [21] za svaki algoritam navedena uspješnost klasifikacije, a u ovom radu se ispitivala pogreška klasifikacije, jedna vrijednost je vrlo jednostavno pretvorena u drugu. Zbroj točnosti i pogreške klasifikacije jednak je jedan [52].

- *LANL-CM5-1994-3.1-cln.swf*

Uspješnost klasifikacije za ovaj skup podataka bila je velika kad su se koristile Bayesove mreže, što se može vidjeti na gornjem grafu slike 8.16. Za veliki broj uzoraka u skupu za učenje uspješnost klasifikacije kreće se od najmanje 20 do najviše oko 65%. Za manji broj uzoraka u skupu za učenje, uspješnost raste od oko 30 do 55%. Stabla odluke su slabiji klasifikatori. Najveća postignuta uspješnost je

20%, a rezultati slučajnih šuma slični su rezultatima koje su dobile Bayesove mreže za manji broj uzoraka. Najmanja srednja pogreška klasifikacije dobivena jednostavnim GP – om jednaka je otprilike 16%, dok se najmanje srednje pogreške dobivene drugim dvjema vrstama GP – a kreću između 15 i 30%.



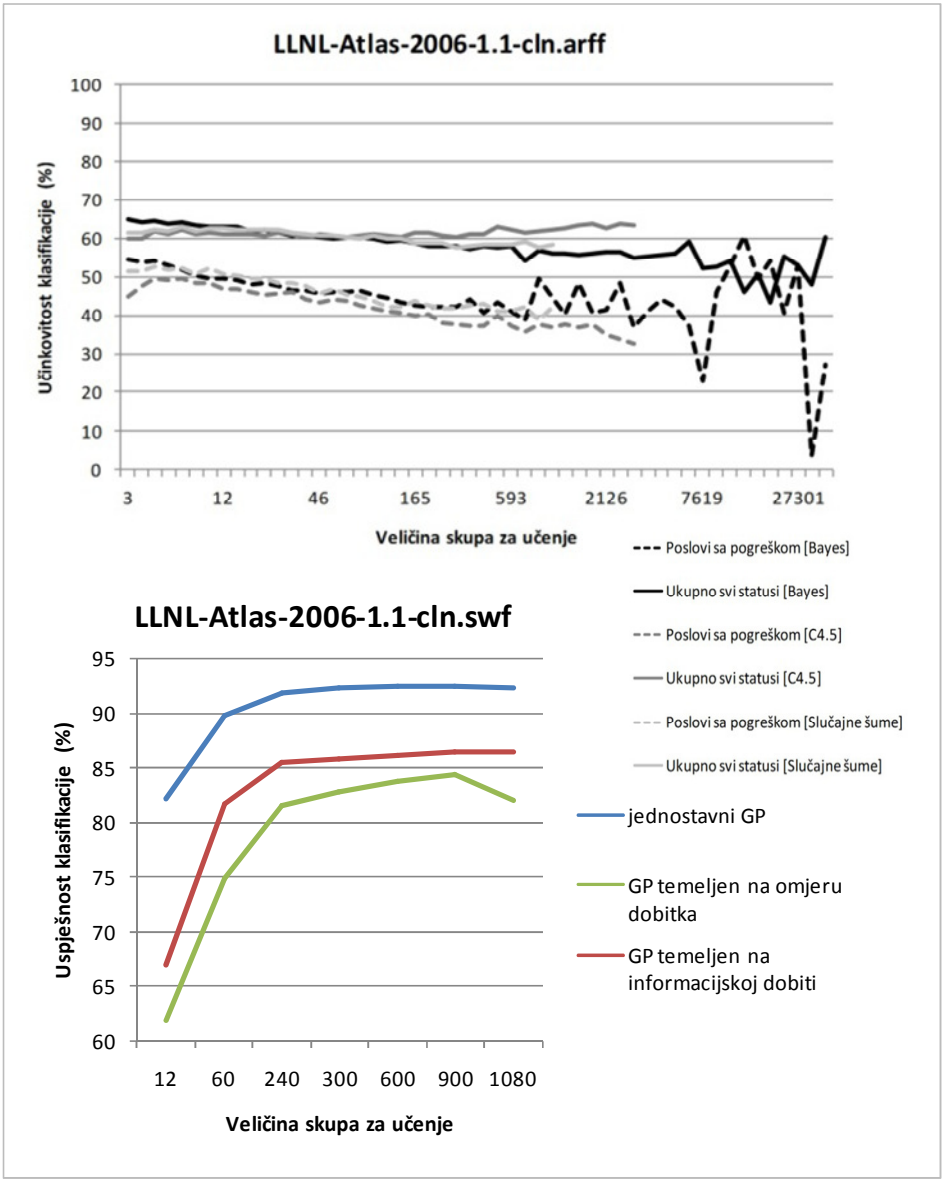
Slika 8.16 Usporedba rezultata različitih algoritama

Usporedba rezultata dobivenih algoritmima C4.5, Bayesove mreže i slučajne šume i rezultata dobivenih eksperimentima u praktičnom dijelu ovog rada prikazana je na slici 8.16. Gornji graf sa slike preuzet je iz [21], a donji graf je dobiven iz podataka koji se nalaze u tablici 8.14. Iz donjeg grafa može se vidjeti kako jednostavni GP daje

bolje rezultate za manje veličine skupa za učenje, a za veće je GP temeljen na informacijskoj dobiti bolji od ostalih dviju vrsta.

- *LLNL-Atlas-2006-1.1-cln.swf*

Za ovaj skup podataka Bayesove mreže dale su najveću uspješnost u iznosu od 60%.



Slika 8.17 Usporedba rezultata različitih algoritama

Za manje uzoraka uspješnost je bila oko 50% (najmanja oko 20% za nekoliko tisuća uzoraka u skupu za učenje).

Jednostavnim GP – om za ovaj skup najmanja pogreška iznosi oko 37%. Najmanje pogreške dobivene drugim dvjema vrstama kreću se od 8 do 20%.

Usporedba rezultata dobivenih algoritmima C4.5, Bayesove mreže i slučajne šume i rezultata dobivenih eksperimentima u praktičnom dijelu ovog rada prikazana je na slici 8.17. Gornji graf sa slike preuzet je iz [21]. Donji graf dobiven je iz podataka koji se nalaze u tablici 8.15. U ovom je slučaju jednostavni GP opet bolji od ostalih dviju vrsta.

9. Zaključak

Genetsko programiranje uspješno je upotrijebljeno za rješavanje problema klasifikacije kako na relativno jednostavnom skupu podataka od samo tisuću uzoraka, tako i na nekoliko složenijih skupova koji za učenje na raspolaganju imaju više desetaka pa i stotina tisuća primjera.

Rezultati dobiveni nizom eksperimenata u ovom radu pokazali su kako je za oba skupa podataka iz arhive paralelnog opterećenja grozdova jednostavni GP bolji od GP - a temeljenog na informacijskoj dobiti i GP - a temeljenog na omjeru dobitka promatrano na ukupno 1200 uzoraka. Za oba skupa podataka GP temeljen na informacijskoj dobiti dao je bolje rezultate od GP – a temeljenog na omjeru dobitka.

10. Literatura

[1] Bay, S., D., Nearest Neighbor Classification from Multiple Feature Subsets, Intelligent Data Analysis, Volume 3, Issue 3, Department of Information and Computer Science, University of California, Irvine, 1998, str. 191 – 209, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.51.9040&rep=rep1&type=pdf

[2] Bojarczuk, C. C., A constrained-syntax genetic programming system for discovering classification rules: application to medical data sets, Artificial Intelligence in Medicine, 30(1), 2004., str. 27 - 48

[3] Bojarczuk, C. C. et al., Discovering Comprehensible Classification Rules using Genetic Programming: A Case Study in a Medical Domain, Proceedings of the Genetic and Evolutionary Computation Conference, Volume 2, 1999., Morgan Kaufmann, str. 953 – 958, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.9861&rep=rep1&type=pdf

[4] Bojarczuk, C. C., Lopes H. S., Freitas A. A., An innovative application of a constrained-syntax genetic programming system to the problem of predicting survival of patients, Genetic Programming Proceedings of EuroGP2003, Volume 2610, Springer – Verlag, 2003., str. 11 – 21, http://sci2s.ugr.es/keel/pdf/specific/congreso/springerlink.com3.pdf

[5] Bot, M., Langdon, W., Application of Genetic Programming to Induction of Linear Classification Trees, Genetic Programming Proceedings of EuroGP2000, Volume 1802, Springer – Verlag, 1999., str. 247 – 258, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.8630&rep=rep1&type=pdf

[6] Confusion matrix, http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html, 9. 5. 2011.

[7] Dahyot R., Bootstrap, Jackknife and other resampling methods, Department of Statistics, Trinity College Dublin, Ireland, 2005., http://www.scss.tcd.ie/Rozenn.Dahyot/453Bootstrap/05_Permutation.pdf

[8] Dalbelo Bašić, B., Šnajder, J., Predavanja iz kolegija Strojno učenje, FER, Zagreb, ak. god. 2010./2011.

[9] Dalbelo Bašić, B., Sustavi temeljeni na pravilima, predavanje iz kolegija Umjetna inteligencija, FER, Zagreb, travanj 2008.

[10] Data Sets Error Rates, 24. 8. 1999., http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume11/opitz99a-html/node9.html, 23. 6. 2011.

[11] Eggermont, J., Data Mining using Genetic Programming: Classification and Symbolic Regression, doktorska disertacija, Laiden University, 2005., http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/eggermont_thesis.html

[12] Eggermont J., Eiben, A. E., van Hemert, J. I., A comparison of genetic programming variants for data classification, Advances in intelligent data analysis, Lecture Notes in Computer Science, Volume 1642, Leiden University, Leiden, Nizozemska, 1999., str. 281 - 290

[13] Eggermont, J., Kok, N., J., Kosters, A., W., Genetic Programming for Data Classification: Refining the Search Space, Universiteit Leiden, 2003., http://www.liacs.nl/~kosters/bnaic03-eggermont.ps

[14] Eggermont, J., Kok, N., J., Kosters, A., W., Genetic Programming for Data Classification: Partitioning the Search Space, Universiteit Leiden, Proceedings of the 2004 ACM Symposium on Applied Computing (SAC), Nicosia, Cyprus, 2004., http://www.liacs.nl/~kosters/SAC2003final.pdf

[15] Engelbrecht, A. P., Schoeman, L., Rouwhorst S., A Building Block Approach to Genetic Programming for Rule Discovery, Data Mining: A Heuristic Approach, Idea Group Publishing, 2002., str. 174 – 190, http://www.smesfair.com/pdf%5Cstatistics%5Csmesfair01.pdf

[16] Folino, G., Pizzyti, C., Spezzano, G., Genetic Programming and Simulated Annealing: A Hybrid Method to Evolve Decision Trees, Genetic Programming Proceedings of EuroGP2000, Volume 1802, Springer – Verlag, 2000., str. 294 – 303, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.36.6517&rep=rep1&type=ps

[17] Freitas, A. A., A Genetic Programming Framework For Two Data Mining Tasks: Classification And Generalized Rule Induction, Morgan Kaufmann, In Genetic Programming 1997: Proceedings of the Second Annual Conference, 1997., str. 96 – 101, http://kar.kent.ac.uk/21483/

[18] Freitas, A. A., A Survey of Evolutionary Algorithms for Data Mining and Knowledge Discovery, Pontificia Universidade Catolica do Parana Rua Imaculada

Conceicao, Curitiba, Brazil, Advances in Evolutionary Computation, Springer – Verlag, 2002., str. 819 – 845, http://www.cs.kent.ac.uk/pubs/2002/1582

[19] Gamberger, D., Šmuc, T., Marić, I., Poslužitelj za analizu podataka (DMS), http://dms.irb.hr/tutorial/hr_tut_mod_eval_1.php, Institut Ruđer Bošković, 2001.

[20] Grudenić, I., Bogunović, N., Lecture Notes in Computer Science: Job Status Prediction – Catch Them Before They Fail, Advances in Grid and Pervasive Computing, Volume 6646/2011, 2011., str. 3 - 12

[21] Grudenić, I., Prilagodljivo dinamičko raspoređivanje skupnih poslova na grozdu računala, doktorska disertacija, FER, Sveučilište u Zagrebu, 2010.

[22] Grupa autora, Otkrivanje znanja dubinskom analizom podataka, Priručnik za istraživače i studente, Institut Ruđer Bošković, http://lis.irb.hr/Prirucnik/prirucnik-otkrivanje-znanja.pdf

[23] Gutierrez-Osun, R., Intelligent Sensor Systems, Lecture 13: Validation, Wright State University, http://research.cs.tamu.edu/prism/lectures/iss/iss_l13.pdf

[24] Huang, J. J., Tzeng G. H., Ong C. S., Two-stage Genetic Programming (2SGP) for the credit scoring model, Applied Mathematics and Computation, Volume 174, Elsevier, 2006., str. 1039 – 1053, http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/Huang_Tgp_06.html

[25] Intuitive Biostatistics: Choosing a statistical test, http://www.graphpad.com/www/book/choose.htm, 23. 6. 2011.

[26] Jabeen, H., Baig, A. R., Review of Classification Using Genetic Programming, National University of Computer and Emerging Sciences, Islamabad, Pakistan, 2010., http://www.ijest.info/docs/IJEST10-02-02-06.pdf

[27] Jakobović, D., Uvod u genetsko programiranje, predavanje, FER, Sveučilište u Zagrebu, http://www.zemris.fer.hr/~yeti/studenti/Uvod%20u%20genetsko%20programiranje.pdf

[28] Jović, A., Postupci dubinske analize podataka, FER, Sveučilište u Zagrebu, http://www.fer.hr/_download/repository/Jovic,KDI.pdf

[29] Kechadi, T., Carthy, J., *Data Mining*, http://www.csi.ucd.ie/staff/jcarthy/home/DataMining/DM-Lecture01.pdf, 9. 5. 2011.

[30] Kosinski, W., Advances in Evolutionary Algorithms, InTech, studeni 2008.

[31] Koza, J. R., Concept formation and decision tree induction using the genetic programming paradigm, Parallel Problem Solving from Nature - Proceedings of

1st Workshop,Volume 496, Springer – Verlag, 1991., str. 124 – 128, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.5800&rep=rep1&type=pdf

[32] Koza, J. R., Genetic Programming – on the programming of computers by means of natural selection, MIT Press, Cambridge, 1992., http://www.ru.lv/~peter/zinatne/ebooks/MIT%20-%20Genetic%20Programming.pdf

[33] Lanzi, P., L., Data Mining and Text Mining, predavanja, Politecnico di Milano, Italy, 2007., http://www.pierlucalanzi.net/wp-content/teaching/dmtm/DM2011-02-DataMining.pdf

[34] Li, Q. et al., Dynamic Split-Point Selection Method for Decision Tree Evolved by Gene Expression Programming, Evolutionary Computation, 2009., str. 736 - 740

[35] Lin, J. Y., Classifier design with feature selection and feature extraction using layered genetic programming, Expert Systems with Applications, Volume 34, Issue 2, 2007., str. 1384-1393, http://myweb.nutn.edu.tw/~bcchien/Papers/J_EWSA2008_2.pdf

[36] Marmelstein, R. E. et al., Pattern Classification using a Hybrid Genetic Program – Decision Tree Approach, Proceedings of the Third Annual Conference of Genetic Programming, Morgan Kaufmann, 1998., str. 223 – 231,

http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/marmelstein_1998_pchGPpdta.html

[37] Mendes R. R. F., et al., Discovering Fuzzy Classification Rules with Genetic Programming and Co-Evolution, Proceedings of the Genetic and Evolutionary Computation Conference GECCO2001, Volume 2168, Springer – Verlag, 2003., str. 314 – 325, http://sci2s.ugr.es/keel/pdf/specific/congreso/springer1_5.pdf

[38] Muni, D., P., Pal, N., R., Das, J., A Novel Approach To Design Classifiers using GP, IEEE Transactions of Evolutionary Computation, Volume 8, Issue 2, 2004., str. 183 – 196, http://www.isical.ac.in/~muni_r/munipaldas_TEC.pdf

[39] Oltean, M., Diosan L., An autonomous GP-based system for regression and classification problems, Applied Soft Computing, Volume 9, Issue 1, 2009., str. 49 – 60

[40] Parallel Workloads Archive, http://www.cs.huji.ac.il/labs/parallel/workload/

[41] Poli, R., Langdon B. W., McPhere, F. N., A Field Guide to Genetic Programming, http://www.gp-field-guide.org.uk/

[42] Quinlan, J., Induction of decision trees, Mach. Learn. 1, 1, Machine Learning, 1986., str. 81 – 106, http://www.di.unipi.it/~coppola/didattica/ccp0506/papers/ML_1.1.81_Quinlan.pdf

[43] Refaeilzadeh, P., Tang, L., Liu, H., Cross-Validation, Arizona State University, 2008., http://www.public.asu.edu/~ltang9/papers/ency-cross-validation.pdf

[44] Ribarić, S., Predavanja iz kolegija Raspoznavanje uzoraka, FER, Sveučilište u Zagrebu, ak. god. 2009. / 2010.

[45] Savio, A., Termenón, M., Graña, M., Machine learning in fMRI, Validation, Computational Intelligence Group, University of the Basque Country, prosinac 2010., http://www.ehu.es/ccwintco/uploads/c/c8/Validation_uji_2010.pdf

[46] Silva, S., Controlling bloat: individual and population based approaches in Genetic Programming, doktorska disertacija, Dep. Informatics Engineering, University Coimbra, 2008.

[47] Silva, S., Costa, E., Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories, Genetic Programming and Evolvable Machines, Volume 10, Number 2, 2009., str. 141 - 179

[48] Smart, W., Zhang, M., Using Genetic Programming For Multiclass Classification By Simultaneously Solving Component Binary Classification Problems, Lecture Notes in Computer Science, Volume 3447/2005, Springer, 2005., str. 227 – 239, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.96.621&rep=rep1&type=pdf

[49] Squier, L., What is Data Mining?, IBM SPSS, studeni 2001., www.dama-ncr.org/Library/2001.11.14-Laura%20Squier.ppt

[50] Table of critical values for the Wilcoxon test, http://www.sussex.ac.uk/Users/grahamh/RM1web/WilcoxonTable2005.pdf, 23. 6. 2011.

[51] Takac, A., Application of Cellular Genetic Programming in Data Mining, Znalosti (2004), Slovakia, 2004., http://ii.fmph.uniba.sk/~takaca/KN04.PDF

[52] Tan, P. - N., Steinbach, M., Kumar, V., Introduction to Data Mining, Addison – Wesley, 2006., http://www-users.cs.umn.edu/~kumar/dmbook/ch4.pdf

[53] Tsakonas, A., A comparison of classification accuracy of four genetic programming-evolved intelligent structures, Information Sciences (2006), Volume 176, Issue 6, 2006., str. 691 - 724

[54] UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/

[55] Weise, T., Achler, S., Göb, M., Voigtmann, C., Zapf, M., Evolving Classifiers – Evolutionary Algorithms in Data Mining, Reproduction (2007) , Volume: 2007, Issue: 2007, 4, University of Kassel, Germany, 2007., str. 1 – 20, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.6165&rep=rep1&type=pdf

[56] Wong, M. L., Leung K. S., Learning Programs in Different Paradigms using Genetic Programming, Proceedings of the Fourth Congress of the Italian Association for Artificial Intelligence, Springer – Verlag, Berlin, 1995., str. 353 – 364,

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.5283&rep=rep1&type=pdf

11. Sažetak

U ovom radu navedeni su različiti načini primjene genetskog programiranja na problem klasifikacije podataka, a detaljnije je opisan i analiziran jedan od načina koji koristi genetsko programiranje za razvoj stabala odluke kao jednog vrlo poznatog i često korištenog klasifikacijskog algoritma. Klasifikacija podataka vezana je za dubinsku analizu podataka, tj. za otkrivanje korisnog i zanimljivog znanja u podacima i do sad su se za rješavanje ovog problema koristili različiti algoritmi iz domene umjetne inteligencije i strojnog učenja. Glavna značajka tehnika strojnog učenja je poboljšanje vlastitih svojstava na temelju prikupljenog iskustva, a genetsko programiranje se ovdje koristi kao jedna od tih tehnika, jer ono uči na određenoj količini primjera i poslije je sposobno prikupljeno iskustvo i znanje iskoristiti za rad s dosad neviđenim podacima.

11.1 Abstract

In this paper are presented different ways of applying genetic programming on a data classification problem. It is described and analyzed in more detail one of many ways of applying genetic programming on decision tree induction, a commonly used classification algorithm. Data classification is related to data mining and discovering interesting and useful knowledge in datasets. For solving this kind of problem several techniques from artificial inteligence and machine learning field are used. The main feature of machine learning techniques is improving own abilities based on gained experience. Genetic programming is used as one of these techniques because it learns on some amount of examples (patterns) and afterwards can use the gained experience and knowledge to work on yet unseen data.

11.2 Ključne riječi

Genetsko programiranje, klasifikacija, pogreška klasifikacije, stabla odluke, informacijska dobit, omjer dobitka, grupiranje, skup za učenje, skup za ispitivanje