SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Primjena genetskog programiranja u strojnom učenju

Karlo Knežević

Voditelj: *doc.dr.sc. Domagoj Jakobović*

Zagreb, svibanj, 2012.

Sadržaj

1. Uvod

Dubinskom analizom podataka (engl. *data mining*) naziva se primjena računarskih postupaka i alata koji pomažu u analizi podataka. To je relativno novo područje računarske znanosti koje se intenzivno razvija te je praktično nemoguće obuhvatiti sve tehnike koje ono uključuje. Što god danas nabrojali već sutra može biti prošireno nekim novim pristupom. Isto tako, ne postoji usuglašen pristup da se neki zadaci analize podataka trebaju obavljati određenim postupcima. Postoje samo pozitivna i manje pozitivna iskustva sa određenim postupcima i njihovom primjenom na konkretnim domenama [3].

Osnovno svojstvo po kojem se dubinska analiza podataka razlikuje od tradicionalne ili „obične“ analize primjena je postupaka strojnog učenja. Strojno učenje (engl. *machine learning*) dio je područja računarstva poznatog kao umjetna inteligencija (engl. *artificial intelligence*). Umjetna inteligencija bavi se razvojem računarskih postupaka koji su u stanju računalima simulirati inteligentno ponašanje, a strojno učenje važnim podskupom tih postupaka koje karakterizira mogućnost učenja na osnovi prethodnog iskustva.

Genetsko programiranje (engl. *genetic programming*, GP) jedan je od pokušaja odgovora na pitanje kako navesti računalo da riješi neki zadatak bez davanja izravnih uputa o postupku rješavanja; drugim riječima, kako postići da računalo učini nešto korisno, a da mu nismo rekli kako to treba učiniti (ovakav način rješavanja problema može se još i nazvati automatsko programiranje).

U drugom poglavlju opisan je pojam strojnog učenja te postupci i vrste strojnog učenja, kao i rješavanje problema prenaučenosti i podnaučenosti. U trećem poglavlju objašnjena je metoda genetskog programiranja, načini prikaza rješenja i problemi u generiranju rješenja. Uporaba genetskog programiranja u strojnom učenju opisana je u četvrtom poglavlju.

2. Strojno učenje

Strojno učenje jest programiranje računala na način da optimiziraju neki kriterij uspješnosti temeljem podatkovnih primjera ili prethodnog iskustva [7]. To znači ukoliko postoji neki model koji je definiran do na neke parametre, učenje bi bilo optimizacija parametara modela temeljem podataka. Npr, ukoliko je model polinom *n*-tog stupnja, $f(x) = a_n x^n + \cdots + a_1 x + a_0$, učenje je pronalazak i optimizacija koeficijenata polinoma $(a_1, \ldots, a_n)$. Model može biti predikcijski ili deskriptivan [7].

Temeljni pojmovi u strojnom učenju su indukcija i generalizacija. Cilj je za zadani uzorak, ograničene veličine, pronaći opće pravilo koje objašnjava podatke.

U ovom poglavlju bit će opisani samo osnovni pojmovi i problemi strojnog učenja. Naredna potpoglavlja vrlo sažeto objašnjavaju neke pojmove i ukoliko čitatelja interesira više, preporuča se dodatna literatura.

2.1 Vrste strojnog učenja

Postoje 3 vrste strojnog učenja [7]:

1. nadzirano učenje,
2. nenadzirano učenje i
3. podržano (ojačano) učenje.

Kod nadziranog učenja podaci za učenje su u obliku (ulaz, izlaz). Cilj učenja jest pronaći preslikavanje $\hat{y} = f(x)$ s ulaza na izlaz. Ukoliko je y diskretna vrijednost, tada se problem naziva klasifikacija, a ukoliko je y kontinuirana vrijednost, tada se naziva regresija.

Kod nenadziranog učenja, u skupu za učenje, nalaze se podaci bez ciljne vrijednosti. Cilj nenadziranog učenja jest pronaći pravilnosti u podacima.

Podržano ili ojačano učenje jest učenje optimalne strategije na temelju pokušaja s odgođenom nagradom. Tipične primjene podržanog učenja su igranje igara, robotika i upravljanje i višeagentski sustavi.

2.2 Nadzirano učenje [7]

Postupcima nadziranog učenja mogu se rješavati dvije vrste problema: klasifikacija i regresija. Kod klasifikacije primjeru pridružujemo klasu (razred) kojoj taj primjer pripada. Kod regresije primjeru pridružujemo neku kontinuiranu vrijednost. Razlika je dakle u tome je li ciljna varijabla diskretna ili nominalna (klasifikacija) ili kontinuirana (regresija).

2.2.1 Osnovni pojmovi

Svrha klasifikacije jest odrediti klasu *C* kojoj pripada primjer *x*. Primjer se definira kao vektor značajki, $x = (x_1, \dots, x_n)^T$, gdje je *n* dimenzija vektora. Primjeri se mogu interpretirati kao točke u *n*-dimenzijskom vektorskom prostoru koji se naziva ulazni prostor (engl. *input space*) ili prostor primjera (engl. *instance space*). Skup primjera za učenje, *D*, sastoji se od parova primjera i pripadnih oznaka, $D = \{(x^i, y^i)\}_{i=1}^{N}$, gdje je *N* ukupan broj primjera za učenje, a *i* je indeks primjera, odnosno njemu pripadne oznake.

Neka je *X* skup svih mogućih primjera. Pretpostavka svih algoritama strojnog učenja jest da su primejri iz *X* uzorkovani nezavisno i iz iste zajedničke distribucije, $P(x, y)$. Ta pretpostavka skraćeno se označava s *iid* (engl. *independent and identically distribuited*).

2.2.2 Hipoteza i model

Zadaća klasifikacijskog algoritma jest inducirati hipotezu $h : \mathrm{X} \rightarrow \{0,1\}$ koja određuje pripada li neki primjer *x* klasi *C* ili ne (2.1) .

$$h(x) = \begin{cases} 1 \; x \; pripada \; klasi \; C \\ 0 \; x \; ne \; pripada \; klasi \; C \end{cases} \tag{2.1}$$

Primjer $x \in X$ zadovoljava hipotezu $h \in H$ akko $h(x) = 1$. Hipoteza *h* konzistentna je s primjerom za učenje *(x,y)* akko $h(x) = y$.

Hipoteze se odabiru iz pomno odabranog skupa mogućih hipoteza. Skup mogućih hipoteza *H* naziva se model ili prostor hipoteza. Učenje se svodi na pretraživanje prostora hipoteza *H* i nalaženje najbolje hipoteze $h \in H$. Koliko dobro hipoteza *h*

klasificira primjere za učenje iskazuje empirijska pogreška ili pogreška učenja (engl. *training error*) (2.2).

$$E(h|D) = \frac{1}{N}\sum_{i=1}^{N} 1\{h(x^i \neq y^i\} \tag{2.2}$$

Oznaka $1\{P\}$ označava indikatorsku funkciju čija vrijednost je 1 ako $P \equiv T$, a 0 inače. Primjeri koje hipoteza klasificira pozitivno, a zapravo su negativni, nazivaju se lažno pozitivni primjeri (engl. *false positives,* FP). Obrnuto, primjeri koje hipoteza klasificira negativno, a zapravo su pozitivni, nazivaju se lažno negativni primjeri (engl. *false negatives*, FN).

Moguće je da za neki skup za učenje *D* postoji više hipoteza modela *H* koje ispravno klasificiraju primjere iz *D*. Skup takvih hipoteza naziva se prostor inačica (engl. *version space*).

Svojstvo hipoteze da odredi (predvidi) klasifikaciju još neviđenih primjera naziva se generalizacija.

2.2.3 Problem šuma

Šum je nepoželjna anomalija u podacima. Mogući uzroci šuma su:

- nepreciznost pri mjerenju značajki,
- pogreške u označavanju (engl *teacher noise*),
- postojanje skrivenih značajki (latentnih varijabli),
- nejasne granice klasa (subjektivnost).

U prisutstvu šuma ne postoji jednostavna granica između pozitivnih i negativnih primjera, čak i onda kada je problem jednostavan.

2.2.4 Regresija

Kod regresije ciljna vrijednost *y* je kontinuirana, $y \in \boldsymbol{R}$. Na temelju primjera $D = \{(x^i, y^i)\}$ potrebno je naučiti nepoznatu funkciju $f: X \to \boldsymbol{R}$ tako da $y^i = f(x^i)$. Zbog prisutstva šuma, uči se funkcija $y^i = f(x^i) + \varepsilon$, gdje je ε slučajni šum.

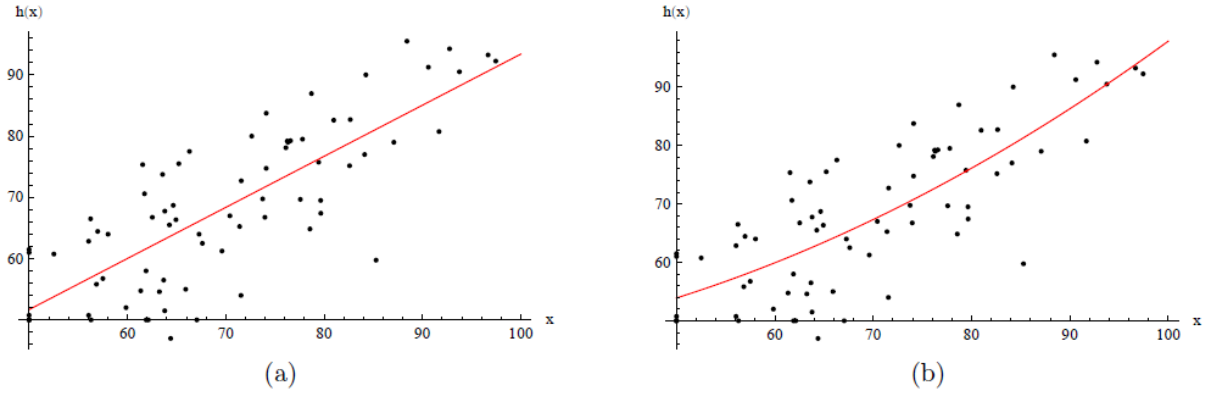
Kod regresije, empirijska pogreška izražena je formulom (2.3).

$$E(h|D) = \frac{1}{2}\sum_{i=1}^{N}(y^i - h(x^i))^2 \tag{2.3}$$

Ukoliko se odabere linearan model $h \in H$ koji minimizira empirijsku pogrešku, dobiva se izraz (2.4).

$$h(x) = \sum_{i=1}^{N} w_i x_i + w_0 = w^T x + w_0 \tag{2.4}$$

Pomoću metode najmanjih kvadrata (engl. *least squares*) računaju se vrijednosti težinskih koeficijenata hipoteze (slika 1). Ukoliko je prostor *X* jednosdimenzijski, *X*=***R***, tada se rješenje dobiva u zatvorenoj formi (engl. *closed-form solution*). U slučajevima kada analitičko rješenje ne postoji, učenje se provodi iterativnim metodama.



Slika 1. a) linearna regresija polinomom 1. stupnja, b) linearna regresija polinomom 2. stupnja [7].

2.2.5 Odabir modela

Odabir jednog modela *H* iz skupa modela naziva se odabir modela (engl. *model selection*). Budući da se odabir svodi na optimizaciju hiperparametara, često se koristi i naziv optimizacija modela. Glavna svrha odabira modela jest izabrati onaj model koji ima najbolje svojstvo generalizacije (§2.2.2).

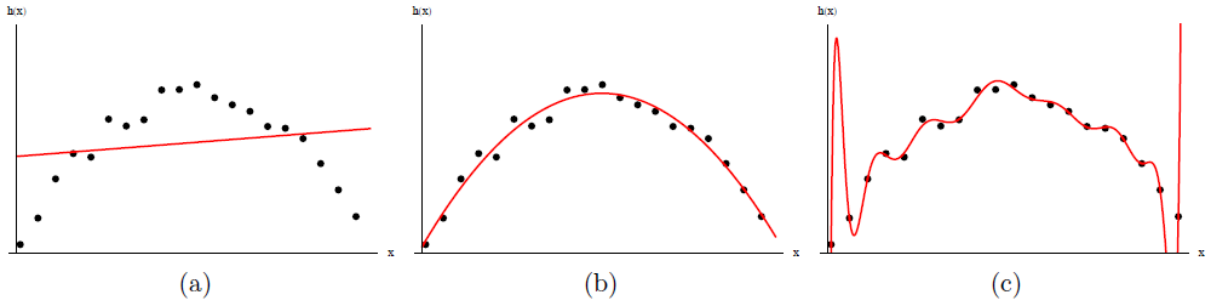
U načelu odabiru se što jednostavniji modeli. Razlozi za jednostavne modele su:

1. jednostavan (ali ne prejednostavan) model bolje generalizira,
2. jednostavan model lakše je koristiti (manja računalna složenost),
3. jednostavan model lakše je naučiti (složeniji modeli imaju više parametara koje treba optimirati),

4. jednostavan model lakše je tumačiti te iz njega ekstrahirati znanje (npr. pravila).

Najveći problem jest odabrati jednostavan, ali ne prejednostavan model, i uopće kako znati da je odabran jednostavan model, a ne složen (slika 2). Dvije krajnosti su:

- Prenaučenost (engl. *overfitting*) – ako je model *H* previše složen u odnosu na stvarnu funkciju, hipoteze $h \in H$ previše su prilagodljive, pa podaci *D* nisu dovoljni da ih ograniče.
- Podnaučenost (engl. *underfitting*) – ako je model *H* prejednostavan u odnosu na stvarnu funkciju, hipoteza se može dovoljno prilagoditi podacima, pa onda loše opisuje i podatke iz skupa za učenje *D*. Ako hipoteza ne može ispravno klasificirati podatke za učenje, izgledno je da će još lošije klasificirati nove primjere, tj. hipoteza će imati loše svojstvo generalizacije.

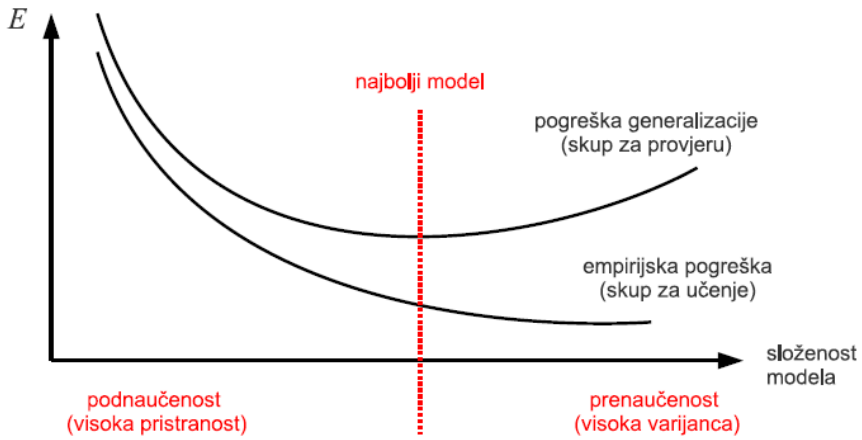


Slika 2. Regresija funkcije $f(x) = x^2 + \varepsilon$: a) podnaučenost (polinom 1. stupnja), b) optimalan model (polinom 2. stupnja), c) prenaučenost (polinom 15. stupnja) [7].

2.2.6 Unakrsna provjera

Jednostavan način da se kvantitativno utvrdi je li model prenaučen ili podnaučen jest unakrsna provjera (engl. *cross-validation*). Kod unakrsne provjere skup podataka za učenje podijeli se na dva dijela: skup za učenje (engl. *training set*) i skup za provjeru. Model se uči na skupu za učenje, a generalizacijska sposobnost provjerava se na skupu na provjeru. Pogreška hipoteze mjerena na skupu koji nije korišten za učenje naziva se pogreška generalizacije (slika 3).

Postoje i drugi načini odabira modela, ali unakrsna provjera je najčešći način.



Slika 3. Empirijska pogreška i pogreška generalizacije u ovisnosti o složenosti modela [7].

2.2.7 Komponente algoritma i pristupi nadziranom učenju

Svaki algoritam strojnog učenja mora definirati sljedeć tri komponente:

1. model ili prostor hipoteza,
2. funkciju gubitka i
3. optimizacijski postupak.

Funkcija gubitka za dane parametre modela θ izračunava razliku između ciljne vrijednosti y^i i njezine aproksimacije $h(x^i|\theta)$.

Optimizacijski postupak je postupak kojim se nalaze vrijednosti parametara modela θ^* za koje je empirijska pogreška najmanja (2.5).

$$\theta^* = argmin_\theta E(\theta|D) \tag{2.5}$$

Funkcija *argmin* vraća vrijednost argumenta koji minimizira funkciju.

Postoje tri pristupa nadziranom učenju prema odabiru modela:

1. generativni ili diskriminativni modeli,
2. parametarski i neparametarski modeli,
3. linearni i nelinearni modeli.

Generativni modeli modeliraju zajedničku vjerojatnost $P(x, C_j)$ te se pomoću te vjerojatnosti može odrediti posteriorna vjerojatnost $P(C_j|x)$. Diskriminativni modeli ne modeliraju zajedničku vjerojatnost, već izravno modeliraju pripadnost primjera x klasi

C_j. Diskriminativni modeli dijele se na probabilističke i neprobabilističke, u ovisnosti o interpretaciji izlaza klasifikatora.

Kod parametarskih modela složenost modela ne ovisi o broju primjera za učenje, dok neparametarski modeli ovise o broju primjera za učenje.

Linearni modeli povlače linearnu granicu između primjera dviju klasa, odnosno funkciju aproksimiraju linearnim modelom. Nelinearni modeli povlače nelinearnu granicu između primjera dviju klasa, odnosno funkciju aproksimiraju nelinearnom niperravninom.

2.3 Nenadzirano učenje [9]

Ukoliko skup primjera za učenje nije označen pripadnošću razreda, tj. ukoliko skup za učenje *D* izgleda kao $D = \{x^i\}_{i=1}^N$, tada govorimo o nenadziranom učenju (engl. *unsupervised learning*). Tri tipična zadatka nenadziranog učenja su:

1. grupiranje (engl. *clustering*),
2. otkrivanje novih vrijednosti ili vrijednosti koje odskaču,
3. smanjenje dimenzionalnosti (engl. *dimensionality reduction*).

Najpoznatiji algoritmi grupiranja su: algoritam k-srednjih vrijednsti, algoritam k-medoida, EM-algoritam (primjenjen na Gaussovu mješavinu ili druge mješavine) i hijerarhijsko grupiranje.

Svi pojmovi i definicije u §2.2 jednake su i u ovom poglavlju. Pošto je genetsko programiranje tipičan predstavnik nadziranog učenja, za detaljnije informacije o nenadziranom učenju čitatelj se upućuje na dodatnu literaturu.

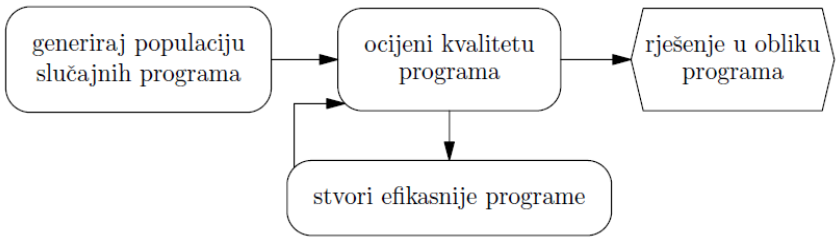
3. Genetsko programiranje

Genetsko programiranje (engl. *genetic programming*, *GP*) je optimizacijska tehnika koja pripada skupini evolucijskih algoritama (engl. *evolutionary algorithms*). U računarskoj znanosti, evolucijsko računanje dio je umjetne inteligencije, polje računarska inteligencija, koje se bavi kombinatornim optimizacijama. Evolucijsko računanje primjer je metaheuristike i danas se dijeli na tri velika područja[6]:

1. evolucijski algoritmi (engl. *evolutionary algorithms*),
2. algoritmi zasnovani na inteligenciji roja (engl. *swarm intelligence*),
3. ostali algoritmi.

Evolucijski algoritmi definiraju se kao postupci optimiranja, učenja i modeliranja, koji se temelje na mehanizmu evolucije u prirodi. Predložio ga je 1992. godine John R. Koza[1]. 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.

Pojednostavljen shematski prikaz genetskog programiranja prikazan je na slici 4.



Slika 4. Pojednostavljeni shematski prikaz genetskog programiranja [2].

3.1 Glavne značajke genetskog programiranja

Naziv *genetsko programiranje* sintagma je dviju riječi: *genetika* i *programiranje*. Pridjev *genetski* upotrijebljen je zbog korištenja genetskih operatora i zbog toga što se postupak imitira evoluciju, a imenica *programiranje* zbog toga što se rješenje prikazuje kao računalni program. Jedinka u populaciji programa prikazuje se u ovisnosti o problemu i načinu efikasnog rješenja. Kroz niz generacija razvijaju se 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 stabla se mijenjanju veličina i dubina stabla). U terminologiji povezanoj s genetskim programiranjem vrlo se često spominje izraz *bloat*, što je zapravo nekontrolirani rast duljine rješenja, tj. jedinke u populaciji i zato se mora uvesti način ograničavanja duljine rješenja [1].

Iz navedenog su očite razlike u odnosu na genetski algoritam (koji mnogi miješaju s genetskim programiranjem):

- prikaz rješenja (i same jedinke): $GA \rightarrow kromosom$ | $GP \rightarrow program$
- promjenjivost veličine jedinke: $GA \rightarrow nepromjenjiva$ | $GP \rightarrow promjenjiva$
- djelovanje genetskih operatora (kod *GP* ovisi o prikazu programa, a kod GA o sadržaju kromosoma)

3.2 Vrste genetskog programiranja

Genetsko programiranje dijeli se na nekoliko vrsta, ovisno o načinu prikaza računalnog programa, kao rješenja zadanog problema [1]:

- stablasti GP (engl. *tree-based GP*),
- linearni GP (engl. *linear GP*),
- GP temeljen na grafovima (engl. *graph-based GP*),
- celularni GP (engl. *cellular GP*),
- GP temeljen na gramatici (engl. *grammar-based GP*),
- GP ograničene sintakse (engl. *constrained syntax GP*),
- kartezijski GP (engl. *Cartesian GP*).

U ovom seminarskom radu naglasak je stavljen na genetsko programiranje temeljenom na stablima.

3.3 Formalna predodžba računalnog programa temeljenog na stablima

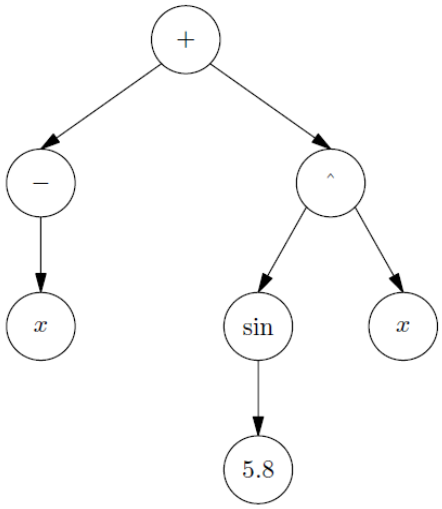
Kod stablastog genetskog programiranja program je predstavljen stablom. Stablo se, kao nelinearna struktura, vrlo često koristi kao prikaz rješenja; njime se mogu jednostavno prikazati različiti matematički i logički izrazi i sl. [2].

Izrazi se sastoje od:

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

Skupovi završnih (koji se uvijek nalaze u listovima), u oznaci *T*, i funkcijskih znakova, u oznaci *F*, zajedno čine skup primitiva (engl. *primitives*) za GP. Za primitive vrijede sljedeća pravila:

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



Slika 5. Stablasta predodžba izraza – $x + sin^x 5.8$ [2]

$Arg(f)$ je broj argumenata funckije *f*. Funkcije mogu imati i druge funkcije kao argumente (slika 5).

Stabla ne moraju biti samo binarna, već mogu imati i više djece. Izrazi se najčešće predstavljaju u prefiks notaciji (engl. *prefix, polish notation*). Također, funkcije mogu

imati promjenjiv broj argumenata ili mogu imati fiksan broj argumenata. 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.

3.4 Genetski operatori

Evolucijski apsekt genetskog programiranja očituje se u načinu optimiranja promatranog problema, gdje su u ulogama genetskih operatora prisutne metode reprodukcije (engl. *reproduction*), križanja (engl. *crossover*) i mutacije (engl. *mutation*).

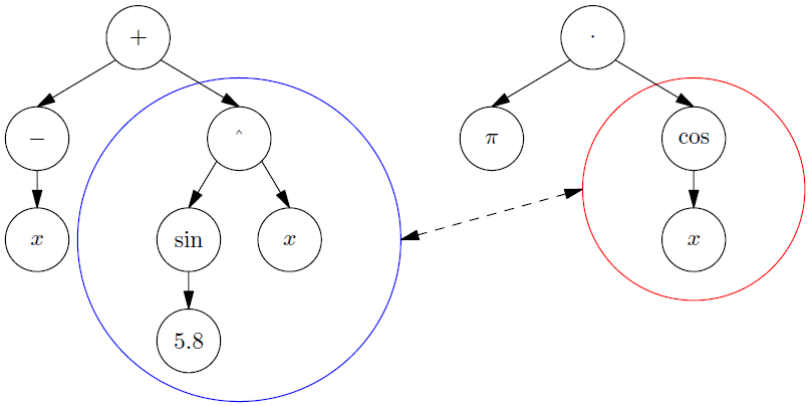
Osim navedenih operatora, koriste se još i permutacije (engl. *permutation*), inverzije (engl. *inversion*) itd. [2].

3.4.1 Reprodukcija

Reprodukcija je jednostavno kopiranje odabrane jedinke i umetanje iste u novu populaciju. Semantika ovog genetskog operatora je preživljavanje odabrane jedinke, a očekivana posljedica je povećanje prosječne dobrote populacije kroz iteracije algoritma [2].

3.4.2 Križanje

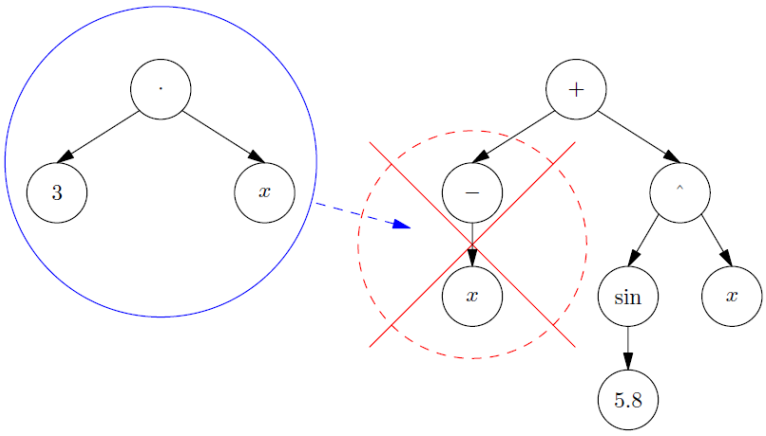
Križanje je analogno biološkoj splonoj rekombinaciji. Naime, u tom postupku dolazi do zamjene nekih podstabala dviju odabranih jedinki (slika 6). Najučestaliji oblik je obavljanje opisane radnje na dva slučajno odabrana podstabla [2].



Slika 6. Križanje [2].

3.4.3 Mutacija

Analogno biološkoj mutaciji, dolazi do poremećaja genetskog materijala, najčešće slučajno odabrane ili nekom drugim operatorom stvorene jedinke (slika 7). Radi se o zamjeni nekog podstabla jedinke novim, slučajno generiranim, stablom [2].



Slika 7. Mutacija [2].

3.5 Selekcija

Tijekom generiranja populacija odvija se selekcija, i to na temelju vjerojatnosti odabira genetskih operatora i dobrota jedinki (dobrotu određuje funkcija dobrote, engl. *fitness function*). Uobičajeno je da selekcija izravno (proporcionalno) ovisi o dobroti (jednostavna selekcija uz tzv. *roulette whee* metodu odabira jedinki).

Često se koriste i turnirske selekcije (engl. *tournament selection*), kod kojih se slučajno odabire *k* jedinki te se najgora, odnosno najgore jedinke, zamjenjuju novim jedinkama nastalim primjenom genetskih operatora nad najboljima. Turnirske selekcije imaju svojstvo elitizma, što znači da najbolja jedinka sigurno preživljava selekciju u promatranoj iteraciji.

Jedan od boljih načina za seleciju je sljedeći [2]:

[1] populacija se podijeli na dvije podpopulacije,

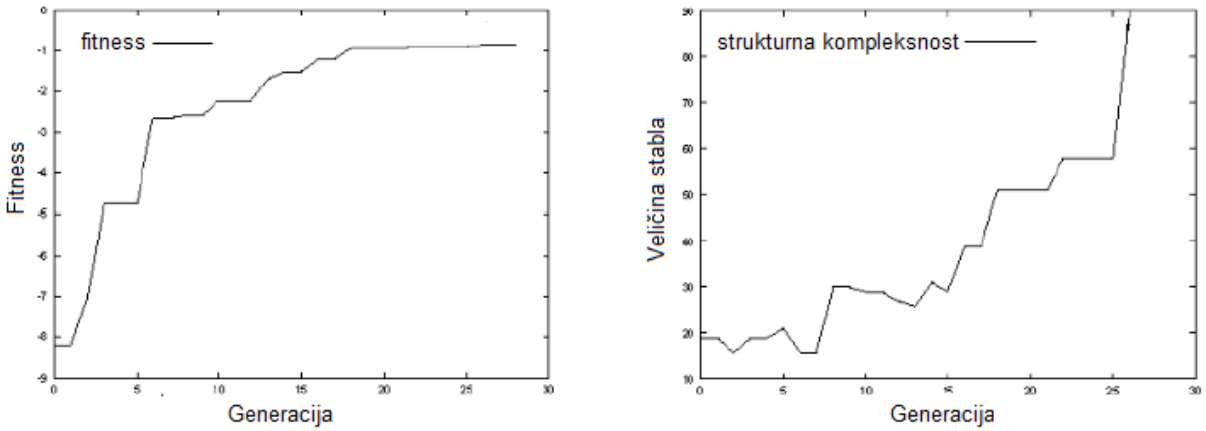
[2] većina operacija (≈ 80%) određenog genetskog operatora provodi se nad jednom podpopulacijom (uz *k*-turnirsku selekciju), a na ostatak populacije genetski operatori ne djeluju te se jedinke automatski prebacuju u sljedeću generaciju,

[3] u svakoj generaciji iterativno se ponavlja postupak [2] i [3].

3.6 Kontrola rasta jedinke i zaustavljanje generativnog procesa

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 (slika 8) [5]. 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. Moguća rješenja za kontrolu rasta jedinke su:

- uvođenje mjera kažnjavanja prevelikih jedinki,
- unaprijed odrediti supremum dubine sabla, D, ili broja funkcijskih čvorova, N_f,
- podrezivanje stabla itd.



Slika 8. Lijevo: prikaz odnosa vrijednosti funkcije dobrote i broja generacija, desno: prikaz odnosa veličine stabla i broj generacija. Bloat[5].

Obzirom na činjenicu da koncept genetskog programiranja, s dobro definiranom funkcijom dobrote nad određenim problemom i adekvatvnim skupovima F i T, relativno brzo dolazi do zadovoljavajućeg rješenja problema, u velikom broju slučajeva generativni proces zaustavlja se pri ispunjenju određenog logičkog predikata. Dodatno, u obzir se uzima i konstanta G, kao supremum broja iteracija, odnosno broja generacija populacije i supremum M, kao broj jedinki u populaciji.

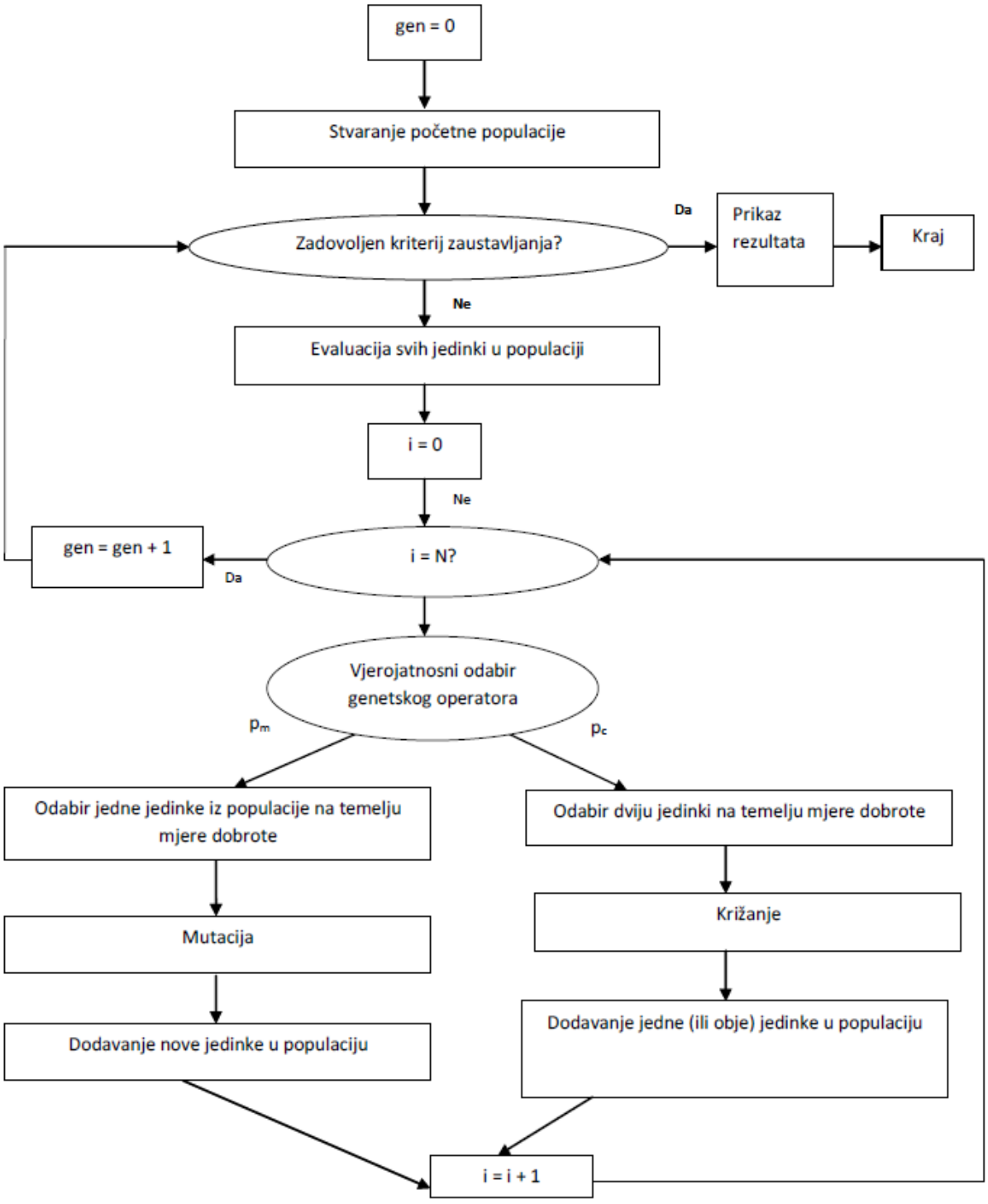
Jednom kada se ispuni kriterij zaustavljanja, iz populacije rješenja odabire se najkvalitetnije te se semantički opisuje, ako je moguće. Pošto je genetsko

programiranje stohastička metoda, poželjne su daljne alanize rješenja, kao i ponavljanje cijelog postupka. Bolja rješenja mogu se postići [2]:

- redefiniranjem funkcije dobrote,
- redefiniranjem skupova *F* i *T*,
- povećanjem supremuma broja iteracija ili jedinki,
- promjenom vjerojatnosti odabira genetskog operatora ili jedinke.

3.7 Blok-dijagram genetskog programiranja

Genetsko programiranje može se predočiti blok-dijagramom na slici 9 :



Slika 9. Blok-dijagram genetskog programiranja [1].

4. Primjena genetskog programiranja u strojnom učenju

U ovom poglavlju bit će opisan način rješavanja problema klasifikacije i regresije kod nadziranog učenja, uz pomoć genetskog programiranja. Ponuđeni postupci pripadaju stablastom genetskom programiranju. Također, bit će opisana samo jedna kategorija GP za klasifikaciju i za regresiju jer postoji više kategorija, koje sadrže više postupaka. Ukoliko čitatelj želi saznati više, upućuje se na dodatnu literaturu.

Stablasti GP (u daljnjem kontekstu samo GP) često se koristi kod klasifikacije različitih vrsta podataka i može se podijeliti u tri različite kategorije [1]:

- razvoj drugih klasifikacijskih algoritama,
- razvoj klasifikacijskih pravila (engl. *classification rules*),
- razvoj klasifikacijskih aritmetičkih izraza.

Prvoj kategoriji pripada razvoj različitih drugih klasifikacijskih algoritama, kao što su stabla odluke, neuronske mreže i sl. Drugoj kategoriji pripada razvoj pravila za klasifikaciju ili logičkih izraza s logičkim operatorima, a trećoj razvoj aritmetičkih izraza ili funkcija. Od navedenih klasifikacijskih kategorija, detaljnije će biti opisan razvoj drugih klasifikacijskih algoritama.

4.1 Razvoj klasifikacijskih algoritama

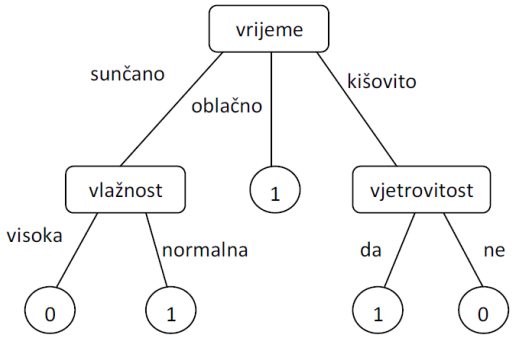
Kada se genetsko programiranje koristi za razvoj drugih klasifikacijskih algoritama postoji unaprijed niz pravila uz čiju se pomoć stvaraju moguća rješenja. Drugi klasifikacijski algoritmi koji se mogu razviti uz pomoć GP-a najčešće su [1]:

- stabla odluke (engl. *decision trees*),
- stabla odluke temeljena na neizrazitoj logici (engl. *fuzzy decision trees*),
- neuronske mreže (engl. *neural networks*, *multilayer perceptron*) i sl.

4.1.1 Razvoj stabala odluke genetskim programiranjem

Genetsko programiranje često se koristi za stvaranje stabala odluke koja klasificiraju podatke. John Koza prvi je predstavio ovaj način rješavanja problema klasifikacije (slika 10)[1]. Kao primjer, izgradio je stablo odluke koji na temelju meteoroloških uvjeta klasificira svaki uzorak u jedan od dva moguća razreda. Skup atributa čine

$\{vrijeme, temperatura, vlažnost, vjetrovitost\}$; vrijeme poprima vrijednosti *sunčano*, *oblačno* ili *kišovito*; temperatura poprima vrijednosti *visoka*, *srednja* ili *niska*; vlažnost može biti *visoka* ili *niska*; vjetrovitost poprima vrijednost *da* ili *ne* (ima li ili nema vjetra). Atributi su pretvoreni u funkcijske znakove $\{TEMP, HUM, OUT, WIND\}$, a razredi su pretvoreni završne ili terminalne znakove $\{0,1\}$. Svaka funkcija ima onoliko argumenata koliko vrijednosti taj atribut može poprimiti. Npr., atribut vrijeme pretvoren je u funkciju koja, ako je vrijeme oblačno, vraća svoj drugi argument, drugo podstablo. Ovakvo stablo odluke slično je stablima izgrađenom algoritmom *ID3* ili *C4.5*: funkcijski znakovi su atributi, a listovi su razredi.



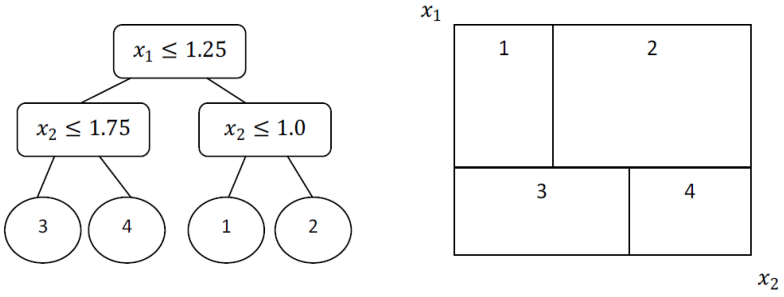
Slika 10. Primjer stabla iz primjera Johna Koza-e [1].

Stablo odluke može biti i linearno. Ovo stablo razlikuje se od uobičajenih stabala odluke jer može imati puno djece (puno više od 2 ili 3). Linearna stabla odluke sadrže čvorove koji se sastoje od linearnih kombinacija, tj. od aritmetičkih izraza više varijabli. Čitajući redak stabla slijeva, prvi čvorovi djece čvora roditelja predstavljaju konstante i varijable, a zadnji čvorovi djece predstavljaju oznake razreda. Ako redak na dubini *d* ima *n* čvorova djece, tada prvih $n-3$ djece čine linearnu kombinaciju, sljedeće dijete služi za suporedbu s linearnom kombinacijom (operator usporedbe može biti izabran proizvoljno; najčešće operator $\leq$), a zadnja dva djeteta označavaju razred. Ukoliko je usporedba istinita, uzorak se klasificira u razred koji sadrži $(n-1)$. dijete, u suprotnom u razred koji sadrži posljednje dijete (slično ternarnom operatoru u *C* jeziku, *?:*). Oznaka razreda može biti iz skupa završnih znakova, ali može biti i funkcijski znak pa se postupak rekurzivno ponavlja nadubini *d+1*. Iz navedenog je očito da ovakav klasifikator predstavlja linearnu kombinaciju binarnih klasifikatora.

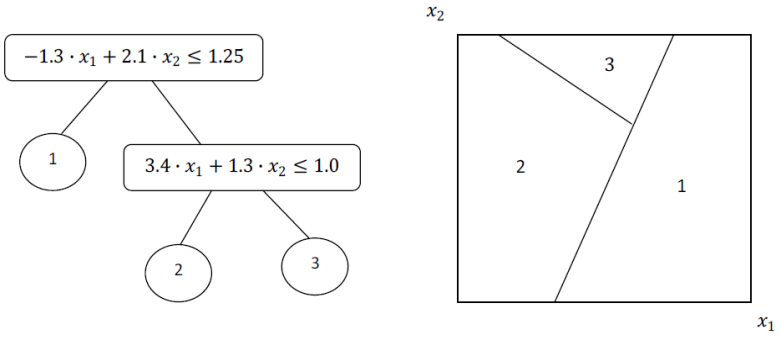
Stabla koja u čvorovima sadrže aritmetički izraz, dijele se u [1]:

[1] *axis – parallel decision trees* (nelinearna stabla),

[2] *oblique decision trees* (linearna stabla).



Slika 11. Nelinearno stablo odluke i odgovarajuća podjela 2D prostora atributa [1].



Slika 12. Linearno stablo odluke i odgovarajuća podjela 2D prostora atributa [1].

Linearna stabla (slika 12) odluke u čvorovima sadrže izraze koji mogu imati jednu, ali i više od jedne varijable, dok nelinearna stabla (slika 11) u čvoroima 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 hiperravnina koje su paralelne s osima.

Ovakav način učenja pripada nadziranom učenju.

4.1.2 Usporedba razvoja stabla izgrađenog *GP* i algoritmom *ID3*

Algoritam *ID3* (engl. *Induction of Decision Trees*) razvio je J.R.Quinlan (1986.), a proširenje je algoritam *C4.5* (Quinlan, 1993.)[9]. Algoritam *ID3* pohlepan je algoritam koji pretražuje potpun prostor hipoteza. Nepotpuno pretražuje prostor, sve dok ne nađe hipotezu konzistentnu s podacima. Algoritam preferira izgradnju kraćih stabala i preferira stabla koja stavljaju atribute s većom informacijskom dobiti bliže korijenu stabla. Na kraju postupka izgradnje stabla, dobiva se stablo koje je konzistentno sa

svim primjerima iz skupa za učenje [9]. Međutim, ovakav način izgradnje stabla sadrži dva potencijalna problema:

- podaci sadrže šum,
- skup za učenje je premalen.

U tom slučaju može doći do prenaučenosti stabla odluke. Prenaučenost se rješava podrezivanjem, zaustavljanjem rasta stabla prije savršene klasifikacije, uporabom statističkih testova i uvođenjem eksplicitne mjere kompleksnosti. Svaki od navedenih postupaka zahtijeva eksplicitnu intervenciju čovjeka i onemogućuje da sustav samostalno uči i izgrađuje stablo po svojoj odluci. Podrezivanje krivog čvora, ograničenje dubine ili manjak primjera u skupu za učenje može dovesti do izgradnje lošeg klasifikatora. To je bitna razlika između izgradnje stabla genetskim programiranjem i algoritmima *ID3* ili *C4.5*.

Usporedbom rezultata istraživanja u [1], kao i prikazom na slikama 14. i 15., vidljivo je da algoritam genetskog programiranja postiže bolje rezultate na većim skupovima podataka. Uporabom *GP* u [1], na većim skupovima podataka postignuto je poboljšanje klasifikacije za 5% - 7%. Tablicom 1 prikazana je usporedba točnosti klasifikacije algoritama *C4.5* i *GP* (funkcija dobrote je informacijska dobit).

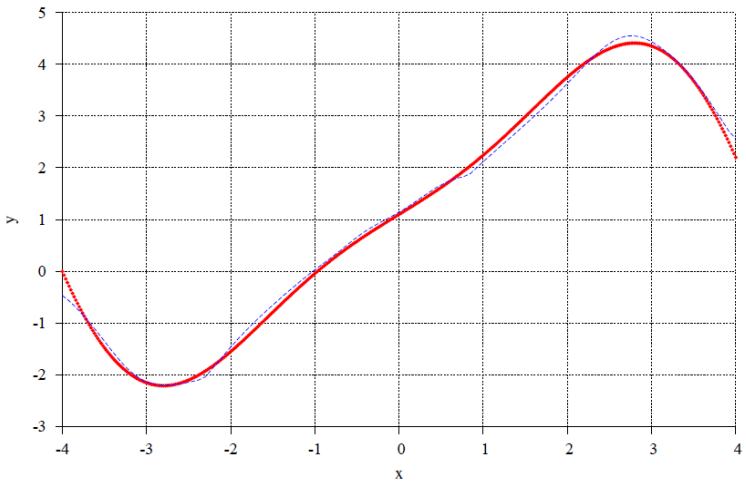
Tablica 1. Usporedba točnosti klasifikacije algoritama C4.5 i GP na problemu [1], u ovisnosti o veličini skupa za učenje.

Veličina skupa za učenje	Točnost klasifikacije algoritma (%)	
	C4.5	GP
12	**80**	68
60	82	**83**
240	83	**86**
300	83	**87**
600	82	**88**
900	84	**88**

1100	84	88

4.2 Regresija izvedena genetskim programiranjem

U strojnom učenju regresija se izvodi interpolacijom polinoma *n*-tog stupnja između točaka ili se koristi metoda lokalne regresije (s težinskim faktorima). Kod genetskog programiranja regresija se efikasno može izvesti izračunavanjem algebarskog izraza (funkcije) koju čini skup točaka iz skupa za učenje. Unaprijed se mora odrediti sadržaj skupova *F* i *T*. Genetskim programiranjem može se dobiti vrlo dobar rezultat jer evaluirano stablo dobro opisuje funkciju koja je generirala primjere skupa za učenje (slika 13).



Slika 13. Plavo: izvorna generirajuća funkcija primjera iz skupa za učenje; crveno: regresijska funkcija.

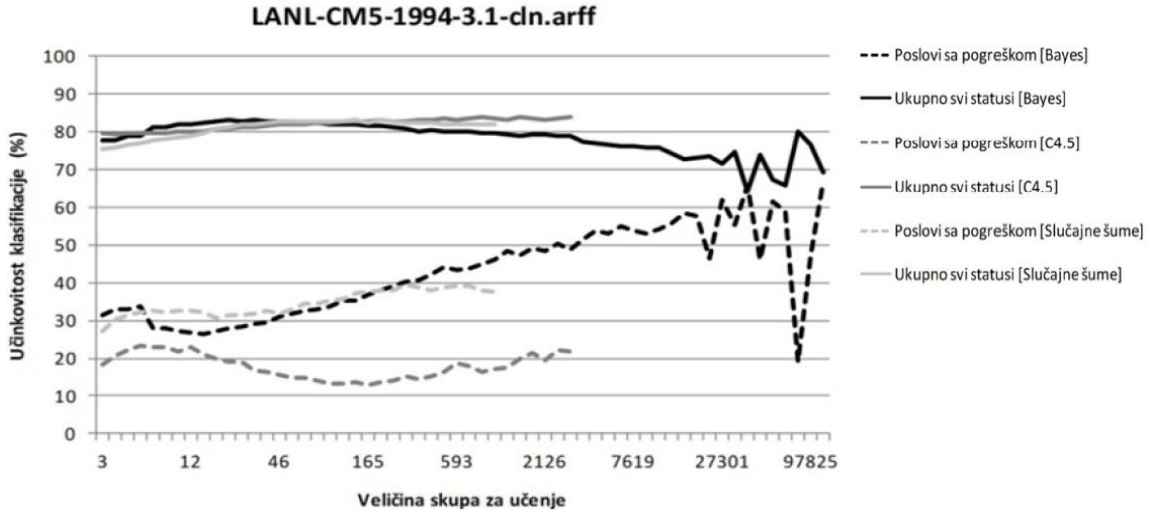
4.3 Prednosti i nedostaci primjene genetskog programiranja za klasifikaciju i regresiju

Jedna od prednosti korištenja genetskog programiranja je njihova robusnost i mogućnost rada sa zašumljenim podacima. Genetsko programiranje izvodi globalno pretraživanje prostora, dok većina ostalih algoritama strojnog učenja koriste pohlepno (engl. *greedy*) pretraživanje. Zbog globalnog pretraživanja prostora veća je vjerojatnost pornalaženja ispravnog rješenja, tj. globalnog optimima, dok ostali algoritmi, poput neuronskih mreža, gotovo uvijek pronalaze lokalne optimume (metoda gradijentnog spusta!). Za razliku od genetskih algoritama, jedinke u *GP*

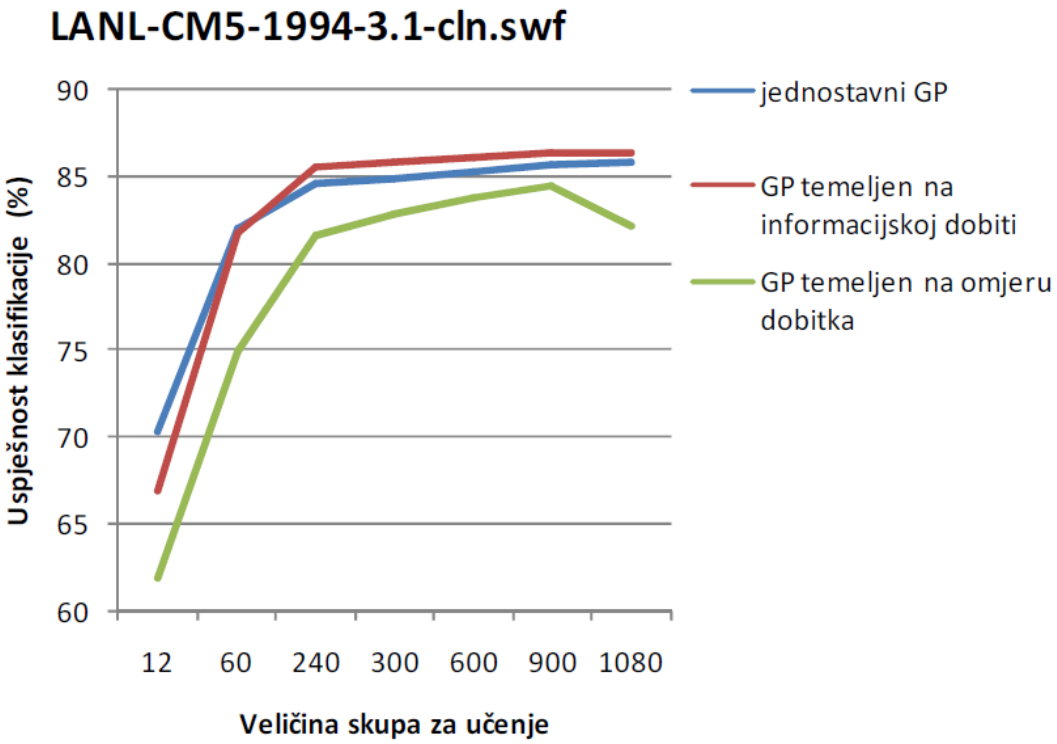
varijabilne su veličine i veličinom se mogu prilagođavati podacima. GP može raditi s malim skupom primjera za učenje, iščitavanje rješenja puno je razumljivije od ostalih klasifikatora i zbog robusnosti algoritma, GP može raditi sa sirovim podacima (nije nužna transformacija i pretprocesiranje podataka) [1].

Glavni nedostatak korištenja GP-a česta je potreba za evaluacijom jedinki tijekom rada. Broj evaluacija jednak je umnošku broja veličine populacije i broja generacija. Također, česti problem je nekontroliran rast jedinki (*bloat*). Tijekom rada jedinke počinju rasti, a točnost klasifikatora se na povećava.

Slike 14. i 15. prikazuju korištenje različitih algoritama na problem rješavan u [1]. Usporedbom grafova primjećuje se da klasifikacija genetskim algoritmom postiže puno bolje rezultate.



Slika 14. Učinkovitost klasifikacije podataka Bayesovim algoritmom, algoritmom C4.5 i algoritmom slučajne šume [1].



Slika 15. Učinkovitost klasifikacije genetskim programiranjem [1].

5. Zaključak

Genetsko programiranje je metoda stohastičkog pretraživanja koja predstavlja rješenja u obliku računalnih programa i na taj način omogućava rješavanje problema koji bi inače zahtijevao izradu programa. Zbog svoje strukture, genetsko programiranje može se smatrati najopćenitijim oblikom evolucijskog računanja te jednom od najopćenitijih metoda strojnog učenja [5].

U ovom seminarskom radu prikazana je primjena genetskog programiranja u strojnom učenju. Genetskim programiranjem može se izgraditi stablo odluke koje se koristi za klasifikaciju i regeresiju. Navedene su prednosti i nedostaci genetskog programiranja. Pokazano je da stablasti GP može puno bolje klasificirati od stabala generiranih drugim algoritmima zbog potpunog pretraživanja prostora stanja.

Smatram da se genetskim programiranjem učinkovito mogu riješiti mnogi problemi klasifikacije i regresije u strojnom učenju.

6. Literatura

[1] Stokić, I., Primjena genetskog programiranja u strojnom učenju, Diplomski rad 213., Fakultet elektrotehnike i računarstva, Zagreb, lipanj 2011.

[2] Kokan, I., Primjena genetskog programiranja u postupku simboličkog integriranja, Završni rad 317., Fakultet elektrotehnike i računarstva, Zagreb, lipanj 2008.

[3] Otkrivanje znanja dubinskom analizom podataka, Institut Ruđer Bošković, http://lis.irb.hr/Prirucnik/prirucnik-otkrivanje-znanja.pdf , 28. ožujka 2012.

[4] Herrmann, M., Genetic Programming: Examples and Theory, Natural Computing, Lecture 9, http://www.genetic-programming.org/ , 29. ožujka 2012.

[5] Jakobović, D., Raspoređivanje zasnovano na prilagodljivim pravilima, Doktorska disertacija, Fakultet elektrotehnike i računarstva, Zagreb, 2005.

[6] Knežević, K., Raspoređivanje u okružju nesrodnih strojeva uporabom evolucijskog računanja, završni rad 1991., Fakultet elektrotehnike i računarstva, Zagreb, srpanj 2011.

[7] Dalbelo Bašić, B., Šnajder, J., Nadzirano učenje, bilješka 1, predavanje iz predmeta Strojno učenje, Fakultet elektrotehnike i računarstva, Zagreb, 2011.

[8] Dalbelo Bašić, B., Šnajder, J., Grupiranje podataka, bilješka 5, predavanje iz predmeta Strojno učenje, Fakultet elektrotehnike i računarstva, Zagreb, 2011.

[9] Dalbelo Bašić, B., Šnajder, J., Stabla odluke, predavanje iz predmeta Strojno učenje, Fakultet elektrotehnike i računarstva, Zagreb, 2011.

7. Sažetak

U seminarskom radu opisani su pojmovi nadziranog učenja. Objašnjena je metoda genetskog programiranja, zajedno sa genetskim operatorima i predloženim rješenjem za rješavanje problema prebrzog rasta jedinki bez izrazitog poboljšanja funkcije dobrote. Prikazani su postupci razvoja stabala odluke i regresije genetskim programiranjem.

Navedeni su prednosti i nedostaci genetskog programiranja, kao i usporedba učinkovitosti GP i ostalih algoritama na problemu klasifikacije.

U seminarskom radu pokazano je da je genetsko programiranje učinkovita metoda za problem klasifikacije i regresije u strojnom učenju.