SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4334

Primjena genetskog programiranja na problem klasifikacije podataka

Ivan Vlašić

Zagreb, svibanj 2016.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Klasifikacija je problem razvrstavanja podataka u kategorije na osnovu skupine podataka za učenje čija je podjela već poznata. Kao primjer, koji se u praksi često koristi, možemo navesti prepoznavanje spada li novi email pod neželjeni sadržaj ili ne. Razvojem računarske znanosti, a pogotovo strojnog učenja, došlo je do pojave različitih metoda rješavanja tog problema. Neke su bile više uspješne od drugih. Ovaj rad će pokušati pojasniti rješavanje problema klasifikacije podataka upotrebom genetskog programiranja koje se pokazalo kao dosta precizna metoda iako može dovesti do problema prevelike složenosti kod velike skupine podataka.

U početku ćemo se upoznati sa problemom klasifikacije, njegovim najčešćim primjenama i prihvaćenim metodama rješavanja te osnovnim karakteristikama genetskog programiranja koje ga čine pogodnim za rješavanje ovog problema. Posebnu pažnju ćemo posvetiti algoritmima klasifikacije u obliku regresijskog modela i logičkih pravila. Pojasniti ćemo programski sustav implementiran uz ovaj rad, modele i operatore koje smo koristili kao i detaljne upute za korištenje. Analizirati ćemo rezultate dobivene primjenom navedenog sustava na testne primjere te donijeti zaključke o njegovoj optimiziranosti i kvaliteti dobivenih rješenja.

2. Strojno učenje i problem klasifikacije

U ovom poglavlju ćemo dati kratki pregled strojnog učenja i upoznati se sa njegovim vrstama. Objasniti ćemo problem klasifikacije podataka i proučiti njegove najčešće primjene. Na kraju ćemo opisati neke od najznačajnijih algoritama koji se u današnje vrijeme koriste za rješavanje tog problema kao i njihovu uspješnost.

2.1. Strojno učenje

Strojno učenje je grana računarske znanosti koja se razvila iz prepoznavanja uzoraka i teorije učenja u umjetnoj inteligenciji. Podrazumijeva programiranje računala na način da optimiziraju neki kriterij uspješnosti temeljem podatkovnih primjera ili prethodnog iskustva. Strojno učenje je veliki razvoj doživjelo osamdesetih godina prošlog stoljeća kad se počelo udaljavati od umjetne inteligencije i razvijati kao zasebna grana računarske znanosti.

Strojno učenje možemo podijeliti na nadzirano, nenadzirano i podržano učenje. Nadzirano učenje se koristi ulaznim skupom podataka kako bi učilo i izgradilo model predviđanja. Podaci za učenje se sastoje od primjera koji imaju vektor ulaznih podataka i predviđen izlaz. Algoritmima nadziranog učenja se ti podaci analiziraju i stvara se funkcija koja se može koristiti za mapiranje novih primjera. Ovim postupcima mogu se rješavati dvije vrste problema: klasifikacija i regresija. Kod klasifikacije primjeru pridružujemo klasu kojoj taj primjer pripada, dok kod regresije primjeru pridružujemo neku kontinuiranu vrijednost.

Za razliku od nadziranog učenja, nenadzirano učenje ne koristi podatke za učenje. Kod ove vrste učenja dani su nam podaci bez ciljne vrijednosti i naš je zadatak pronaći pravilnosti među njima. Tipično se koristi kod eksplorativne dubinske analize podataka, u biologiji za grupiranje organizama prema njihovim značajkama, grupiranju sličnih dokumenata, grupiranju DNA-mikropolja i slično. Koriste se grupiranje,

procjena gustoće i smanjenje dimenzionalnosti.

Podržano učenje je učenje strategije na temelju serije izlaza. Tipično se koristi kod igara, robotike i upravljanja te višeagentnim sustavima. Pristup temelji na agentima koji ovisno o kvaliteti postignutih rezultata postižu nagrade. Cilj svakog agenta je maksimizirati dobivenu nagradu i time temelji uspjeh svojih strategija.

2.2. Klasifikacija podataka

Kao što je već objašnjeno u uvodu poglavlja, problemi klasifikacije podataka se rješavaju postupcima nadziranog učenja. Svakom primjeru pridružujemo određenu klasu, odnosno razred kojoj pripada. Primjer možemo definirati kao vektor značajki gdje *n* predstavlja dimneziju vektora, $X = (x_1, x_2, ..., x_n)^T$. Vektor predstavlja ulazni prostor (engl. *input space*) ili prostor primjera (engl. *instance space*). Kod nadziranog učenja unaprijed nam je poznata klasa kojoj pripada primjer iz skupa učenja. Cilj klasifikacije je određivanje nepoznatih klasa ili grupa u podacima, to jest svrstati nove, već neviđene podatke u neki od već unaprijed poznatih klasa. Najjednostavniji je slučaj u kojem se podaci moraju svrstati u samo jedan razred. Kod više razreda, radi se o klasifikaciji s višestrukim oznakama. Ako se ograničimo samo na dvije klase, klasifikator nazivamo binarni klasifikator.

Skup primjera za učenje sastoji se od parova primjera i pripadnih oznaka. Možemo ga prikazati tablično kao:

x_1	x_2	...	y
$x_1^{(1)}$	$x_2^{(1)}$	...	$y^{(1)}$
$x_1^{(2)}$	$x_2^{(2)}$	...	$y^{(1)}$
$\vdots$	$\vdots$		$\vdots$
$x_N^{(1)}$	$x_N^{(1)}$	...	$y^{(N)}$

gdje je N ukupan broj primjera, a *i* indeks primjera. Skup ulaznih oznaka označavamo s $(x^{(i)})_{i=1}^N$, a izlaznih s *y*. Zadaća klasifikacijskog algoritma jest naučiti hipotezu koja određuje pripada li neki primjer određenoj klasi ili ne. Da bi u tome uspio, potrebno je izgraditi klasifikator. Ako klasifikator dobro generalizira ispravno će klasificirati još neviđene podatke. Važno je napomenuti da uspjeh klasifikatora u velikoj mjeri ovisi o o svojstvima podataka koje je potrebno klasificirati. Ne postoji klasifikator koji radi najbolje za sve probleme (pojava koja je poznatija pod imenom

no free lunch theorem). Određivanje primjerenog klasifikatora za problem je dosta složen i težak problem koji nije uvijek moguće riješiti. Učenje hipoteze je loše definiran problem (engl. 1itill-based problem). Primjeri za učenje nisu sami po sebi dovoljni da bi se hipoteza jednoznačno definirala. Svojstvo hipoteze da odredi klase još neviđenih razreda naziva se generalizacija.

Jedan od glavnih problema do kojih može doći je problem šuma. Šum je neželjena anomalija u podacima koja je uzrokovana različitim uzrocima poput: nepreciznosti pri mjerenju značajki, pogreški u označavanju, nejasnih granica klasa i postojanju skrivenih značajki. U načelu, šum nije moguće razdvojiti od pravih podataka. Danas klasifikacija ima mnoge primjene. Koristi se u računalnom vidu, prepoznavanju govora, klasifikaciji dokumenata i slično. Svi ovi i mnogi drugi problemi se rješavaju različitim algoritmima od kojih svi imaju svoje pozitivne i negativne strane, kao i različite uspješnosti. U sljedećem poglavlju ćemo razmotriti neke od najčešćih algoritama koji se koriste u praksi kao i njihove značajke.

2.3. Najznačajniji algoritmi za rješavanje problema klasifikacije

Algoritama za rješavanje problema klasifikacije postoji mnogo. Razlikuju se po svojoj jednostavnosti primjene, točnosti i razumljivosti. Uspješnost algoritama u velikoj mjeri ovisi o problemu kojeg rješavamo. Ponekad nam je važnija brzina klasifikatora od njegove točnosti, a u drugim slučajevima je važnije da klasifikator ima veliku točnost nego brzinu ili jednostavnost. Među algoritme linearne klasifikacije spadaju Fisherova linearna diskriminanta i Naivni Bayes klasifikatori koji su se u početku pokazali dosta učinkovitima za rješavanje složenih problema, no detaljna analiza 2006. godine je pokazala da postoje drugi algoritmi koji su dosta bolji, poput algoritma *random forest*. U tu skupinu još spadaju logistička regresija i *perceptron*. Jedan od češće korištenih algoritama je i metoda potpornih vektora (engl. *Support Vector Machine*) koji se bazira na učenju razdvajanjem funkcija i estimacije funkcije u regresiji. U velikoj upotrebi su i neuronske mreže koje su u širom smislu replika ljudskog mozga kojom se nastoji simulirati postupak učenja. To je skup međusobno povezanih jednostavnih procesnih elemenata koji se nazivaju čvorovi ili neuroni, i čija se funkcionalnost zasniva na biološkom neuronu. Neuronske mreže su vrlo dobre u procjeni nelinearnih odnosa uzoraka, mogu raditi s nejasnim ili manjkavim podacima, mogu raditi s velikim brojem varijabli i prilagođavati se okolini što ih čini pogodnim za probleme klasifikacije. Još neki

od poznatijih metoda i algoritama su kvadratna metoda koja je dizajnirana za situacije kada karakteristike svake grupe imaju normalnu distribuciju te metoda kernel procjene.

Za probleme klasifikacije također se može koristiti i genetsko programiranje koje se zasniva na razvoju stabala odluke. Prednost genetskog programiranja leži u činjenici da može otkriti manje vidljive veze između podataka. Važna primjena je postignuta u otkrivanju gena. Više o tome će biti objašnjeno u sljedećih nekoliko poglavlja.

3. Genetsko programiranje

U ovom poglavlju ćemo se detaljnije upoznati sa genetskim programiranjem i vrstama koje se danas koriste. Objasniti ćemo probleme za čije je rješavanje pogodno genetsko programiranje kao i načini do kojih algoritmi dolaze do rješenja.

Genetsko programiranje je optimizacijska tehnika koja spada pod skupinu evolucijskih algoritama. Radi na principu simulacije evolucije, bolja rješenja preživljavaju i svoja svojstva prenose u sljedeće generacije dok loša izumiru. Kao i u prirodi, koriste se mehanizmi selekcije, križanja i mutacije čime možemo doći do raznolikog broja jedinki i uspješno se približiti dovoljno dobrom rješenju promatranog problema. Razvoj genetskog programiranja možemo pratiti od pedesetih godina prošlog stoljeća, no tek šezdesetih i sedamdesetih godina su postali šire prepoznati kao pogodna tehnika za rješavanje optimizacijskih problema. Većem napretku je pridonio John R. Koza koji je GP primjenio na velikom broju optimizacijskih problema i problema pretraživanja različite složenosti. Devedesetih se genetsko programiranje uglavnom koristilo za relativno jednostavne probleme budući da su izračuni dosta računalno intezivni čemu računala tog doba nisu bila dorasla. U zadnjim godinama, zbog velikog napretka u razvoju procesora, genetsko programiranje se koristi za sve širi spektar problema poput sortiranja, pretraživanja, elektroničkog dizajna i mnogih drugih.

Genetsko programiranje se često uspoređuje sa genetskim algoritmima. Međutim, postoje određene razlike. Kod genetskog programiranja rješenje je prikazano računalnim programom. Rješenje je onaj program koji daje rješenje zadanog problema u u najkraćem roku. Kod genetskih algoritama postoje različite reprezentacije rješenja ovisno o problemu koji rješavamo. Rješenje se može predstavljati brojkom, nizom bitova ili brojeva i drugima. Još jedna od većih razlika je duljina rješenja. Kod genetskog algoritma duljina rješenja se kroz evaluaciju ne mjenja, dok kod GP-a stabla često mjenjaju dubinu, veličinu i širinu. To često dovodi do problema prekomjernog rasta zbog čega je potrebno implementirati određena ograničenja duljine rješenja, poput dubine stabla ili broja čvorova.

U sljedećim potpoglavljima ćemo se detaljnije upoznati o načinu na koji genetsko

programiranje radi.

3.1. Stvaranje početne populacije

Algoritam započinje nultom generacijom koja se stvara na temelju slučajnog odabira. Ova generacija nema nikakva optimizacijska svojstva, služi samo kao temelj za stvaranje i razvoj boljih jedinki u sljedećim generacijama. Postoje različiti načini kojima gradimo početnu generaciju, no najčešće korištene metode su *Full* i *Grow*.

3.1.1. Full metoda

Stvaranje stabala kod *Full* metode započinje slučajno izabranom funkcijom koja predstavlja korijen. Potom rekurzivno stvaramo sljedeće razine stabla dok ne dođemo do maksimalno predviđene dubine. Tijekom ovog procesa čvorove biramo samo iz skupa operatora. Na posljednjoj razini odabiremo čvorove isključivo iz skupa završnih znakova.

3.1.2. Grow metoda

Ova metoda je slična prijašnjoj s razlikom da u svakom trenutku stvaranja čvorove možemo birati i iz skupa završnih znakova. Gradnjom stabala na ovaj način nećemo uvijek doći do maksimalne dubine zbog čega su stabla izgrađena ovom metodom dosta raznolikija i manje složena.

3.1.3. Ramped half-and-half metoda

Najčešća metoda korištena za stvaranje nulte popoulacije je metoda *ramped half-and-half*. Polovica jedinki se stvara *full* metodom, a druga polovica *grow*. Određena je samo maksimalna dubina stabala, a jedinke se stvaraju za svaku dubinu, sve do maksimalne.

Prednost ove metode je što stvara jedinke s dobrom raspodjelom po veličini i strukturi.

3.2. Procjena dobrote jedinke

Procjena dobrote jedinke uvelike ovisi o problemu kojim se bavimo. Predstavljena je *fitnes* funkcijom ili funkcijom dobrote. Ovisno o veličinama koje dobijemo primje-

nom fitnes funkcije možemo utvrditi kvalitetu pojedinog rješenja. Pravilno implementirana funkcija dobrote je važna jer o njoj u velikoj mjeri ovisi koja će rješenja ostati u razmatranju, a koja ne. Dobrotu neke jedinke možemo definirati kao mjeru kvalitete tog rješenja u zadanom prostoru rješenja. Primjerice, kod traženja maksimuma neke funkcije, dobrotu možemo prikazati kao vrijednost koju ta funkcija ima za pojedinu jedinku. Kod problema stvaranja rasporeda, dobrotu možemo procjeniti kao postotak predavanja koja se ne poklapaju, vremenskom intervalu koji studenti imaju između uzastopnih predavanja i slično.

Definiranje funkcije dobrote je jedan od ključnih problema genetskog programiranja jer je potrebno da bude što preciznija, a što jednostavnija budući da je njeno evaluiranje potrebno obavljati u svakoj generaciji na cjelokupnom skupu jedinki.

3.3. Genetski operatori

3.3.1. Selekcija

Jedan od centralnih mehanizama genetskog programiranja je selekcija. Ona omogućava boljim jedinkama da prenesu svoje "gene" u sljedeće generacije algoritma. Dobrota jedinke se zasniva na fitnes funkciji ili funkciji dobrote koju smo već opisali. Operatori selekcije i njihov utjecaj se mogu okarakterizirati kroz nekoliko parametara:

- vrijeme preuzimanja,
- selekcijski intezitet,
- momenti razdiobe dobrote

Vrijeme preuzimanja je vrijeme potrebno operatoru selekcije da generira populaciju u kojoj se nalazi samo najbolje rješenje. To je vrijeme od prve generacije pa do nastanka generacije koja je popunjena samo jedinkama najboljeg rješenja ako koristimo samo mehanizam selekcije. Ako je mehanizam selekcije pravilan, u svakoj generaciji bi trebali imati bolja rješenja i sve veći broj kopija najboljeg rješenja.

Selekcijski intezitet je povećanje srednje dobrote populacije prije i nakon primjene operatora selekcije, podijeljenog sa standardnom devijacijom populacije:

$$I = \frac{\bar{f}_s - \bar{f}}{\sigma}$$

$\bar{f}$ predstavlja prosječnu dobrotu rješenja u populaciji roditelja, $\bar{f}_s$ predstavlja prosječnu dobrotu rješenja u populaciji djece, dok je σ standardno odstupanje u populaciji roditelja.

Moment razdiobe dobrote populacije se računa prema izrazu:

$$\mu_r = \frac{1}{n}\sum_{i=1}^{n}(f_i - \bar{f})^r$$

gdje je f_i dobrota rješenja, a $\bar{f}$ srednja vrijednost dobrote za čitavu populaciju.

U nastavku ćemo se upoznati s nekim od često korištenih vrsta selekcije.

Proporcionalna selekcija

Proporcionalnu selekciju možemo slikovito prikazati kao kotač ruleta gdje je svakom rješenju u populaciji pridružen isječak čija je površina proporcionalna dobroti rješenja. Tako najbolja jedinka ima najveći isječak kotača i najveću šansu za odabirom, dok će kod onih manje dobrih biti obrnuto.

Turnirska selekcija

Kod turnirske selekcije iz populacije izvlačimo slučajni uzorak od *s* rješenja te odabiremo ono rješenje iz tog uzorka koje ima najveću dobrotu.

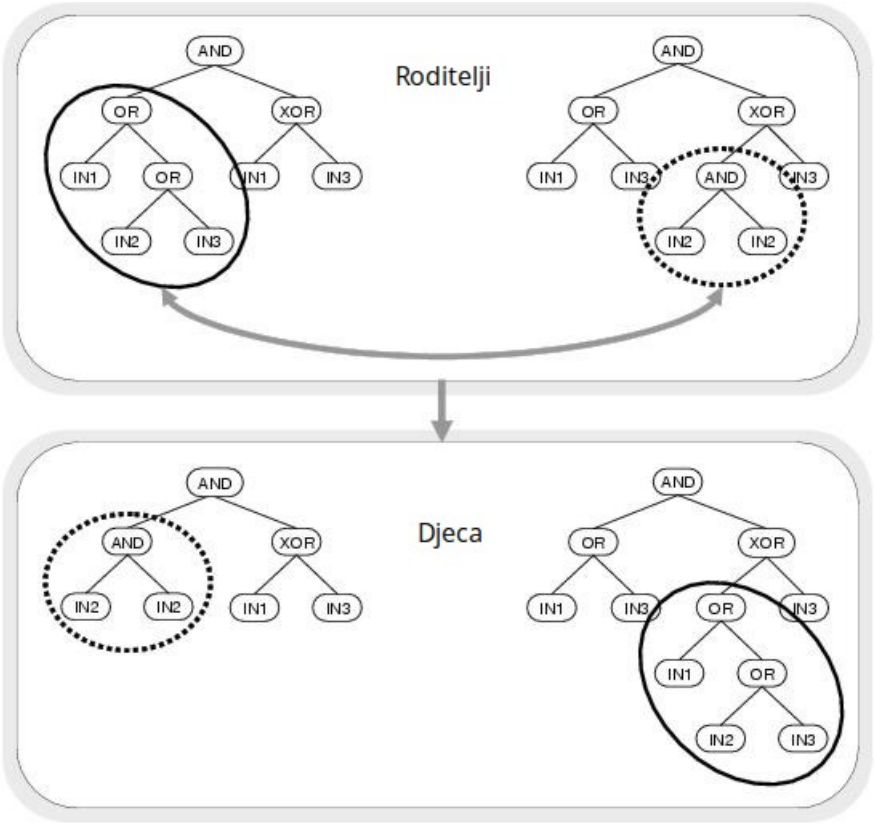
Turnirska selekcija koja u populaciji od *n* jedinki izvlači uzorak od njih *s* naziva se *s*-turnirska selekcija.

3.3.2. Križanje

Križanje je analogno biološkoj spolnoj reprodukciji. Križanjem dolazi do rekombiniranja genetskog materijala dvaju roditelja. Rezultat su dvije jedinke djece koje od svakog roditelja nasljeđuju dio genetskog materijala i koje će vjerojatno imati bolju dobrotu. U slučaju da se to ne dogodi loša rješenja će brzo "odumrijeti" mehanizmima selekcije koji će ih rjeđe birati za prijenos u sljedeće generacije. Ovisno o načinu prikaza jedinke postoje različite vrste križanja koje između ostaloga uključuju uniformno križanje koje se koristi kod jedinki predstavljenih kao niz bitova, aritmetičko, diskretno linearno križanje i drugi.

3.3.3. Mutacija

Mutacija omogućava genetsku raznolikost jedinki čime se mogu dobiti bolja ili lošija rješenja. U generaciji se bira udio jedinki koji će sudjelovati u mutaciji ili vjerojatnost odabira mutacije za svaku pojedinu jedinku. Kod jedinki prikazanih stablima mutacija

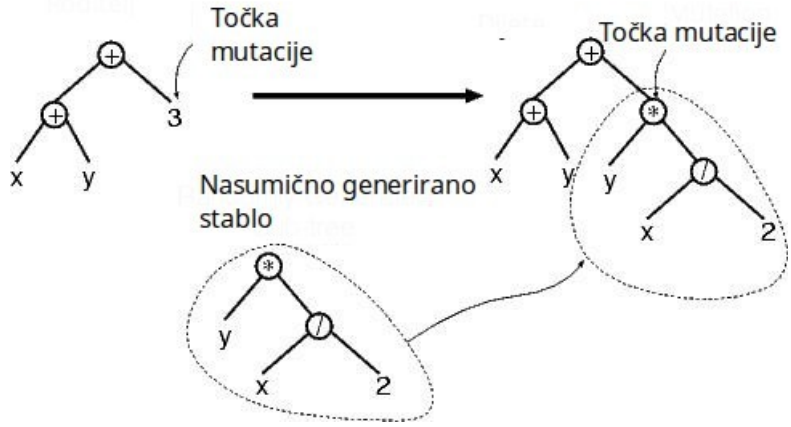


Slika 3.1: Primjer križanja

se obavlja tako da se nasumično odabere čvor koji se briše zajedno sa svojim podstablima. Na njegovom mjestu se generira novo podstablo stvoreno slučajnim odabirom. Pri tome je potrebno paziti da ne dođe do prekoračenja postavljenih ograničenja poput dubine stabla ili broja čvorova.

3.4. Problemi genetskog programiranja

Kao što smo već napomenuli, kod genetskog programiranja možemo naići i na određene probleme. To je prije svega problem prebrzog rasta jedinki (engl. *bloat*). Ako je jedinka prikazana u obliku stabla, problem se očituje kao nekontrolirani rast veličine stabla, to jest broja čvorova i dubine bez poboljšanja dobrote. To može dovesti do dugog trajanja evaluacije većih jedinki i nerazumljivosti i nečitljivosti rješenja. Predloženi su različiti mehanizmi rješavanja tog problema koji između ostalog uključuju kažnjavanje prevelikih jedinki, odbacivanje onih jedinki koji prelaze dopuštena ograničenja te njihova zamjena novim ili podrezivanje stabala (engl. *tree pruning*).



Slika 3.2: Mutacija jedinke prestavljene stablom

3.5. Primjena genetskog programiranja na problem klasifikacije

U ovom potpoglavlju ćemo se detaljnije upoznati sa načinom na koji se genetsko programiranje koristi za rješavanje problema klasifikacije podataka. Klasifikacija je raširen problem s kojim se i nesvjesno suočavamo svaki dan. Na primjer, prepoznavanje lice pojedinaca koje smo prije upoznali, vrsta životinja, biljaka i hrane i drugih objekata iz okoline. Koristi se i u složenim znanstvenim domenama poput medicine gdje može pomoći u dijagnozi pacijenata na osnovu povijesti bolesti drugih pacijenata sa sličnim simptomima.

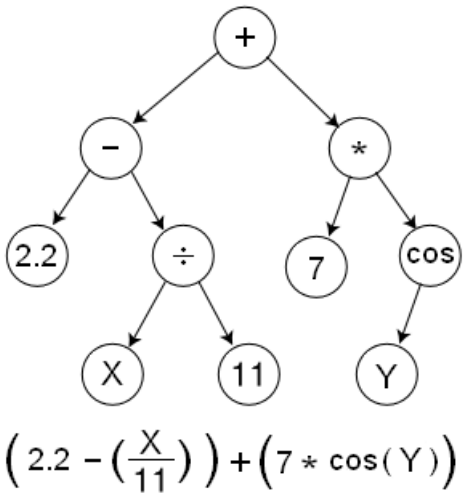
S obzirom na raširenost ovog problema važno je razviti računalne mehanizme sposobne rješavanja problema klasifikacije u slučajevima kada je to za ljude presloženo i vremenski zahtjevno. Generalni oblik klasificiranja podrazumjeva prepoznavanje klasa kojima određeni primjerci problema pripadaju na osnovu atributa ili svojstava testnih primjeraka. Na osnovu testnih primjeraka možemo razviti model koji će predviđati klasu kojoj pripadaju novi, još nesvrstani primjerci.

Kroz povijest razvoja klasifikacije koristili su se različiti algoritmi i načini rješavanja tog problema. Genetsko programiranje je relativno novo i brzo razvijajuće područje koje se u široj primjeni našlo tek u zadnjih nekoliko desetljeća.

Model na kojem ćemo se zadržati u ovom radu je reprezentacija stablima odluke koje koristimo za klasifikaciju numeričkim izrazima. U kombinaciji sa aritmetičkim operacijama koristit ćemo se i logičke izraze poput *if-then-else* operacija koje se često koriste u računalnim programima. Ovakav način prikaza rješenja se pokazao relativno uspješnim u praktičnim primjenama poput klasifikacije rukopisa, bolesti, prepoznava-

nja objekata i slično.

Primjer stabla i pripadajuće funkcije koja ovisi o ulaznim parametrima X i Y prikazan je na slici 3.3



Slika 3.3: Stablo odluke

Klasifikator numeričkih izraza se definira kao bilo koja funkcija koja vraća numeričku vrijednost. Funkcija se sastoji od matematičkih operacija, uključujući aritmetičke operacije i druge poput sinusa, kosinusa ili logaritma. Kao što smo prije napomenuli, funkcije mogu sadržavati i operatore usporedbe kao ako-onda-ili izraz. Klasifikator numeričkih izraza provodi klasifikaciju pomoću varijabli koje su mu predane kao ulaz i na izlazu daje numeričku vrijednost. Ta numerička vrijednost se onda mora interpetirati u jednu od mogućih klasa za zadani problem.

Važnost će biti na preciznosti modela koji nastaju genetskim programiranjem. Iako postoje drugi faktori o kojima može ovisiti kvaliteta promatranog rješenja poput veličine programa, razumljivosti, vremenskoj izvedbi i slično, njih ćemo svrstati u drugi plan. Na osnovu toga, kvalitetna rješenja će biti ona koja imaju najveći preciznost izvođenja klasifikacije, bez obzira na ostale promatrane uvjete.

Cilj je za zadane primjere proizvesti program koji će što preciznije klasificirati primjere i u velikom postotku predviđati pripadnost primjeraka problema zadanim klasama.

4. Programsko ostvarenje

U ovom poglavlju ćemo detaljnije opisati programsko ostvarenje i opisati na koji smo način rješavali problem klasifikacije podataka upotrebom genetskog programiranja. Upoznati ćemo se s radnim okruženjem ECF-a, opisati genetske operatore koje smo koristili i objasniti prikaz dobivenog rješenja.

4.1. Radno okruženje ECF

ECF (engl. *Evolutionary Computation Framework*) je radno okruženje za rješavanje problema evolucijskog računanja koje se razvija na Fakultetu elektrotehnike i računarstva u Zagrebu pod vodstvom izv. prof. dr. sc. Domagoja Jakobovića. Napisano je u programskom jeziku C++.

Za korištenje ECF-a možemo kombinirati i parametrizirati sljedeće komponente:

- Genotype
- Algorithm
- Evolutionary system

Genotype je osnovna komponenta ECF-a. Svaka jedinka u populaciji ima jednog ili više genotipa koje je najjednostavnije definirati u konfiguracijskoj datoteci. Genotip je jedina komponenta koju je potrebno definirati, sve ostale mogu koristiti pretpostavljene vrijednosti. Svaki genotip ima svoju strukturu i genetske operatore križanja i mutacije. U ovom radu ćemo koristiti izvedeni razred **Tree** koji predstavlja stablasti kromosom genetskog programiranja.

Algorithm predstavlja algoritam koji koristimo. Željeni algoritam i njegovi parametri se mogu definirati u konfiguracijskoj datoteci.

Evolutionary system je kontekst u kojem postoje genotip i algoritam. Kontekst se brine za sve ostalo: veličinu populacije, uvjete završavanja i slično.

Parametri se zadaju u konfiguracijskoj datoteci XML formata (engl. *Extensible Markup Language*), a dohvaćaju se prilikom inicijaliziranja ECF-a. Primjer konfigu-

racijske datoteke dan je na slici 4.1

```
<ECF>
    <Algorithm>
        <SteadyStateTournament>
            <Entry key="tsize">3</Entry>
        </SteadyStateTournament>
    </Algorithm>
    <Genotype>
        <FloatingPoint>
            <Entry key="lbound">-50</Entry>
            <Entry key="ubound">50</Entry>
            <Entry key="dimension">3</Entry>
        </FloatingPoint>
    </Genotype>
    <Registry>
        <Entry key="function">6</Entry>
        <Entry key="population.size">500</Entry>
        <Entry key="mutation.indprob">0.3</Entry>
        <Entry key="term.maxgen">500</Entry>
        <Entry key="term.fitnessval">1e-12</Entry>
        <Entry key="log.frequency">10</Entry>
        <Entry key="log.level">3</Entry>
    </Registry>
</ECF>
```

Slika 4.1: Primjer konfiguracijske datoteke

4.2. Korišteni klasifikatori

U ovom radu ćemo implementirati dva različita klasifikatora genetskog programiranja. To su :

- Regresijski GP (klasifikacijski aritmetički izrazi)
- Regresijski GP uz glasanje - posebno stablo za svaku klasu

Regresijski GP s jednim stablom uspoređuje numeričke vrijednosti dobivene evaluacijom stabla s intervalima definiranim za svaku klasu. Razred kojem primjer pripada je onaj u čijem se intervalu nalazi dobivena numerička vrijednost. Funkcijom dobrote potom određujemo uspješnost predviđanja klasa svake jedinke.

Kod regresijskog GP-a uz glasanje svaka klasa ima definirano svoje stablo. Tijekom evaluacije računamo izlaznu numeričku vrijednost za svako od stabala. Odlučujuće je ono stablo koje daje najmanju apsolutnu vrijednost.

4.3. Funkcije čvorova

Stabla odluke se sastoje od skupa funkcijskih znakova i skupa završnih znakova. U skup završnih znakova spadaju atributi klasifikacijskog problema. Ako je ukupan broj atributa jednak n varijable tada imaju oznake od x_0 do x_{n-1}. Prilikom evaluacije za

vrijednosti varijabli iz skupa završnih znakova uzimaju se odgovarajuće vrijednosti iz skupa podataka definiranih za svaki primjer klasifikacije. U skup funkcijskih znakova spadaju:

- Aritmetički operatori
- Sinus, kosinus
- Max, min
- Sqrt, Log
- IfPositive
- IfLessThanEq

U nastavku ćemo se pobliže upoznati s njima.

U aritmetičke operatore koje koristimo spadaju zbrajanje (+), oduzimanje (-), množenje (*) i dijeljenje (/). Pored njih, koristit ćemo i funkcije sinusa i kosinusa.

Ostale korištene funkcije je potrebno implementirati unutar radnog okruženja ECF-a. Pri tome im je potrebno definirati pridružena svojstva **nArguments_** koje označava broj djece koje će čvor s navedenom funkcijom imati te **name_** koje označava ime navedene funkcije. U svakoj definiranoj funkciji je potrebno implementirati metodu **execute** u kojoj vrijednosti čvorova djece dohvaćamo pomoću metode **getNextArgument**. U svaku od funkcija čvorova roditelja se šalje vrijednost samo jednog djeteta dok se sve ostale čvorove preskače i zanemaruje.

Max i *Min* funkcije definiraju čvorove koji imaju dvoje djece. *Max* funkcija vraća vrijednost onog djeteta koje ima veću vrijednost, dok *Min* funkcija vraća manju od vrijednosti svoje djece.

Funkcija *Sqrt* definira čvor koji ima jedno dijete te vraća korijen apsolutne vrijednosti tog djeteta.

Kod funkcije *Log* je potrebno pripaziti na vrijednost koju joj predajemo. Budući da je logaritamska funkcija definirana samo za pozitivne vrijednosti veće od nule, u slučaju da je vrijednost djeteta manja ili jednaka nuli funkcija vraća logaritam od jedan, to jest nula.

IfPositive definira čvor koji ima troje djece. U slučaju da je vrijednost prvog djeteta veća od 0 vraća vrijednost drugog djeteta, a inače vraća vrijednost trećeg.

IfLessThanEq definira čvor koji ima četvero djece. U slučaju da je vrijednost prvog djeteta manja ili jednaka vrijednosti drugog vraća vrijednost trećeg djeteta, a u suprotnom slučaju vrijednost četvrtog djeteta.

Uvođenje logičkih izraza pored onih aritmetičkih je važno jer omogućava veću raznolikost rješenja i pomaže u preciznijoj evoluciji stabala.

4.4. Funkcije dobrote

Funkcija dobrote koju ćemo koristiti u treniranju klasifikatora je *F1 score*. Nastala je iz matrice (textitconfusion matrix koja omogućava vizualizaciju uspješnosti algoritma, obično nadziranog učenja kod problema umjetne inteligencije među koje spada i problem klasifikacije.

Sastoji se od dvije dimenzije ("stvarno" i "predviđeno") i identičnog skupa klasa u obe dimenzije.

Za primjer možemo uzeti klasifikator koji razvrstava pse, mačke i zečeve u njihove pripadajuće klase. Ako uzmemo skup od 27 životinja - 8 mačaka, 6 pasa i 13 zečeva, rezultati nakon klasifikacije bi mogli izgledati ovako:

		Predviđeno		
		Mačka	Pas	Zec
Stvarno	Mačka	5	3	0
	Pas	2	3	1
	Zec	0	2	11

Od osam mačaka, klasifikator je za tri predvidio da su psi, a od šest pasa za jednog je predvidio da je zec, a za dva da su mačke. Svi pogotci su prikazani dijagonalom pa je lako vidjeti sve pogreške.

Matrica iz koje ćemo dobiti funkciju dobrote ima dva retka i dva stupca sa sljedećim mogućim vrijednostima:

- *false positive (FP)* - Svrstavanje pogrešnog primjerka u klasu
- *false negative (FN)* - Primjerci klase koji su svrstani u neku drugu klasu
- *true positive (TP)* - Pogodak
- *true negative (TN)* - Točno neprihvaćanje primjerka u klasu

Za navedeni primjer i klasu Mačka, dobili bi sljedeću matricu.

5 TP (točno svrstane mačke)	3 FN (mačke svrstane u pse)
2 FP (psi svrstani u mačke)	17 TN (sve ne-mačke svrstane u druge klase)

Naša funkcija dobrote će tada imati oblik:

$$F1 = \frac{2 * TP}{2 * TP + FP + FN}$$

Moguće vrijednosti su iz intervala $[0, 1]$ gdje veći broj predstavlja veću dobrotu rješenja.

4.5. Parametri i konfiguracijske datoteke

Parametre zadajemo u konfiguracijskoj datoteci XML formata koju je potrebno predati prilikom inicijalizaciranja ECF-a. Tu možemo definirati mehanizme križanja, veličinu populacije, vjerojatnost mutacije, minimalnu i maksimalnu dubinu stabla i druge.

U slučaju ne navođenja nekog od parametara, uvode se sljedeće pretpostavljene vrijednosti:

- Veličina populacije: 100
- Vjerojatnost mutacije: 0.3
- Prekid: 50 generacija bez napretka najbolje jedinke

Parametri genotipa koji će nama biti od značaja su *functionset* pomoću kojeg definiramo skup funkcijskih znakova te *terminalset* koji predstavlja skup završnih znakova. Skup završnih znakova ćemo u svim slučajevima definirati oznakama od x_0 do x_{n-1} gdje n predstavlja broj atributa problema.

Pored navedenih možemo definirati i parametar *classesNum* koji predstavlja broj klasa koji koristimo u klasificiranju. U slučaju ne navođenja navedenog parametra koristi se pretpostavljena vrijednost dva.

Primjer parametarske datoteke gdje možemo vidjeti navedene atribute nalazi se na slici 4.2.

```
<ECF>
    <Genotype>
        <Tree>
            <Entry key="maxdepth">4</Entry>
            <Entry key="mindepth">1</Entry>
            <Entry key="functionset">sqrt log iflte max min ifpos sin cos + - * /</Entry>
            <Entry key="terminalset">X0 X1 X2 X3</Entry>
        </Tree>
    </Genotype>
    <Registry>
        <Entry key="classesNum">2</Entry>
    </Registry>
</ECF>
```

Slika 4.2: Primjer konfiguracijske datoteke

Kod definiranja intervala klasa koristimo datoteku definiranu parametrom *classesfile*. U slučaju ne navođenja parametra koristi se pretpostavljeni naziv *classes.txt*. Po-

moću ove datoteke definiramo sve klase koje koristimo kao i intervale kojima pojedina klasa pripada. Ako datoteka ne postoji, koriste se pretpostavljene klase s oznakama od 0 do $n-1$ gdje n predstavlja broj klasa definiranih parametrom *classesNum*. Za intervale se koriste negativna i pozitivna potencija broja dva, gdje eksponent intervala odgovara numeričkoj oznaci klase. U klasi $n-1$ definiramo dva intervala koja pokrivaju sve preostale realne vrijednosti.

Naprimjer, ako smo definirali postojanje tri klase bez datoteke sa konfiguracijom dobit ćemo sljedeće vrijednosti intervala:

- Klasa 0: $[-1, 1]$
- Klasa 1: $[-2, -1] \cup [1, 2]$
- Klasa 2: $(-\infty, -2] \cup [2, +\infty)$

Primjer strukture datoteke za definiranje razreda nalazi se na slici 4.3.

```
a 1:4 6:12
b :1 4.5:6
c 4:4.5 12:
```

Slika 4.3: Primjer definiranja klasa

Na početku retka definirano ime klase, a intervale odvajamo dvotočkama. Beskonačne vrijednosti označavamo ne navođenjem vrijednosti s odgovarajuće strane intervala.

Za primjer naveden na slici definirane su tri klase i intervali:

- Klasa a: $[1, 4] \cup [6, 12]$
- Klasa b: $(-\infty, 1] \cup [4.5, 6]$
- Klasa c: $[4, 4.5] \cup [12, +\infty)$

Primjere za učenje u datoteci definiranoj parametrom *inputfile* s pretpostavljenom vrijednosti *learning.txt* u kojoj navodimo atribute svakog primjera i razred kojem pripada.

Svaki primjer je napisan u novom retku, a atributi su odvojeni zarezima. Kod primjera sa n atributa imat ćemo $n+1$ zapisa u retku gdje posljednji predstavlja klasu.

Na slici 4.3 prikazana je datoteka za učenje čiji primjeri imaju četiri atributa i dva razreda.

Datoteka *test.txt* je pretpostavljane vrijednost parametra *testfile* u kojoj se nalazi primjeri za testiranje ima identičnu strukturu.

Detaljnije objašnjenje o atributima i šta točno znače za navedene datoteke ćemo opisati u sljedećem poglavlju.

```
2,1,0,0,1
0,0,1,1,0
2,2,0,0,1
1,2,0,1,1
0,1,1,0,0
1,0,0,0,1
0,2,0,0,1
1,1,1,1,1
2,2,0,1,0
2,1,1,0,1
0,1,0,1,1
1,0,1,0,1
2,1,1,1,0
0,0,1,0,0
```

Slika 4.4: Datoteka sa primjerima za učenje

Ime datoteke u koju se zapisuju rezultati određeno je parametrom *resultsfile* koji ima pretpostavljenu vrijednost *results.txt*.

U datoteci se za svaku generaciju zapisuje broj generacije i najveće vrijednosti dobrote skupa za učenje i testiranje. Vrijednosti su odvojene zarezima.

5. Primjena i analiza rezultata

Implementirane klasifikatore ćemo primjeniti na dva primjera problema klasifikacije. John Koza, koji je prvi predstavio ovaj način rješavanja problema klasifikacije, u svom radu je uzeo jednostavan skup podataka od četiri atributa koji određuju jeli dan pogodan za igranje tenisa ili ne. Klasifikatore ćemo upotrijebiti i na podacima koji se jako često koriste u svrhu testiranja klasifikatora gdje je cilj odrediti pripadnost porodice cvijeta iris.

5.1. Klasifikacija vremenskih prilika

Prvi problem koji ćemo pokušati riješiti je predstavio John Koza. U svom radu je uzeo jednostavan skup podataka od četiri atributa (vrijeme, temperatura, vlažnost i vjetrovitost) gdje svaki od atributa može poprimiti nekoliko različitih vrijednosti. Svaki uzorak se svrstava u jedan od dva moguća razreda (pozitivno ili negativno) koja predstavljaju mogućnost igranja tenisa.

Vrijednosti svakog od atributa ćemo preslikati u cijele brojeve kako je prikazano sljedećom tablicom.

Atribut	Vrijednost	Oznaka
Vrijeme	Sunčano	0
	Oblačno	1
	Kišovito	2
Temperatura	Vruće	0
	Umjerena	1
	Hladno	2
Vlažnost	Visoka	0
	Niska	1
Vjetrovitost	Ne	0
	Da	1

Uzorke ovog problema smo već pokazali u prethodnom poglavlju kao primjer datoteke sa primjerima za učenje.

5.2. *Iris flower data set*

Iris flower data set je vjerojatno najpoznatiji skup podataka koji se koristi kod klasifikacije podataka. Predstavio ga je Ronald Fisher 1936. godine u svom radu *The use of multiple measurements in taxonomic problems*.

Podaci se sastoje od tri klase po 50 primjeraka gdje je svaki primjerak definiran sa četiri atributa. Klase predstavljaju tri porodice cvijeta iris - porodice su *iris setosa, iris versicolor, iris virginica*.

Atributi koji se koriste su:

- dužina lapa
- širina lapa
- dužina latica
- širina latica

Sve vrijednosti su predstavljene u centimetrima.

Nekoliko uzoraka je prikazano sljedećom tablicom.

Dužina lapa	Širina lapa	Dužina latica	Širina latica	Porodica	Oznaka
5.1	3.5	1.4	0.2	*I. setoosa*	0
4.9	3.0	1.4	0.2	*I. setosa*	0
7.0	3.2	4.7	1.4	*I. versicolor*	1
6.4	3.2	4.5	1.5	*I. versicolor*	1
6.3	3.3	6.0	2.5	*I. virginica*	2
5.8	2.7	5.1	1.9	*I. virginica*	2

6. Zaključak

Zaključak.

Primjena genetskog programiranja na problem klasifikacije podataka

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Classification of data with genetic programming

Abstract

Abstract.

Keywords: Keywords.