SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 491

PRIMJENA GENSKI EKSPRESIVNOG PROGRAMIRANJA NA PROBLEMU SIMBOLIČKE REGRESIJE

Tomislav Krog

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 491

PRIMJENA GENSKI EKSPRESIVNOG PROGRAMIRANJA NA PROBLEMU SIMBOLIČKE REGRESIJE

Tomislav Krog

Zagreb, lipanj 2022.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 11. ožujka 2022.

ZAVRŠNI ZADATAK br. 491

Pristupnik:	**Tomislav Krog (0036519636)**
Studij:	Elektrotehnika i informacijska tehnologija i Računarstvo
Modul:	Računarstvo
Mentor:	doc. dr. sc. Marko Đurasević

Zadatak: **Primjena genski ekspresivnog programiranja na problemu simboličke regresije**

Opis zadatka:

Proučiti problem simboličke regresije. Proučiti metodu genetski ekspresivnog programiranja te njenu primjenu za problem simboličke regresije. Oblikovati i ostvariti programski sustav za rješavanje proizvoljnog problema simboličke regresije korištenjem genetski ekspresivnog programiranja. Odabrati skup problema za ispitivanje razvijenog sustava te ocijeniti njegovu uspješnost. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Rok za predaju rada: 10. lipnja 2022.

Sadržaj

Uvod 1

1. Evolucijski algoritmi 2

1.1. Općenito o evolucijskim algoritmima 2

1.2. Genski ekspresivno programiranje 3

1.2.1. Reprezentacija kromosoma u GEP-u 4

1.2.2. Genotip – fenotip 5

1.2.3. K – ekspresije i geni 6

1.2.4. Multigeni kromosomi 7

1.2.5. GEP algoritam 8

2. Strojno učenje 12

2.1. Općenito o strojnom učenju 12

2.2. Nadzirano učenje 14

2.3. Regresijska analiza 15

2.4. Simbolička regresija 16

3. Statističke mjere 17

3.1. Srednja kvadratna pogreška (MSE) 17

3.2. R – kvadrat 18

3.3. Točnost 19

4. Implementacija i rezultati 20

4.1. Skup podataka 20

4.2. Programska implementacija 21

4.3. Rezultati 26

Zaključak 28

Literatura 29

Sažetak 30

Summary.. 31

Skraćenice.. 32

Uvod

Simbolička regresija zaboravljena je metoda strojnog učenja. Tužna istina je da ova metoda nikada nije stekla popularnost, tj. nije postala mainstream [1]. U akademskom smislu, istraživanje o popularnijim temama kao što su neuronske mreže mnogo je lakše shvatljivo. Genski ekspresivno programiranje je potpuni sustav genotip/fenotip koji razvija računalne programe kodirane u linearnim kromosomima fiksne duljine. Strukturna organizacija linearnih kromosoma omogućuje neograničen i plodan rad važnih genetskih operatora kao što su mutacija, transpozicija i rekombinacija jer genska ekspresija (ostvarenje informacije iz gena radi proizvodnje genskog proizvoda) uvijek rezultira valjanim programima.

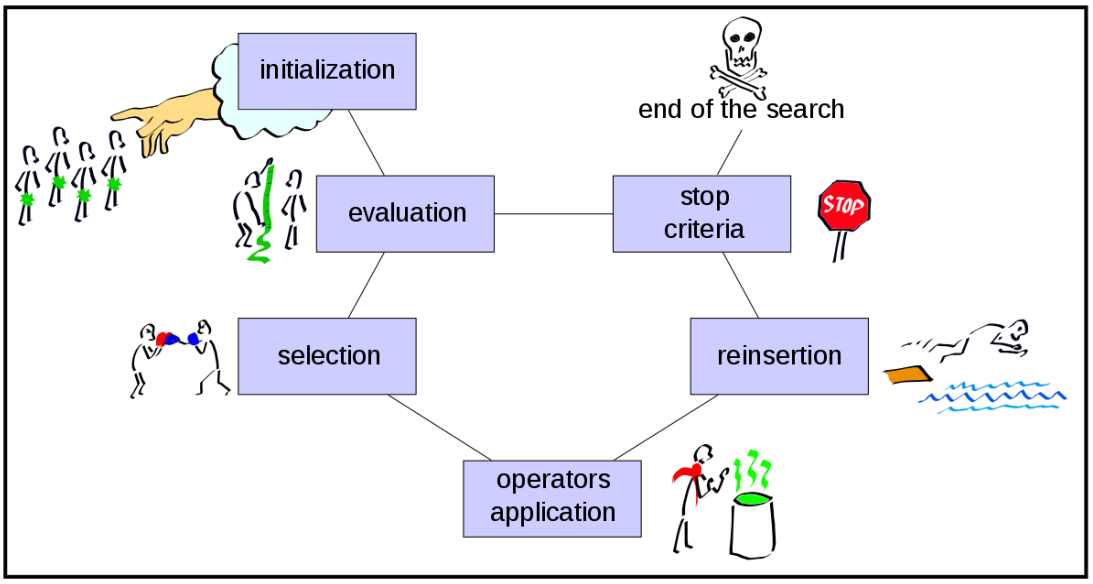
Cilj ovog rada je koristeći genski ekspresivno programiranje riješiti problem te „zaboravljene“ metode – simboličke regresije.

U prvom dijelu rada opisuju se općenito evolucijski algoritmi, dok se naglasak stavlja na algoritam koji je i tema ovog rada – genski ekspresivno programiranje. U drugom dijelu naglasak je na simboličkoj regresiji, ali se hijerarhijski uz nju opisuju regresijska analiza, nadzirano učenje te općenito strojno učenje. Treći dio rada bavi se opisivanjem statističkih mjera koje nam pokazuju kvalitetu naše implementacije, a u zadnjem dijelu opisan je skup podataka na kojem se obavlja testiranje, dio implementacije te su ukratko opisani dobiveni rezultati.

1. Evolucijski algoritmi

1.1. Općenito o evolucijskim algoritmima

Evolucijski algoritmi (EA) učinkovite su heurističke metode pretraživanja temeljene na Darwinovoj evoluciji sa snažnim karakteristikama robusnosti i fleksibilnosti za dohvaćanje globalnih rješenja složenih problema optimizacije [2]. Korištenjem EA-a vjerojatnost pronalaženja optimuma u ranoj fazi procesa optimizacije vrlo je visoka [2]. Evolucijski algoritam glavni je predmet interesa u evolucijskom računarstvu. Postoji problem koji treba riješiti, a rješenje je zamišljeno da leži negdje u prostoru mogućih rješenja – prostoru pretraživanja. U nastavku Slika 1.1 prikazuje generalnu shemu evolucijskog algoritma.



Slika 1.1 Prikaz generalne sheme evolucijskog algoritma (EA) [3]

1.2. Genski ekspresivno programiranje

Genski ekspresivno programiranje (GEP) evolucijski je algoritam koji stvara računalne programe ili modele [4]. Ti računalni programi kompleksne su strukture stabla koje uče i prilagođavaju se promjenom svoje veličine, oblika i kompozicije. GEP programi kodirani su u vrlo jednostavne linearne strukture – kromosome fiksnih duljina [4].

Računalni programi koje GEP stvara mogu imati različite forme: oni mogu biti konvencionalni matematički modeli, stabla odluke, neuronske mreže, modeli logističke regresije kao i drugih regresija, nelinearni klasifikatori, složene polinomske strukture itd. [4].

GEP je usko povezan s genetskim algoritmom (GA) i genetskim programiranjem (GP) [4]. Najznačajnija razlika između GEP-a i GP-a je u tome što GEP usvaja linearnu reprezentaiju računalnih programa fiksne duljine, koja se kasnije može prevesti u stablo izraza. Nasuprot tome, GP obično direktno koristi prikaz sintaksnih stabala promjenjive veličine.

Mnoge studije tvrde da je GEP superioran u odnosu na tradicionalni GP u učinkovitosti i djelotvornosti [5]. Kao i svi drugi evolucijski algoritmi, GEP uzima početnu populaciju, a zatim razvija tu populaciju selekcijom, mutacijom i križanjem, dok je dobrota (eng. fitness) svake jedinke određena specifičnom funkcijom procjene dobrote u skladu s definicijom problema. Jedina razlika GEP-a je u tome što ima vlastiti skup genetskih operatora posebno dizajniranih za rad s njegovim linearnim i multigenskim prikazom fiksne duljine.

1.2.1. Reprezentacija kromosoma u GEP-u

U genski ekspresivnom programiranju (GEP), genom ili kromosom sastoji se od linearnog, simboličkog niza fiksne duljine sastavljenog od jednog ili više gena. Svaki gen je u suštini niz fiksne duljine sastavljen od raznih primitiva. Slijedeći terminologiju GP-a, u GEP-u postoje dvije vrste primitiva: funkcije i terminali. Funkcija je primitiv koji može prihvatiti jedan ili više argumenata i vratiti rezultat nakon evaluacije, dok terminal predstavlja konstantu ili varijablu u programu. U GEP-u, gen je podijeljen na dva dijela. Prvi dio, koji se naziva glava (eng. head), čine funkcije i terminali, dok drugi dio, koji se naziva rep (eng. tail), čine samo terminali.

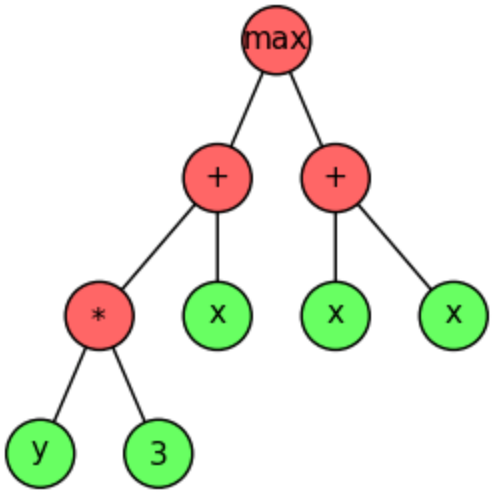
Slika 1.2 u nastavku prikazuje primjer gena sastavljenog od glave (head) veličine 5 te repa (tail) veličine 6.

Slika 1.2 Struktura gena u GEP-u. Glava je sastavljena od 4 funkcije(max, +, +, *) te jednog terminala(x) dok je rep sastavljen samo od terminala(x, x, y, 3, 4, y). U GEP-u je obavezno da se u repu nalaze samo terminali [7].

1.2.2. Genotip – fenotip

U genski ekspresivnom programiranju, linearni kromosomi rade kao genotip, a stabla analize (parse trees) kao fenotip, stvarajući sustav genotip/fenotip. Ovaj sustav je multigeničan (izgrađen od više gena iste duljine), tako da kodira više stabala za analizu u svakom kromosomu. To znači da se računalni programi koje je stvorio GEP sastoje od više stabala analize. Budući da su ta stabla rezultat ekspresije gena, u GEP-u se nazivaju stablima ekspresije (expression trees).

Konkretno, u GEP-u, počevši od prve pozicije gena, možemo konstruirati stablo ekspresije prelaskom s razine na razinu u skladu s brojem argumenata koje svaka funkcija prihvaća. U nastavku, Slika 1.3 prikazuje stablo ekspresije za gen sa Slika 1.2.



Slika 1.3 Ova slika prikazuje stablo ekspresije za program max(x + 3 ∗ y , x + x). Listovi stabala na ovoj slici obojani zelenom bojom nazivaju se terminali, dok se unutarnji čvorovi, obojani crvenom bojom, nazivaju primitivima. Terminali su podijeljeni u dvije podvrste: konstante i argumente. Konstante ostaju iste tijekom čitave evolucije, dok su argumenti programski ulazi. Kod stabla ekspresije sa slike argumenti su varijable x i y, a konstanta je broj 3. [7]

Možemo primijetiti da se posljednja dva elementa u genu ne pojavljuju u stablu ekspresije. Iako je u GEP početno mjesto uvijek prva pozicija gena, krajnja točka ne podudara se uvijek s posljednjom pozicijom gena. Uobičajeno je da GEP geni imaju nekodirajuće regije. U tom smislu, dekodiranje gena (ili kromosoma) u stabla ekspresije slično je ekspresiji gena u prirodi.

1.2.3. K – ekspresije i geni

Kodirajuća regija gena naziva se otvoreni okvir čitanja (ORFs), što se u GEP-u također naziva K-ekspresija (K – expression). K-ekspresija gornjeg gena je [max, +, +, *, x, x, x, y, 3]. Razlog za nekodirajuće regije u genu je taj da se osigura međuspremnik terminala takav da sve k-ekspresije kodirane u GEP genima uvijek odgovaraju valjanim programima ili ekspresijama.

Duljina ORF-a može biti jednaka ili manja od duljine gena. Zbog mogućeg postojanja nekodirajućih regija u GEP genima, gen fiksne duljine može kodirati različita stabla ekspresije. Kako bi se osigurala valjanost gena, za svaki problem se bira duljina glave h, dok se duljina repa t određuje automatski pomoću formule (1):

$$t = h(n-1) + 1 \qquad (1)$$

gdje je n broj argumenata funkcije s najviše argumenata (max arity).

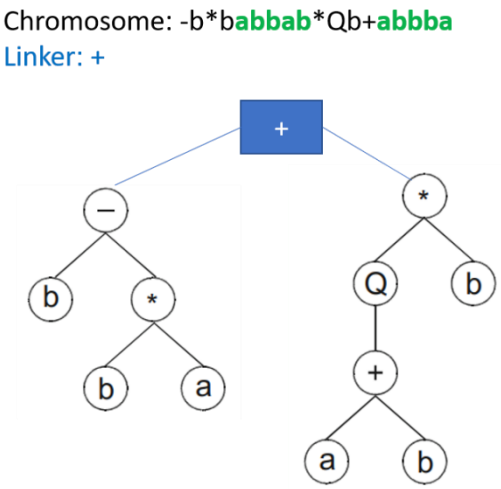
Primjer: Uzmemo li skup funkcija F = {∗, /, −, +} i skup terminala T = {a,b}, te pretpostavimo da je veličina glave h = 10. Kako je maksimalan broj argumenata funkcije s najviše argumenata (max arity) u ovom slučaju n = 2 , veličina repa se dobije iz izraza

t = h(n − 1) + 1 = 10(2-1) = 11.

1.2.4. Multigeni kromosomi

U GEP-u, jedinka je predstavljena kromosomom, koji može sadržavati jedan ili više gena. Kromosom sa samo jednim genom naziva se monogeni kromosom, dok se kromosom sastavljen od više gena naziva multigenskim. U multigenskom kromosomu, svaki gen se prevodi u sub-ET (podstablo ekspresije), a tipično se funkcija povezivanja ili povezivač (linker) koristi za kombiniranje sub-ET-ova u jedno veliko stablo ekspresije.

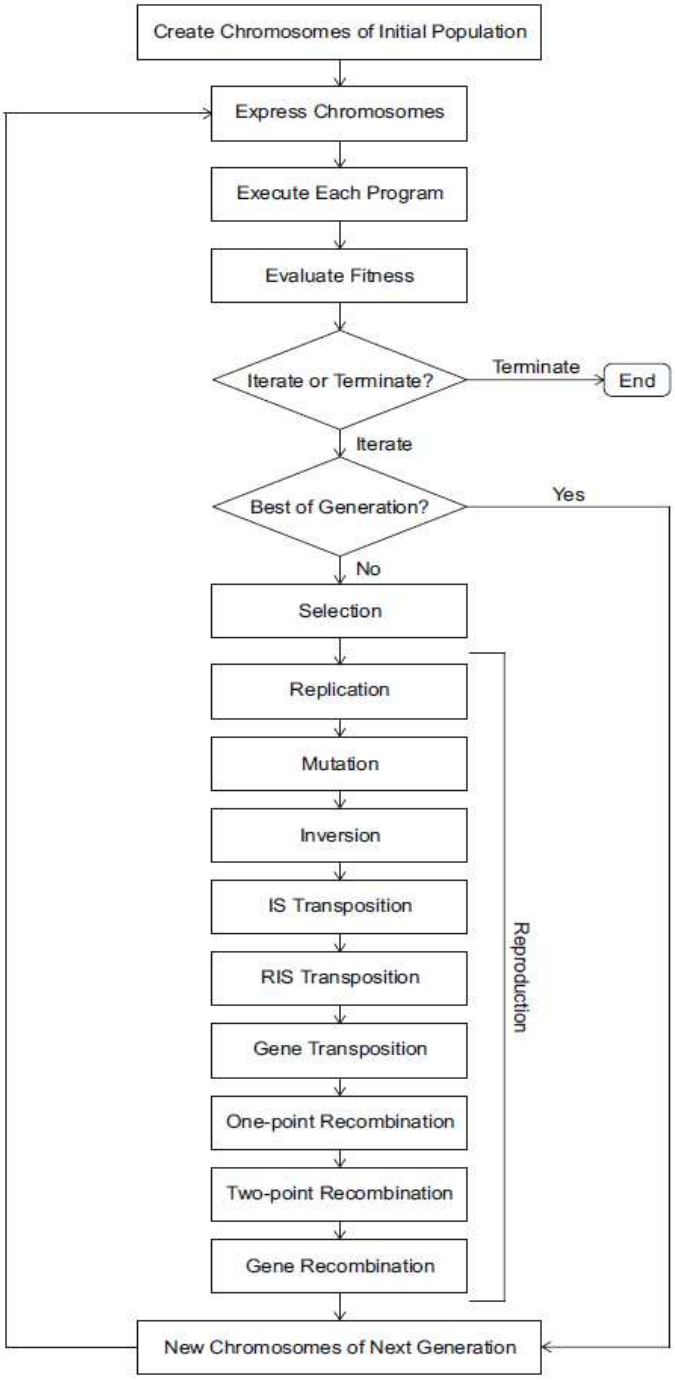
U nastavku Slika 1.4 prikazuje multigenski kromosom s dva gena duljine 9 te način izgradnje stabla iz dva sub-ET-a koji su povezani funkcijom povezivanja koja je u ovom slučaju operator zbrajanja.



Slika 1.4 Stablo ekspresije multigenog kromosoma (kromosoma izgrađenog od dva gena jednake duljine) [7]

1.2.5. GEP algoritam

Slika 1.5 prikazuje slijed koraka algoritma genski ekspresivnog programiranja.



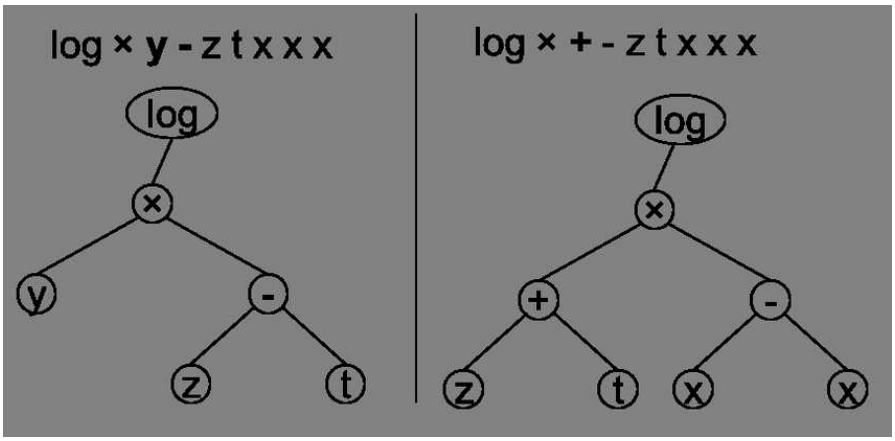
Slika 1.5 Primjer slijeda algoritma genski ekspresivnog programiranja [7]

Proces algoritma genski ekspresivnog programiranja započinje slučajnim generiranjem kromosoma početne populacije. Nakon toga dolazi do ekspresije kromosoma te se procjenjuje dobrota (fitness) pojedinke jedinke. Zatim dolazi do primjene mnogih operatora te su oni opisani u tekstu koji slijedi.

Operator selekcije odabire jedinke koje će operator replikacije replicirati. Ovisno o shemi odabira, broj kopija koje proizlazi iz jednog programa može varirati, pri čemu se neki programi repliciraju više puta dok se drugi repliciraju samo jednom ili uopće ne. Operator selekcije koristi se u svrhu odabira roditelja za stvaranje potomstva na temelju njihove dobrote (fitnessa).

Operator replikacije kopira genom i prenosi ga na sljedeću generaciju. Kromosomi su kopirani u sljedeću generaciju na temelju dobrote (fitness) i pomoću proporcionalne selekcije. Što je jedinka bolja, to će vjerojatno proizvesti više potomaka. Rulet se okreće onoliko puta koliko ima jedinki u populaciji, a veličina populacije ostaje konstantna. Naravno, replikacija ne može sama uvesti varijaciju: samo s djelovanjem preostalih operatora moguće je uvesti svojstvo genetske varijabilnosti u populaciju

Operator mutacije daleko je najučinkovitiji operator u genski ekspresivnom programiranju (GEP). Uz mutaciju, populacije jedinki vrlo se učinkovito prilagođavaju, omogućujući evoluciju dobrih rješenja za gotovo sve probleme. U genski ekspresivnom programiranju, mutacije se mogu pojaviti bilo gdje u kromosomu. No, očito je da struktura kromosoma mora ostati netaknuta, to jest, u glavama (eng. heads) svaki se simbol može promijeniti u drugi (funkciju ili terminal), dok se u repovima terminali mogu mijenjati samo u druge terminale. Na taj je način očuvana struktura kromosoma, a sve nove jedinke proizvedene mutacijom su strukturno ispravni programi. Slika 1.6 prikazuje operator mutacije u GEP-u te kratak opis istog.



Slika 1.6 Operator mutacije u GEP-u. Mutirani simbol je y te se mijenja u funkcionalni simbol +. Početni kodirani izraz je log(y × (z − t)). Nakon mutacije, kodirani izraz je log((z + t) × (x − x)). Dva simbola x u repu postaju aktivni nakon mutacije. [12]

Operator inverzije nasumično bira kromosom, gen koji treba modificirati te početnu i završnu točku slijeda koji se invertiraju. U genski ekspresivnom programiranju, operator inverzije ograničen je na glave (heads) gena. Ovdje bilo koji slijed može biti nasumično odabran i invertiran.

Operator IS transpozicije nasumično bira kromosom, početnu i završnu točku IS elementa i ciljno mjesto. Elementi umetnutog slijeda ili IS elementi su kratki fragmenti genoma s funkcijom ili terminalom na prvoj poziciji koji se transponiraju u glavu gena (head) osim korijena (root).

Operator RIS transpozicije nasumično odabire kromosom, gen koji treba modificirati te početnu i završnu točku RIS elementa. Elementi sekvence umetanja korijena ili RIS elementi su kratki fragmenti s funkcijom na prvoj poziciji koja se transponira na početnu poziciju gena.

Operator transpozicije gena nasumično odabire kromosom koji će se modificirati, a zatim nasumično odabire jedan od svojih gena (osim prvog, očito) za transponiranje. U operatoru transpozicije gena cijeli gen radi kao transpozon i transponira se na početak kromosoma. Za razliku od ostalih oblika transpozicije, kod transpozicije gena, transpozon (gen) briše se na izvornom mjestu. Zbog toga, duljina kromosoma je sačuvana.

U rekombinaciji s jednom točkom prekida (eng. one-point recombination) roditeljski kromosomi se uparuju i dijele u točno istoj točki. Materijal desno od točke rekombinacije se nakon toga izmjenjuje između dva kromosoma. U GEP-u, događaj rekombinacije uvijek uključuje dva roditeljska kromosoma i uvijek rezultira s dvije nove jedinke. Obično se kromosomi kćeri međusobno razlikuju kao što se razlikuju i od svojih majki.

U rekombinaciji s dvije točke prekida (eng. two-point recombination) dva su roditeljska kromosoma uparena i dvije su točke nasumično odabrane kao točke križanja. Materijal između rekombinacijskih točaka se nakon toga izmjenjuje između roditeljskih kromosoma, tvoreći dva nova kromosoma kćeri. Vrijedno je naglasiti da je rekombinacija u dvije točke križanja razornija od rekombinacije u jednoj točki križanja u smislu da temeljitije rekombinira genetski materijal, neprestano uništavajući stare, a stvarajući nove građevne blokove.

U rekombinaciji gena, cijeli geni se izmjenjuju između dva roditeljska kromosoma, tvoreći dva kromosoma kćeri koji sadrže gene oba roditelja. Izmijenjeni geni nasumično su odabrani i zauzimaju potpuno istu poziciju u roditeljskim kromosomima. Vrijedno je naglasiti da rekombinacija gena ne može stvoriti nove gene: jedinke koje je stvorio ovaj operator samo su različiti rasporedi postojećih gena.

2. Strojno učenje

U ovom poglavlju opisane su osnove strojnog učenja, nadzirano učenje, regresijska analiza i naravno simbolička regresija.

2.1. Općenito o strojnom učenju

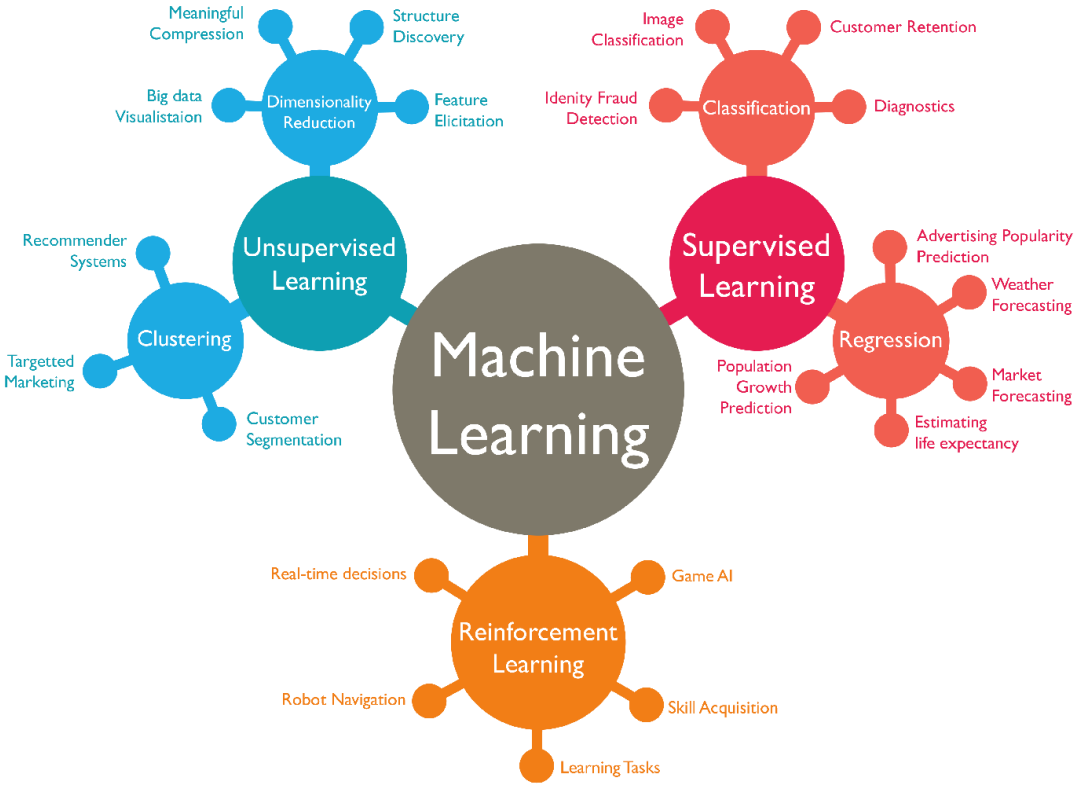
Strojno učenje (ML) je područje umjetne inteligencije i računalne znanosti koje se usredotočuje na korištenje podataka i algoritama u svrhu imitiranja načina na koji ljudi uče, s ciljem stalnog poboljšanja točnosti. ML aplikacije uče iz iskustva (točnije iz podataka) kao što ljudi rade. Posebice je interesantno da to uspijevaju bez izravnog programiranja.

Kada su izložene novim podacima, ML aplikacije uče, rastu, mijenjaju se i razvijaju same. Drugim riječima, strojno učenje uključuje računala koja pronalaze pronicljive informacije bez da im se kaže gdje da traže. Umjesto toga, ona to čine korištenjem algoritama koji uče iz podataka u iterativnom procesu.

Ukratko, cilj strojnog učenja je izvući značenje iz podataka. Dakle, podaci su ključ strojnog učenja. Što više kvalificiranih podataka ima strojno učenje, to algoritam strojnog učenja postaje točniji.

Strojno učenje je složeno, zbog čega je podijeljeno u dva primarna područja, nadzirano učenje i nenadzirano učenje. Svako od njih ima specifičnu svrhu i djelovanje, daje različite rezultate i koristi različite oblike podataka. Smatra se da otprilike 70 posto strojnog učenja otpada na nadzirano učenje, dok nenadzirano učenje čini od 10 do 20 posto. Ostatak čini podržano učenje.

U nastavku, Slika 2.1 prikazuje podjelu strojnog učenja.



Slika 2.1 Prikaz podjele strojnog učenja na mnoge grane i podgrane [13]

2.2. Nadzirano učenje

Nadzirano učenje podskup je umjetne inteligencije (AI) i strojnog učenja. Cilj je nadziranog učenja pod nadzorom razumijeti podatke u kontekstu određenog pitanja.

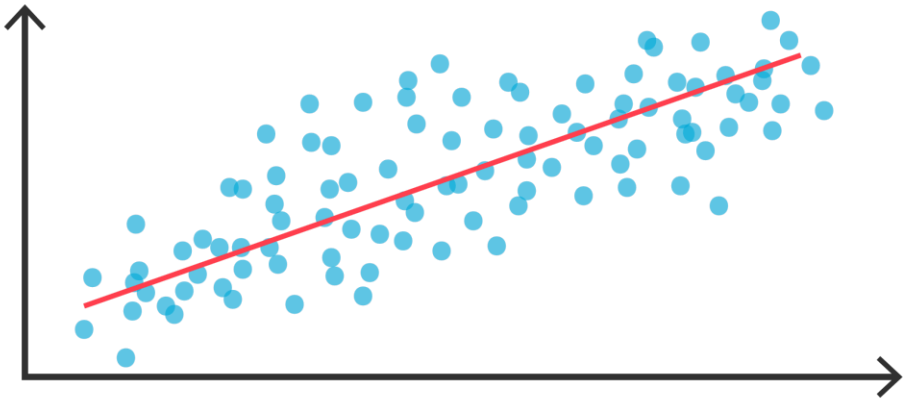
Nadzirano učenje uključuje korištenje označenih skupova podataka za treniranje računalnih algoritama za određeni rezultat. Kako korisnik unosi ulazne podatke u model, sustav se prilagođava kako bi predvidio ishode i preciznije klasificirao podatke unakrsnom provjerom valjanosti – prilagođavajući svoje težine kako bi se što bolje uklopile u model.

Prikupljanje označenih podataka za treniranje prvi je korak u procesu nadziranog učenja. Sljedeći korak je podjela tih označenih podataka u tri skupa: skup za treniranje, skup za validaciju i skup za testiranje. Algoritam nadziranog učenja minimizira pogreške u modelu sa skupom za treniranje. Korisnici mogu provjeriti napredak algoritma učenja neovisno sa skupom za provjeru valjanosti. Testni skup nudi testne podatke iz stvarnog svijeta koji se koriste samo kada skup za provjeru valjanosti dokaže da je model optimalan i može se generalizirati na nove podatke.

2.3. Regresijska analiza

U statističkom modeliranju, **regresijska analiza** je skup statističkih procesa za procjenu odnosa između zavisne varijable i jedne ili više nezavsnih varijabli. Regresijska analiza testira odnos između zavisne varijable u odnosu na neovisne varijable. Tipično, nezavisne varijable mijenjaju se s ovisnom varijablom, a regresijska analiza pokušava odgovoriti na to koji su čimbenici najvažniji za tu promjenu.

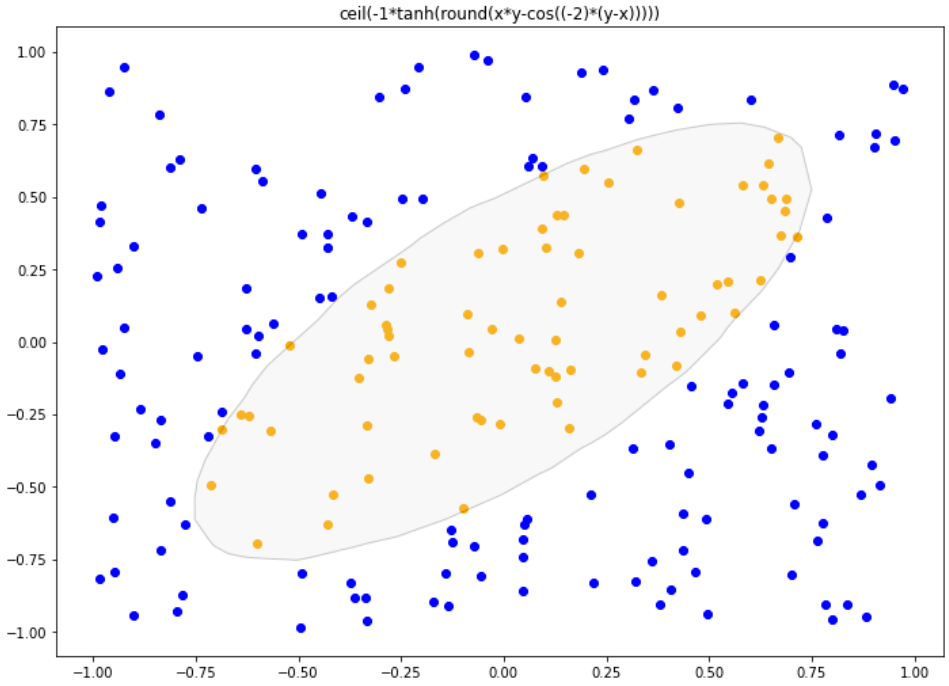
Regresijske analize ne predviđaju uzročnost, već samo odnos između varijabli. Nezavisne varijable nikada neće biti savršeni prediktor zavisne varijable. Pojam pogreške je brojka koja pokazuje sigurnost s kojom možemo vjerovati formuli. Što je veći pojam pogreške, manje je prihvatljiva regresijska linija. Pogreška može biti 50 posto, što ukazuje da varijabla nije ništa bolja od slučajnosti. Ona može biti i 85 posto, što pokazuje da postoji značajna vjerojatnost da neovisna varijabla utječe na zavisnu varijablu. Slika 2.2 prikazuje općeniti regresijski pravac.



Slika 2.2 Crveni pravac prikazuje regresijski pravac, dok plave točke predstavlju točke podataka [14]

2.4. Simbolička regresija

Simbolička regresija je proces identificiranja matematičkih izraza koji odgovaraju promatranom rezultatu iz procesa crne kutije. To je problem diskretne optimizacije za koji se općenito vjeruje da je NP-težak. Simbolička regresija je vrsta regresijske analize koja pretražuje prostor matematičkih izraza kako bi pronašla model koji najbolje odgovara danom skupu podataka, kako u smislu točnosti tako i jednostavnosti. U usporedbi s klasičnom regresijom ovaj pristup ima nedostatak što ima znatno veći prostor za pretraživanje. Budući da je prostor pretraživanja u simboličkoj regresiji neograničen, postoji beskonačan broj modela koji će precizno odgovarati konačnom skupu podataka. To znači da algoritam simboličke regresije može potrajati dulje od standardnih tehnika regresije da otkrije prihvatljiv model i parametrizaciju. Ipak, ova karakteristika simboličke regresije također ima prednosti: budući da evolucijski algoritam zahtijeva raznolikost kako bi učinkovito istražio prostor pretraživanja, rezultat će vjerojatno biti odabir modela s visokim rezultatom. U nastavku Slika 2.3 prikazuje problem klasifikacije riješen simboličkom regresijom.



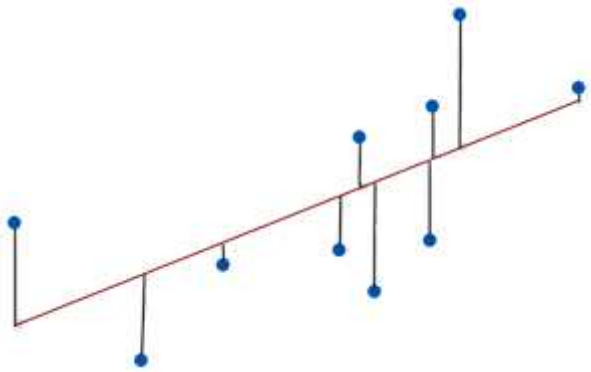
Slika 2.3 Problem klasifikacije riješen simboličkom regresijom [1]

3. Statističke mjere

U ovom poglavlju opisane su statističke mjere koje su korištene u ovom radu pri evaluaciji pouzdanosti regresijskog modela.

3.1. Srednja kvadratna pogreška (MSE)

Srednja kvadratna pogreška (MSE) mjeri količinu pogreške u statističkim modelima. Procjenjuje prosječnu kvadratnu razliku između promatranih i predviđenih vrijednosti. Kada model nema greške, MSE je jednak nuli. Srednja kvadratna pogreška poznata je i kao srednja kvadratna devijacija (MSD). Primjerice, u regresiji, srednja kvadratna pogreška predstavlja prosječni kvadratni rezidual, a reziduali predstavljaju razliku između promatrane vrijednosti i srednje vrijednosti koju model predviđa za to promatranje. Slika 3.1 grafički prikazuje odnos stvarnih i predviđenih vrijednosti.



Slika 3.1 Kako točke podataka padaju bliže regresijskoj liniji, model ima manju pogrešku, smanjujući MSE. Model s manje pogreške daje preciznija predviđanja. [15]

Ako je $\widehat{y_i}$ predviđena vrijednost i-tog uzorka i y_i odgovarajuća stvarna vrijednost tada je srednja kvadratna pogreška (MSE) procijenjena na ukupno $n_{samples}$ uzoraka, definirana formulom (2):

$$MSE(y, \hat{y}) = \frac{1}{n_{samples}} \sum_{i=0}^{n_{samples}-1} (y_i - \widehat{y_i})^2. \quad (2)$$

3.2. R – kvadrat

R-kvadrat (R^2 ili koeficijent determinacije) je statistička mjera u regresijskom modelu koja određuje udio varijance u ovisnoj varijabli koja se može objasniti nezavisnom varijablom. Drugim riječima, r-kvadrat pokazuje koliko se podaci uklapaju u regresijski model. R-kvadrat može imati bilo koju vrijednost između 0 i 1 (odnosno 0% - 100%). Koeficijent determinacije (R-kvadrat) interpretira se kao mjera koja pokazuje koliko dobro regresijski model objašnjava promatrane podatke. Na primjer, r-kvadrat od 70% otkriva da je 70% varijabilnosti uočene u ciljnoj varijabli objašnjeno regresijskim modelom. Općenito, veći r-kvadrat ukazuje na to da se modelom objašnjava veća varijabilnost. Model je reprezentativniji što je koeficijent determinacije bliži jedinici.

Ako je $\widehat{y_i}$ predviđena vrijednost i-tog uzorka i y_i odgovarajuća stvarna vrijednost za ukupno n uzoraka, procijenjeni R^2 je definiran izrazom (3):

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \widehat{y_i})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \qquad (3)$$

gdje je $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ i $\sum_{i=1}^{n}(y_i - \widehat{y_i})^2 = \sum_{i=1}^{n} \in_i^2$.

4. Implementacija i rezultati

U ovom poglavlju opisuje se skup podataka koji se koristi te kako je taj skup podijeljen. Također se opisuju osnove implementacije algoritma te dobiveni rezultati.

4.1. Skup podataka

U ovom radu skup podataka nad kojima se provodi testiranje sastoji se od 100 pojedinačnih ulaznih podataka simboličke regresije (od kojih smo mi uspjeli testirati njih 30-ak) koje su opisane u radu AI Feynman: a Physics-Inspired Method for Symbolic Regression, Udrescu & Tegmark (2019). Svaka zagonetka odgovara tablici brojeva gdje su retci u obliku {x1, x2,..., y) gdje je y = f(x1, x2,...). Zadatak je otkriti odgovarajući simbolički izraz za nepoznatu funkciju f. Prilikom svakog pojedinog pokretanja programa predviđa se jedna od tih 100 zagonetki te se određuje uspješnost tog predviđanja.

Strukturu pojedine nepoznate funkcije (tekstne datoteke) nakon sitnih preinaka pomoću pandas toola prikazuje Slika 4.1. U ovom primjeru m, g, z su ulazni parametri funkcije U te su stoga i stupci nazvani u skladu s tim.

	m	g	z	U
0	2.846020	4.883283	2.521483	35.043365
1	1.286130	2.335288	2.646706	7.949335
2	1.396436	3.038298	4.351214	18.461287
3	3.029507	1.305495	1.225331	4.846193
4	4.735766	3.624687	2.018787	34.653839

Slika 4.1 Prikaz prvih 5 redaka nepoznate funkcije U (m, g, z su parametri funkcije U)

Skup podataka podijeljen je na dva dijela: skup za treniranje i skup za testiranje. Omjer veličina ovih dvaju skupova je 80/20. Model ćemo trenirati isključivo na skupu za treniranje. Razlog za podjelu skupova je taj što je cilj strojnog učenja da uspješno predvidi podatke koje nije vidio u treniranju.

4.2. Programska implementacija

U ovom radu koristimo alat geppy [10] za stvaranje GEP aplikacije. Alat geppy [10] izgrađen je na temeljima većeg „evolucijskog frameworka“ DEAP [9]. Tipična GEP aplikacija započinje stvaranjem primitivnog skupa pozivanjem klase `PrimitiveSet` s imenom primitivnog skupa i imenima ulaznih varijabli kao argumentima. Specificiranjem imena ulaznih varijabli terminali za takve ulaze bit će automatski konstruirani.

Funkcijom `add_function` dodajemo operatore u primitivni set te se s tim operatorima konstruira konačna funkcija simboličke regresije. Ti operatori mogu biti funkcije iz pythonovih modula kao što su operator, math i drugi, a također mogu (i u nekim slučajevima moraju) biti ručno implementirani. Primjer takvog operatora je operator dijeljenja, eksponencijalni operator, operator korjenovanja itd.

Glavni razlog je taj što su navedene funkcije i mnoge druge na neki način „opasne“. Odnosno, lako je generirati nerazumno velike brojeve (kao što je pow(10, 100)), kao i NaN vrijednosti ili čak pogreške (sqrt(-1)). Budući da je evolucija uvijek nasumična, ne možemo pretpostaviti razumne ulaze ili izlaze za danu funkciju. Umjesto toga, potrebno je zaštititi takve funkcije.

Kako bi ih zaštitili potrebno je generirati „sigurne funkcije“ koje ispunjavaju svrhu tražene funkcije, odnosno funkcije koje na neki način sprječavaju generiranje pogrešaka (limitiranje izlazne vrijednosti, zadavanje „default“ vrijednosti izlaza prilikom pojave NaN vrijednosti ili iznimaka itd.).

Sljedeći isječak koda prikazuje dodavanje matematičkih operatora(funkcija) u primitivan skup.

```
primitive_set.add_function(operator.add, 2)
primitive_set.add_function(operator.sub, 2)
primitive_set.add_function(operator.mul, 2)
primitive_set.add_function(math.sin, 1)
primitive_set.add_function(math.cos, 1)
primitive_set.add_function(math.tan, 1)
primitive_set.add_function(math.asin, 1)
primitive_set.add_function(math.acos, 1)
primitive_set.add_function(protected_sqrt, 1)
primitive_set.add_function(protected_pow, 2)
primitive_set.add_function(protected_div, 2)
primitive_set.add_function(protected_exp, 1)
primitive_set.add_rnc_terminal()
```

Kôd 4.1 – Program za stvaranje primitivnog skupa te dodavanje operatora u isti skup

Sljedeći isječak koda prikazuje definiciju „sigurne“ eksponencijalne funkcije.

```
def protected_exp(x1):
if x1 < 1e-6:
return 1
if x1 > 100:
return 1
if x1 < -100:
return 1
return math.exp(x1)
```

Kôd 4.2 – Definicija sigurne funkcije exp

U sljedećem koraku potrebno je definirati dobrotu i tipove jedinki. U GEP-u jedinka se naziva kromosom, koji je sastavljen od jednog ili više gena. Veoma je bitno napomenuti da svi geni u kromosomu moraju imati jednaku veličinu glave (eng. head length) te jednaku veličinu repa (eng. tail length). Gen predstavlja linearni niz fiksne duljine sastavljen od funkcija i terminala. Također, jedinka bi trebala imati svojstvo dobrote (fitness property), a dobrota (fitness) predstavlja mjeru kvalitete rješenja. Minimiziranje dobrote (fitnessa) gradi se korištenjem negativnih težina, dok se maksimiziranje dobrote (fitnessa) gradi pozitivnim težinama .

```
creator.create("min_fitness", base.Fitness, weights=(-1,))
class Individual(gep.Chromosome):
        def __init__(self, gene_gen, n_genes, linker=None):
                super().__init__(gene_gen, n_genes, linker)
                self.fitness = creator.min_fitness()
```

Kôd 4.3 – Program za definiranje fitnessa te kreiranje jedinke i populacije

Idući korak u algoritmu predstavlja stvaranje jedinke i populacije. Također se određuje broj gena te veličina glave (head) svakog gena. Veličina repa (tail) je automatski izračunata pomoću poznate veličine glave i maksimalnog broja argumenata funkcije s najviše argumenata. Bitno je i spomenuti da broj argumenata funkcije povezivanja treba odgovarati broju gena. Općenito, složeniji problemi zahtijevaju veću duljinu glave i duže kromosome formirane s više gena. Stvorena populacija je samo popis jedinki koji generira DEAP funkcija `initRepeat` [9].

Određivanje dobrote (fitness) određene jedinke idući je korak GEP algortima. Linearni kromosom prvo se prevodi u stablo ekspresije (expression tree) te takvo stablo ekspresije zapravo predstavlja računalni program, ili češće, matematički izraz, koji se može izvršiti i evaluirati s obzirom na ulazne vrijednosti. Takav proces dekodiranja genotipa u fenotip postiže se funkcijom `compile_()`. Tom funkcijom jedinka se prevodi u Python lambda izraz. Sada kada imamo dostupan lamba izraz lako možemo procijeniti njegovu dobrotu (fitness) umetanjem konkretnih ulaznih vrijednosti te definiramo funkciju evaluacije. Funkcija evaluacije koristi i tehniku linearnog skaliranja koja pomaže pri pronalasku konstantnih koeficijenata.

```
def linear_scaling_evaluate(individual):
    f = toolbox.compile(individual)
    Yp = np.array(list(map(f, m, g, z)))
    if isinstance(Yp, np.ndarray):
        Q = np.hstack((np.reshape(Yp, (-1, 1)),
np.ones((len(Yp), 1))))
        (individual.a,individual.b),residuals,_,_=
np.linalg.lstsq(Q,U)
        if residuals.size > 0:
            return residuals[0] / len(U)
    individual.a = 0
    individual.b = np.mean(U)
    return np.mean((U - individual.b) ** 2)
```

Kôd 4.4 – Primjena linearnog skaliranja na jedinku, evaluacija dobrote (fitnessa)

U idućem koraku algoritma potrebno je registrirati operatore u Toolbox. Operatori koje registriramo u ovom koraku su: operator selekcije, operator mutacije, operator inverzije operator IS transpozicije, operator RIS transpozicije, operator transpozicije gena, rekombinacija s jednom točkom dijeljenja (one-point recombination), rekombinacija s dvije točke dijeljenja (two-point recombination) te rekombinacija gena.

```
toolbox.register('select', tools.selTournament, tournsize=3)
toolbox.register('mut_uniform', gep.mutate_uniform,
pset=primitive_set, ind_pb=0.05, pb=1)
toolbox.register('mut_invert', gep.invert, pb=0.1)
```

Kôd 4.5 – Isječak koda s registracijom operatora selekcije, mutacije, inverzije

U sljedećem isječku kodu prikazana je definicija nekih statistike koje pratimo, uključujući minimalnu/maksimalnu dobrotu (fitness) u svakoj generaciji i prosječnu/standardnu devijaciju dobrote svake generacije koristeći paket numpy.

```
stats = tools.Statistics(key=lambda ind:
ind.fitness.values[0])
stats.register("avg", np.mean)
stats.register("std", np.std)
stats.register("min", np.min)
stats.register("max", np.max)
```

Kôd 4.6 – Registracija statistika

Zadnji korak je inicijalizacija veličine populacije te određivanje broja generacija. Nakon što je utvrđeno da nema velikih promjena u kasnijim generacijama kao broj generacija uzima se njih 30. Također spremamo 3 najbolje jedinke koje su se pojavile u svim generacijama. U GEP-u je veoma bitan elitizam jer neki genetski operatori imaju destruktivni karakter te bi mogli uništiti najbolju pronađenu jedinku. Nakon što inicijaliziramo sve parametre pokreće se evolucija.

```
population_size = 120
num_of_gen = 30
pop = toolbox.population(n=population_size)
hof = tools.HallOfFame(3)
pop,log=gep.gep_simple(pop,toolbox,n_generations=num_of_gen,
n_elites=2, stats=stats, hall_of_fame=hof, verbose=True)
```

Kôd 4.7 – Pokretanje evolucije

4.3. Rezultati

Za potrebe prikaza rezultata testirane su funkcije prikazane u tablici. Svaka funkcija testirana je 5 puta te je za svaku pokrenuto 30 generacija s veličinom populacije 120. U nastavku Tablica 4.1 prikazuje testirane funkcije kao i broj ulaznih parametara tih funkcija.

Tablica 4.1 Pregled funkcija na kojima je provedeno testiranje

Funkcija	Broj varijabli
$v1 = (u + v)/(1 + u \cdot \frac{v}{c^2})$	3
$Ef = q1 \cdot r/(4 \cdot \pi \cdot \varepsilon \cdot r^3)$	3
$f = \frac{e^{-\frac{\theta^2}{2}}}{\sqrt{2\pi}}$	1
$F = G \cdot m1 \cdot \frac{m2}{(x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2}$	9

U nastavku, tabličnim prikazima(Tablica 4.2, Tablica 4.3, Tablica 4.4, Tablica 4.5) izloženi su rezultati testiranja svake pojedine funkcije. Svaka funkcija testirana je pet puta te su bilježene statističke mjere, srednja kvadratna pogreška i R-kvadrat.

Tablica 4.2 v1=(u+v)/(1+u·v/c^2)

Pokretanje	MSE	R^2	Najbolja jedinka
1.	0.09	0.93	add(sub(u, sub(protected_div(add(u, u), add(v, 2)), u)), mul(c, protected_div(sub(u, 2), u)))
2.	0.07	0.95	add(mul(add(π, protected_div(sub(2, u), add(v, c))), 2), sub(u, protected_div(protected_div(1, -9), 1)))
3.	0.08	0.94	add(sub(protected_div(u, mul(mul(1, c), v)), u), protected_div(π, u))
4.	0.08	0.94	add(mul(add(mul(π, u), add(1, -9)), protected_div(mul(π, c), add(c, 1))), mul(add(mul(π, u), add(1, -9)), protected_div(mul(v, c), add(c, u))))
5.	0.09	0.93	add(sub(c, protected_div(mul(u, sub(π, u)), protected_div(v, π))), mul(mul(sub(u, π), sub(-8, c)), mul(1, 2)))

Tablica 4.3 Ef = q1· r/(4·π·ε·r^3)

Pokretanje	MSE	R^2	Najbolja jedinka
1.	0.00	1.00	add(1, protected_div(protected_div(protected_div(protected_div(q1, ε), r), protected_div(π, -8)), ε))
2.	0.00	1.00	add(protected_div(mul(protected_div(add(-1,π), mul(ε, ε)), q1), r), protected_div(add(1, π), -8))
3.	0.00	1.00	add(protected_div(q1, mul(r, mul(mul(ε, ε), 2))), protected_div(q1, mul(r, protected_div(mul(ε, ε), 2))))
4.	0.00	1.00	add(mul(mul(protected_div(1,ε),protected_div(q1,r)), protected_div(mul(4, 7), ε)), protected_div(sin(2), 1))
5.	0.00	1.00	add(sin(sub(r, r)), mul(protected_div(-5, ε), protected_div(protected_div(q1, ε), r)))

Tablica 4.4 f = e^(-θ^2/2)/√2π

Pokretanje	MSE	R^2	Najbolja jedinka
1.	0.00	1.00	add(mul(Θ, protected_div(add(sub(1, Θ), 5), sub(2, π))), protected_div(Θ, π))
2.	0.00	1.00	add(protected_div(9, Θ), protected_div(protected_div(Θ, mul(add(π, Θ), sub(π, Θ))), 8))
3.	0.00	1.00	add(protected_div(add(Θ, protected_exp(sub(protected_exp(2), Θ))), π), protected_exp(Θ))
4.	0.00	1.00	add(add(add(-7, protected_div(1, π)), mul(2, Θ)), sub(mul(protected_exp(sub(π,Θ)), add(protected_div(2, 2), 2)), -2))
5.	0.00	1.00	add(sub(protected_div(protected_exp(π), sub(sub(2, π), Θ)), Θ), add(mul(protected_div(2, mul(π, theta)), π), 1))

Tablica 4.5 F = G·m1·m2/((x2-x1)^2+(y2-y1)^2+(z2-z1)^2)

Pokretanje	MSE	R^2	Najbolja jedinka
1.	0.00	0.81	add(mul(add(y1, x1), sub(mul(m1, x2), sub(y2, x1))), mul(x1, mul(y1, mul(add(x2, G), z1))))
2.	0.00	0.80	add(mul(y1, add(mul(mul(x2, x1), add(m1, z1)), G)), add(mul(-1, y2), add(x2, G)))
3.	0.00	0.83	add(add(z1, protected_div(sub(G, sub(y2, x2)), z1)), sub(x2, sub(sub(z2, mul(x1, y1)), m1)))
4.	0.00	0.79	add(mul(add(y1, m1), add(protected_div(G, z2), add(x2, x1))), mul(mul(sub(x2, y2), 1), protected_div(sub(m2, z1), 2)))
5.	0.00	0.80	add(add(mul(protected_div(y1, z2), sub(m1, G)), sub(mul(z1, -6), add(z1, z2))), mul(x2, mul(mul(mul(y1, z1), add(m1, G)), x1)))

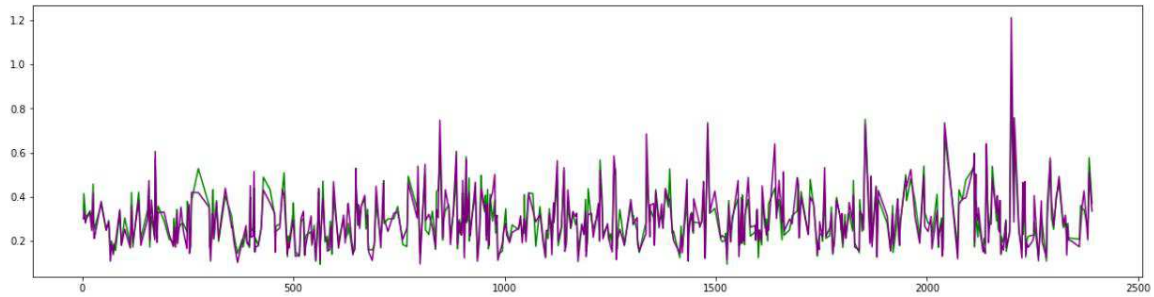
Tablica 4.6 Sumarne statistike MSE

Funkcija	min	avg	std
v1=(u+v)/(1+u·v/c^2)	0.07	0.082	0.0075
Ef = q1· r/(4·π·ε·r^3)	0.00	0.00	0.00
f = e^(-θ^2/2)/√2π	0.00	0.00	0.00
F = G·m1·m2/((x2-x1)^2+(y2-y1)^2+(z2-z1)^2)	0.00	0.00	0.00

Tablica 4.7 Sumarne statistike R^2

Funkcija	min	avg	std
v1=(u+v)/(1+u·v/c^2)	0.93	0.938	0.0075
Ef = q1· r/(4·π·ε·r^3)	1.00	1.00	0.00
f = e^(-θ^2/2)/√2π	1.00	1.00	0.00
F = G·m1·m2/((x2-x1)^2+(y2-y1)^2+(z2-z1)^2)	0.79	0.806	0.0014

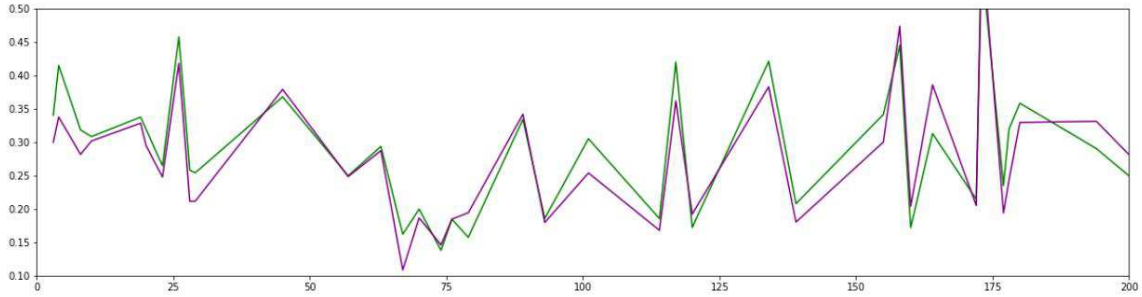
Slika 4.2 grafički prikazuje odnos predviđenih i stvarnih vrijednosti za određenu funkciju.



Slika 4.2 Odnos predviđenih(zelena boja) i stvarnih vrijednosti(ljubičasta boja) za funkciju

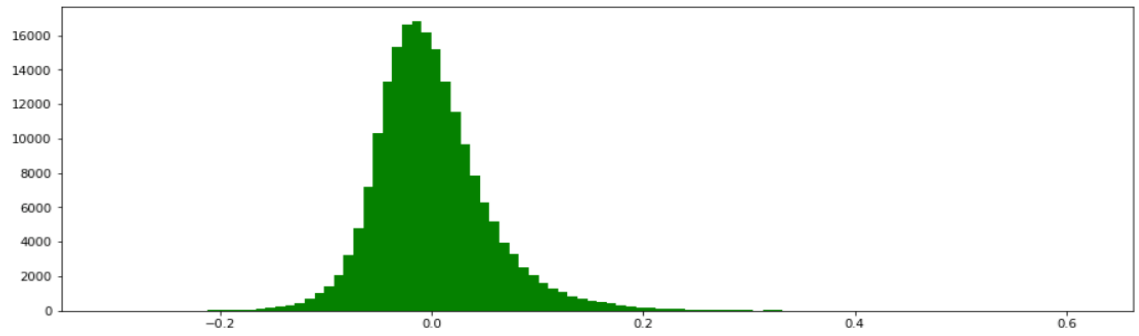
F = G·m1·m2/((x2-x1)^2+(y2-y1)^2+(z2-z1)^2)

Kako bi bolje uvidjeli razlike između tih vrijednosti Slika 4.3 prikazuje isti graf limitiran na manji interval.



Slika 4.3 Uvećani prikaz grafa predviđenih i stvarnih vrijednosti

U nastavku, Slika 4.4 prikazuje histogram grešaka predikcije na testnom skupu određene funkcije.



Slika 4.4 Histogram grešaka predikcije na testnom skupu za funkciju
F = G·m1·m2/((x2-x1)^2+(y2-y1)^2+(z2-z1)^2)

Kako možemo vidjeti, statističke mjere, odnosno R – kvadrat i srednja kvadratna pogreška pokazuju nam da naš model veoma dobro predviđa nepoznate funkcije na kojima je testiran. Visoke vrijednosti R – kvadrat nam govore da je veliki postotak varijabilnosti uočene u ciljnoj varijabli objašnjen regresijskim modelom, a male vrijednosti srednje kvadratne pogreške upućuju na malu količinu pogreške našeg modela. Naš skup podijeljen je u dva odvojena skupa; skup za treniranje i skup testiranje. Model se jednako dobro pokazao na skupu za testiranje kao što na skupu za treniranje. Iz toga možemo zaključiti da model nije prenaučen, odnosno dobar je i za podatke koje još nije vidio.

Zaključak

U ovom radu koristili smo genski ekspresivno programiranje kako bi riješili problem simboličke regresije. Testirajući naš model na više od 30 različitih zagonetnih funkcija dobili smo zadovoljavajuće rezultate.

Rezultati testiranja ovise o funkciji koja se predviđa. Model će lakše i bolje predvidjeti linearne funkcije s manje parametara nego nelinearne s puno parametara. Naš model zadovoljavajuće predviđa većinu testiranih funkcija, odnosno s ne prevelikom srednjom kvadratnom pogreškom otkriva takozvane zagonetne funkcije.

Eureqa je softver za simboličku regresiju temeljen na genetskom programiranju koji je osmislila tvrtka Nutonian. Za taj softver predviđala se mogućnost otkrivanja zakona fizike sa simboličkom regresijom. Iako se ta predikcija možda nikada neće ispuniti, možemo se složiti da su mnogi modeli strojnog učenja koji se danas primjenjuju složeniji nego što je potrebno, a u suštini rade posao koji bi mogla obaviti jednostavna matematička formula [1]. Simbolička regresija vjerojatno nikada neće biti korisna za probleme poput klasifikacije slika, ali prijelaz na eksplicitne simboličke modele mogao bi iznijeti na vidjelo mnoge skrivene obrasce u moru skupova podataka kojima danas raspolažemo [1].

Literatura

[1] *Symbolic Regression: The Forgotten Machine Learning Method*, Poveznica: https://towardsdatascience.com/symbolic-regression-the-forgotten-machine-learning-method-ac50365a7d95

[2] *Evolutionary Algorithm,* Poveznica: https://www.sciencedirect.com/topics/mathematics/evolutionary-algorithm

[3] *File:Evolutionary Algorithm.svg,* Poveznica: https://commons.wikimedia.org/wiki/File:Evolutionary_Algorithm.svg

[4] *Gene expression programming,* Wikipedia, 2022. Poveznica: https://en.wikipedia.org/wiki/Gene_expression_programming

[5] *Gene expression programming* , Poveznica: https://www.gene-expression-programming.com/

[6] *Symbolic regression*, Wikipedia, 2022. Poveznica: https://en.wikipedia.org/wiki/Symbolic_regression

[7] *Max Tegmark, Feynman Symbolic Regression Database* Poveznica: https://en.wikipedia.org/wiki/Symbolic_regression

[8] *Introduction to gene expression programming* , Poveznica: https://geppy.readthedocs.io/en/latest/intro_GEP.html

[9] *Accuracy and precision* , Poveznica: https://www.math.net/accuracy-and-precision

[10] *DEAP documentation* , Poveznica: https://deap.readthedocs.io/en/master/

[11] *geppy's documentation,* Poveznica: *https://geppy.readthedocs.io/en/latest/*

[12] *The mutation operator in GEP, Poveznica:* https://www.researchgate.net/figure/The-mutation-operator-in-GEP-The-mutated-symbol-is-y-it-is-changed-into-functional_fig6_283872823

[13] *Machine learning, Poveznica:* https://www.wordstream.com/wp-content/uploads/2021/07/machine-learning.png

[14] *Regression analysis diagram, Poveznica:* https://www.tibco.com/sites/tibco/files/media_entity/2020-09/regression-analysis-diagram.svg

[15] *Mean squared error, Poveznica:* https://www.statisticshowto.com/probability-and-statistics/statistics-definitions/mean-squared-error/

[16] Silviu-Marian Udrescu, Max Tegmark, AI Feynman: a Physics-Inspired Method for Symbolic Regression, 2020

[17] Luiz Otavio V. B. Oliveira, Joao Francisco B. S. Martins, Luis F. Miranda, Gisele L. Pappa, Analysing Symbolic Regression Benchmarks under a Meta-Learning Approach, 2018

[18] Yuzhong Peng, An improved Gene Expression Programming approach for symbolic regression problems, 2014

[19] Cândida Ferreira†, Gene Expression Programming: A New Adaptive Algorithm for Solving Problems

Sažetak

Tema ovog rada je „Primjena genski ekspresivnog programiranja na problemu simboličke regresije”. Implementacija genski ekspresivnog programiranja izvedena je pomoću alata geppy koji je zasnovan na DEAP frameworku za evolucijsko računarstvo. Preinakama gotovih algoritama geppy alata te isto tako dodavanjem inovacija u alat pokušava se riješiti središnji problem ovog rada – simbolička regresija. Drugim riječima, pokušava se pronaći simbolički izraz koji odgovara podacima iz nepoznate funkcije. Kao skup podataka u ovom radu koristi se baza od 100 zagonetki simboličke regresije iz udžbenika Richarda Feynmana „The Feynman Lectures on Physics”.

Ključne riječi: simbolička regresija, genski ekspresivno programiranje, strojno učenje, regresija, evolucijsko računarstvo

Summary

The topic of this paper is "Application of gene expression programming for the symbolic regression problem". The implementation of gene expression programming was performed using an evolutionary algorithm framework called geppy which is built on top of the evolutionary computing framework called DEAP. By modifying algorithms of the geppy tool and also by adding innovations to the tool, we tried to solve the central problem of this work - symbolic regression. In other words, an attempt is made to find a symbolic expression that corresponds to data from unknown functions. As a dataset in this paper, a database of 100 puzzles of symbolic regression from Richard Feynman's textbook "The Feynman Lectures on Physics" is used.

Keywords: symbolic regression, gene expression programming, machine learning, regression, evolutionary computation

Skraćenice

GEP	*Gene expression programming*	genski ekspresivno programiranje
MSE	*Mean squared error*	srednja kvadratna pogreška
EA	*An evolutionary algorithm*	evolucijski algoritam
GA	*A genetic algorithm*	genetski algoritam
ORF	*Open reading frame*	otvoreni okvir čitanja
ET	*Expression tree*	stablo ekspresije
IS	*Insertion sequence*	-
RIS	*Root insertion sequence*	-
ML	*Machine learning*	strojno učenje
AI	*Artificial intelligence*	umjetna inteligencija
MSD	*Mean squared deviation*	srednja kvadratna devijacija