Podešavanje težina neuronske mreže upotrebom optimizacionog algoritma

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet

Nikola Stamenić, Lea Petković
mi16177@alas.matf.bg.ac.rs, mi16163@alas.matf.bg.ac.rs

16. jun 2020.

Sažetak

Ovaj rad se bavi podešavanjem težina neuronske mreže uz pomoć algoritma optimizacije rojem čestica. Pored osnovnog teorijskog pristupa datim pojmovima, ANN i PSO, rad sadrži i praktičnu primenu ovakvog pristupa podešavanja težina. Kako bi imali osećaj koliko su rezultati dobri, autori ovog rada svoje rezultate upoređuju sa rezultatima autora rada *Designing Artificial Neural Networks Using Particle Swarm Optimization Algorithms* [3].

Ključne reči: PSO, neuronske mreže, ANN, optimizacija rojem čestica, Keras, Iris, Wine, Breast Cancer.

Sadržaj

1 Uvod

Podešavanje težina neuronske mreže (eng. *Artificial Neural Network, ANN*) za cilj ima nalaženje optimalnog skupa težina, kako bi čitava neuronska mreža za određene podatke davala što tačnije izlaze, kako u primeni za klasifikaciju, tako i u primeni za regresiju. Neuronska mreža može biti korišćena na jako bitnim istraživanjima, tako da podešavanje težina može igrati jako bitnu ulogu. Jedan od načina na koje možemo podesiti težine jeste algoritmom optimizacije rojem čestica, skraćeno (eng. *Particle Swarm Optimization*) [4]. PSO je algoritam nastao inspirisan prirodom, odnosno kretanjem jata ptica u potrazi za hranom, pa može predstavljati dobar algoritam za podešavanje težina neuronske mreže. Problem sa samim algoritmom može biti to što postoji mogućnost zaglavljivanja u lokalnom optimumu, što u startu ne garantuje nalaženja najboljeg mogućeg rešenja. Problemom podešavanja težina neuronske mreže, pomoću optimizacije rojem čestica, bavili su se mnogi istraživači [3].

2 Neuronske mreže

Neuronska mreža je sistem koji vrši mapiranje između ulaza i izlaza problema. Neuronske mreže zapravo predstavljaju parametrizovanu reprezentaciju koja se može koristiti za aproksimaciju raznih funkcija [3]. Matematičkom optimizacijom nekog od kriterijuma kvaliteta vrši se pronalaženje odgovarajućih parametara.

Neuronske mreže uče informacije kroz proces treniranja u nekoliko iteracija. Kada je proces učenja završen, neuronska mreža je spremna i sposobna da klasifikuje nove informacije, predvidi ponašanje, ili aproksimira nelinearnu funkciju problema. Njena struktura sastoji se od skupa neurona, predstavljenih funkcijama, koji su međusobno povezani sa ostalim neuronima organizovanim u slojeve.

Struktura neuronske mreže se razlikuje po broju slojeva. Prvi sloj jeste ulazni sloj, poslednji sloj jeste izlazni, a svi slojevi između se nazivaju skrivenim slojevima. Slojevi su međusobno potpuno povezani. Slojevi komuniciraju zahvaljujući tome što je izlaz svakog neurona, iz prethodnog sloja, povezan sa ulazima svih neurona iz narednog sloja. Jačina veza kojom su neuroni međusobno povezani se naziva težinski faktor (eng. *weight*). Najčešće ima 3 sloja.

Postoje različite vrste neuronskih mreža. Mozemo ih klasifikovati prema: broju slojeva (jednoslojne i višeslojne), vrsti veza između neurona, smeru prostiranja informacija (neuronske mreže sa propagacijom unapred ili unazad) [1], vrsti podataka itd.

Njihove primene su mnogobrojne, obzirom da predstavljaju najčešće primenjivanu metodu mašinskog učenja. Neke od primena su: kategorizacija teksta, medicinska dijagnostika, prepoznavanje objekata na slikama, autonomna vožnja, igranje igara poput igara na tabli ili video igara, mašinsko prevođenje prirodnih jezika, prepoznavanje rukom pisanih tekstova itd.

2.1 Keras Pajton biblioteka

Keras je biblioteka za neuronske mreže, napisana u Pajtonu. Keras radi na platformi za mašinsko učenje *TensorFlow*. Razvijena je sa ciljem da omogući brzo eksperimentisanje sa neuronskim mrežama [2], da bude razumljiva, modularna i proširiva.

Pored, već pomenute, *TensorFlow* platforme, ova biblioteka radi i na: *Microsoft Cognitive Toolkit, R, Theano, ili PlaidML* platformama. Nastala je kao deo istraživanja u okviru projekta ONEIROS (eng. *Open-ended Neuro-Electronic Intelligent Robot Operating System*), a njen autor i održavaoc je gugl inženjer - François Chollet.

3 Optimizacija rojem čestica

U ovoj sekcije biće objašnjen sam algoritam optimizacije rojem čestica. Najviše vremena biće posvećeno originalnom algoritmu PSO-a kao i drugoj generaciji algoritma PSO (eng. *The Secong Generation of PSO*), kao i novom modelu algoritma PSO (eng. *A New Model of PSO*).

3.1 Originalni algoritam PSO

Algoritam PSO je metod za optimizaciju neprekidne nelinearne funkcije, koji je predložio Eberhart. Sam algoritam je inspirisan posmatranjem socijalnog i kolektivnog ponašanja u kretanju jata ptica pri potrazi za hranom ili preživaljavanjem. PSO je nadahnut kretanjem najboljeg člana populacije i njegovog iskustva. Metafora govori da se skup rešenja kreće prostorom pretrage sa ciljem da nađe što bolju poziciju, rešenje [3].

Populacijom se smatra grupa čestica i gde svaka predstavlja poziciju $x_i \in R^D$, **i = 1,...,M** u višedimenzionom prostoru. Čestice se evaluiraju u posebnoj funkciji optimizacije, kako bi se odredila njihova prilagođenost i sačuvala najbolja vrednost. Svaka čestica se kreće po prostoru pretrage u zavisnosti od funkcije brzine v_i koja u obzir uzima globalno najbolju poziciju u populaciji ($p_g \in R^D$ - socijalna komponenta) kao i najbolju poziciju date čestice ($p_i \in R^D$ - kognitivna komponenta). Čestice će se kretati u svakoj iteraciji na drugu poziciju, dok ne dostignu optimalnu poziciju. U svakom momentu t, brzina čestice i se ažurira koristeći:

$$v_i(t+1) = \omega v_i(t) + c_1 r_1(p_i(t) - x_i(t)) + c_2 r_2(p_g(t) - x_i(t))$$

gde je ω inertna težina i obično je postavljena da varira linearno od 1 do 0 tokom iteracije, c_1 i c_2 su koeficijenti ubrzanja, r_1 i r_2 su slučajni brojevi iz uniformne (0,1) raspodele. Ubrzanje v_i je ograničeno između $[v_{min}, v_{max}]$. Ažuriranjem ubrzanja na ovaj način dozvoljavamo čestici i da traži najbolju poziciju $p_i(t)$, dok se najbolje globalno rešenje računa [3]:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

3.2 Druga generacija PSO algoritma

Druga generacija PSO algoritma je unapređenje originalnog PSO algoritma, koja u obzir uzima tri aspekta: lokalni optimum svih čestica, globalno najbolje rešenje, i novi koncept - geometrijski centar optimalne populacije. Autor knjige *"Second Generation Particle Swarm Optimization"* objašnjava da ptice održavaju određenu distancu između centra jata (hrane). Jata ptica uvek ostaju u istom regionu neko vreme, tokom kojeg će centar jata ostati nepomeren u očima čestica. Nakon toga, jato se kreće na sledeći region, tada sve čestice moraju održati određenu distancu sa centrom jata.

3.3 Novi model PSO-a

Ovaj algoritam je predložio Garo (eng. *Garro*), bazirao ga je na osnovu ideja drugih autora koji su predlagali unapređenje originalnog PSO algoritma. Shi i Eberhart su predlagali linearno variranje inertnih težina kroz generacije, što je znatno unapredilo performanse originalnog PSO algoritma. Yu je razvio strategiju da kada se kroz generacije globalno najbolje rešenje ne poboljšava, svaka čestica i biva izabrana sa predefinisanom verovatnoćom, a zatim je dodat slučajni šum svakom vektoru brzine dimenzije v_i izabrane čestice i. Bazirano na nekim evolutivnim shemama Genetičkih algoritama, nekoliko efektnih mutacija i ukrštanja su predložene za PSO.

4 Podešavanje težina neuronske mreže pomoću optimizacije rojem čestica

Treniranje neuronske mreže, odnosno podešavanje težina, izvršeno je pomoću optimizacije rojem čestica. Napravljena je troslojna neuronska mreža (ulazni, skriveni i izlazni sloj) pomoću Keras biblioteke u Pajtonu, pomoću koje se rešavaju klasifikacioni problemi. Dve trećine podataka korišćene su za trening podatke, a ostatak čini test podatke. Skriveni sloj sadrži 100 jedinica/čvorova, dok je broj izlaza ekvivalentan broju različitih klasa u odgovarajućem skupu podataka. Pri kompiliranju Keras modela korišćen je *adam* optimizator, ali on nema nikakvog uticaja, već je naveden iz sintaksnih razloga. Kao metrika korišćena je preciznost (eng. *accurancy*).

Kao što je ranije napomenuto, ANN trenirana je pomoću PSO algoritma. PSO koristi prvo 30, čestica i 300 iteracija, a zatim 50 čestica i 1000 iteracija. Šalju se trening podaci i neuronska mreža. Težine neuronske mreže predstavljaju pozicije čestica. Inicijalne težine biraju se nasumično unutar intervala [-2, 2], a ažuriranje težina se vrši prema formuli:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

objašnjenoj u poglavlju 3.1. Parametri c_1, c_2 i w izabrani su u odnosu na vrednosti istih tih parametara u radu [3], odnosno w je 0.3, dok su c_1 i c_2 za 0.5 manji, tačnije 0.5 i 1.0 redom, jer tako algoritam daje bolje rezultate.

Kao funkcija cilja korišćena je preciznost. Takođe, u cilju izbegavanja zaglavljivanja u lokalnom minimumu algoritam se pokreće nekoliko puta i pozicije najgorih pet čestica se nasumično menjaju. Najbolja čestica je upravo ona sa najvećom preciznošću.

4.1 Rezultati

U ovoj sekciji biće predstavljeni eksperimentalni rezultati dobijeni nad skupovima: *Iris, Breast Cancer i Wine*, kao i upoređeni sa rezultatima drugih istraživača koji su se bavili ovom temom i koristili navedene skupove [3]. Stopa prepoznavanja neuronske mreže procenjena je i računata formulom:

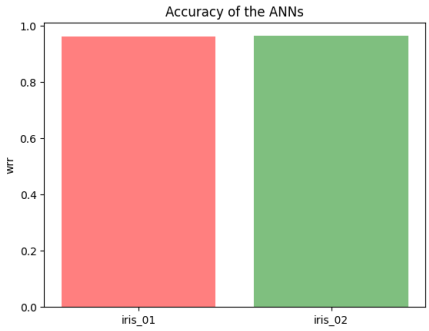
$$wrr = 0.4 * (Tr_r r) + 0.6 * (Te_r r),$$

gde je $Tr_r r$ preciznost nad trening podacima, a $Te_r r$ preciznost dobijenu nad test podacima. Korišćeni su faktori 0.4 i 0.6 kako bi se izbegla visoka vrednost wrr (eng. *weighted recognition rate*) usled visoke preciznosti nad trening podacima (kada to nije slučaj i sa test podacima). U nastavku sledi više o rezultatima na svakom od skupova pojedinačno.

4.1.1 Skup Iris

Ovaj skup podataka nalazi se u Pajton paketu za mašinsko učenje: Scikit-learn. Sastoji se od 3 vrste cveta Iris - Setosa, Versicolour, Virginica. Sadrži 150 instanci i ima 5 atributa koji predstavljaju dužinu i širinu latica (eng. *petal length, petal width*), dužinu čašičnog listića (eng. *sepal length, sepal width*) i vrstu kojoj cvet pripada (eng. *species*).

Procenjena stopa prepoznavanja (wrr) neuronske mreže, koja je dobijena na skupu Iris, najveća je od svih testiranih i ona iznosi vise od 95%, kako na trening, tako i na test podacima. Na slici 1 može se videti i zaključiti da je dobijena wrr autora ovog rada i preciznost autora ranije pomenutog rada [3] približno ista.



Slika 1: Preciznost na podacima skupa Iris. Sa leve strane nalaze se rezultati autora, a desno rezultati iz literature [3]

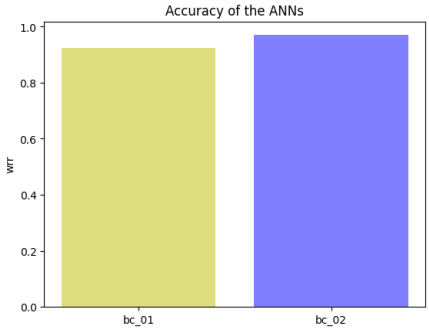
4.1.2 Skup Breast cancer

Skup podataka Breast cancer se, takođe, nalazi u Pajton paketu za mašinsko učenje: Scikit-learn. Predstavlja jednostavan skup podataka koji se koristi za binarnu klasifikaciju. Više informacija o skupu dato je u tabeli 1.

Tabela 1: Informacije o skupu Breast Cancer

Klase	2
Instance po klasi	212(M), 357(B)
Ukupno instanci	569
Dimenzija	30
Atributi	real, positive

Rezultati dobijeni na neuronskoj mreži za skup Breast Cancer neznatno su slabiji od rezultata nad skupom podataka Iris. Dobijena wrr iznosi oko 90-92%. Na slici 2 može se videti poređenje rezultata. Primećuje se da su rezultati autora [3] nešto bolji, ali i dalje su obe preciznosti zadovoljavajuće.



Slika 2: Preciznost na podacima skupa Breast Cancer. Sa leve strane nalaze se rezultati autora, a desno rezultati iz literature [3]

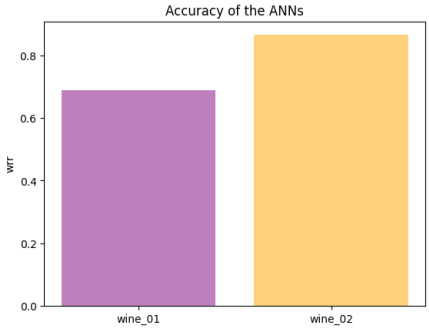
4.1.3 Skup Wine

Wine skup podataka predstavlja klasičan, jednostavan i višeklasni klasifikacioni skup podataka u paketu za Scikit-learn. Više informacija o skupu dato je u tabeli 2.

Tabela 2: Informacije o skupu Wine

Klase	3
Instance po klasi	59,71,48
Ukupno instanci	178
Dimenzija	13
Atributi	real, positive

Najslabiji rezultati dobijeni su upravo na ovom skupu, kako u literaturi, tako i u ovom radu. U radu [3] dobijena je wrr oko 85%, dok su autori ovog rada dobili slabije rezultate - oko 70%, što se može videti na priloženom grafiku 3.



Slika 3: Preciznost na podacima skupa Wine. Sa leve strane nalaze se rezultati autora, a desno rezultati iz literature [3]

5 Zaključak

Istraživanjem podešavanja težina neuronske mreže pomoću optimizacije rojem čestica, zaključeno je da rezultati donekle zavise od skupa, što bi moglo sugerisati da je potrebno posvetiti više pažnje pretprocesiranju onih skupova na kojima su rezultati slabiji ili promenu određenih parametara (npr. povećan broj iteracija pri učenju), u cilju veće preciznosti kasnije.

Dalje istraživanje moglo bi da se fokusira na korišćenju drugih optimizacionih algoritama za podešavanje težina neuronskih mreža, kao što su: optimizacija mravljom kolonijom (eng. *Ant Colony Optimization, ACO*), optimizacija kolonijom pčela (eng. *Bee Colony Optimization,*

BCO), simulirano kaljenje (eng. *Simulated Annealing, SA*) itd. Drugi pravac istraživanja mogao bi se baviti unapređenjem samog PSO algoritma i njegovih parametara, radi dobijanja boljih rezultata.

Literatura

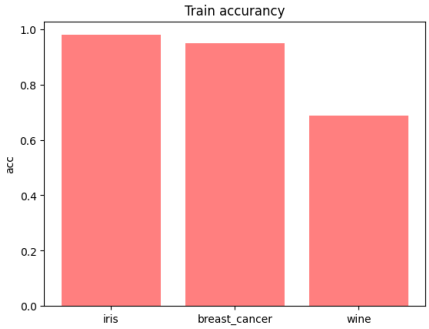
[1] Internet Archive Wayback Machine. on-line at: https://web.archive.org/web/20090215055110/http://learnartificialneuralnetworks.com/#Intro.

[2] Keras. on-line at: https://keras.io/.

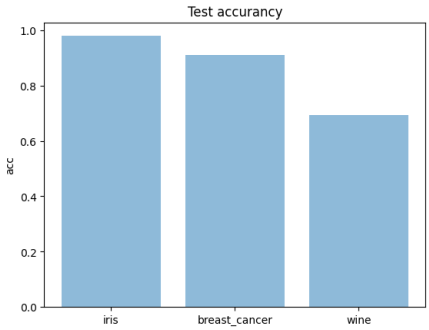
[3] Beatriz A. Garro and Roberto A. Vázquez. Designing Artificial Neural Networks Using Particle Swarm Optimization Algorithms. 2 June 2015.

[4] J. Yu, L. Xi, and S. Wang. *An improved particle swarm optimization for evolving feedforward artificial neural networks.* Neural Processing Letters vol 26 no 3, 2007.

A Prikaz rezultata



Slika 4: Uporedni prikaz rezultata dobijenih na trening podacima, za sva tri skupa



Slika 5: Uporedni prikaz rezultata dobijenih na test podacima, za sva tri skupa

B Kod

```
import pandas as pd                # data processing, CSV file I/O (e.g. pd.read_csv)
import numpy as np                 # linear algebra
from numpy.random import seed
from numpy.random import rand
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from keras.models import Sequential
from keras.layers import Dense
from keras import losses, optimizers
from keras.utils import to_categorical

from matplotlib import pyplot as plt
```

```
class Particle:

    def __init__(self, weights, velocity, omega, c1, c2,
                 lower_bound, upper_bound, particle_id):
        self.id = particle_id
        self.weights = weights
        self.best_weights = weights
        self.velocity = np.array(velocity)
        self.omega = omega
        self.c1 = c1
        self.c2 = c2
        self.lower_bound = lower_bound
```

```
        self.upper_bound = upper_bound
        self.best_valuation = 0
        self.history = []

    def update(self, best_particle):
        self.update_velocity(best_particle)
        self.check_velocity()
        self.update_particle()

        return self.weights

    def update_velocity(self, bp):
        # v = Wv + c1r1(pi - xi) + c2r2(g - xi)
        # v         - ubrzanje
        # W         - omega
        # c1 i c2   - unapred zadati parametri
        # r1 i r2   - slucajni brojevi iz (0,1) uniformne raspodele
        # pi        - najbolje resenje trenutne jedinke
        # g         - najbojle globalno resenje
        r1 = np.random.random()
        r2 = np.random.random()

        self.velocity = self.omega * self.velocity + self.c1*r1*(np.add(np.array(
    self.best_weights), (-1)*np.array(self.weights))) + self.c2*r2*(np.add(np.array
    (bp), (-1)*np.array(self.weights)))

    def check_velocity(self):
        for i in range(len(self.velocity)):
            if(i%2 == 1):
                for j in range(len(self.velocity[i])):
                    if(self.velocity[i][j] < self.lower_bound):
                        self.velocity[i][j] = self.lower_bound

                    if(self.velocity[i][j] > self.upper_bound):
                        self.velocity[i][j] = self.upper_bound
            else:
                for j in range(len(self.velocity[i])):
                    for k in range(len(self.velocity[i][j])):

                        if(self.velocity[i][j][k] < self.lower_bound):
                            self.velocity[i][j][k] = self.lower_bound

                        if(self.velocity[i][j][k] > self.upper_bound):
                            self.velocity[i][j][k] = self.upper_bound

        return


    def update_particle(self):
        # xi = xi + vi
        self.weights = np.add(np.array(self.weights), self.velocity)

    def update_valuation(self, valuation):
        self.history.append(valuation)

        if(valuation > self.best_valuation):
            self.best_valuation = valuation
            self.best_weights = self.weights

    def get_weights(self):
        return self.weights

    def set_weights(self, new_weights):
        self.weights = new_weights
```

```
class PSO:
    def __init__(self, num_particles, num_iters, training_x, training_y, model,
    shapes, c1, c2, w, lower_bound, upper_bound):
        self.num_iters = num_iters
        self.training_x = training_x
        self.training_y = training_y
        self.model = model
        self.best_particle = self.make_velocity(shapes)
        self.best_evaluation = 0
        self.history = []
        self.shapes = shapes
        self.lower_bound = lower_bound
        self.upper_bound = upper_bound
        self.particles = self.make_particles(num_particles, shapes, c1, c2, w,
    lower_bound, upper_bound)
        self.evaluate_particles()
```

```
        self.combo = 0
        self.w = w
        self.c1 = c1
        self.c2 = c2


    def make_particles(self, size, shape, c1, c2, w, low, upp):
        particles = []

        velocity = self.make_velocity(shape)

        for i in range(size):
            weights = self.make_weights(shape, low, upp)
            particles.append(Particle(weights, velocity, w, c1, c2, low, upp, i))

        return particles

    def make_velocity(self, shapes):
        velocity = []

        for shape in shapes:
            velocity.append(np.zeros(shape))

        return velocity


    def make_weights(self, shapes, low, upp):
        weights = []

        for shape in shapes:
            if(len(shape) == 1):
                # wght da dobijemo vrednosti iz (0, upp-low)
                wght = rand(shape[0])*(upp-low)
                rec = np.full(shape, low)

                weights.append(np.add(wght, rec))

            else:
                wght = rand(shape[0], shape[1])*(upp-low)
                rec = np.full(shape, low)

                weights.append(np.add(wght, rec))

        return weights


    def evaluate_particles(self):

        for particle in self.particles:
            self.model.set_weights(particle.get_weights())
            train_loss, train_acc = self.model.evaluate(self.training_x, self.
 training_y, verbose = 0)

            particle.update_valuation(train_acc)

            if(train_acc > self.best_evaluation):
                self.best_particle = particle.get_weights()
                self.best_evaluation = train_acc


    def isclose(self, a, b, rel_tol=1e-09, abs_tol=0.0):
        return abs(a-b) <= max(rel_tol * max(abs(a), abs(b)), abs_tol)

    def update_particles(self):

        for particle in self.particles:
            particle.update(self.best_particle)

    def get_particles(self):
        return self.particles

    def start_pso(self):

        for i in range(self.num_iters):
            partly_best_solution = self.best_evaluation
            self.update_particles()
            self.evaluate_particles()

            #print("Iteracija: {}\nPreciznost: {}\n".format(i+1, self.
 best_evaluation))

            if(self.isclose(partly_best_solution, self.best_evaluation, 0.001)):
                self.combo += 1
            else:
                self.combo = 0
```

```
            if(self.combo == 5):
                self.particles.sort(key = lambda x: x.best_valuation)

                velocity = self.make_velocity(self.shapes)

                weights = self.make_weights(self.shapes, self.lower_bound, self.
    upper_bound)
                self.particles[0] = Particle(weights, velocity, self.w, self.c1,
    self.c2,
                                             self.lower_bound, self.upper_bound,
    self.particles[0].id)

                weights = self.make_weights(self.shapes, self.lower_bound, self.
    upper_bound)
                self.particles[1] = Particle(weights, velocity, self.w, self.c1,
    self.c2,
                                             self.lower_bound, self.upper_bound,
    self.particles[1].id)

                weights = self.make_weights(self.shapes, self.lower_bound, self.
    upper_bound)
                self.particles[2] = Particle(weights, velocity, self.w, self.c1,
    self.c2,
                                             self.lower_bound, self.upper_bound,
    self.particles[2].id)

                weights = self.make_weights(self.shapes, self.lower_bound, self.
    upper_bound)
                self.particles[3] = Particle(weights, velocity, self.w, self.c1,
    self.c2,
                                             self.lower_bound, self.upper_bound,
    self.particles[3].id)

                weights = self.make_weights(self.shapes, self.lower_bound, self.
    upper_bound)
                self.particles[4] = Particle(weights, velocity, self.w, self.c1,
    self.c2,
                                             self.lower_bound, self.upper_bound,
    self.particles[4].id)
                self.combo = 0
                self.evaluate_particles()
                #print("Change bad particles")

            self.history.append(self.best_evaluation)


        return self.best_particle
```

```
accs = []
print("IRIS DATASET: \n\n")
data1 = datasets.load_iris()

x_train, x_test, y_train, y_test = train_test_split(data1.data,
                                                    data1.target,
                                                    test_size=0.33)

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)


# Pravljenje neuronske mreze sa 3 sloja - ulazni, skriveni i izlazni
model = Sequential()
model.add(Dense(units=100, input_dim=x_train.shape[1], activation='relu'))
model.add(Dense(units=y_train.shape[1], activation='sigmoid'))
model.summary()

shapes = [i.shape for i in model.get_weights()]

model.compile(optimizer='adam',
              loss=losses.categorical_crossentropy,
              metrics=['accuracy'])

test_iris_acc = 0
train_iris_acc = 0
best_iris_pso = None

PSOS = []
for i in range(5):
    pso = PSO(30, 300,
              x_train, y_train,
```

```
            model,
            shapes,
            0.5, 1.0, 0.3, -2, 2)

    PSOS.append(pso)

for pso in PSOS:

    best_particle = pso.start_pso()

    model.set_weights(best_particle)

    train_loss, train_acc = model.evaluate(x_train, y_train)
    test_loss, test_acc = model.evaluate(x_test, y_test)

    if(test_iris_acc < test_acc):
        test_iris_acc = test_acc
        best_iris_pso = pso
        train_iris_acc = train_acc

    print("Current PSO train accuracy: " + str(train_acc))
    print("Current PSO test accuracy:   " + str(test_acc) + "\n")

print()
print("Global best accuracy: " + str(test_iris_acc))
accs.append((train_iris_acc, test_iris_acc))
```

```
print("\n---------------------------------\nBREAST CANCER DATASET: \n\n")
data2 = datasets.load_breast_cancer()

x_train, x_test, y_train, y_test = train_test_split(data2.data,
                                                    data2.target,
                                                    test_size=0.33)

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Pravljenje neuronske mreze sa 3 sloja - ulazni, skriveni i izlazni
model = Sequential()
model.add(Dense(units=100, input_dim=x_train.shape[1], activation='relu'))
model.add(Dense(units=y_train.shape[1], activation='sigmoid'))
model.summary()

shapes = [i.shape for i in model.get_weights()]

model.compile(optimizer='adam',
            loss=losses.categorical_crossentropy,
            metrics=['accuracy'])

test_cancer_acc = 0
train_cancer_acc = 0
best_cancer_pso = None

PSOS = []
for i in range(5):
    pso = PSO(30, 300,
            x_train, y_train,
            model,
            shapes,
            0.5, 1.0, 0.3, -2, 2)

    PSOS.append(pso)

for pso in PSOS:

    best_particle = pso.start_pso()

    model.set_weights(best_particle)

    train_loss, train_acc = model.evaluate(x_train, y_train)
    test_loss, test_acc = model.evaluate(x_test, y_test)

    if(test_cancer_acc < test_acc):
        test_cancer_acc = test_acc
        train_cancer_acc = train_acc
        best_cancer_pso = pso

    print("Current PSO train accuracy: " + str(train_acc))
    print("Current PSO test accuracy:   " + str(test_acc) + "\n")

print()
print("Global best accuracy: " + str(test_cancer_acc))
```

```
accs.append((train_cancer_acc, test_cancer_acc))
```

```
print("\n--------------------------------\nWINE DATASET: \n\n")
data3 = datasets.load_wine()

x_train, x_test, y_train, y_test = train_test_split(data3.data,
                                                    data3.target,
                                                    test_size=0.33)

y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# Pravljenje neuronske mreze sa 3 sloja - ulazni, skriveni i izlazni
model = Sequential()
model.add(Dense(units=100, input_dim=x_train.shape[1], activation='relu'))
model.add(Dense(units=y_train.shape[1], activation='sigmoid'))
model.summary()

shapes = [i.shape for i in model.get_weights()]

model.compile(optimizer='adam',
              loss=losses.categorical_crossentropy,
              metrics=['accuracy'])

test_wine_acc = 0
train_wine_acc = 0
best_wine_pso = None

PSOS = []
for i in range(5):
    pso = PSO(30, 300,
              x_train, y_train,
              model,
              shapes,
              0.5, 1.0, 0.3, -2, 2)

    PSOS.append(pso)

for pso in PSOS:

    best_particle = pso.start_pso()

    model.set_weights(best_particle)

    train_loss, train_acc = model.evaluate(x_train, y_train)
    test_loss, test_acc = model.evaluate(x_test, y_test)

    if(test_wine_acc < test_acc):
        test_wine_acc = test_acc
        train_wine_acc = train_acc
        best_wine_pso = pso

    print("Current PSO train accuracy: " + str(train_acc))
    print("Current PSO test accuracy:   " + str(test_acc) + "\n")

print()
print("Global best accuracy: " + str(test_wine_acc))
accs.append((train_wine_acc, test_wine_acc))
```