



Projektni zadatak

Predmet: CS324-Skripting jezici

Oblast: Python Machine Lering

Tema: K-nearest neighbors algorithm

Student: *Nebojša Bosijok* Index: 3440

Mentor: *Nemanja Zdravković*

Sadržaj

1. Uvod i teoretska postavka izabrane teme.....	3
1.1. Mašinsko učenje.....	3
1.1.1. Definicija.....	3
1.1.2. Funkcionisanje	3
1.1.3. Prednosti	3
1.1.4. Metode	4
1.1.5. Vrste	4
1.1.6. Najpopularnije aplikacije	5
1.1.7. Komercijalna primena.....	5
1.1.8. Tehnološki lideri	6
1.2. K-nearest neighbors(K-NN) : Moćan algoritam mašinskog učenja.....	6
1.2.1. Šta je K-NN?	6
1.2.2. Klasifikacija K-NN	7
1.2.3. Euklidova udaljenost.....	7
1.2.4. Koraci koje treba izvršiti tokom K-NN algoritma	8
1.2.5. Uticaj na performanse K-NN algoritma.....	8
1.2.6 Prednosti K-NN	9
1.2.7. Mane K-NN	9
1.2.8. Oblasti u kojim se primenjuje algoritam.....	9
1.2.9. “Birds of a feather flock together”	9
2. Primer/studija slučaja primene izabrane teme.....	10
2.1. Skup podataka o vrstama cvetova Irida.....	10
2.1.1. Korak 1: Izračunajte euklidsku udaljenost.....	11
2.1.2. Korak 2: Naći najbliže komšije.....	14
2.1.3. Korak 3: Napraviti predviđanja.....	15
2.1.4. Studija slučaja vrste Irida	16
3. Zaključak.....	24
4. Literatura.....	24

1. Uvod i teoretska postavka izabrane teme

1.1. Mašinsko učenje

1.1.1. Definicija

Mašinsko učenje je pod-oblast veštačke inteligencije, pri čemu se termin odnosi na sposobnost IT sistema da samostalno pronađu rešenja za probleme prepoznavanja obrazaca u bazi podataka. Odnosno, mašinsko učenje omogućava IT sistemima da prepoznaju obrasce na osnovu postojećih algoritama i skupova podataka i da razviju adekvatne koncepte rešenja. Stoga se u mašinskom učenju veštačko znanje generiše na osnovu iskustva.

Da bi softver mogao da samostalno generiše rešenja, neophodno je prethodno delovanje ljudi. Na primer, potrebni algoritmi i podaci moraju se unapred uneti u sisteme i moraju se definisati odgovarajuća pravila analize za prepoznavanje obrazaca u fondu podataka. Kada su ova dva koraka završena, sistem može mašinskim učenjem da izvršava sledeće zadatke:

- Pronalaženje, izdvajanje i rezimiranje relevantnih podataka
- Izrada predviđanja na osnovu podataka analize
- Izračunavanje verovatnoće za određene rezultate
- Prilagođavanje određenim događajima autonomno
- Optimizacija procesa zasnovanih na prepoznatim obrascima
-

1.1.2. Funkcionisanje

Mašinsko učenje deluje na sličan način kao i ljudsko učenje. Na primer, ako se detetu prikazu slike sa određenim predmetima, ono može da nauči da ih identifikuje i razlikuje. Mašinsko učenje funkcioniše na isti način: Kroz unos podataka i određene komande, računaru je omogućeno da 'nauči' da identifikuje određene objekte(osobe, predmete...) i da ih razlikuje. U tu svrhu softver se isporučuje sa podacima i obučava se. Na primer, programer može reći sistemu da je određeni objekat 'čovjek', a drugi objekat 'nije čovek'. Softver neprekidno prima povratne informacije od programera. Ove povratne signale algoritam koristi za prilagođavanje i optimizaciju modela. Sa svakim novim skupom podataka koji se unose u sistem.

1.1.3. Prednosti

Mašinsko učenje nesumljivo pomaže ljudima da rade kreativnije i efikasnije. U osnovi, može se delegirati prilično složen ili monoton posao računaru putem mašinskog učenja – počev od skeniranja, čuvanja i arhiviranja papirnih dokumenata kao što su fakture pa sve do organizovanja i uređivanje slika

Pored ovih prilično jednostavnih zadataka, mašine za samo učenje mogu obavljati i složene zadatke. To uključuje, na primer, prepoznavanje obrazaca grešaka. To je glavna prednost, posebno u oblastima kao što je prerađivačka industrija: industrija se oslanja na kontinuiranu proizvodnju bez grešaka. Iako čak ni stručnjaci često ne mogu biti sigurni gde i kojom korelacijom nastaje proizvodna greška u floti potrojenja, mašinsko učenje nudi mogućnost rane identifikacije greške – to štedi zastoje i novac. U intervjuu, Damian

Heimel, suosnivač i glavni direktor kompanije Deevio, objasnio je kako se njihov softver za mašinsko učenje koristi u livačkoj industriji. S obzirom da se ovde proizvedene komponente često podvrgavaju strogim bezbednosnim zahtevima, mašinsko učenje je popularan metod u automoatizaciji kontrole kvaliteta na kraju linije. Defektni na limenim komponentima se mogu široko razlikovati, od pukotina do rupa, a kada se proizvodi nekoliko hiljada delova dnevno, postupak inspekcije je sklon ljudskim greškama. Iako se oči ljudskog inspektora vremenom umore, softver za mašinsko učenje može da uvede standardizovani postupak inspekcije kvaliteta.

Programi za samoučenje se sada koriste i u medicinskim oblastima. U budućnosti, nakon što "potroše" ogromne količine podataka (medicinske publikacije, studije itd.). Aplikacije će moći da upozore na slučaj da njegov lekar želi da propiše lek koji ne može da toleriše. Ovo "znanje" takođe znači da aplikacija može da predloži alternativne opcije koje na primer takođe uzimaju u obzir genetske zahteve određenog pacijenta.

1.1.4. Metode

U mašinskom učenju koriste se statističke i matematičke metode za učenje iz skupova podataka. Za to postojedasetine različitih metoda, pri čemu se može napraviti opšta razlika između dva sistema, naime simboličkih pristupa s jedne strane i subsimbolničkih pristupa s druge strane. Dok su simbolički sistemi, na primer, predloženi sistemi u kojima su sadržaj znanja, tj. indukovana pravila i primeri eksplicitno predstavljeni, podsimbolni sistemi su veštačke neuronske mreže. Oni rade na principu ljudskog mozga, pri čemu su sadržaji znanja implicitno predstavljeni.

1.1.5. Vrste

U osnovi, algoritmi igraju važnu ulogu u mašinskom učenju: s jedne strane, oni su odgovorni za prepoznavanje obrazaca, a s druge strane mogu da generišu rešenja. Algoritmi se mogu podeliti u različite kategorije:

Nadgledano učenje: Tokom praćenog učenja, primeri modela su unapred definisani. Da bi se osigurala adekvatna alokacija informacija u odgovarajuće grupe modela algoritama, oni tada moraju biti navedeni. Drugim rečima, sistem uči na osnovu zadatih ulaznih i izlaznih parova. Tokom praćenog učenja programer, koji deluje kao vrsta nastavnika, daje odgovarajuće vrednosti za određeni input. Cilj je obučiti sistem u kontekstu uzastopnih proračuna sa različitim ulazima i izlazima i uspostaviti veze.

Učenje bez nadzora: U učenju bez nadzora veštačka inteligencija uči bez unapred definisanih ciljnih vrednosti i bez nagrada. Uglavnom se koristi za učenje segmentacije (grupisanje). Mašina pokušava dastrukturiše i sortira unesene podatke prema određenim karakteristikama. Na primer, mašina bi mogla (vrlo jednostavno) naučiti da se novčići različitih boja mogu sortirati prema karakterističnoj "boji" kako bi ih strukturirali.

Delimično nadgledanje učenja: Delimično nadzirano učenje je kombinacija nadgledanog i nenadgledanog učenja.

Podsticanje učenja: Pojačavanje učenja – baš kao i klasična Skinner-ova uslovljenost – zasniva se na nagradama i kaznama. Algoritam se podučava pozitivno ili negativnom interakcijom koja reaguje na određenu situaciju koja treba da se dogodi.

Aktivno učenje: U okviru aktivnog učenja algoritam dobija priliku da traži rezultate za određene ulazne podatke na osnovu unapred definisanih pitanja koja se smatraju značajnim. Obično sam algoritam bira pitanja od velike važnosti.

Generalno, baza podataka može biti van mreže ili na mreži, u zavisnosti od odgovarajućeg sistema. Pored toga, može biti dostupan samo jednom ili više puta za mašinsko učenje. Druga karakteristika koja se razlikuje je postepeni razvoj ulaznih i izlaznih parova ili njihovo istovremeno prisustvo. Na osnovu ovog aspekta pravi se razlika između takozvanog sekvencijalnog učenja i takozvanog grupnog učenja.

1.1.6. Najpopularnije aplikacije

Mašinsko učenje se primenjuje na Netflix-u i Amazon-u, kao i za Facebook prepoznavanje lica. Za vas kao korisnika, mašinsko učenje se na primer ogleda u mogućnosti označavanja ljudi na postavljenim slikama. Zapravo, Facebook ima najveću bazu podataka o licima u svetu. Podatke koje korisnici unose u društvenu mrežu koristi Facebook za optimizaciju i obuku sistema za mašinsko učenje u pogledu vizuelnog prepoznavanja.

Još jedna primena mašinskog učenja je sada čvrsto integrisana u svakodnevni život jeste automatsko otkrivanje neželjene pošte koja je integrisana u gotovo sve programe e-pošte. U okviru otkrivanja neželjene pošte, podaci sadržani u e-porukama se analiziraju i kategoriziraju. U tom pogledu se koriste obrasci ‘neželjene pošte’ i ‘neželjeni sadržaj’. Ako je e-pošta prepoznata kao neželjena pošta, program uči da još efikasnije prepoznaje neželjenu poštu. Druga područja primene mašinskog učenja uključuju rangiranje na pretraživačima, borbu protiv kibernetičkog kriminala i sprečavanje računarskih napada.

1.1.7. Komercijalna primena

Uz pomoć mašinskog učenja, ekonomski podaci mogu se pretvoriti u novac. Kompanije koje se oslanjaju na metode mašinskog učenja ne samo da mogu da povećaju zadovoljstvo svojih kupaca, već i da istovremeno postignu smanjenje troškova. Kroz mašinsko učenje mogu se proceniti želje i potrebe kupaca i presonalizovati sledeće marketinške mere. Ovo optimizuje korisničko iskustvo i povećava lojalnost kupaca.

Pored toga, mašinsko učenje može pomoći kompanijama da saznaju da li postoji opasnost od migracije kupaca u bliskoj budućnosti. To se postiže, na primer, automatskom procenom zahteva za podršku. Alternativa je analiziranje onih karakteristika koje su zajedničke kupcima koji su već migrirali. Ako se oni postojeći kupci koji takođe imaju ove karakteristike zatim filtriraju na osnovu karakteristika koje proizilaze iz analize, kompanija dobija spisak grupa kupaca kojima preči migracija. Tada se mogu preduzeti odgovarajuće mere za zadržavanje ovih kupaca.

Pored toga, sve više i više ‘chat’ botova se koristi u oblasti telefonske službe za korisnike. To su automatizovani programi koji komuniciraju sa kupcima. Na ovaj način ‘chat’ botovi mogu da optimizuju svoje kognitivne sposobnosti s obzirom na interpretaciju tona u različitim situacijama. Pored toga, ‘chat’

botovi su u stanju da proslede poziv – na primer, ako je reč o složenim zahtevima – zaposlenom u za pružanje informacija.

Pored toga, mašinsko učenje je ključna tehnologija u razvoju autonomnih sistema: pored automobila bez vozača, mašinsko učenje se koristi i u zajedničkim robotima. Ostale oblasti primene za mašinsko učenje bile bi:

- Analize berze
- Otkrivanje prevara sa kreditnim karticama
- Automatizovane dijagnostičke procedure
- Akvizicija nagaznih mina u akustičnim podacima senzora i radara

1.1.8. Tehnološki lideri

Pored Microsoft-a, Google-a, Facebook-a, IBM-a i Amazon-a, Apple takođe troši ogromna finansijska sredstva na upotrebu i dalji razvoj mašinskog učenja. IBM-ov super računar Watson i dalje je najpoznatiji uređaj za mašinsko učenje. Watson se uglavnom koristi u medicinskom i finansijskom sektoru. Kao što je već pomenuto, Facebook koristi mašinsko učenje za prepoznavanje slika, Microsoft za sistem za prepoznavanje govora "Cortana", Apple za "Siri". Naravno, mašinsko učenje se koristi i u Google-u, kako u oblasti usluga slika, tako i u rangiranju pretraživača.

provajderi u oblaku kao što su Google, Microsoft, AWS i IBM sada su stvorili usluge za mašinsko učenje. Uz njihovu pomoć programeri koji nemaju određeno znanje o mašinskom učenju takođe mogu da razvijaju aplikacije. Ove aplikacije mogu da uče iz slobodno definisanog skupa podataka. U zavisnosti od dobavljača, ove platforme imaju različita imena:

- IBM: Watson
- Amazon: Amazonsko mašinsko učenje
- Microsoft: Azure ML Studio
- Google: Tensorflow

Pored gore pomenutih platformi, postoji širok spektar visokokvalitetnih, besplatnih programa otvorenog koda kroz koje je mašinsko učenje učinjeno dostupnim široj publici, tkao da vi kao programer ili specijalista za podatke možete sa njima raditi.

1.2. K-nearest neighbors(K-NN) : Moćan algoritam mašinskog učenja

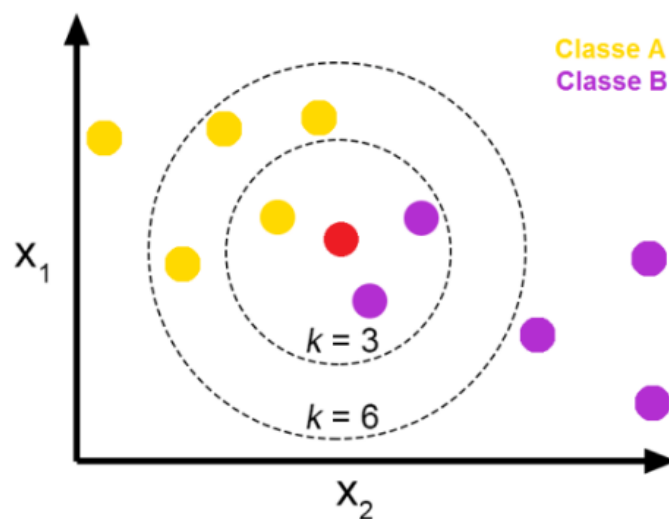
1.2.1. Šta je K-NN?

Najjednostavnija analogija ovog algoritma jeste čuvena rečenica "Reci mi sa kojim prijateljima se družiš i rećiću ti ko si". KNN je neparametrijski i "lazy learning" algoritam. Neparametrijski znači da ne postoji pretpostavka za osnovnu distribuciju podataka, odnosno struktura modela određuje se iz skupa podataka, a "lenji algoritam" jer mu za generisanje modela nisu potrebne tačke podataka o obuci. Svi

podaci o treningu koriste se u fazi testiranja što trening čini bržim, a faza testiranja sporijim i skupljim. On čuva sve dostupne slučajeve i klasifikuje nove podatke ili slučajeve na osnovu mere sličnosti.

1.2.2. Klasifikacija K-NN

U klasifikaciji K-NN, izlaz je članstvo u klasi. Objekat se klasifikuje sa više glasova od svojih suseda, sav objekat je dodeljen klasi najčešće među svojim k najbližim susedima (K je pozitivan ceo broj). Ako je $k = 1$, onda se objekat jednostavno dodeljuje klasi tog najbližeg suseda. Da bi se utvrdilo koje su K instance u skupu podataka o obuci najbližije novom ulazu, koristi se mera udaljenosti (Slika 1). Za ulazne promenljive sa stvarnom vrednošću najpopularnija mera udaljenosti je Euklidova udaljenost.



Slika 1. Crvena tačka je klasifikovana u klasu koja je najčešća među najbližim komšijama

1.2.3. Euklidova udaljenost

Euklidska udaljenost je najčešća metrika rastojanja koja se koristi u skupovima podataka sa niskim dimenzijama. Takođe je poznata i kao norma L_2 . Euklidska udaljenost je uobičajeni način merenja udaljenosti u stvarnom svetu. Gde su p i k n -dimenzionalni vektori i označeni sa $p = (n_1, n_2, \dots, n_p)$ i $k = (k_1, k_2, \dots, k_n)$ predstavljaju n vrednosti atributa dva zapisa.

Iako je euklidska udaljenost korisna u malim dimenzijama, ne funkcioniše dobro u velikim dimenzijama i za kategorijalne promenljive. Nedostatak euklidove distance je u tome što ignoriše sličnosti između atributa. Svaki atribut se tretira kao potpuno različit od svih atributa.

Ostale popularne mere daljine:

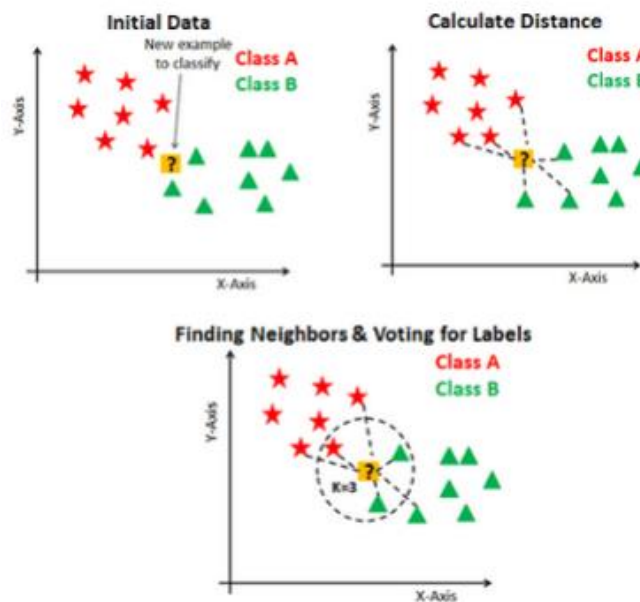
- **Hamming Distance:** Izračunavanje udaljenosti između binarnih vektora.

- **Manhattan Distance:** Izračunavanje udaljenosti između stvarnih vektora koristeći zbir njihove apsolutne razlike. Naziva se i **City Block Distance**.
- **Minkowski Distance:** Generalizacija između Euklida i Manhattan distance

1.2.4. Koraci koje treba izvršiti tokom K-NN algoritma

1. Podeliti podatke na podatke o obuci i testiranju.
2. Izabrati vrednost K
3. Odrediti koju funkciju rastojanja treba koristiti
4. Izabrati uzorak iz podataka testa koji treba klasifikovati i izračunati udaljenost od njegovih n uzoraka za obuku
5. Poređati dobijene udaljenosti i uzeti k-najbliže uzorke podataka
6. Dodeliti test klasu klasi na osnovu većine glasova k suseda

(Slika2)



Slika2. Koraci u KNN algoritmu

1.2.5. Uticaj na performanse K-NN algoritma

1. Funkcija rastojanja ili metrika rastojanja koja se koristi za određivanje najbližih suseda.
2. Pravilo odluka korišćene za izvođenje klasifikacije iz K-najbližih suseda.
3. Broj suseda se koristi za klasifikaciju novih primera.

1.2.6 Prednosti K-NN

1. K-NN algoritama je vrlo jednostavan za implementaciju.
2. Gotovo optimalno u velikom ograničenju uzoraka.
3. Koristi lokalne informacije koje mogu dovesti do visoko prilagodljivog ponašanja.
4. Veoma se lako podvrgava paralelnoj primeni.

1.2.7. Mane K-NN

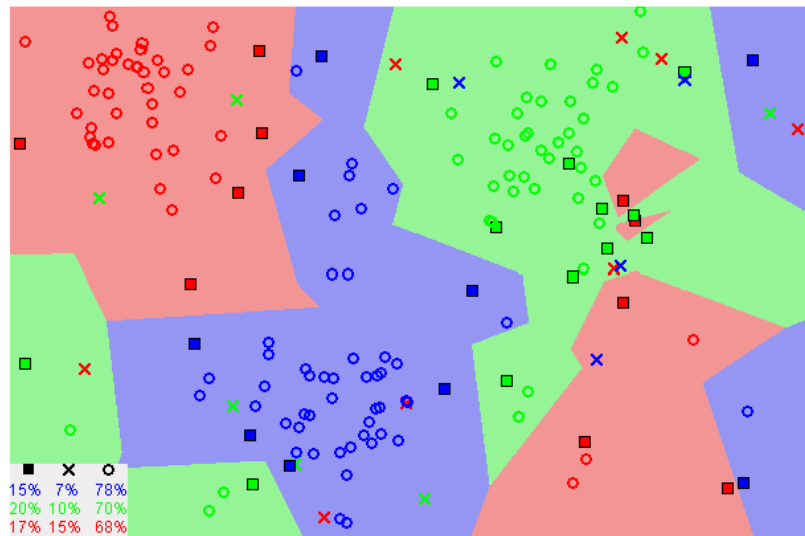
1. Veliki zahtevi za skladištenje.
2. računarki intenzivan opoziv.
3. Vema podložan problemu dimenzionalnosti.

1.2.8. Oblasti u kojim se primenjuje algoritam

1. Finansije – finansijski instituti će predvideti kreditni rejting kupaca.
2. Zdravstvo – Ekspresija gena.
3. Političke nauke – klasifikacija potencijalnih glasača u dve klase glasaće ili neće.
4. Otkrivanja rukopisa.
5. Prepoznavanje slika.
6. Video identifikacija.
7. Šablonska identifikacija.

1.2.9. ‘‘Birds of a feather flock together’’

KNN algoritam pretpostavlja da slične stvari postoje u neposrednoj blizini. Drugim rečima, slične stvari su blizu jedna drugoj.



Slika3. Prikaz kako slične tačke obično slične tačke podataka obično postoje blizu jedna drugoj

Na slici iznad (Slika3) može se primetiti da su slične tačke podataka uglavnom bliske jedna drugoj. KNN algoritam zavisi od ove pretpostavke da je dovoljno istinita da algoritam bude koristan. KNN beleži ideju sličnosti (koja se ponekad naziva i udaljenost ili blizina) sa nekom matematikom koju smo možda naučili u detinjstvu – izračunavanjem udaljenosti između tačaka na grafikonu.

2. Primer/studija slučaja primene izabrane teme

2.1. Skup podataka o vrstama cvetova Irisa

Skup podataka o cvetovima irisa uključuje predviđanje vrsta cveća na osnovu merenja cvetova irisa.

To je problem klasifikacije više klasa. Broj zapažanja za svaki razred je uravnotežen. Postoji 150 posmatranja sa 4 ulazne promenljive i 1 izlaznom promenljivom. Ulazne promenljive su sledeće:

- Dužina čašićnog listića u cm.
- Širina čašićnog listića u cm.
- Dužina latica u cm.
- Širina latica u cm.
- Klasa

Osnovne performanse problema su približno 33%.

Prvo ćemo razviti svaki deo algoritma u ovom odeljku, a zatim ćemo sve elemente povezati u radnu implementaciju primenjenu na stvarni skup podataka u sledećem odeljku.

Ovaj K-NN vodič je podeljen na 3 dela:

- **Korak 1:** Izračunati euklidsku udaljenost.
- **Korak 2:** Naći najbliže komšije.
- **Korak 3:** Napraviti predviđanja.

Ovi koraci će vas naučiti osnovama primene i primene algoritma K-NN za probleme klasifikovanja i regresijskog prediktivnog modeliranja.

Za implementaciju ovog problema korišće se programski jezik Python 3.

2.1.1. Korak 1: Izračunajte euklidsku udaljenost

Prvi korak je izračunavanje udaljenosti između dva reda u skupu podataka. Redovi podataka uglavnom se sastoje od brojeva, a jednostavan način izračunavanja udaljenosti između dva reda ili vektora brojeva je crtanje prave linije. Ovo ima smisla u 2D ili 3D i lepo se skalira do većih dimenzija.

Ravnu rastojanja možemo izračunati između dva vektora pomoću Euklidove mere rastojanja. Izračunava se kao kvadratni koren iz zbega kvadratnih razlika između dva vektora.

- Formula Euklidove udaljenosti: $\sqrt{\sum_{i=1}^N (x1_i - x2_i)^2}$

Gde je $k1$ prvi red podataka, $k2$ je drugi red podataka, a i indeks određene kolone dok zbrajamo sve kolone.

Sa Euklidovom udaljenošću, što je vrednost manja, to će biti više slična dva zapisa. Vrednost 0 znači da nema razlike između dva zapisa(Slika4).

```
1 import math
2 #Proracun Euklidove udaljenosti izmedju dva vektora
3 def euclidean_distance(row1, row2):
4     distance = 0.0
5     for i in range(len(row1)-1):
6         distance += (row1[i] - row2[i])**2
7     return math.sqrt(distance)
```

Slika4. Euklidova funkcija za izračunavanje udaljenosti između dva vektora

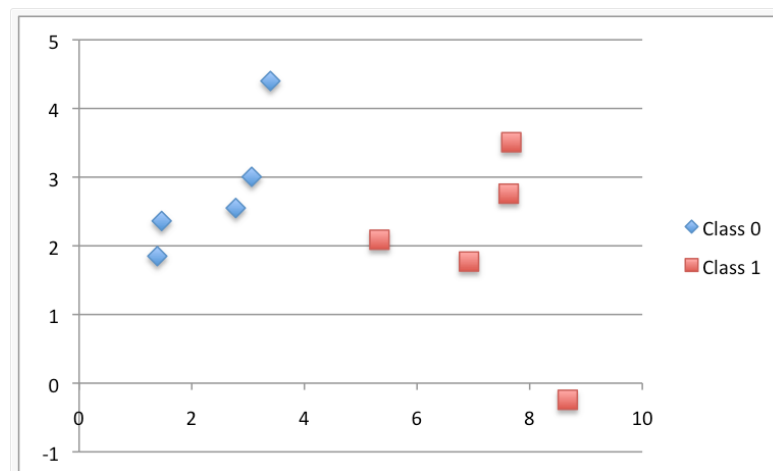
Može se videti da funkcija pretpostavlja da je poslednja kolona u svakom redu izlazna vrednost koja se zanemaruje pri izračunavanju udaljenosti.

Ovu funkciju udaljenosti možemo testirati pomoću malog izmišljenog skupa podataka o klasifikaciji. Ovaj skup podataka korišćemo nekoliko puta dok kontruišemo elemente potrebne za K-NN algoritam(Slika5).

X1	X2	Y
2.7810836	2.550537003	0
1.465489372	2.362125076	0
3.396561688	4.400293529	0
1.38807019	1.850220317	0
3.06407232	3.005305973	0
7.627531214	2.759262235	1
5.332441248	2.088626775	1
6.922596716	1.77106367	1
8.675418651	-0.242068655	1
7.673756466	3.508563011	1

Slika5. Kreiran skup podataka

Ispod je grafikon skupa podataka koji koristi različite boje za prikaz različitih klasa za svaku tačku(Slika6).



Slika6. Raseijana parcela malog izmišljenog skupa podataka za testiranje K-NN algoritma

Sastavljajući ovo sve zajedno, možemo napisati mali primer kako bismo testirali našu funkciju rastojanja ispisujući rastojanje između prvog reda i svih ostalih redova. Očekivali bismo da je udaljenost između prvog reda i samog njega 0, na šta treba dobro povesti računa.

Na slici ispod predstavljen je kompletan primer u nastavku(Slika7):

```

# Proracun Euklidove udaljenosti izmedu dva vektora
def euclidean_distance(row1, row2):
    distance = 0.0
    for i in range(len(row1) - 1):
        distance += (row1[i] - row2[i]) ** 2
    return sqrt(distance)

# Testiranje udaljenosti
dataset = [[2.7810836, 2.550537003, 0],
           [1.465489372, 2.362125076, 0],
           [3.396561688, 4.400293529, 0],
           [1.38807019, 1.850220317, 0],
           [3.06407232, 3.005305973, 0],
           [7.627531214, 2.759262235, 1],
           [5.332441248, 2.088626775, 1],
           [6.922596716, 1.77106367, 1],
           [8.675418651, -0.242068655, 1],
           [7.673756466, 3.508563011, 1]]
row0 = dataset[0]
for row in dataset:
    distance = euclidean_distance(row0, row)
    print(distance)

```

Slika7. Predstavljena koda celog primera

Pokretanje ovog primera ispisuje razdaljine između prvog reda i svakog reda u skupu podataka(Slika8).

```

Run: main x
0.0
1.3290173915275787
1.9494646655653247
1.5591439385540549
0.5356280721938492
4.850940186986411
2.592833759950511
4.214227042632867
6.522409988228337
4.985585382449795

Process finished with exit code 0

```

Slika8.Rezultati predstavljenog primera

Sada sa Euklidovom udaljenošću možemo da lociramo najbliže u skupu podataka.

2.1.2. Korak 2: Naći najbliže komšije

Komšije za novi podatak u skupu podataka su k najbliže instance, kako je definisano našom merom udaljenosti.

Da bismo locirali komšije za novi komad podataka u skupu podataka, prvo moramo izračunati udaljenost između svakog zapisa u skupu podataka do novog dela podataka. To možemo učiniti pomoću naše funkcije daljine pripremljene gore.

Jednom kada se izračunaju udaljenosti, moramo sve zapise u skupu podataka o treningu sortirati prema njihovoj udaljenosti od novih podataka. Tada možemo odabrati vrh k da se vratimo kao najbližiji susedi.

To možemo da uradimo tako što ćemo evidentirati udaljenost za svaki zapis u skupu podataka kao skup, sortirati listu korpi prema udaljenosti (u opadajućem redosledu), a zatim potražiti komšije.

Ispod je funkcija nazvana `get_neighbors` koja ovo implementira:

```
29 # Locirati najsljednije komšije
30 def get_neighbors(train, test_row, num_neighbors):
31     distances = list()
32     for train_row in train:
33         dist = euclidean_distance(test_row, train_row)
34         distances.append((train_row, dist))
35     distances.sort(key=lambda tup: tup[1])
36     neighbors = list()
37     for i in range(num_neighbors):
38         neighbors.append(distances[i][0])
39     return neighbors
40
```

Slika9. Funkcija koja locira najsljednije komšije

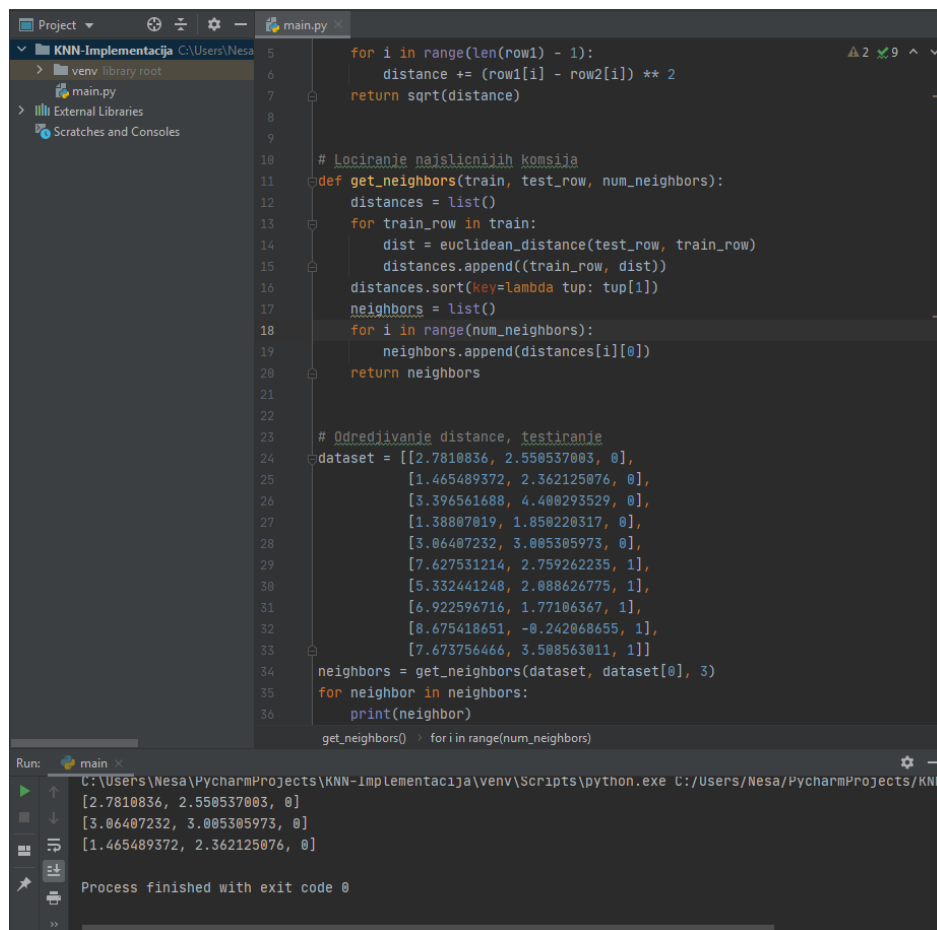
Možemo da vidimo se se funkcija `euclidean_distance` razvijena u prethodnom koraku za izračunavanje udaljenosti između svake `test_row` i `train_row`.

Lista `train_row` i razdaljina razdvajanja sortirana je tamo gde se koristi prilagođeni ključ koji osigurava da se druga stavka u listi (tuple(`tup[1]`)) koristi u operaciji sortiranja.

Konačno, vraća se lista `num_neighbors` najsljednijih suseda liste `test_row`.

Ovu funkciju možemo testirati pomoću malog izmišljenog skupa podataka pripremljenog u prethodnom odeljku.

Kompletni primer je naveden ispod (Slika10)



```
Project
venv library root
main.py
External Libraries
Scratches and Consoles

5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36

for i in range(len(row1) - 1):
    distance += (row1[i] - row2[i]) ** 2
return sqrt(distance)

# Lociranje najsljednijih komšija
def get_neighbors(train, test_row, num_neighbors):
    distances = list()
    for train_row in train:
        dist = euclidean_distance(test_row, train_row)
        distances.append((train_row, dist))
    distances.sort(key=lambda tup: tup[1])
    neighbors = list()
    for i in range(num_neighbors):
        neighbors.append(distances[i][0])
    return neighbors

# Određivanje distance, testiranje
dataset = [[2.7810836, 2.550537003, 0],
           [1.465489372, 2.362125076, 0],
           [3.396561688, 4.400293529, 0],
           [1.38807019, 1.850220317, 0],
           [3.06407232, 3.005305973, 0],
           [7.627531214, 2.759262235, 1],
           [5.332441248, 2.088626775, 1],
           [6.922596716, 1.77106367, 1],
           [8.675418651, -0.2428068655, 1],
           [7.673756466, 3.508563011, 1]]

neighbors = get_neighbors(dataset, dataset[0], 3)
for neighbor in neighbors:
    print(neighbor)

get_neighbors() > for i in range(num_neighbors)

Run:
C:\Users\Nesa\PycharmProjects\KNN-Implementacija\venv\Scripts\python.exe C:\Users\Nesa\PycharmProjects\KNN-Implementacija\main.py
[2.7810836, 2.550537003, 0]
[3.06407232, 3.005305973, 0]
[1.465489372, 2.362125076, 0]
Process finished with exit code 0
```

Slika 10. Kompletna primena Koraka 1 i Koraka 2

Pokretanje ovog primera ispisuje 3 najsljednija zapisa u skupu podataka na prvi zapis, po redosledu sljednosti.

Kao što se i očekivalo, prvi zapis je najsljedniji sebi i nalazi se na vrhu liste.

Sada kada znamo kako da dobijemo komšije iz skupa podataka, možemo ih koristiti za predviđanje.

2.1.3. Korak 3: Napraviti predviđanja

Najsljednije komšije prikupljene iz skupa podataka o obuci mogu se koristiti za predviđanje. U slučaju klasifikacije, možemo vratiti najzastupljenije klasu među komšije.

To možemo postići izvođenjem funkcije *max()* na listi izlaznih vrednosti suseda. S obzirom na listu vrednosti klasa koje se primećuju u susedima, funkcija *max()* uzima skup jedinstvenih vrednosti klase i poziva brojanje na listi vrednosti klase za svaku vrednost klase u skupu.

Ispod je funkcija *predict_classification()* koja ovo implementira (Slika 11):

```

26 # Napraviti klasifikacijsko predviđanje sa komsijama
27 def predict_classification(train, test_row, num_neighbors):
28     neighbors = get_neighbors(train, test_row, num_neighbors)
29     output_values = [row[-1] for row in neighbors]
30     prediction = max(set(output_values), key=output_values.count)
31     return prediction

```

Slika11. Funkcija koja definiše klasifikacijsko predviđanje sa komsijama

Ovu funkciju možemo testirati nad gore zadatim skupom podataka i upotpuniti naš primer(Slika12).

```

4 def euclidean_distance(row1, row2):
5     distance = 0.0
6     for i in range(len(row1) - 1):
7         distance += (row1[i] - row2[i]) ** 2
8     return sqrt(distance)
9
10
11 # Locirati najslicnije komsije
12 def get_neighbors(train, test_row, num_neighbors):
13     distances = list()
14     for train_row in train:
15         dist = euclidean_distance(test_row, train_row)
16         distances.append((train_row, dist))
17     distances.sort(key=lambda tup: tup[1])
18     neighbors = list()
19     for i in range(num_neighbors):
20         neighbors.append(distances[i][0])
21     return neighbors
22
23
24 # Napraviti klasifikacijsko predviđanje sa komsijama
25 def predict_classification(train, test_row, num_neighbors):
26     neighbors = get_neighbors(train, test_row, num_neighbors)
27     output_values = [row[-1] for row in neighbors]
28     prediction = max(set(output_values), key=output_values.count)
29     return prediction
30
31
32 # Testiranje distance
33 dataset = [[(2.7810836, 2.550537003, 0),
34             (1.465489372, 2.362125076, 0),
35             (3.396561688, 4.400293529, 0),

```

Run: main

C:\Users\Nesa\PycharmProjects\KNN-Implementacija\venv\Scripts\python.exe C:/Users/Nesa/PycharmProjects/KNN-
Expected 0, Got 0.

Process finished with exit code 0

Slika12. Testiranje dodatog Koraka 3

Izvođenje ovog primera ispisuje očekivanu klasifikaciju od 0 i stvarnu klasifikaciju predviđenu od 3 najslićnija suseda u skupu podataka.

Možemo zamisliti kako se funkcija *predict_classification()* može promeniti kako bi se izračunala srednja vrednost vrednosti ishoda.

Sada imamo sve delove za predviđanje sa K-NN-om. Možemo da ga primenimo na stvarni skup podataka.

2.1.4. Studija slučaja vrste Irida

Ovaj odeljak primenjuje K-NN algoritam nad skupom podataka Iris cvetova. Prvi korak je učitavanje skupa podataka i pretvaranje učitanih podataka u brojeve koje možemo koristiti sa proračunima srednje i standardne devijacije. Za ovo ćemo koristiti pomoćnu funkciju *load_csv()* za učitavanje datoteke, *str_column_to_float()* za pretvaranje brojeva nizova u tip podataka *float* i *str_column_to_int()* za pretvaranje kolone klasa u celobrojne vrednosti.

Algoritam ćemo proceniti pomoću k-fold unakrsne provere sa 5 proklopa. To znači da će $150/5 = 30$ zapisa biti u svakom perklopu. Koristićemo pomoćne funkcije *evaluate_algorithm()* za procenu algoritma sa unakrsnom validacijom i *accuracy_metric()* za izračunavanje tačnosti predviđanja.

Nova funkcija pod nazivom *k_nearest_neighbors()* je razvijena za upravljanje primenom K-NN algoritma, prvo učeći statistiku iz skupa podataka za obuku i koristeći ih za predviđanje skupa testova.

Kompletan primer je dat iz delova u nastavku(Slika13, Slika14, Slika15, Slika16):

```
1  from random import seed
2  from random import randrange
3  from csv import reader
4  from math import sqrt
5
6
7  # Ucitavanje CSV fajla
8  def load_csv(filename):
9      dataset = list()
10     with open(filename, 'r') as file:
11         csv_reader = reader(file)
12         for row in csv_reader:
13             if not row:
14                 continue
15             dataset.append(row)
16     return dataset
17
18
19 # Konvertovanje string kolone u float
20 def str_column_to_float(dataset, column):
21     for row in dataset:
22         row[column] = float(row[column].strip())
23
24
25 # Konvertovanje string kolone u celobrojnu vrednost
26 def str_column_to_int(dataset, column):
27     class_values = [row[column] for row in dataset]
28     unique = set(class_values)
29     lookup = dict()
30     for i, value in enumerate(unique):
31         lookup[value] = i
32     for row in dataset:
33         row[column] = lookup[row[column]]
34     return lookup
```

Slika13. Predstavljanje celog koda(Prvi deo)

```

37 # Naci min i max vrednost za svaku kolonu
38 def dataset_minmax(dataset):
39     minmax = list()
40     for i in range(len(dataset[0])):
41         col_values = [row[i] for row in dataset]
42         value_min = min(col_values)
43         value_max = max(col_values)
44         minmax.append([value_min, value_max])
45     return minmax
46
47
48 # Prilagodjavanje kolone skupa podataka na 0-1
49 def normalize_dataset(dataset, minmax):
50     for row in dataset:
51         for i in range(len(row)):
52             row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])
53
54
55 # Podela skupa podataka u k preklopa
56 def cross_validation_split(dataset, n_folds):
57     dataset_split = list()
58     dataset_copy = list(dataset)
59     fold_size = int(len(dataset) / n_folds)
60     for _ in range(n_folds):
61         fold = list()
62         while len(fold) < fold_size:
63             index = randrange(len(dataset_copy))
64             fold.append(dataset_copy.pop(index))
65         dataset_split.append(fold)
66     return dataset_split
67
68
69 # Izracunavanje procene tacnosti
70 def accuracy_metric(actual, predicted):
71     correct = 0
72     for i in range(len(actual)):
73         if actual[i] == predicted[i]:
74             correct += 1
75     return correct / float(len(actual)) * 100.0
76

```

Slika14. Predstavljanje celog koda(Drugi deo)

```

78 # Procena algoritma pomocu unakrsne validacije
79 def evaluate_algorithm(dataset, algorithm, n_folds, *args):
80     folds = cross_validation_split(dataset, n_folds)
81     scores = list()
82     for fold in folds:
83         train_set = list(folds)
84         train_set.remove(fold)
85         train_set = sum(train_set, [])
86         test_set = list()
87         for row in fold:
88             row_copy = list(row)
89             test_set.append(row_copy)
90             row_copy[-1] = None
91         predicted = algorithm(train_set, test_set, *args)
92         actual = [row[-1] for row in fold]
93         accuracy = accuracy_metric(actual, predicted)
94         scores.append(accuracy)
95     return scores
96
97
98 # Proracun Euklidove udaljenosti izmedju dva vektora
99 def euclidean_distance(row1, row2):
100     distance = 0.0
101     for i in range(len(row1) - 1):
102         distance += (row1[i] - row2[i]) ** 2
103     return sqrt(distance)
104
105
106 # Locirati najslabije komsije
107 def get_neighbors(train, test_row, num_neighbors):
108     distances = list()
109     for train_row in train:
110         dist = euclidean_distance(test_row, train_row)
111         distances.append((train_row, dist))
112     distances.sort(key=lambda tup: tup[1])
113     neighbors = list()
114     for i in range(num_neighbors):
115         neighbors.append(distances[i][0])
116     return neighbors

```

Slika15. Predstavljanje celog koda(Treći deo)

```

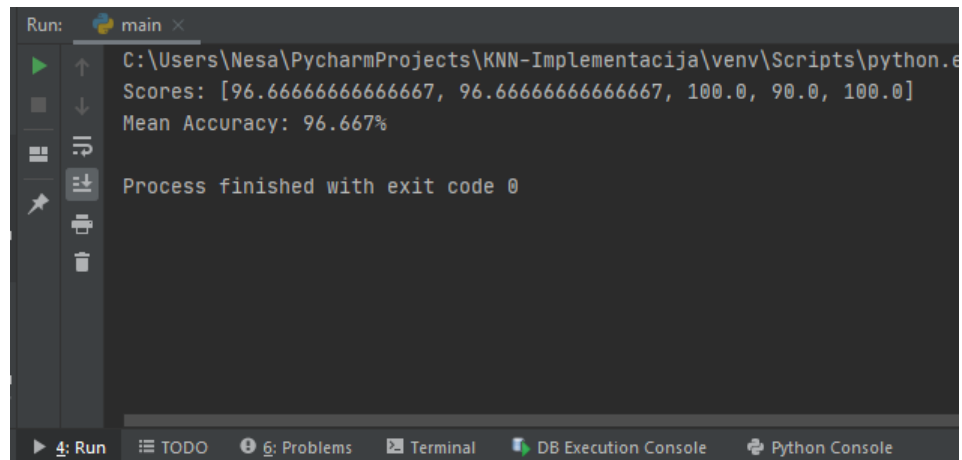
119 # Pravljenje predviđanja sa komsijama
120 def predict_classification(train, test_row, num_neighbors):
121     neighbors = get_neighbors(train, test_row, num_neighbors)
122     output_values = [row[-1] for row in neighbors]
123     prediction = max(set(output_values), key=output_values.count)
124     return prediction
125
126
127 # KNN algoritam
128 def k_nearest_neighbors(train, test, num_neighbors):
129     predictions = list()
130     for row in test:
131         output = predict_classification(train, row, num_neighbors)
132         predictions.append(output)
133     return predictions
134
135
136 # testiranje KNN algoritma nad Iris cvetovima skupa podataka
137 seed(1)
138 filename = 'iris.csv'
139 dataset = load_csv(filename)
140 for i in range(len(dataset[0]) - 1):
141     str_column_to_float(dataset, i)
142 # Konvertovanje klasne kolone u celobrojnu vrednost
143 str_column_to_int(dataset, len(dataset[0]) - 1)
144 # algoritam procene
145 n_folds = 5
146 num_neighbors = 5
147 scores = evaluate_algorithm(dataset, k_nearest_neighbors, n_folds, num_neighbors)
148 print('Scores: %s' % scores)
149 print('Mean Accuracy: %.3f%%' % (sum(scores) / float(len(scores))))

```

Slika16. Predstavljanje celog koda(Četvrti deo)

Pokretanjem primera ispisuje srednje ocene tačnosti klasifikacije na svakom prelazu unakrsne provere, kao i srednju ocenu tačnosti.

Možemo videti da je srednja tačnostod oko 96,6% dramatično bolja od osnovne tačnosti od 33%(Slika17):



```

Run: main x
C:\Users\Nesa\PycharmProjects\KNN-Implementacija\venv\Scripts\python.exe
Scores: [96.66666666666667, 96.66666666666667, 100.0, 90.0, 100.0]
Mean Accuracy: 96.667%

Process finished with exit code 0

```

Slika17. Predstavljanje rešenja, gore postavljenog zadatka

Skup podataka za obuku možemo koristiti za predviđanje novih zapažanja(redovi podataka):

To uključuje upućivanje poziva funkciji `predict_classification()` sa redom koji predstavlja naše zapažanje za predviđanje oznake klase.

Takođe bismo možda želeli da znamo oznaku klase (niz) za predviđanje.

Možemo ažurirati funkciju `str_column_to_int()` tako da štampa mapiranje imena klasa nizova za cele brojeve kako bismo mogli da protumačimo predviđanje koje je napravio model(Slika18).

```
157 # Pretvaranje stringa u celobrojni izraz
158 def str_column_to_int(dataset, column):
159     class_values = [row[column] for row in dataset]
160     unique = set(class_values)
161     lookup = dict()
162     for i, value in enumerate(unique):
163         lookup[value] = i
164         print('[%s] => %d' % (value, i))
165     for row in dataset:
166         row[column] = lookup[row[column]]
167     return lookup
```

Slika18. Izmena funkcije za pretvaranje stringa u celobrojni izraz

Povezujući ovo, u nastavku je naveden potpun primer korišćenja KNN-a sa celim skupom podataka i pravljenja jednog predviđanja za novo posmatranje(Slika19, Slika20, Slika21, Slika22).

```
1 # Poređenje sa najbližim susedima nad skupom podataka Iris.csv 21 57 ^
2 from csv import reader
3 from math import sqrt
4
5
6 # Ucitavanje fajla csv
7 def load_csv(filename):
8     dataset = list()
9     with open(filename, 'r') as file:
10         csv_reader = reader(file)
11         for row in csv_reader:
12             if not row:
13                 continue
14             dataset.append(row)
15     return dataset
16
17
18 # Konvertovanje string kolone u tip podataka float
19 def str_column_to_float(dataset, column):
20     for row in dataset:
21         row[column] = float(row[column].strip())
22
```

Slika19. Predstavljanje celog koda(Prvi deo)

```

24 # Konvertovanje string kolone u celobrojne vrednosti
25 def str_column_to_int(dataset, column):
26     class_values = [row[column] for row in dataset]
27     unique = set(class_values)
28     lookup = dict()
29     for i, value in enumerate(unique):
30         lookup[value] = i
31         print('[%s] => %d' % (value, i))
32     for row in dataset:
33         row[column] = lookup[row[column]]
34     return lookup
35
36
37 # Naci min i max vrednost za svaku kolonu
38 def dataset_minmax(dataset):
39     minmax = list()
40     for i in range(len(dataset[0])):
41         col_values = [row[i] for row in dataset]
42         value_min = min(col_values)
43         value_max = max(col_values)
44         minmax.append([value_min, value_max])
45     return minmax
46

```

Slika20. Predstavljanje celog koda(Drugi deo)

```

48 # Prilagodjavanje razmera kolona skupa podataka u opseg 0-1
49 def normalize_dataset(dataset, minmax):
50     for row in dataset:
51         for i in range(len(row)):
52             row[i] = (row[i] - minmax[i][0]) / (minmax[i][1] - minmax[i][0])
53
54
55 # Proracun Euklidove udaljenosti izmedju dva vektora
56 def euclidean_distance(row1, row2):
57     distance = 0.0
58     for i in range(len(row1) - 1):
59         distance += (row1[i] - row2[i]) ** 2
60     return sqrt(distance)
61
62
63 # Lociranje najblizih suseda
64 def get_neighbors(train, test_row, num_neighbors):
65     distances = list()
66     for train_row in train:
67         dist = euclidean_distance(test_row, train_row)
68         distances.append((train_row, dist))
69     distances.sort(key=lambda tup: tup[1])
70     neighbors = list()
71     for i in range(num_neighbors):
72         neighbors.append(distances[i][0])
73     return neighbors

```

Slika21. Predstavljanje celog koda(Treći deo)

```

76 # predviđanje sa susedima
77 def predict_classification(train, test_row, num_neighbors):
78     neighbors = get_neighbors(train, test_row, num_neighbors)
79     output_values = [row[-1] for row in neighbors]
80     prediction = max(set(output_values), key=output_values.count)
81     return prediction
82
83
84 # Pravljanje predviđanja pomocu KNN-a nad skupom podataka Iriisa
85 filename = 'iris.csv'
86 dataset = load_csv(filename)
87 for i in range(len(dataset[0]) - 1):
88     str_column_to_float(dataset, i)
89 # Konvertovanje klasne kolone u celobrojnu vrednost
90 str_column_to_int(dataset, len(dataset[0]) - 1)
91 # definisanje parametra modela
92 num_neighbors = 5
93 # definisanje novog zapisa
94 row = [5.7, 2.9, 4.2, 1.3]
95 # predviđanje labela
96 label = predict_classification(dataset, row, num_neighbors)
97 print('Data=%s, Predicted: %s' % (row, label))

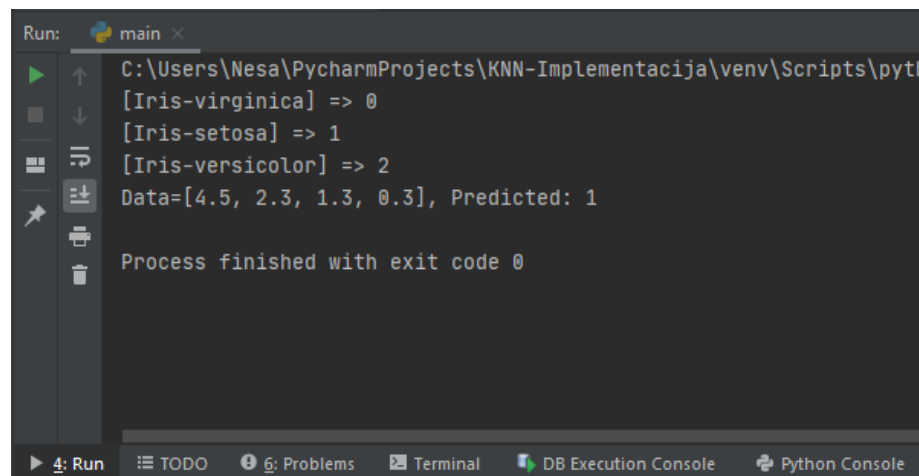
```

Slika22. Predstavljanje celog koda (Četvrti deo)

Pokretanjem podataka prvo sažima mapiranje oznaka klasa na cele brojeve, a zatim uklapa model u ceo skup podataka.

Tada se definiše novo zapažanje (u ovom slučaju uzео sam red iz skupa podataka) i izračunava se predviđena oznaka.

U ovom slučaju se predviđa da naše zapažanje pripada 1 za koju znamo da je "Iris-setosa" (Slika23).



```

Run: main x
C:\Users\Nesa\PycharmProjects\KNN-Implementacija\venv\Scripts\python.exe
[Iris-virginica] => 0
[Iris-setosa] => 1
[Iris-versicolor] => 2
Data=[4.5, 2.3, 1.3, 0.3], Predicted: 1
Process finished with exit code 0

```

Slika23. Predstavljanje rešenja sa imenima

3. Zaključak

U ovom projektu bilo je reči o tome kako da se primeni KNN algoritam ispočetka u Pythonu. Konkretno predstavljeno je na koji način kodirati algoritam K-NN korak po korak. Kako proceniti K-NN nad stvarnim skupom podataka i kako koristiti K-NN za predviđanje novih podataka.

4. Literatura

1. https://link.springer.com/chapter/10.1007/978-3-642-38652-7_2
2. <https://ieeexplore.ieee.org/abstract/document/1672890>
3. <https://link.springer.com/chapter/10.1007/BFb0033288>
4. https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm
5. <https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>
6. <https://scikit-learn.org/stable/modules/neighbors.html>
7. <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>
8. <https://www.geeksforgeeks.org/k-nearest-neighbours/>
9. https://www.saedsayad.com/k_nearest_neighbors.htm
10. <https://www.unite.ai/what-is-k-nearest-neighbors/>