VISOKA ŠKOLA ELEKTROTEHNIKE I RAČUNARSTVA STRUKOVNIH STUDIJA

BEOGRAD

PROJEKTNI ZADATAK

PRVI DEO

Predmet: Veštačka inteligencija

Profesor: Emilija Kisić

Datum: *6.6.2021*

Student: **Grupa:**

Aleksa Aleksić *Danijel Stokić RT-74/19*

RT-9/19

Zadatak 1

(Bajesovski klasifikator)

Osnovni cilj klasifikacija jeste da posmatrajuci uzorak, po atributima ili merenjima, mozemo da odlučimo da li nekoj klasi neki odredjni objekat pripada. Najlakse bi bilo objasniti sa obilicima (kocka, krug, elipsa). Cilj klasifikacija jeste odredjene oblike razedilmo u klase kojima oni pripadaju.

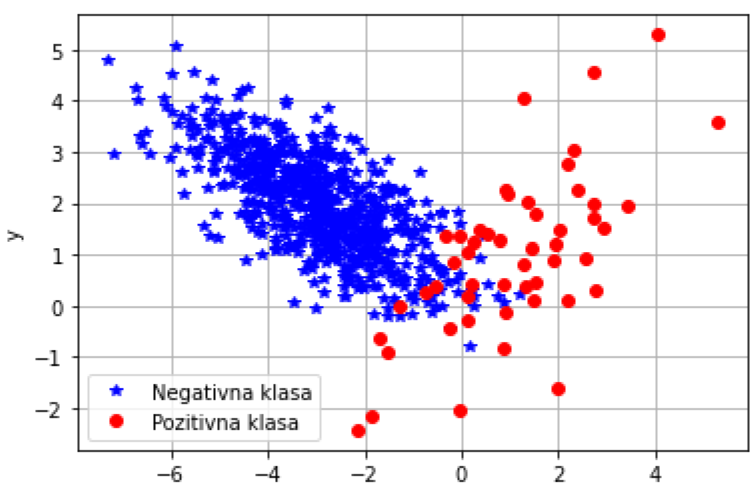
Bajesovski klasifikator se takodje naziva i naivni bajes jer u startu ima predpostavku da atributi klasa nisu zavisni.

Konkretno, što se tiče zadatka, za početak od dobijenih vektora i matrica:

```
M1 = [-3, 2]
Sigma1 = [[2, -1], [-1, 1]]
M2 = [1, 1]
Sigma2 = [[2, 1], [1, 2]]
```

Slika 1. Vektori srednje vrednosti i kovariacijone matrice

Nacrtaćemo rasuti dijagram:



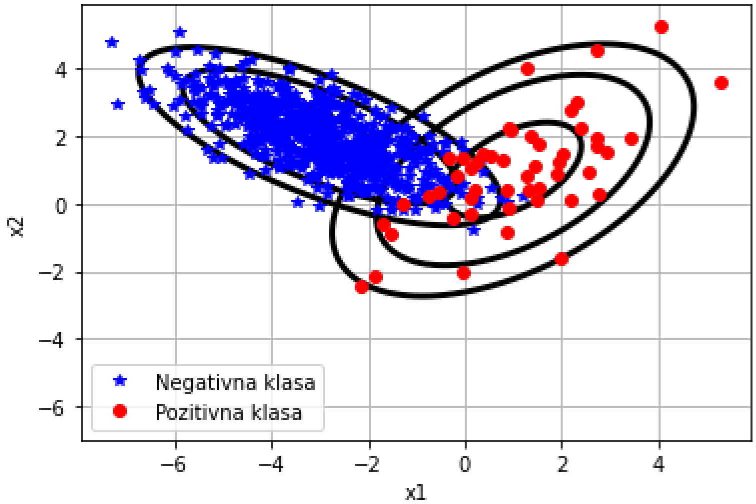
Slika 2. Rasuti dijagram klasa

Na samoj slici odmah možemo da ceključimo da su nam obe klase korelisane.

-Negativna klasa ima negativnu korelaciju (opada)

-Pozitivna klasa ima pozitivnu korelaciju (raste)

Ovo smo mogli da zaključimo i iz sporedne dijagonale kovariacijone matrice sa početka.



Slika 3. Klase sa d^2 krivama.

Na slici 3 pomoču d^2 možemo jasno da vidim da je naš zaključak o korelaciji ispravan.

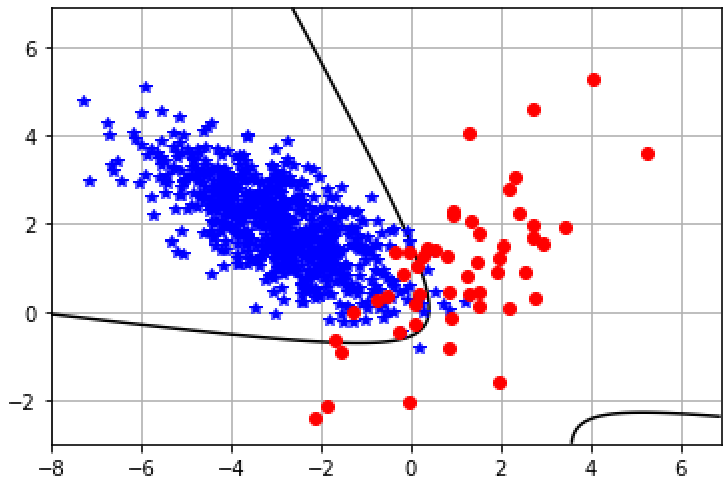
Sada želimo da isprojektujemo bajesovski klasifikator da apriornim verovatnoćama P1 (0.9) i P2 (0.1)

Koristimo formulu sa predavanja za crtanje linije h(x). Takodje, pošto nam kovarijacione matrice nisu iste za obe klase moramo da koristimo proširenu formulu:

$$h(x) = \frac{1}{2}[((X_1 - m_{11})z_{11} + (X_2 - m_{12})z_{21})(X_1 - m_{11}) + ((X_1 - m_{11})z_{12} + (X_2 - m_{12})z_{22})(X_2 - m_{12})] - \frac{1}{2}[((X_1 - m_{21})k_{11} + (X_2 - m_{22})k_{21})(X_1 - m_{21}) + ((X_1 - m_{21})k_{12} + (X_2 - m_{22})k_{22})(X_2 - m_{22})] + \frac{1}{2}\ln\left(\frac{|\Sigma_1|}{|\Sigma_2|}\right) \tag{42}$$

Slika 4. Formula za parabolu h(x)

Pošto nam matrice za obe klase nisu iste sada klasifikator neće biti u obliku linije, več parabole.



Slika 5. Klasifikator sa verovatnoćama 0.9 i 0.1

Iz verovatnoča možemo da zaključimo da se klasifikator „fokusira“ na negativnu klasu, samim time, linija lepo obuhvata odbirke negativne klase, dok imamo i dosta odbiraka pozitivne klase u njoj. Ovo ćemo dokazati i matricom konfuzije:

/	P	N
P	12	38
N	8	692

Slika 6. Matrica konfuzije

Sada vidimo kolike su greske. TP ili tacno klasifikovani odbirci pozitivne klase iznose 12 od ukupno 50 što je dosta mali broj dok je vrednost TN ili tacno klasifikovanih odbiraka negativne klase dosta velika, čak 692 od 700. Ovo i nisu zavidni rezultati, medjutim, ako pogledamo tačnost klasifikatora:

- (P + N) / (P + N + FP + FN)
    - P – broj elemenata u pozitivnoj klasi
    - N – broj elemenata u negativnoj klasi
    - FP – lose klasifikovani elementi pozitivne klase
    - FN – lose klasifikovani elementi negativne klase

Dobijamo rezultat 0.942211...

Ovo je naizgled dobro. Međutim, pošto nam negativna klasa ima mnogo veći broj elemenata u odnosu na pozitivnu klasu a takodje smo videli da je negativna klase dobro klasifikovana, pozitivnu klasu ne uzimamo previše u obzir u toj kalkulaciji. Iz tog razloga ova tačnost nekada nije dobra metrika.

Uvodimo još dve:

- Preciznost (koliko je model pouzdan da predvidi kojoj klasi pripada objekat)
- Odziv (od svih klasifikovanih objekata pozitivne klase, koliko je zapravo pozitivno)

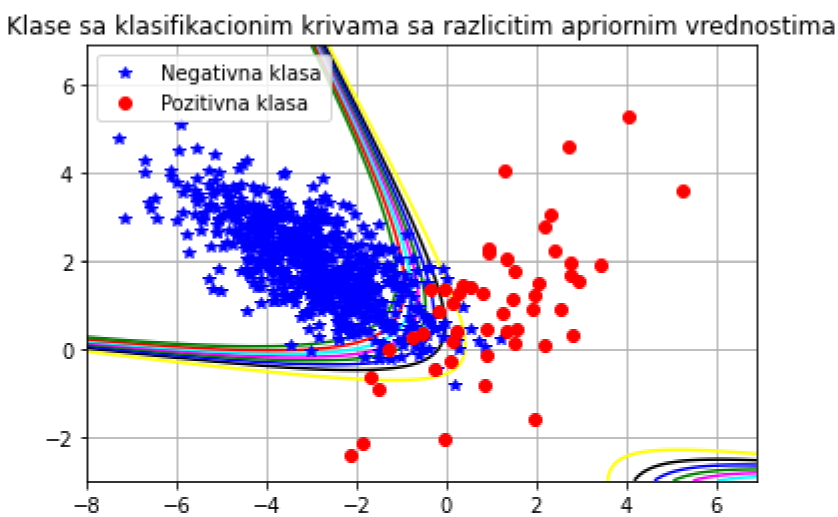
Sada imamo sve vrednosti:

```
Tacnost =  0.9422110552763819
Preciznost =  0.24
Odziv =  0.6
```

Slika 7. Provera tačnosti, tačnost, preciznost i odziv

Sada možemo da sigurnošču da kažemo da nam tačnost nije dobra metrika u ovom slučaju!

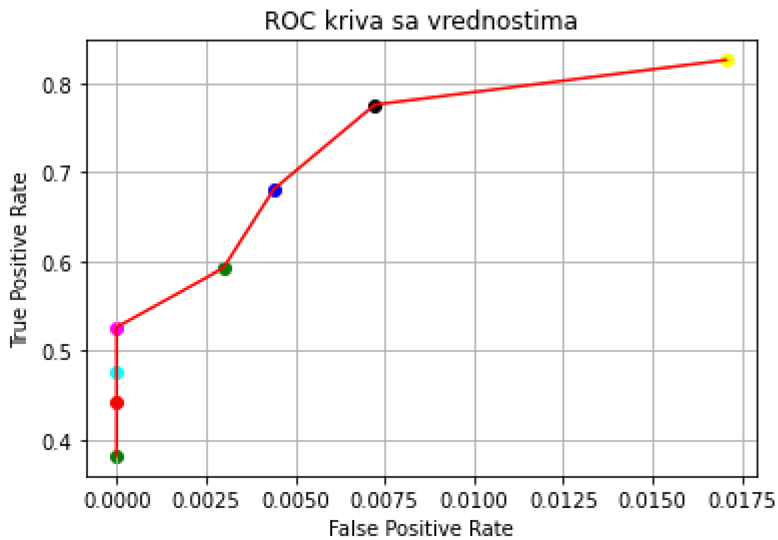
Nakon toga isprojektovaćemo parabole sa ostalim apriornim verovatnoćama. Počinjemo sa već poznatim vrednostima (0.9 i 0.1) i u svakoj iteraciji P1 ćemo smanjivati za 0.1 dok ćemo P2 uvećavati za isto toliko.



Slika 8. Kriva sa različitim verovatnoćama

Kako smo za prvu krivu uzeli žutu boju možemo jasno da vidimo da kako se verovatnoća P1 smanjuje to naš klasifikator počinje da više uzima u obzir elemente pozitivne klase dok respektivno pravi veću grešku nad elementima negativne klase.

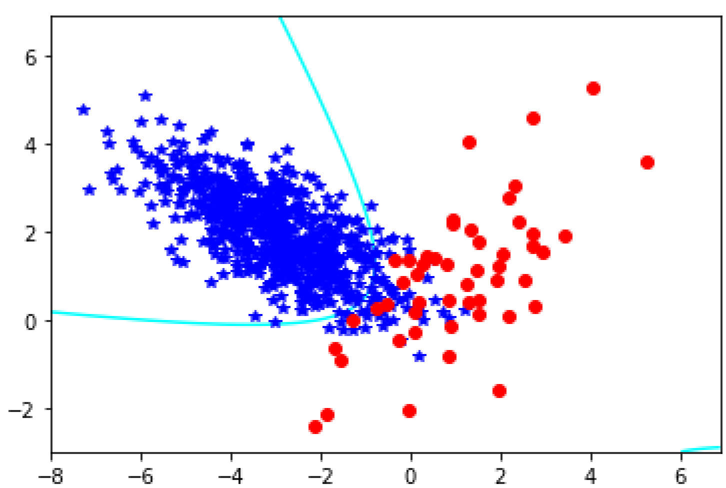
Možemo da prikažemo dijagram sa ROC krivom. ROC kriva nam je najbolja slika problema binarne klasifikacije. Ona predstavlja odnos TPR i FPR gde nam je TPR(True positive rate) metrika koja govori koliko procenat pozitivne klase je dobro klasifikovan dok nam FPR(False positive rate) govori koliko je procenata negativne klase lose klasifikovan ili **koliko procenata negativne klase je predviđeno kao pozitivno.**



Slika 9. ROC kriva

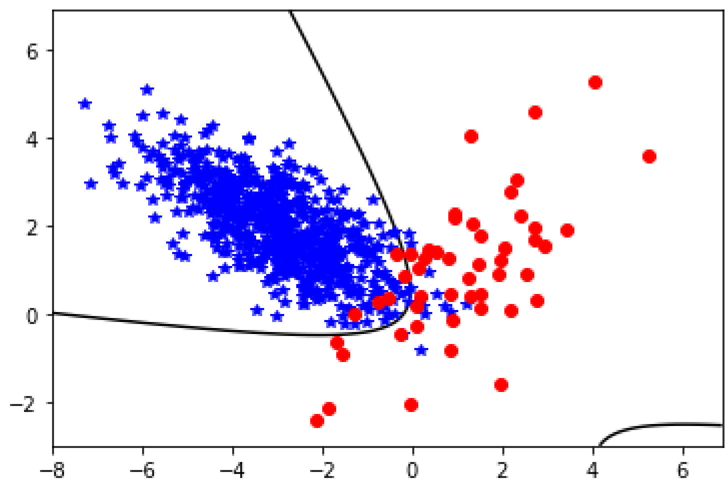
Ovde možemo da primetimo da nam krive zelene, crvene, cian i magenda boje najbolje klasifikuju podatke pozitivne klase jer im je TPR ravan nuli. Takođe možemo da kažemo da bi nam crni klasifikator (crna linija) uopšteno bila najbolja jer ima mali FPR a dosta veliki TPR.

Ako bi nam cilj bio da najbolje klasifikujemo elemente pozitivne klase mogli bi da prikažemo na primer cian krivu:



Slika 10. Klasifikator za pozitivnu klasu

Iz slike 10 možemo da vidimo da je klasifikator sve odbirke pozitivne klase korektno klasifikovano i time imamo FPR = 0 , međutim, vidimo da dosta odbiraka negativne klase ovaj klasifikator svrstava u pozitivne.



Slika 11. Klasifikator sa vrednostima 0.8 i 0.2

Na slici 11 vidimo i crni klasifikator koji bi uopsteno do najbolje rezultate. Odbirci negativne klase su odlično klasifikovani dok imamo malo odbiraka pozitivne klase koji su klasifikovani kao negativni.

Zadatak 2

(Klasifikacija Sklearn novčanice)

Za početak analiziraćemo skup koji smo dobili:

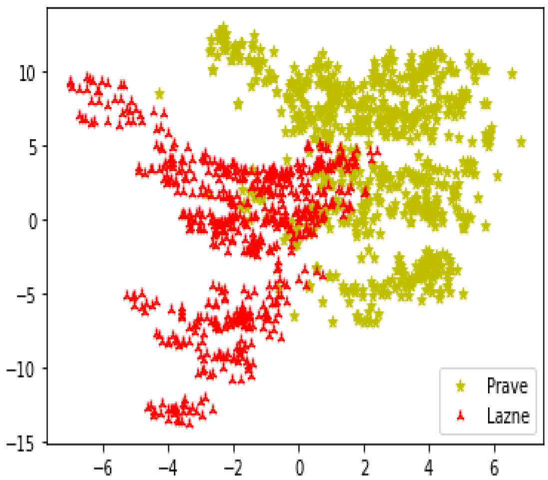
```
Dimenzije skupa = (1372, 5)
```

	0	1	2	3	4
count	1372.000000	1372.000000	1372.000000	1372.000000	1372.000000
mean	0.433735	1.922353	1.397627	-1.191657	0.444606
std	2.842763	5.869047	4.310030	2.101013	0.497103
min	-7.042100	-13.773100	-5.286100	-8.548200	0.000000
25%	-1.773000	-1.708200	-1.574975	-2.413450	0.000000
50%	0.496180	2.319650	0.616630	-0.586650	0.000000
75%	2.821475	6.814625	3.179250	0.394810	1.000000
max	6.824800	12.951600	17.927400	2.449500	1.000000

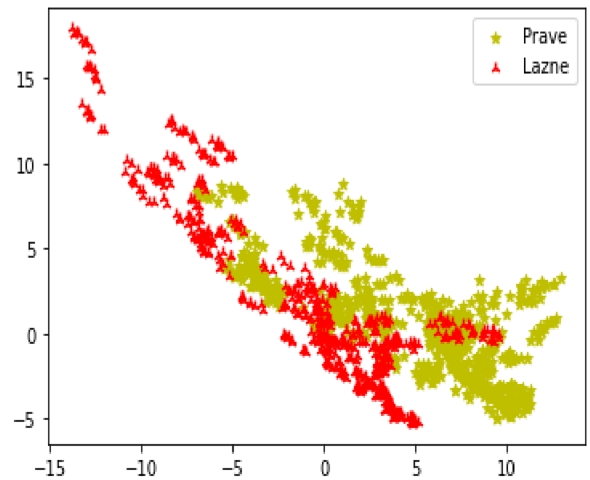
Slika 12.Skup za klasifikaciju laznih novčanica

Vidimo da imamo 4 atributa dok je peta kolona taget vrednost, govori nam da li je naovčanica lažna ili nije.

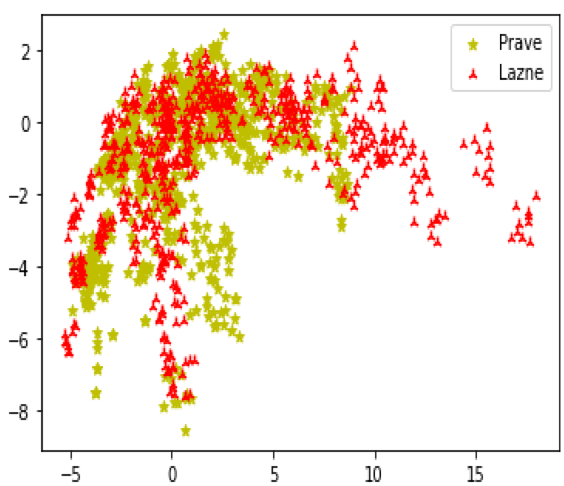
Podelićemo za početak skup tako da možemo da na grafu obojimo klase, uzećemo par atributa o obzir pa čemo ih prikazati na rasutim dijagramima



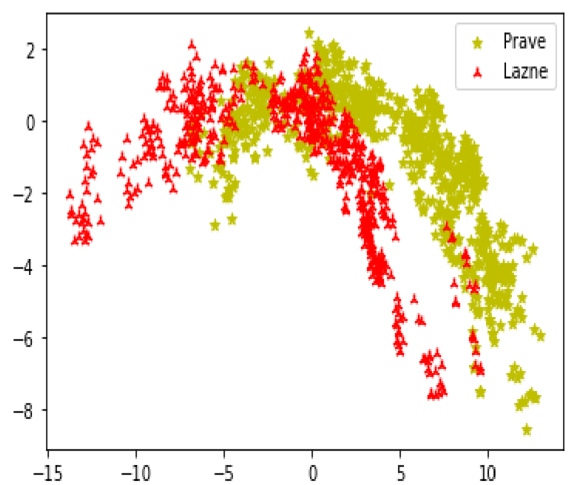
Slika 13. Dijagram atributa 0 i 1



Slika 14. Dijagram atributa 1 i 2



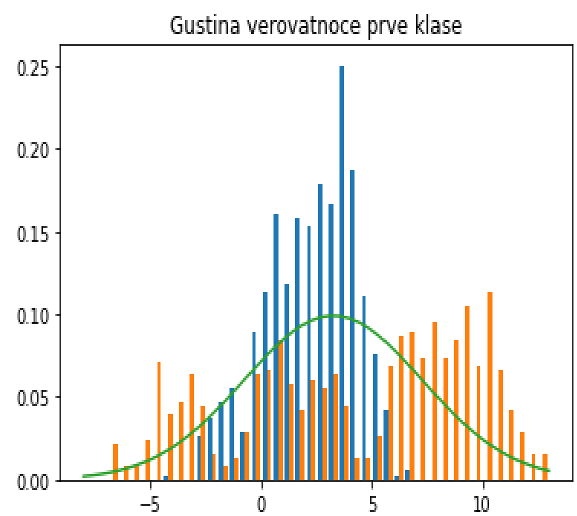
Slika 15. Dijagram atributa 2 i 3



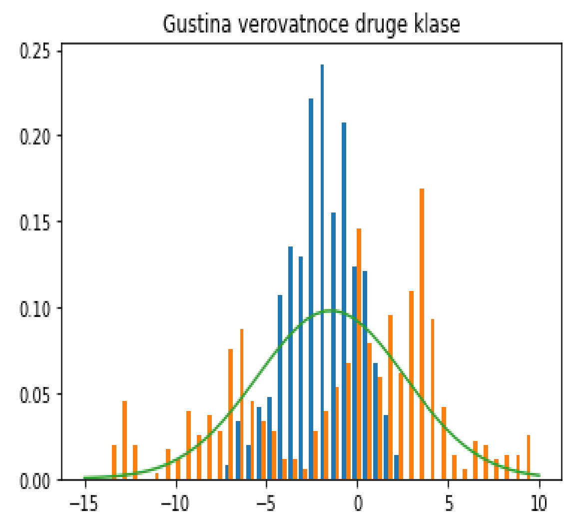
Slika 16. Dijagram atributa 0 i 3

Iz prethodnih slika možemo da vidimo da se većina atributa preklapa što nije baš zgodno za klasifikaciju. Najbolje što bi mogli da uzmemo bi bili atributi 0 i 1 (slika 13).

Prikazaćemo takođe i funkcije gustine verovatnoće klasa.



Slika 17. Gustina raspodele za prvu klasu



Slika 18. Gustina raspodele za drugu klasu

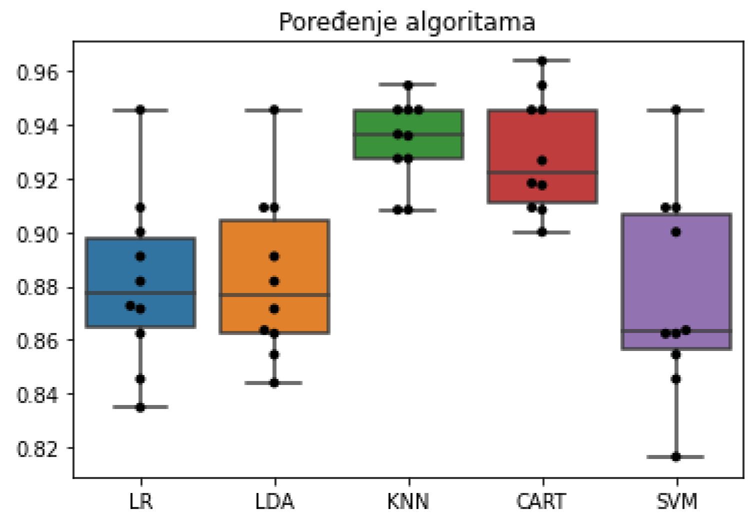
Sada možemo da uvezemo iz Sklearn biblioteke nekoliko klasifikatora i odmah da im proverimo tačnost na našem skupu:

```
LR: 0.881426 (0.030492)
LDA: 0.883253 (0.029298)
KNN: 0.933411 (0.014956)
CART: 0.928866 (0.020779)
SVM: 0.876856 (0.036108)
```

Slika 19. Tačnost klasifikatora

Odmah primećujemo da nam je KNN algoritam dao najbolje rezultate. KNN algoritam uvodi predpostavku da slične stvari (objekti klase) zauzimaju neki zajednički prostor jedan blizu drugoga.

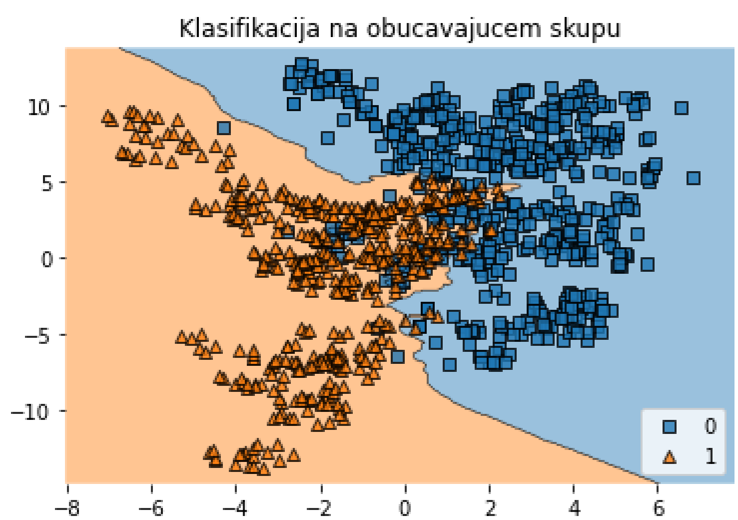
Sada možemo da ovu predpostavku potvrdimo i sa Boxplot-om.



Slika 20. Poređenje algoritama na boxplot-u

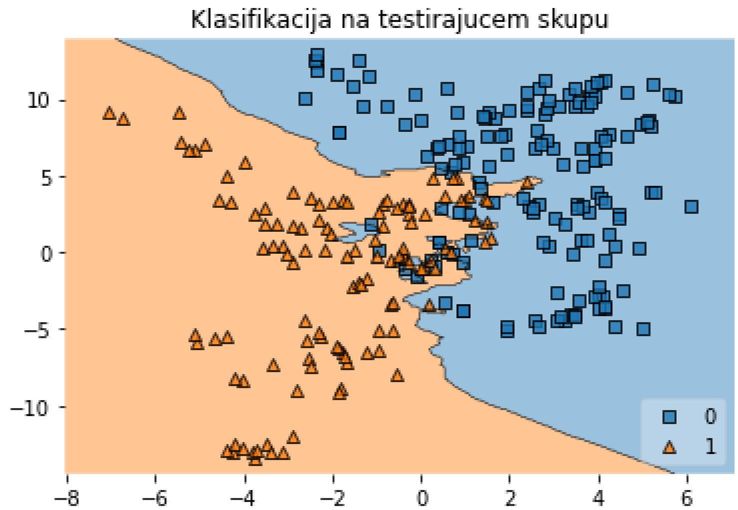
Iz slike gore možemo jasno da vidimo da od 10 iteracija kros – validacije imamo jednu iteraciju sa tačnošću 95% 3 iteracija sa 94% dve sa 93% dok je ostatak malo ispod toga, zbog toga, ostajemo pri odluci da biramo ovaj algoritam.

Prikazaćemo ovaj algoritam prvo na obučavajućem skupu:



Slika 21. KNN na obučavajućem skupu

A zatim konačno i na testirajućem:



Slika 22. KNN na testirajućem skupu.

Sa samih slika možemo da vidimo da nam algoritam prilično dobro klasifikuje podatke, ali da bi se jos bolje uverili možemo da prikažemo i tačnost:

```
Tacnost:  0.9490909090909091
```

Slika 23. Tačnost algoritma

Vidimo ta je tačnost čak 94 procenata što je dosta zavidna brojka. Da smo imali podatke koji se uopšte ne preklapaju, klase ne dolaze toliko u kontakt jedna sa drugom imali bi čak i bolje rezultate.

Zadatak 3

(Linearna regresija osiguranje)

Nakon što smo učitali skup podataka možemo da prikažemo nekoliko značajnih informacija:

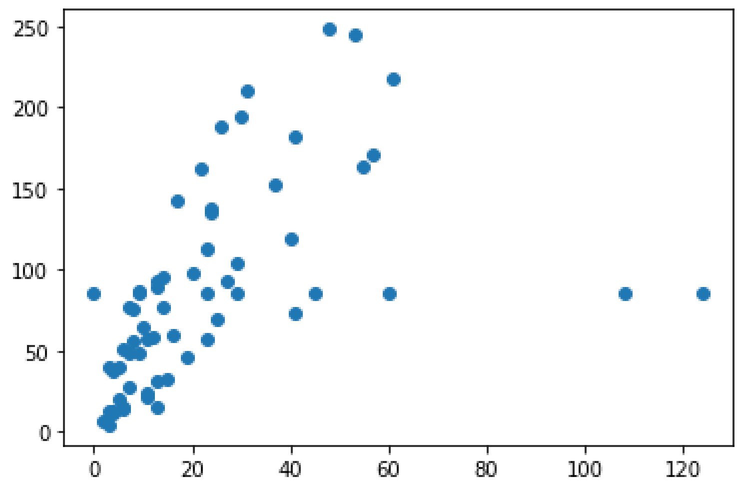
```
Dimenzije:  (63, 2)
      X      Y
0   108    NaN
1    19   46.2
2    13   15.7
3   124    NaN
4    40  119.4
5    57  170.9
6    23   56.9
7    14   77.5
8    45    NaN
9    10   65.3
10    5   20.9
11   48  248.1
12   11   23.5
13   23    NaN
14    7   48.8
15    2    6.6
16   24  134.9
17    6   50.9
18    3    4.4
19   23  113.0
Nan vrednosti:  8
```

Slika 24.Skup osiguranje

Odmah na početku vidimo dimenzije skupa. Imamo 2 atributa od kojih drugi zavisi od prvog. Linearna regresija će nam pomoći da predvidimo Y vrednost za neku novu X vrednost koje trenutno nemamo u skupu.

Prvi problem koji se javlja jestu jedostajuće vrednosti kojih ima 8. Obzirom da nam je skup relativno mali ovo može biti problem.

Pokušaćemo na 2 različita pristupa. Prvo ćemo promeniti atribute koji nedostaju sa srednjom vrednosti kolone:



Slika 25. Rasuti dijagram sa popunjenim Nan vrednostima

Odmah sa slike možemo da primetimo da će nam ovi podaci praviti problem. Srednja vrednost ove kolone je negde oko 80 i to su tačno oni podaci koji na slici „štrče“. Takodje, Nan vrednosti su menjane funkcijom fillna() i koriscena je srendja vrednost Y kolone. Mogli smo da koristimo i funkciju iz Sklearn biblioteke (SimpleImputer) koji takodje daje iste vrednosti.

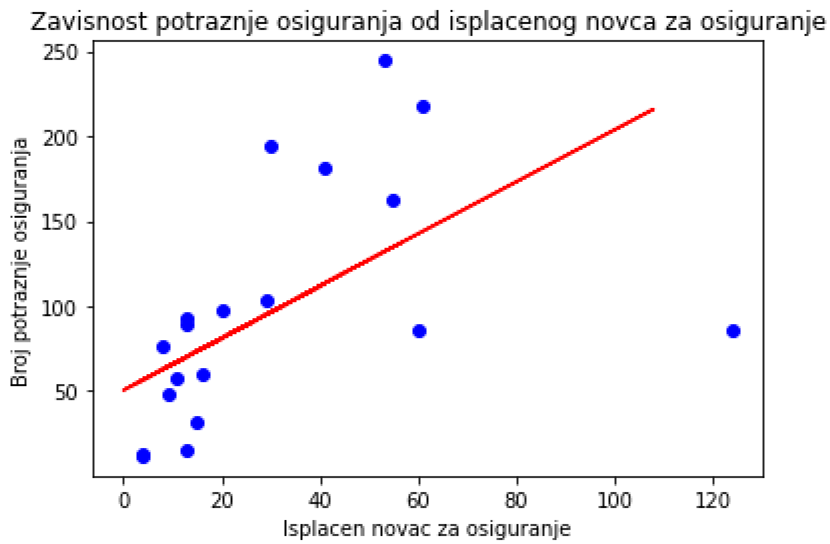
Podelićemo skup na obučavajući i trenirajući i takođe promeniti dimenzije skupova da bi mogli da radimo regresiju

```
X_obucavajuci, X_testirajuci, Y_obucavajuci, Y_testirajuci = train_test_split(X, Y, test_size=0.30, random_state=1)



#Prebacujemo u 2D da bi mogli da radimo regresiju
X_obucavajuci=np.reshape(X_obucavajuci,(-1,1))
Y_obucavajuci=np.reshape(Y_obucavajuci,(-1,1))
X_testirajuci=np.reshape(X_testirajuci,(-1,1))
Y_testirajuci=np.reshape(Y_testirajuci,(-1,1))

print(X_obucavajuci.shape,X_testirajuci.shape,Y_obucavajuci.shape,Y_testirajuci.shape )
```

```
(44, 1) (19, 1) (44, 1) (19, 1)
```

Slika 26. Podela podataka na obučavajući i testirajući i promena dimenzija

Kada prikažemo grafik sa regresivnom pravom vidimo da su nam naknadno uneti podaci napravili problem:



Slika 27. Regrsivna prava na skupu sa dodatim Nan vrednostima

Takodje prikazaćemo i srednju vrednost kvadratne greske i tacnost modela

```
Srednja kvadratna greska =  0.062658601321067733
Tacnost modela =  0.19556902082141647
```

Slika 28.Tacnost modela i srednja vrednost kvadratne greske

Iz samih vrednosti vidimo da model nije zavidan. Tacnost modela je jako mala.

Predikcija vrednosti za X = 90:

```
Broj potraznje osiguranja za isplaceno 90K:  188.30710
```

Slika 28. Predikcija vrednosti 90

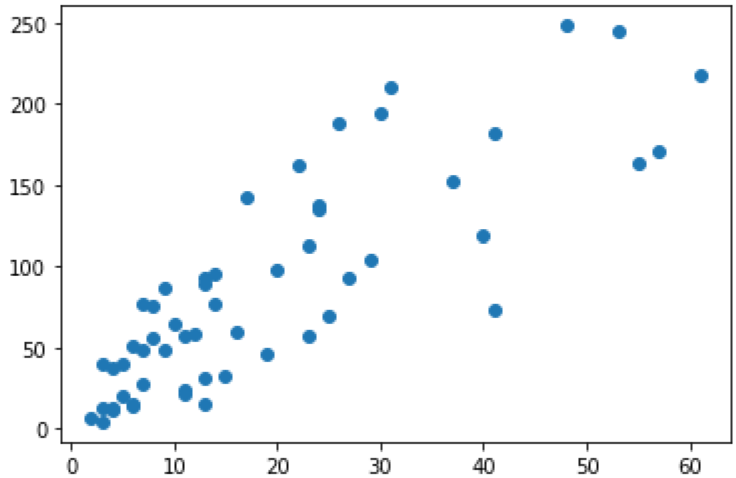
Pošto smo utvrdili da nam model ne daje zavidne rezultate pokušacemo i na sledeći način

Naredna ideja bi se zasnivala na ignorisanju (izbacivanju) nepostojecih vrednosti.

```
Dimenzije sa Nan vrednostima:  (63, 2)
     X      Y
0  108    NaN
1   19   46.2
2   13   15.7
3  124    NaN
4   40  119.4
5   57  170.9
6   23   56.9
7   14   77.5
8   45    NaN
9   10   65.3
Dimenzije bez Nan vrednostima:  (55, 2)
     X      Y
1   19   46.2
2   13   15.7
4   40  119.4
5   57  170.9
6   23   56.9
7   14   77.5
9   10   65.3
10   5   20.9
11  48  248.1
12  11   23.5
```

Slika 30. Isključivanje Nan vrednosti

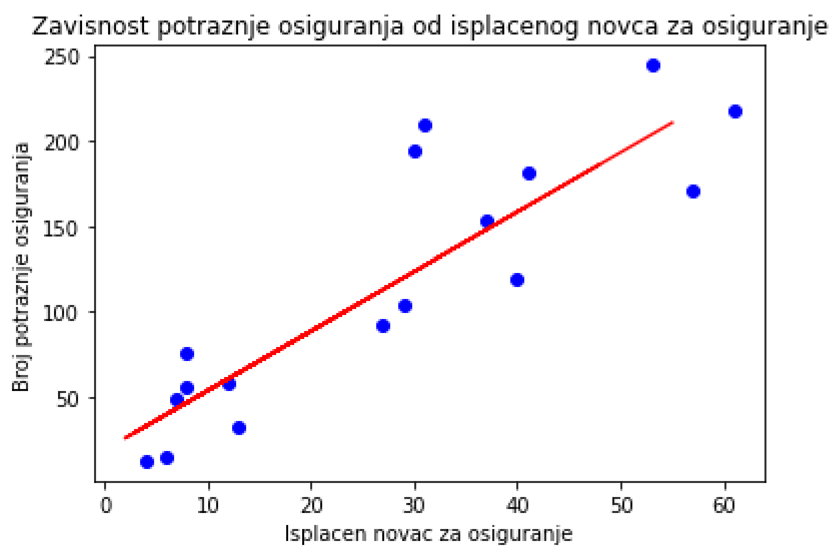
Sada možemo i da prikažemo ponovo rasuti dijagram samo bez Nan vrednosti:



Slika 31. Rasuti dijagram bez Nan vrednosti

Ponovićemo regresiju samo bez vrednosti koje smo izbacili.

```
(38, 1) (17, 1) (38, 1) (17, 1)
```



Slika 32. Linearna regresija bez Nan vrednosti i dimenzije skupova

Sada vidimo da su podaci dosta bolji jer su bliži regresivnoj pravoj. Videćemo da je i tačnost dosta veća pa će i samim time i predikcija biti preciznija.

```
Srednja kvadratna greska =  0.02151700932667341
Tacnost modela =  0.7629038735688708
```

Slika 33. Srednja kvadratna greska i tacnost modela na skupu bez Nan vrednosti

```
Broj potraznje osiguranja za isplaceno 90K:  332.75332
```

Slika 34. Predikcija sa modelom bez Nan vrednosti

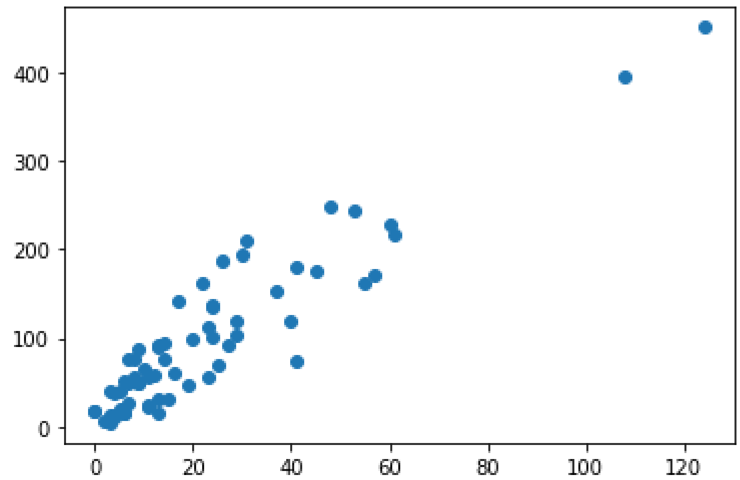
Vrednost sa slike 33 ne možemo nikako da znamo sigurno. Tačnost modela nam kaže koliko je ta vrednost pouzdana ali takodje možemo vizuelno iz skupa da od prilike proverimo tačnost predviđene vrednosti.

Sada možemo da iskoristimo ovakav model da vratimo podatke koje smo sklonili. Koristićemo predikciju da predpostavimo vrednosti koje su inicjalno bile Nan

X	Y
108	[408]
124	[466]
45	[179]
24	[103]
9	[49]
0	[16]
60	[234]
29	[121]
0	[16]

Slika 35. Prediktovane vrednosti koje su nedostajale u skupu

Prikazaćemo i rasuti dijagram sa vrednostima koje je naš model predpostavio:



Slika 36.Prikaz rasutog dijagrama sa prediktovanim vrednostima

Kod korišćen za izradu zadataka

- *Kompletan kod je pisan u Google Colab-u zbog toga je blok znakova '#' postavljen tamo gde je poseban blok koda.*

```
###########################  ZADATAK 1 ###########################

import numpy as np
import matplotlib.pyplot as plt

M1 = [-3, 2]
Sigma1 = [[2, -1], [-1, 1]]
M2 = [1, 1]
Sigma2 = [[2, 1], [1, 2]]

np.random.seed(0)

Negativna1, Negativna2 = np.random.multivariate_normal(M1, Sigma1, 700).T
Pozitivna1, Pozitivna2 = np.random.multivariate_normal(M2, Sigma2, 50).T

plt.plot(Negativna1, Negativna2, 'b*', label = 'Negativna klasa')
plt.plot(Pozitivna1, Pozitivna2, 'ro', label = 'Pozitivna klasa')
plt.grid(True)
plt.xlabel('x')
plt.ylabel('y')
```

```
plt.legend()

plt.show()

##################################################

from scipy import linalg

invSigma1 = linalg.inv(Sigma1)
invSigma2 = linalg.inv(Sigma2)



m11 = M1[0]
m12 = M1[1]
m21 = M2[0]
m22 = M2[1]



z11 = invSigma1[0][0]
z12 = invSigma1[0][1]
z21 = invSigma1[1][0]
z22 = invSigma1[1][1]
```

```
k11 = invSigma2[0][0]

k12 = invSigma2[0][1]

k21 = invSigma2[1][0]

k22 = invSigma2[1][1]


s1 = np.linspace(-7, 5, 100)

s2 = np.linspace(-7, 5, 100)


x1pom, x2pom = np.meshgrid(s1, s2)

d1=(invSigma1[0,0]*(x1pom-M1[0])+invSigma1[0,1]*(x2pom-M1[1]))*(x1pom-M1[0])+(invSigma1[1,0]*(x1pom-M1[0])+invSigma1[1,1]*(x2pom-M1[1]))*(x2pom-M1[1])

d2=(invSigma2[0,0]*(x1pom-M2[0])+invSigma2[0,1]*(x2pom-M2[1]))*(x1pom-M2[0])+(invSigma2[1,0]*(x1pom-M2[0])+invSigma2[1,1]*(x2pom-M2[1]))*(x2pom-M2[1])


plt.plot(Negativna1, Negativna2, 'b*', label = 'Negativna klasa')

plt.plot(Pozitivna1, Pozitivna2, 'ro', label = 'Pozitivna klasa')

plt.contour(x1pom,x2pom,d2,[1,4,7],colors='k',linewidths=3)

plt.contour(x1pom,x2pom,d1,[1,4,7],colors='k',linewidths=3)

plt.xlabel('x1')

plt.ylabel('x2')

plt.legend()

plt.grid(True)

plt.show()


###############################################
```

```
X=np.arange(-8,7,0.1)

Y=np.arange(-3,7,0.1)

x1,x2=np.meshgrid(X,Y)



P1 = 0.9

P2 = 0.1


y=0.5*(((x1-m11)*z11+(x2-m12)*z21)*(x1-m11)+((x1-m11)*z12+(x2-m12)*z22)*(x2-m12))-0.5*(((x1-m21)*k11+(x2-m22)*k21)*(x1-m21)+((x1-m21)*k12+(x2-m22)*k22)*(x2-m22))+0.5*np.log(np.linalg.det(Sigma1)/np.linalg.det(Sigma2))-np.log(P1/P2)


plt.plot(Negativna1, Negativna2, 'b*', label = 'Negativna klasa')

plt.plot(Pozitivna1, Pozitivna2, 'ro', label = 'Pozitivna klasa')

plt.grid(True)

plt.contour(x1,x2,y,0, colors = 'black')

plt.show()


##################################################



x1p1 = Negativna1

x2p1 = Negativna2
```

```
h1=0.5*(((x1p1-m11)*z11+(x2p1-m12)*z21)*(x1p1-m11)+((x1p1-m11)*z12+(x2p1-m12)*z22)*(x2p1-m12))-0.5*(((x1p1-m21)*k11+(x2p1-m22)*k21)*(x1p1-m21)+((x1p1-m21)*k12+(x2p1-m22)*k22)*(x2p1-m22))+0.5*np.log(np.linalg.det(Sigma1)/np.linalg.det(Sigma2)) - np.log(P1/P2)

greska1 = 0
for i in h1:
  if i > 0:
    greska1+=1

x1p2 = Pozitivna1
x2p2 = Pozitivna2

h2=0.5*(((x1p2-m11)*z11+(x2p2-m12)*z21)*(x1p2-m11)+((x1p2-m11)*z12+(x2p2-m12)*z22)*(x2p2-m12))-0.5*(((x1p2-m21)*k11+(x2p2-m22)*k21)*(x1p2-m21)+((x1p2-m21)*k12+(x2p2-m22)*k22)*(x2p2-m22))+0.5*np.log(np.linalg.det(Sigma1)/np.linalg.det(Sigma2)) - np.log(P1/P2)

greska2 = 0
for i in h2:
  if i > 0:
    greska2+=1
```

```
Matrix = [['/', '  P', '  N'],['P ', 50-greska2,' ', greska2],['N ', greska1,' ', 700-greska1]]

[print(*line) for line in Matrix]



Tacnost = (700+50)/(700+50+greska1+greska2)

Preciznost = (50 - greska2)/((50 - greska2) + greska2)

Odziv = (50 - greska2)/((50-greska2) + greska1)

print('Tacnost = ', Tacnost)

print('Preciznost = ', Preciznost)

print('Odziv = ', Odziv)


#Izgled cele matrice konfuzije sa FP i FN vrednostima.

#Zakljucujemo da nam je vrednost False Negative vrednost 8

#Preciznost nam koliko je model pouzdan za predvidjanje klase

#Odziv nam govori da je od ukupnog broja pacijenata koji su klasifikovani kao pozitivni samo
60 posto zapravo pozitivno a ostalo je greska!

#Iz ovoga zakljucujemo da gore navedena Tacnost nije zapravo dobra metrika.



###################################################
```

```
P1 = 0.9

P2 = 0.1

br = 0

Colors = ['yellow','black','blue','green','magenta','cyan','red','green','red']

for i in range(0,8):

  y1=0.5*(((x1-m11)*z11+(x2-m12)*z21)*(x1-m11)+((x1-m11)*z12+(x2-m12)*z22)*(x2-m12))-0.5*(((x1-m21)*k11+(x2-m22)*k21)*(x1-m21)+((x1-m21)*k12+(x2-m22)*k22)*(x2-m22))+0.5*np.log(np.linalg.det(Sigma1)/np.linalg.det(Sigma2))-np.log(P1/P2)

  P1 -= 0.1

  P2 += 0.1

  plt.contour(x1,x2,y1,0, colors = Colors[br])

  br+=1


plt.plot(Negativna1, Negativna2, 'b*', label = 'Negativna klasa')

plt.plot(Pozitivna1, Pozitivna2, 'ro', label = 'Pozitivna klasa')

plt.grid(True)

plt.legend()

plt.title('Klase sa klasifikacionim krivama sa razlicitim apriornim vrednostima')

plt.show()


####################################################



def FPcalc(P1, P2):

 x1p2 = Pozitivna1
```

```
 x2p2 = Pozitivna2

 h2=0.5*(((x1p2-m11)*z11+(x2p2-m12)*z21)*(x1p2-m11)+((x1p2-m11)*z12+(x2p2-m12)*z22)*(x2p2-m12))-0.5*(((x1p2-m21)*k11+(x2p2-m22)*k21)*(x1p2-m21)+((x1p2-m21)*k12+(x2p2-m22)*k22)*(x2p2-m22))+0.5*np.log(np.linalg.det(Sigma1)/np.linalg.det(Sigma2)) - np.log(P1/P2)

 greska2 = 0
 for i in h2:
  if i < 0:
   greska2+=1
 return greska2

def FNcalc(P1, P2):
 x1p1 = Negativna1
 x2p1 = Negativna2

 h1=0.5*(((x1p1-m11)*z11+(x2p1-m12)*z21)*(x1p1-m11)+((x1p1-m11)*z12+(x2p1-m12)*z22)*(x2p1-m12))-0.5*(((x1p1-m21)*k11+(x2p1-m22)*k21)*(x1p1-m21)+((x1p1-m21)*k12+(x2p1-m22)*k22)*(x2p1-m22))+0.5*np.log(np.linalg.det(Sigma1)/np.linalg.det(Sigma2)) - np.log(P1/P2)
```

```
 greska1 = 0
 for i in h1:
   if i > 0:
    greska1+=1
 return (greska1)



FP = FPcalc(0.9, 0.1)
TN = 700 - FNcalc(0.9, 0.1)

FPR = FP/(TN+FP)

TP = 50 - FPcalc(0.9,0.1)
FN = FNcalc(0.9, 0.1)

TPR = TP / (TP + FN)



FPRlist = [FPR]
TPRlist = [TPR]
print('P1 = %.1f || P2 = %.1f'% (P1, P2))
print('False Positive Rate: %.5f' % FPR)
print('True Positive Rate: %.5f'% TPR)
plt.scatter(FPR, TPR, color = 'yellow')
P1 = 0.8
```

```
P2 = 0.2
br = 1
for i in range(1,8):
  FP = FPcalc(P1, P2)
  TN = 700 - FNcalc(P1, P2)

  FPR = FP/(TN+FP)

  TP = 50 - FPcalc(P1, P2)
  FN = FNcalc(P1, P2)

  TPR = TP / (TP + FN)
  FPRlist.append(FPR)
  TPRlist.append(TPR)
  print('P1 = %.1f || P2 = %.1f'% (P1, P2))
  print('False Positive Rate: %.5f' % FPR)
  print('True Positive Rate: %.5f'% TPR)
  plt.scatter(FPR, TPR, color = Colors[br])
  P1 -= 0.1
  P2 += 0.1
  br+=1

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.plot(FPRlist, TPRlist, color = 'red')
```

```
plt.title('ROC kriva sa vrednostima')

plt.grid(True)

plt.show()



###################################################




#Zakljucujemo da, sto se tice klasifikacija pozitivnih odbiraka, mozemo da biramo jednu od 4 linije

#Zelenu, crvenu, cian ili magenda jer ima je false pozitive rate 0


#Crna linija bi uopste bila najbolji klasifikator, jer joj je FPR dosta mali dok je TPR veliki!


P1 = 0.8

P2 = 0.2


y11=0.5*(((x1-m11)*z11+(x2-m12)*z21)*(x1-m11)+((x1-m11)*z12+(x2-m12)*z22)*(x2-m12))-0.5*(((x1-m21)*k11+(x2-m22)*k21)*(x1-m21)+((x1-m21)*k12+(x2-m22)*k22)*(x2-m22))+0.5*np.log(np.linalg.det(Sigma1)/np.linalg.det(Sigma2))-np.log(P1/P2)


P1 = 0.4

P2 = 0.6
```

```
y22=0.5*(((x1-m11)*z11+(x2-m12)*z21)*(x1-m11)+((x1-m11)*z12+(x2-m12)*z22)*(x2-m12))-0.5*(((x1-m21)*k11+(x2-m22)*k21)*(x1-m21)+((x1-m21)*k12+(x2-m22)*k22)*(x2-m22))+0.5*np.log(np.linalg.det(Sigma1)/np.linalg.det(Sigma2))-np.log(P1/P2)


plt.plot(Negativna1, Negativna2, 'b*', label = 'Negativna klasa')

plt.plot(Pozitivna1, Pozitivna2, 'ro', label = 'Pozitivna klasa')

#plt.contour(x1,x2,y11,0, colors = 'green')

plt.contour(x1,x2,y11,0, colors = 'black')


plt.show()



##################################################


        ########################## ZADATAK 2 ##########################

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt


novcanice = pd.read_csv('/content/Novcanice.txt', header = None)


print('Dimenzije skupa =', novcanice.shape)

novcanice.describe()



######################################################
```

```
X = novcanice.iloc[:,0:4]

X = X.to_numpy()

Y = novcanice.iloc[:,4]

Y = Y.to_numpy()


plt.scatter(X[0:763,0], X[0:763,1],c=u'y',marker='*', label='Prave')

plt.scatter(X[763:,0], X[763:,1], c=u'r',marker='2',label='Lazne')




plt.legend()

plt.show()


##################################################



plt.scatter(X[0:763,1], X[0:763,2],c=u'y',marker='*', label='Prave')

plt.scatter(X[763:,1], X[763:,2], c=u'r',marker='2',label='Lazne')



plt.legend()

plt.show()


##################################################
```

```
plt.scatter(X[0:763,2], X[0:763,3],c=u'y',marker='*', label='Prave')

plt.scatter(X[763:,2], X[763:,3], c=u'r',marker='2',label='Lazne')



plt.legend()

plt.show()



#####################################################



plt.scatter(X[0:763,1], X[0:763,3],c=u'y',marker='*', label='Prave')

plt.scatter(X[763:,1], X[763:,3], c=u'r',marker='2',label='Lazne')



plt.legend()

plt.show()



#####################################################



from scipy.stats import norm
```

```
raspodela = norm(np.mean(X[0:763,:2]), np.std(X[0:763,:2]))

broj_tacaka=np.linspace(-8,13)

fgv = [raspodela.pdf(tacke) for tacke in broj_tacaka]

plt.hist(X[0:763,:2], bins = 40, density = True)

plt.plot(broj_tacaka,fgv)

plt.title('Gustina verovatnoce prve klase')


plt.show()





#########################################################





raspodela = norm(np.mean(X[763:,:2]), np.std(X[763:,:2]))

broj_tacaka=np.linspace(-15,10)

fgv = [raspodela.pdf(tacke) for tacke in broj_tacaka]

plt.hist(X[763:,:2], bins = 40, density = True)

plt.plot(broj_tacaka,fgv)

plt.title('Gustina verovatnoce druge klase')


plt.show()
```

```
#####################################################

from sklearn.model_selection import train_test_split

X = X[:,0:2]
X_obucavajuci, X_testirajuci, Y_obucavajuci, Y_testirajuci= train_test_split(X, Y, test_size=0.20, random_state=1)
print(X_obucavajuci.shape, X_testirajuci.shape,Y_obucavajuci.shape,Y_testirajuci.shape)

#####################################################

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import SVC


models = []
models.append(('LR', LogisticRegression(solver='liblinear', multi_class='ovr')))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
```

```
models.append(('CART', DecisionTreeClassifier()))

models.append(('SVM', SVC(C=0.5, kernel='linear')))

results = []

names = []


for name, model in models:

  kfold = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)

  cv_results = cross_val_score(model, X_obucavajuci, Y_obucavajuci,cv=kfold,
scoring='accuracy')

  results.append(cv_results)

  names.append(name)

  print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.std()))




#####################################################



import numpy as np

import seaborn as sns

import warnings

warnings.filterwarnings('ignore')

ax = sns.boxplot(data=results)

ax = sns.swarmplot(data=results, color="black")

plt.xticks(np.arange(0, 5),names)
```

```
plt.title('Poređenje algoritama')
plt.show()

######################################################

model = KNeighborsClassifier()
model.fit(X_obucavajuci, Y_obucavajuci)

predictions = model.predict(X_testirajuci)

import mlxtend
from mlxtend.plotting import plot_decision_regions
plot_decision_regions(X_obucavajuci, Y_obucavajuci, clf=model)
plt.title('Klasifikacija na obucavajucem skupu')
plt.legend(loc = 'lower right')
plt.show()

######################################################

from sklearn.metrics import accuracy_score
from mlxtend.plotting import plot_decision_regions



print("Tacnost: ",accuracy_score(Y_testirajuci, predictions))
plot_decision_regions(X_testirajuci, Y_testirajuci, clf=model, legend=4)
```

```
plt.title('Klasifikacija na testirajucem skupu')
plt.show()


########################### ZADATAK 3 ###########################
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

Names = ['X', 'Y']

osiguranje = pd.read_csv('/content/Osiguranje.csv', names= Names)
print('Dimenzije: ',osiguranje.shape)
print(osiguranje.head(20))
print('Nan vrednosti: ',osiguranje['Y'].isnull().sum())

#######################################################


osiguranje.fillna(osiguranje.mean(), inplace=True)
#osiguranje = osiguranje.dropna()
#Ostavljeno je komentarisano deo gde se menjaju Nan vrednosti sa srednjom vrednosti
#Kada bi pokrenuli sa tim delom, umesto da izbacujemo Nan vrednosti regresija bi bila jako losa
from sklearn.impute import SimpleImputer
```

```
osiguranje.describe()


X = osiguranje['X']

Y = osiguranje['Y']

X = X.to_numpy()

Y = Y.to_numpy()


plt.scatter(X, Y)

plt.show()




##########################################################




from sklearn.linear_model import LinearRegression


X_obucavajuci, X_testirajuci, Y_obucavajuci, Y_testirajuci = train_test_split(X, Y,
test_size=0.30, random_state=1)




#Prebacujemo u 2D da bi mogli da radimo regresiju
```

```
X_obucavajuci=np.reshape(X_obucavajuci,(-1,1))

Y_obucavajuci=np.reshape(Y_obucavajuci,(-1,1))

X_testirajuci=np.reshape(X_testirajuci,(-1,1))

Y_testirajuci=np.reshape(Y_testirajuci,(-1,1))


print(X_obucavajuci.shape,X_testirajuci.shape,Y_obucavajuci.shape,Y_testirajuci.shape )



###########################################################



regresija.fit(X_obucavajuci, Y_obucavajuci)

Y_predikcija = regresija.predict(X_testirajuci)


plt.scatter(X_testirajuci,Y_testirajuci,color='blue')

plt.plot(X_obucavajuci, regresija.predict(X_obucavajuci), color='red')

plt.xlabel('Isplacen novac za osiguranje')

plt.ylabel('Broj potraznje osiguranja')

plt.title('Zavisnost potraznje osiguranja od isplacenog novca za osiguranje')

plt.show()





###########################################################
```

```
from sklearn import metrics

print('Srednja kvadratna greska =
',metrics.mean_squared_error(Y_testirajuci/max(Y_testirajuci),Y_predikcija/max(Y_predikcija))
)

print('Tacnost modela =
',metrics.r2_score(Y_testirajuci/max(Y_testirajuci),Y_predikcija/max(Y_predikcija)))




##########################################################


Predikcija = 90

Predikcija = np.reshape(Predikcija, (-1,1))


predict = regresija.predict(Predikcija)

print('Broj potraznje osiguranja za isplaceno 90K:  %.5f' % predict)



#Zbog zamene Nan vrednosti tačnost je veoma mala 19%




##########################################################
```

```
osiguranje = pd.read_csv('/content/Osiguranje.csv', names= Names)

missing = [[108,0] ,[124,0] ,[45, 0], [24, 0], [9, 0], [0, 0], [60, 0], [29, 0]]

print('Dimenzije sa Nan vrednostima: ', osiguranje.shape)

print(osiguranje.head(10))

osiguranje = osiguranje.dropna()

print('Dimenzije bez Nan vrednostima: ', osiguranje.shape)

print(osiguranje.head(10))




##############################################################




X = osiguranje['X']

Y = osiguranje['Y']

X = X.to_numpy()

Y = Y.to_numpy()


plt.scatter(X, Y)

plt.show()




##############################################################
```

```
X_obucavajuci, X_testirajuci, Y_obucavajuci, Y_testirajuci = train_test_split(X, Y,
test_size=0.30, random_state=1)




#Prebacujemo u 2D da bi mogli da radimo regresiju

X_obucavajuci=np.reshape(X_obucavajuci,(-1,1))

Y_obucavajuci=np.reshape(Y_obucavajuci,(-1,1))

X_testirajuci=np.reshape(X_testirajuci,(-1,1))

Y_testirajuci=np.reshape(Y_testirajuci,(-1,1))


print(X_obucavajuci.shape,X_testirajuci.shape,Y_obucavajuci.shape,Y_testirajuci.shape )


regresija.fit(X_obucavajuci, Y_obucavajuci)

Y_predikcija = regresija.predict(X_testirajuci)


plt.scatter(X_testirajuci,Y_testirajuci,color='blue')

plt.plot(X_obucavajuci, regresija.predict(X_obucavajuci), color='red')

plt.xlabel('Isplacen novac za osiguranje')

plt.ylabel('Broj potraznje osiguranja')

plt.title('Zavisnost potraznje osiguranja od isplacenog novca za osiguranje')

plt.show()
```

```
###########################################################


print('Srednja kvadratna greska =
',metrics.mean_squared_error(Y_testirajuci/max(Y_testirajuci),Y_predikcija/max(Y_predikcija))
)

print('Tacnost modela =
',metrics.r2_score(Y_testirajuci/max(Y_testirajuci),Y_predikcija/max(Y_predikcija)))


###########################################################




predict = regresija.predict(Predikcija)

print('Broj potraznje osiguranja za isplaceno 90K:  %.5f' % predict)



###########################################################



missingX = [108,124 ,45,24, 9,  0, 60,  29, 0]

missingY = []

print(' X\t   Y')

for i in range(0,9):

  Predikcija = missingX[i]

  Predikcija = np.reshape(Predikcija, (-1,1))
```

```
  predict = regresija.predict(Predikcija)
  missingY.append(predict.astype(int))



mX = np.array(missingX)
mY = np.array(missingY)
for i in range(0, 9):
  print(mX[i], '\t', *mY[i])
X1 = np.append(X, mX)
Y1 = np.append(Y, mY)


##############################################################

print(X1.shape, Y1.shape)
plt.scatter(X1, Y1)
plt.show()



##############################################################
```