به نام خدا

دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

تشخیص و شناسایی خطا

امتحان میان‌ترم

شیما سادات ناصری

۴۰۱۱۲۸۱۴

دکتر مهدی علیاری شوره دلی

اردیبهشت ماه ۱۴۰۰

فهرست مطالب

عنوان شماره صفحه

بخش۱: سوال هماهنگ شده

سوال اول

برای الگوریتم least square تابع هزینه به صورت زیر می‌باشد:

$$J = ||Zw - B|| = (Zw - B)^T(Zw - B)$$
$$= (Zw)(Zw)^T - ZwB^T - B(Zw)^T + BB^T$$

معادله بالا در صورت استفاده از روش widrow-hoff برای دو پارامتر B و w با متد گرادیان نزولی در هر epoch، آپدیت می‌شود:

$$B^+ = B^- - \rho_B \frac{\partial J}{\partial B}$$
$$w^+ = w^- - \rho_w \frac{\partial J}{\partial w}$$

که در آن ρ_B و ρ_w به ترتیب لرنینگ ریت بایاس و وزن‌ها می‌باشد. با اعمال مشتق مرتبط با هر پارامتر طبق تابع هزینه بالا، معادلات مشتقی به صورت زیر خواهند بود.

$$\frac{\partial J}{\partial B} = 0 - Zw + B$$
$$\frac{\partial J}{\partial w} = 2Z^TZw - 2Z^TB$$

درنتیجه، معادلات آپدیت به صورت زیر خواهند بود.

$$B^+ = B^- - \rho_B(-Zw + B^-) = B^- + \rho_B Zw - \rho_B B^- \quad \blacksquare$$
$$w^+ = w^- - \rho_w(2Z^TZw^- - 2Z^TB) = w^- + 2\rho_w Z^TB - 2\rho_w Z^TZw^- \quad \blacksquare$$

بخش۲: سوالات هماهنگ نشده

سوال دوم

$$f_X(x) = (1-p)^{x-1}p\ , D = \{x_1, x_2, \dots, x_N\}$$
$$ML\ estimation: Max\{f(x;p)\big|\{x_i\}_{i=1}^N\}$$

این معادله به وضوح نشان می‌دهد که هر آزمایش از آزمایش دیگر مستقل است و به آزمایش‌های دیگر ارتباطی ندارد؛ درنتیجه از هم مستقل‌اند و معادله تخمین maximum likelihood به صورت زیر تبدیل خواهد شد.

$$P(D|p) = \prod_{i=1}^{N}(1-p)^{x_i-1}p$$
$$= p^N(1-p)^{\sum_{i=1}^N(x_i-1)} = p^N(1-p)^{\sum_{i=1}^N x_i - N}$$

حال اگر فرض کنیم که $\sum_{i=1}^N x_i = m$، خواهیم داشت:

$$P(D|p) = p^N(1-p)^{m-N}$$

با اعمال ln نوای این معادله تغییر نخواهد کرد و خواهیم داشت:

$$A = \ln\big(P(D|p)\big) = \ln(\mathrm{p^N}(1-p)^{m-N}) = \ln(p^N) + \ln((1-p)^{m-N})$$
$$= Nln(p) + (m-N)\ln(1-p)$$

برای بدست آوردن مقدار ماکزیمم این تابع باید از A مشتق نسبت به p بگیریم و آن را برابر با صفر قرار دهیم:

$$\frac{\partial A}{\partial p} = \frac{N}{p} + \frac{m-N}{1-p} = 0 \Rightarrow \frac{N-m}{1-p} = \frac{N}{p} \Rightarrow p(N-m) = N - Np \Rightarrow p = \frac{N}{2N-m}\ \blacksquare$$

سوال سوم

https://colab.research.google.com/drive/1tJ0bRMTe8naDcesWhxd_tpaz1nr-nXru?usp=sharing

a) اگر معادله هر بخش (۱ نرمال و ۲ فالت) را به صورت زیر بنویسیم؛

$$g_1 = N_1(\mu_1, \sigma_1^2)$$
$$g_2 = N_2(\mu_2, \sigma_2^2)$$

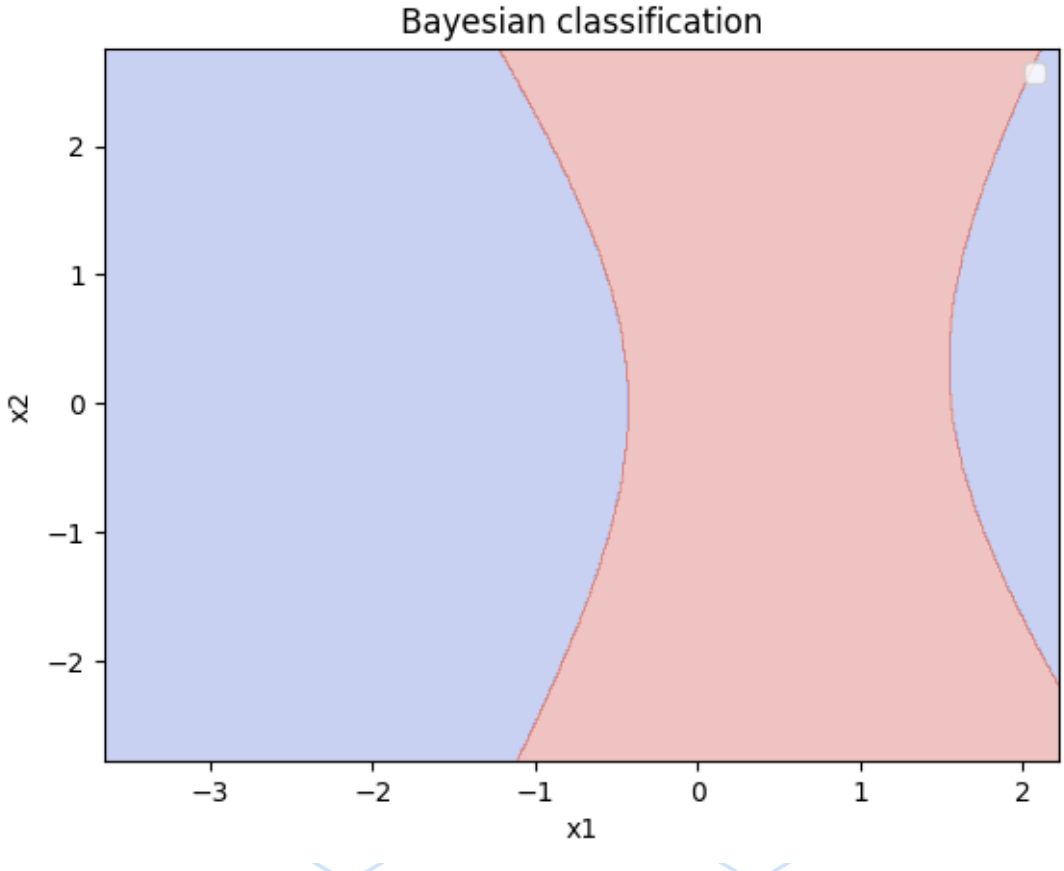
معادله Decision Surface به صورت $g_1 = g_2$ می‌باشد. از آن جایی که معادله هر دو گوسی است، تساوی پس از اعمال ln به صورت زیر خواهد شد:

$$-0.5(X-\mu_1)^T\Sigma_1(X-\mu_1)-0.5\ln(|\Sigma_1|)=-0.5(X-\mu_2)^T\Sigma_2(X-\mu_2)-0.5\ln(|\Sigma_2|)$$
$$X=[x_1,x_2]^T$$
$$\Rightarrow \frac{5\,x_2^2}{6}-5\,x_1^2+\left(\frac{5\,x_1}{3}+\frac{5}{3}\right)(x_1+1)=\frac{5\,(x_1+1)^2}{3}-5\,{x_1}^2+\frac{5\,{x_2}^2}{6}=0$$

پاسخ این معادله در حالت کلی به صورت زیر خواهد شد.

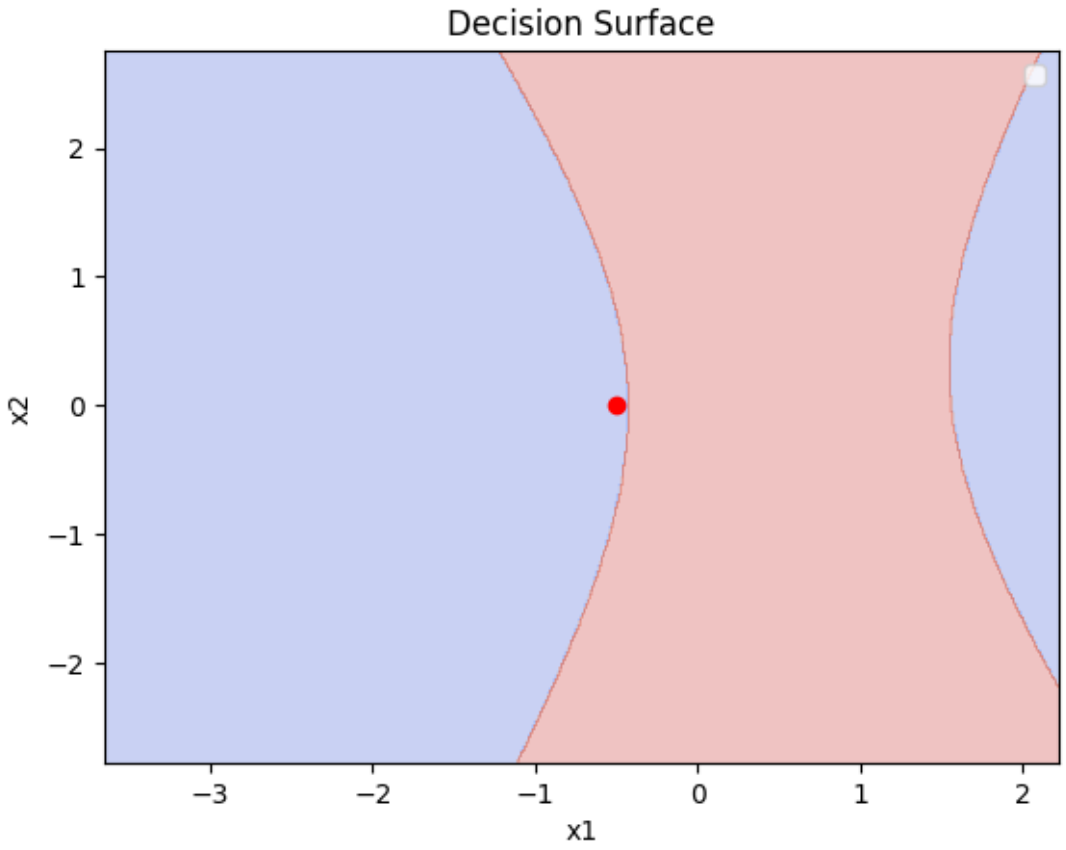
$$X=\begin{pmatrix}\frac{1}{3}-\frac{3\sqrt{\frac{4\,{x_2}^2}{9}+\frac{4}{3}}}{4}\\ \frac{3\sqrt{\frac{4\,{x_2}^2}{9}+\frac{4}{3}}}{4}+\frac{1}{2}\end{pmatrix}$$

نمای آن در صفحه x1-x2 به صورت زیر می‌باشد:



که در آن ناحیه قرمز نرمال و ناحیه آبی فالت است.

b) در حال حاضر همان‌طور که مشاهده می‌شود نقطه [-0.5,0] در ناحیه فالت قرار دارد.



برای این کار نیاز است تا از ضرایب ریسک یا loss matrix (با اعضای بزرگتر مساوی صفر) را به کار گیریم. ضرایب برای سیستم ۲ کلاسه به صورت زیر عمل می‌کنند.

$$loss\ matrix = \begin{bmatrix} \lambda_{11} & \lambda_{12} \\ \lambda_{21} & \lambda_{22} \end{bmatrix}$$

$$r_1 = \lambda_{12} \int_{R_2} f(x|f_1)dx + \lambda_{11} \int_{R_1} f(x|f_1)dx$$

$$r_2 = \lambda_{21} \int_{R_1} f(x|f_2)dx + \lambda_{22} \int_{R_2} f(x|f_2)dx$$

ضرایب روی قطر ماتریس در صورتی که اعتماد کامل به داده ها داشته باشیم صفر خواهند بود. با تغییر این ماتریس در نهایت Decision surface به این صورت خواهد شد:

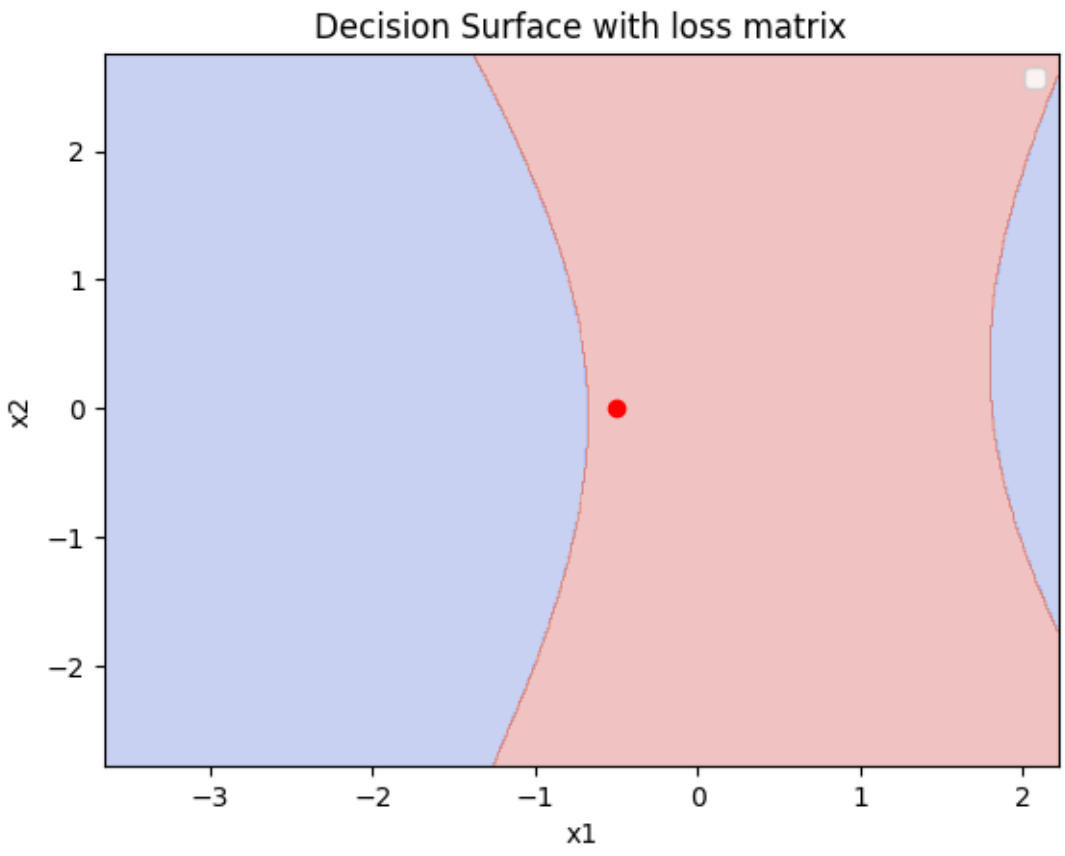
$$R_k = \{x|argmin \sum_{i=1}^{c} \lambda_{ik} f(x|f_i) = k\}$$

کد:

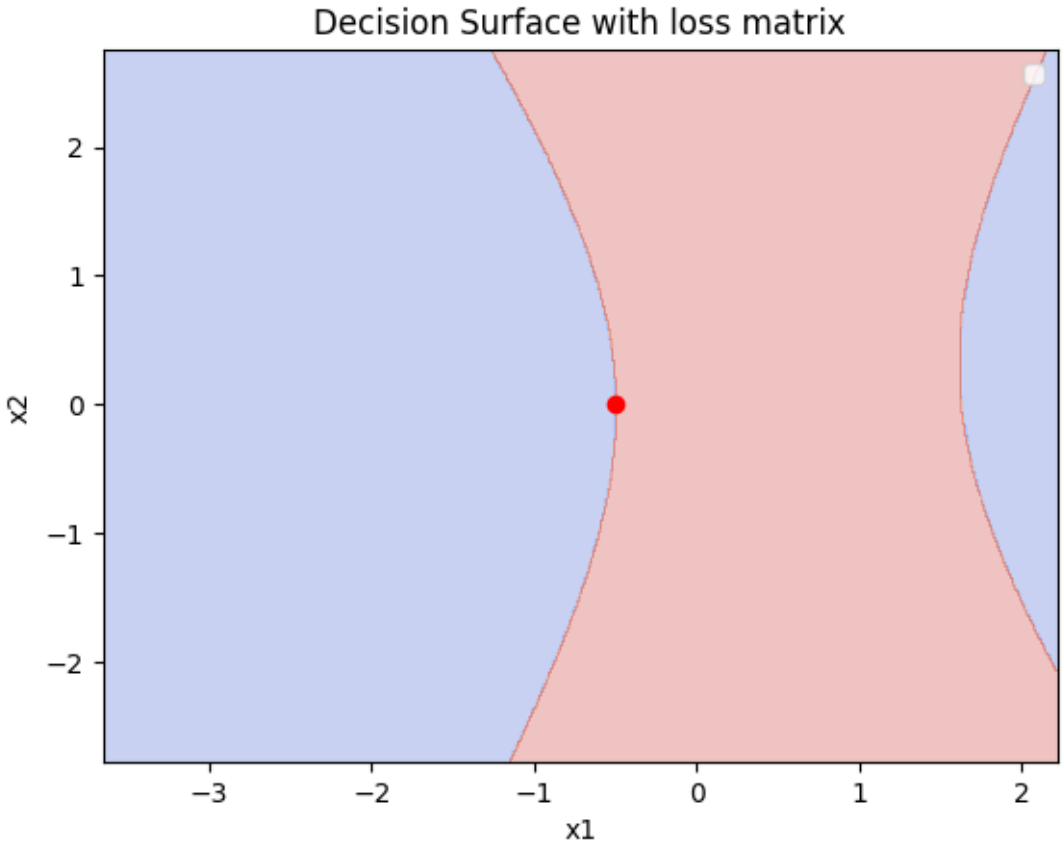
برای این کار از طبقه بندی کننده بیزی استفاده می‌کنیم. در ورودی های بخش predict قسمتی را نیز برای loss matrix در نظر می‌گیریم.

```python
from scipy.stats import multivariate_normal

class BayesianClassifier:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.means = np.array([X[y == c].mean(axis=0) for c in self.classes])
        self.covs = np.array([np.cov(X[y == c].T) + 0.01*np.eye(X.shape[1]) for c in self.classes])
        self.priors = np.array([np.mean(y == c) for c in self.classes])

    def predict(self, X, use_loss=False, loss_matrix=None):
        probs = [multivariate_normal.pdf(X, mean=m, cov=c) for m, c in zip(self.means, self.covs)]
        likelihood = np.array(probs).T
        evidence = np.sum(likelihood * self.priors, axis=1)
        posterior = likelihood * self.priors / evidence[:, np.newaxis]
        if use_loss and loss_matrix is not None:
            expected_loss = np.dot(posterior, loss_matrix)
            return self.classes[np.argmin(expected_loss, axis=1)]
        else:
            return self.classes[np.argmax(posterior, axis=1)]
```

در قسمت predict در صورتی که loss matrix داشته باشیم معادله خروجی آن به صورت معادله ذکر شده خواهد شد. حال اگر برای مثال loss matrix برابر با $\begin{pmatrix} 0 & 1 \\ 5 & 0 \end{pmatrix}$ باشد، نقطه [-0.5,0] در ناحیه نرمال قرار می‌گیرد.



حال اگر $loss\ matrix = \begin{pmatrix} 0 & 1 \\ 1.5 & 0 \end{pmatrix}$ قرار دهیم نقطه مدنظر به صورت زیر جاگیری می‌کند.



این امر نشان می‌دهد که اگر هر یک از مقادیر را کم یا زیاد کنیم، می‌توانیم این نقطه را در کلاس مدنظر خود قرار دهیم. اگر λ_{12} کم شود این نقطه در ناحیه نرمال قرار می‌گیرد و بالعکس. برای λ_{21} نیز اگر مقدار آن زیاد شود، این نقطه در ناحیه نرمال قرار می‌گیرد و بالعکس.

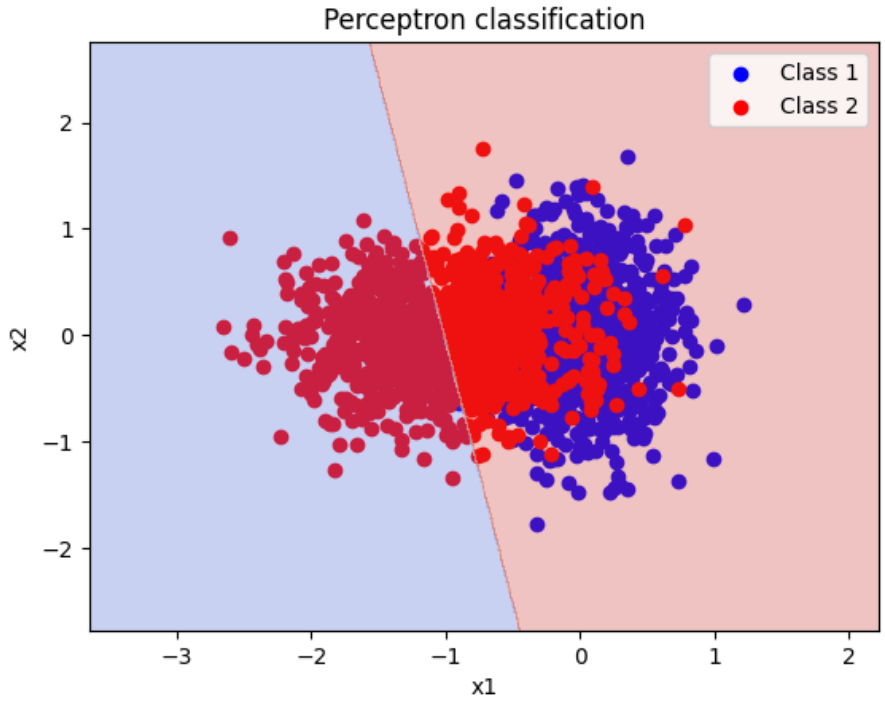
c) ابتدا بر اساس میانگین و کوواریانس داده شده ۱۰۰۰ داده تولید می‌کنیم و سپس آن‌ها را به دو قسمت train و test تقسیم می‌کنیم. به عنوان برچسب، به داده‌های نرمال برچسب ۱ و به داده‌های فالت برچسب ۱- می‌زنیم.

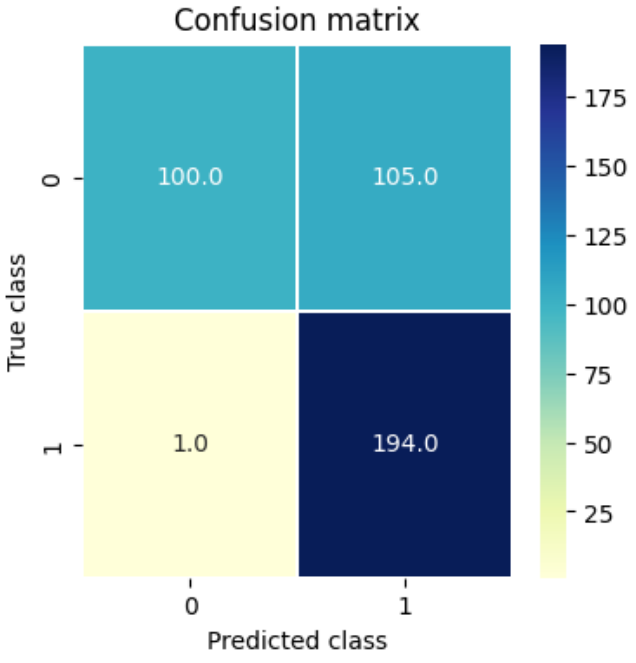
```
np.random.seed(42)
#normal
mean1 = [0, 0]
cov1 = [[0.1, 0], [0, 0.3]]
#fault
mean2 = [-1, 0]
cov2 = [[0.3, 0], [0, 0.2]]

# Generate random data with Gaussian distribution for two classes
class1 = np.random.multivariate_normal(mean1, cov1, 1000)
class2 = np.random.multivariate_normal(mean2, cov2, 1000)

# Concatenate data points and corresponding labels for training the perceptron
X = np.concatenate((class1, class2))
y = np.concatenate((np.ones(len(class1)), -np.ones(len(class2))))


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

یکی از ساده ترین طبقه بندی کننده های خطی، perceptron می‌باشد. دستور آن به صورت زیر می‌باشد.

```
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score, confusion_matrix

perceptron = Perceptron(eta0=0.01)

# fit the classifier to the training data
perceptron.fit(X_train, y_train)

# make predictions on the test data
y_pred = perceptron.predict(X_test)
```

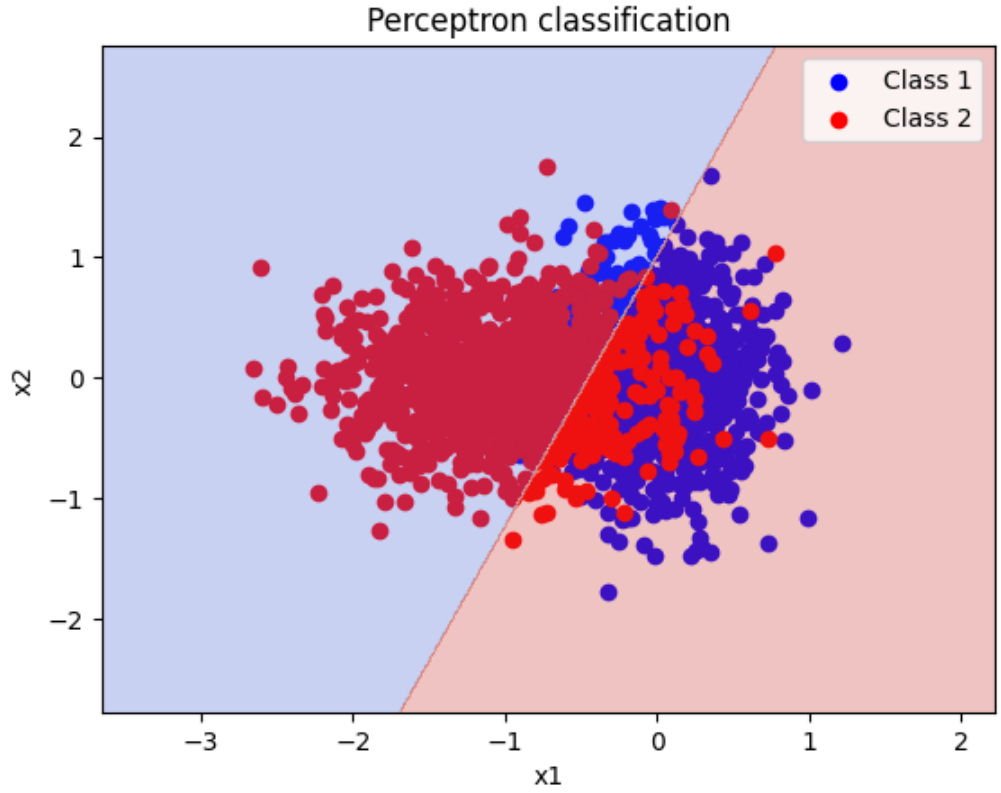
حال براساس این روش، سعی می‌کنیم داده‌ها را دسته بندی کنیم.

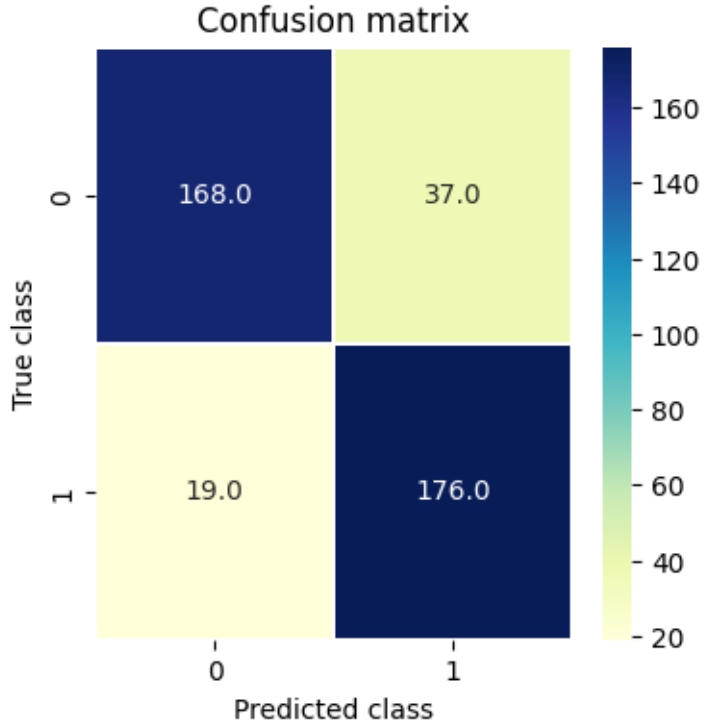





```
Acc= 0.735
MAR= 0.2586616635397123
FAR= 0.34381171823568135
Precision= [0.99009901 0.64882943]
Recall= [0.48780488 0.99487179]
```

حال اگر مقدار لرنینگ ریت (eta0) را ۰.۴۵ قرار دهیم خواهیم داشت.

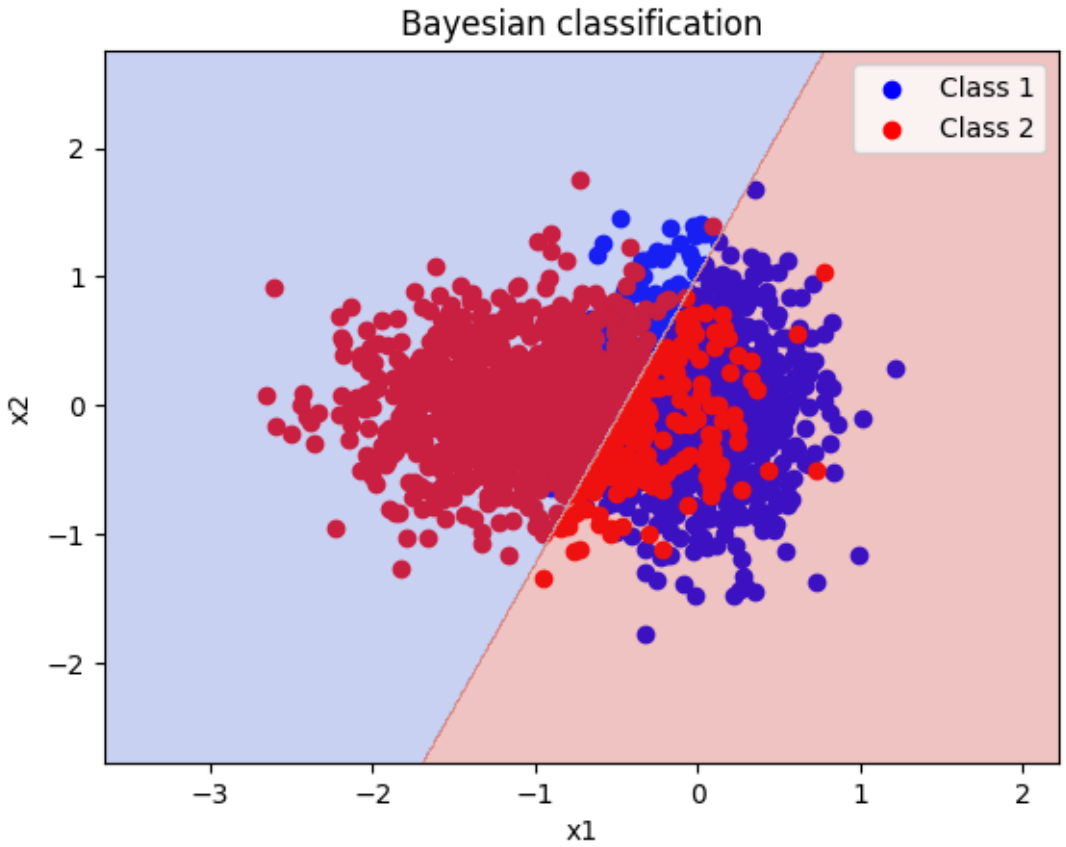


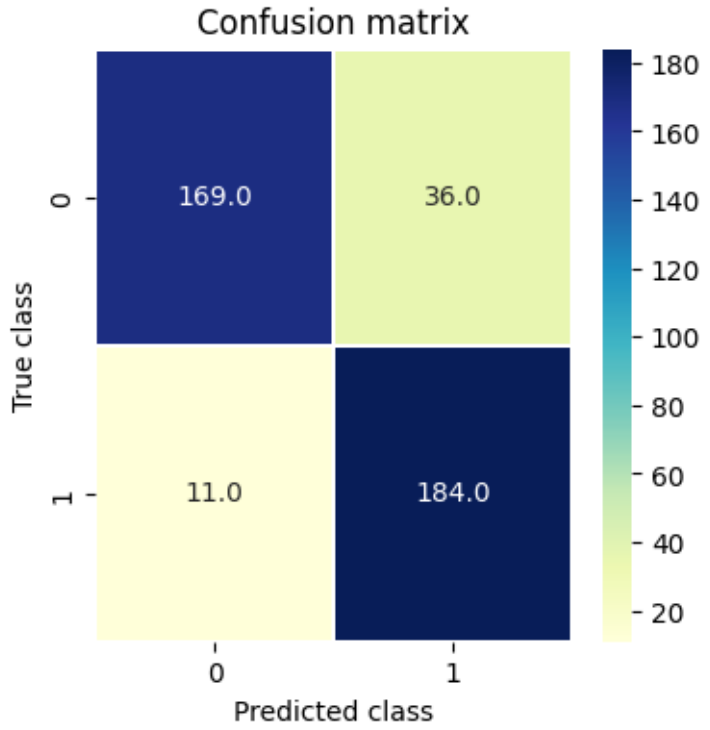



```
Acc= 0.86
MAR= 0.1389618511569731
FAR= 0.24167760871244304
Precision= [0.89839572 0.82629108]
Recall= [0.8195122 0.9025641]
```

مشاهده می‌شود که نتایج بهتر شده اند. یکی از دلایل مهم این امر این است که با تغییر لرنینگ ریت به این مقدار، زاویه Decision surface به یک زاویه جدید تغییر پیدا کرده که در این زاویه بهتر می‌توان دادها را دسته بندی کرد. اگر روی همین راستا یک LDA اعمال شود، احتمال زیاد این داده‌ها به راحتی می‌توانند جداسازی شوند.

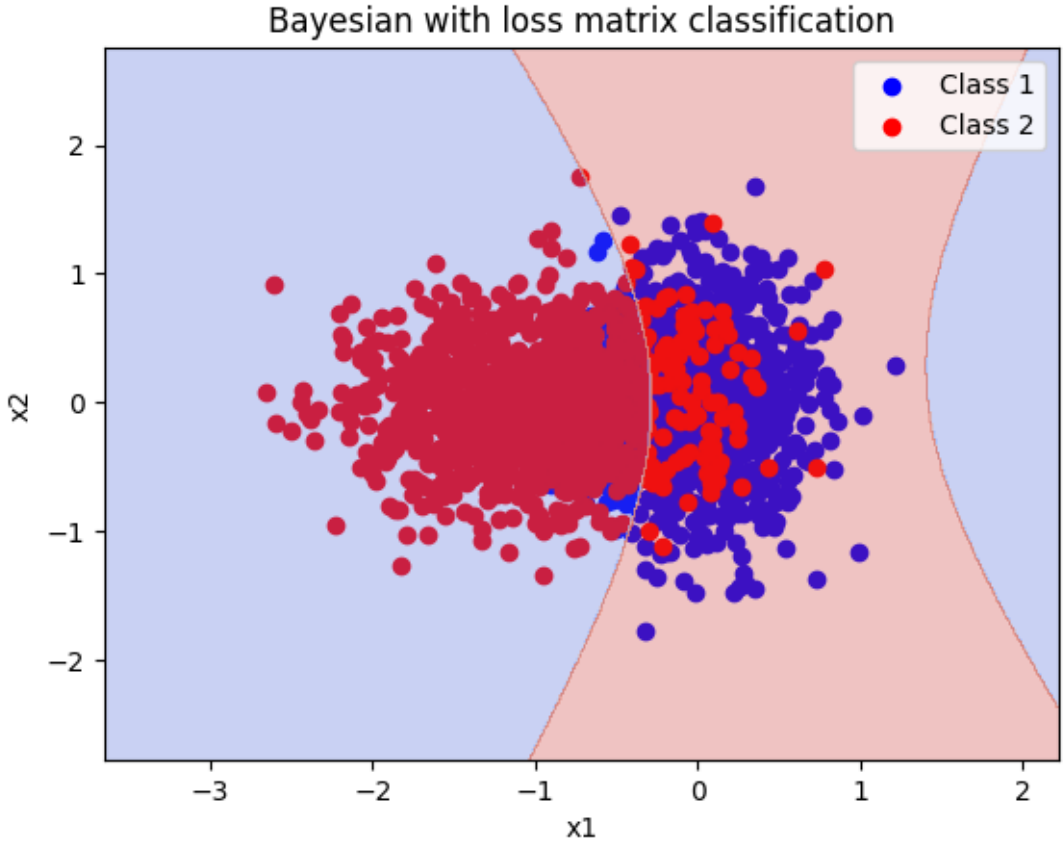
d) اگر طبقه بندی کننده بیزی را روی داده‌ها اعمال کنیم، خواهیم داشت.

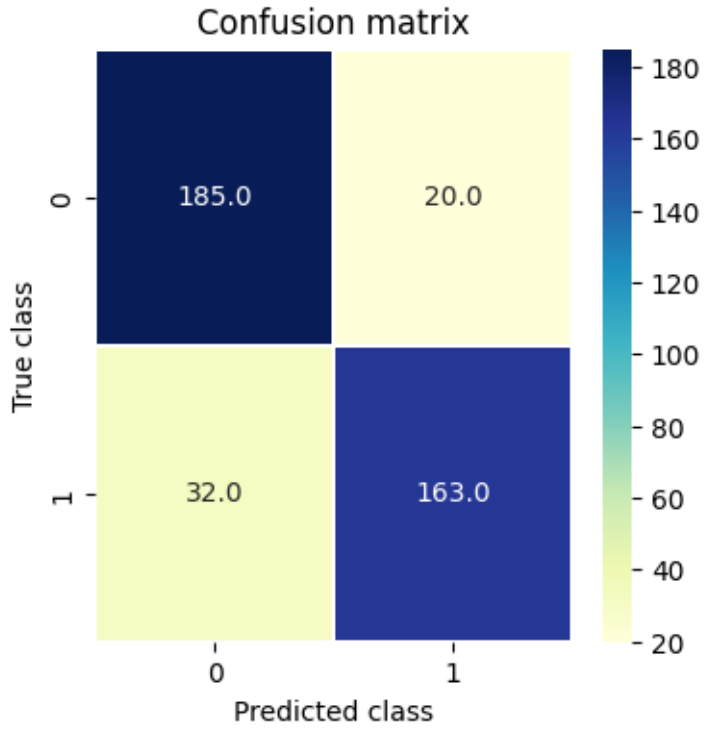





```
Acc= 0.8825
MAR= 0.11601000625390868
FAR= 0.20277565161342304
Precision= [0.93888889 0.83636364]
Recall= [0.82439024 0.94358974]
```

همان‌طور که مشاهده می‌شود، توانسته خیلی بهتر داده ها را دسته بندی کند. بهترین accuracy بدست آمده در قسمت قبلی ۰.۸۶ است که پس از بررسی مقادیر مختلف لرنینگ ریت بدست آمده اما در این طبقه بندی بدون تغییر هیچ چیز خاصی و حتی بدون loss matrix به این مقدار رسیده ایم. نکته بعدی آن این است که این روش توانسته مقادیر FAR و MAR را نیز کاهش دهد. علاوه بر این رفتاری بیزی غیرخطی است و توانایی‌های بیشتری را از یک طبقه بندی کننده خطی دارد. حال با دستکاری loss matrix به مقدار $\begin{pmatrix} 0 & 1.5 \\ 0.7 & 0 \end{pmatrix}$ نتایج زیر حاصل می‌شوند.


Bayesian with loss matrix classification
Class 1
Class 2
x2
x1
2
1
0
−1
−2
−3
−2
−1
0
1
2


Confusion matrix
185.0
20.0
32.0
163.0
True class
Predicted class
0
1
180
160
140
120
100
80
60
40
20

```
Acc= 0.87
MAR= 0.1308317698561601
FAR= 0.2298580518844836
Precision= [0.85253456 0.89071038]
Recall= [0.90243902 0.83589744]
```

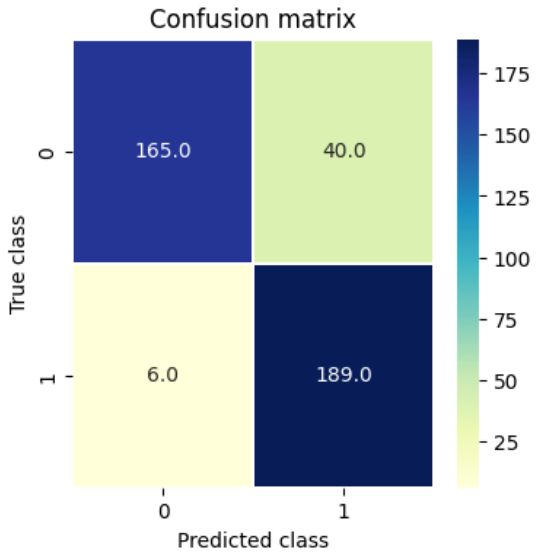
هرچند که مقدار accuracy بهتر شده اما مقادیر FAR و MAR دوباره افزایش داشته‌اند.

سوال چهارم

برای این دسته داده پارزن را ابتدا با bw=0.5 به صورت زیر اعمال می‌کنیم.

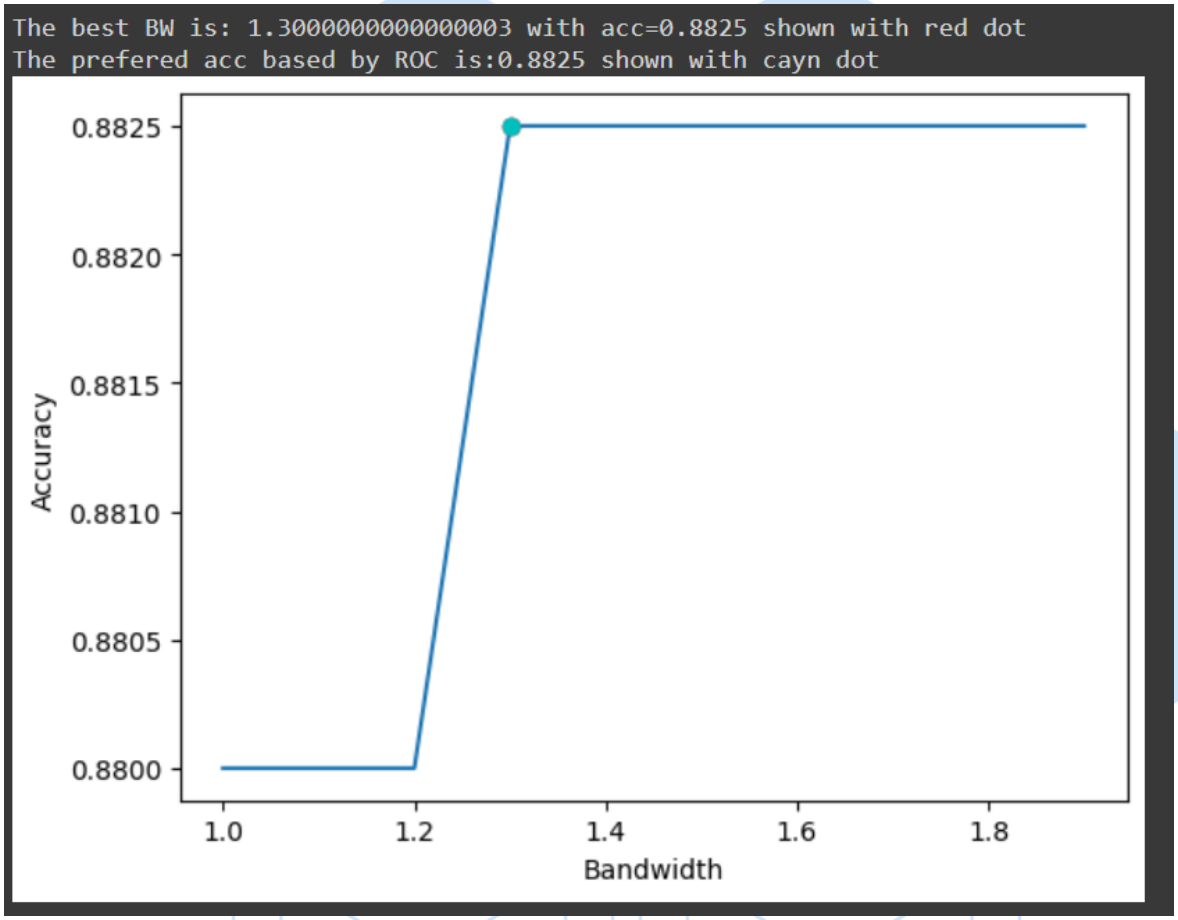
```python
from sklearn.neighbors import KernelDensity as KDE
kde_models = []
for i in range(2): #number of classes
    kde = KDE(kernel='gaussian', bandwidth=0.5)
    kde.fit(X_train[y_train == i])
    kde_models.append(kde)

  # Calculate the probability densities for each observation in the testing dataset
prob_densities = []
for i in range(len(X_test)):
    prob_densities_i = [kde.score_samples([X_test[i]])[0] for kde in kde_models]
    prob_densities.append(prob_densities_i)

  # Assign a class label to each observation based on the highest probability density
y_pred = np.argmax(prob_densities, axis=1)
```

لازم به ذکر است که برچست ها به صورت ۱ برای نرمال و ۰ برای فالت تغییر پیدا کردند. خروجی به صورت زیر است.

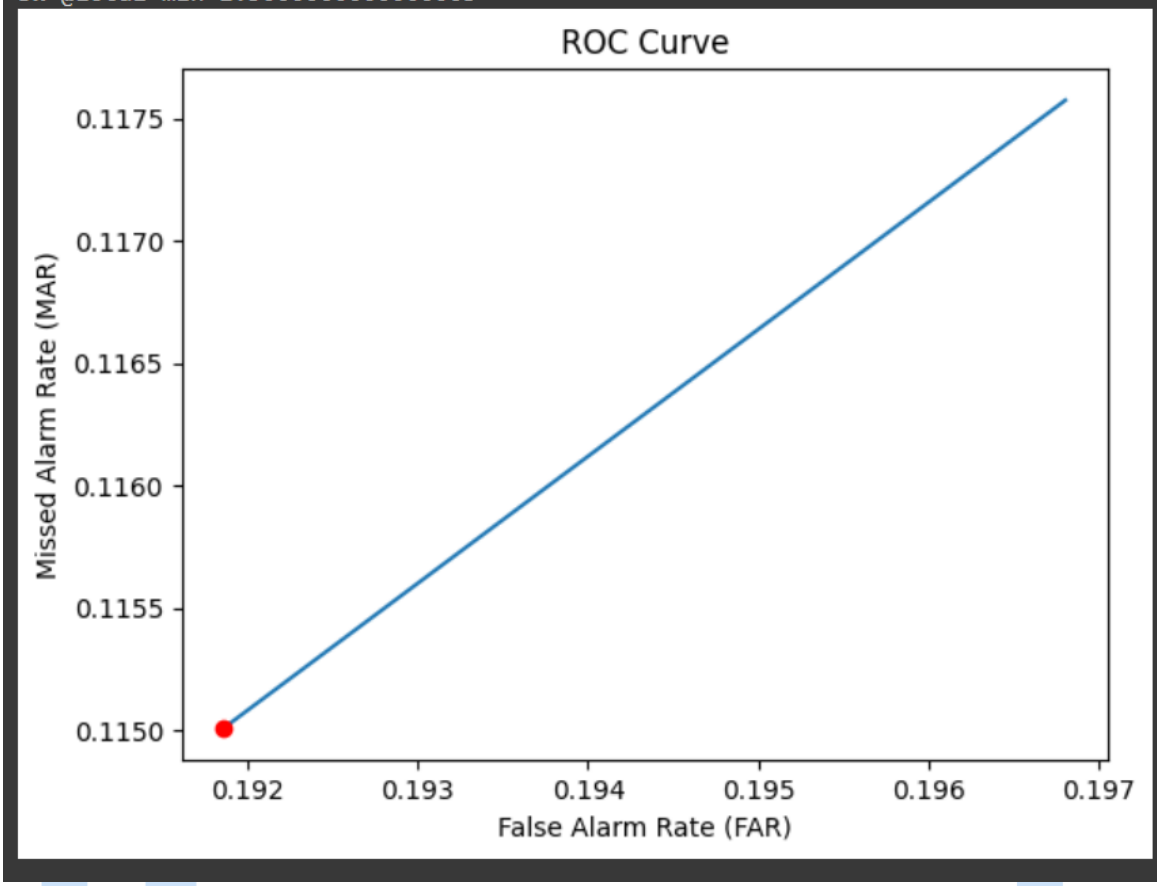


```
Acc= 0.885
MAR= 0.11294559099437149
FAR= 0.19311605239110569
Precision= [0.96491228 0.82532751]
Recall= [0.80487805 0.96923077]
```

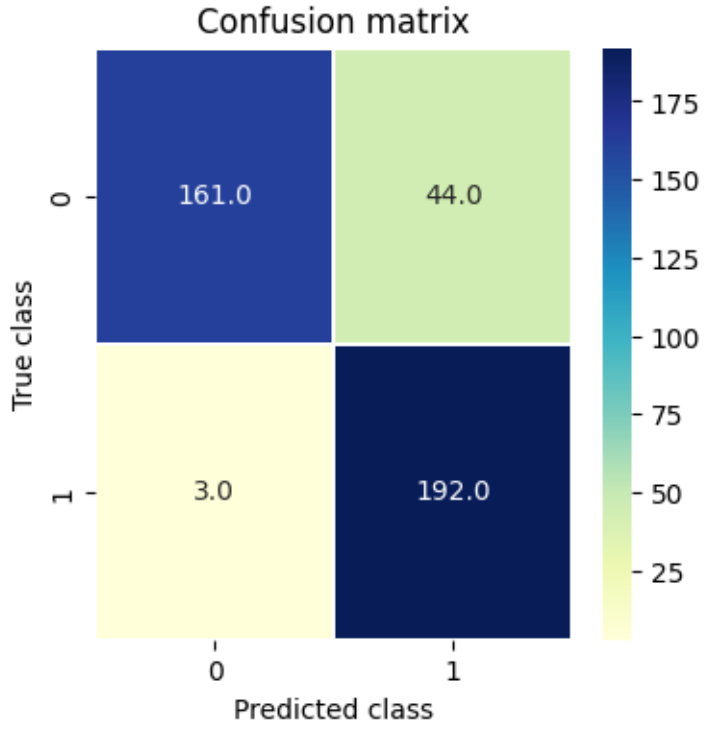
حال سعی می‌کنیم تا برای این داده ها بهترین bw را بدست آوریم. از بازه ۱ تا ۲ با گام ۰.۱ نتایج زیر بدست آمده است.



```
Accuracy @local min=0.8825
BW @local min=1.3000000000000003
```

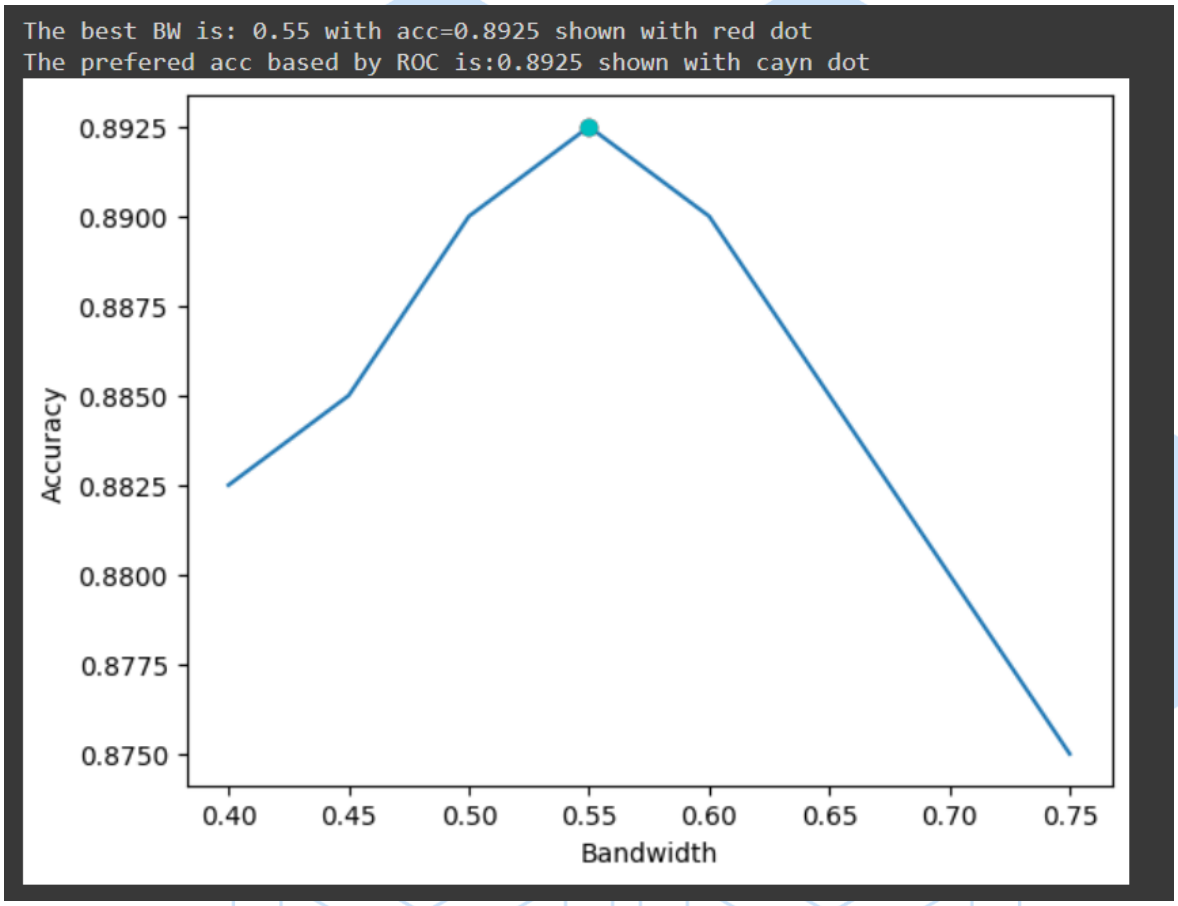


بهترین bw بدست آمده را اعمال می‌کنیم.

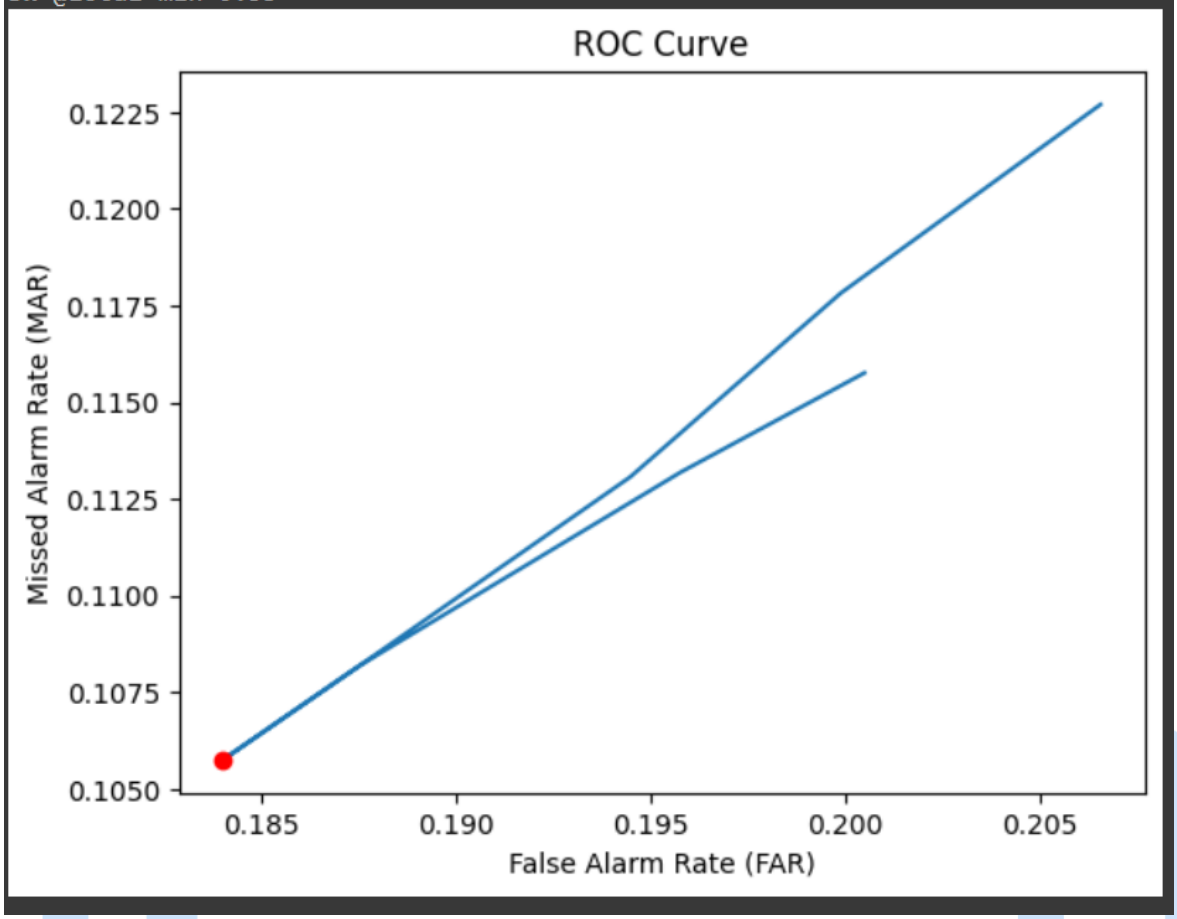


```
Acc= 0.8825
MAR= 0.1150093808630394
FAR= 0.19185834246075212
Precision= [0.98170732 0.81355932]
Recall= [0.78536585 0.98461538]
```

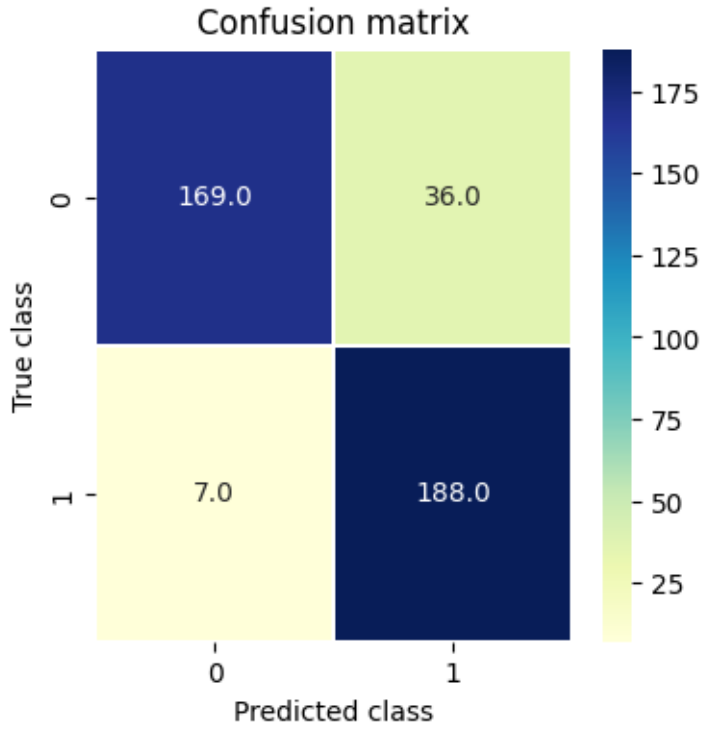
حال به دلیل اینکه داده ها از جنس گوسی هستند یک کرنل دیگر را نیز برای بررسی پیش می‌بریم. با کرنل tophat در بازه ۰.۴ تا ۰.۸ با گام ۰.۰۵ نتایج زیر بدست آمد.



```
Accuracy @local min=0.8925
BW @local min=0.55
```



برای بهترین bw پارزن را اعمال می‌کنیم.



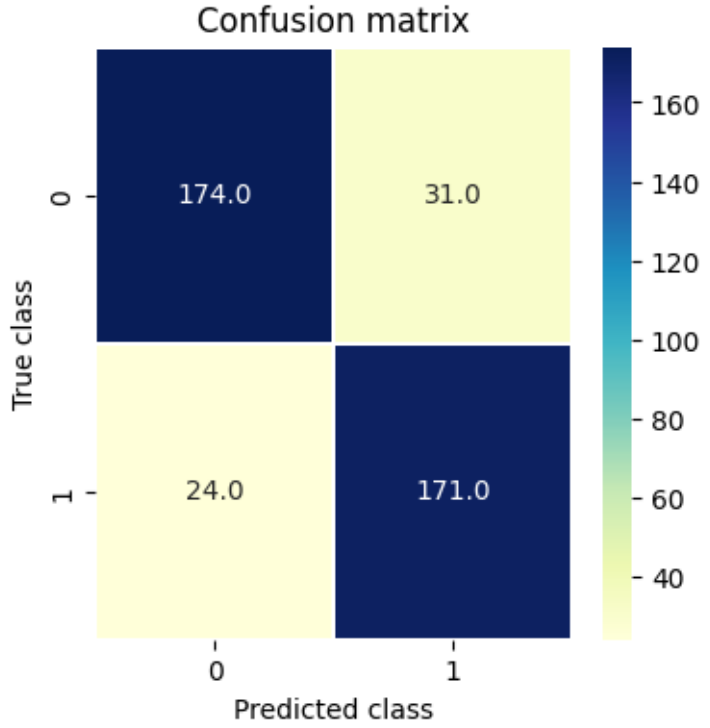

```
Acc= 0.8925
MAR= 0.10575359599749842
FAR= 0.1840310587075305
Precision= [0.96022727 0.83928571]
Recall= [0.82439024 0.96410256]
```

مشاهده می‌شود که در این روش کمی دسته بندی بهتر انجام شده، مقدار accuracy کمی بیشتر و مقادیر FAR و MAR نیز کمی کاهش یافته اند.

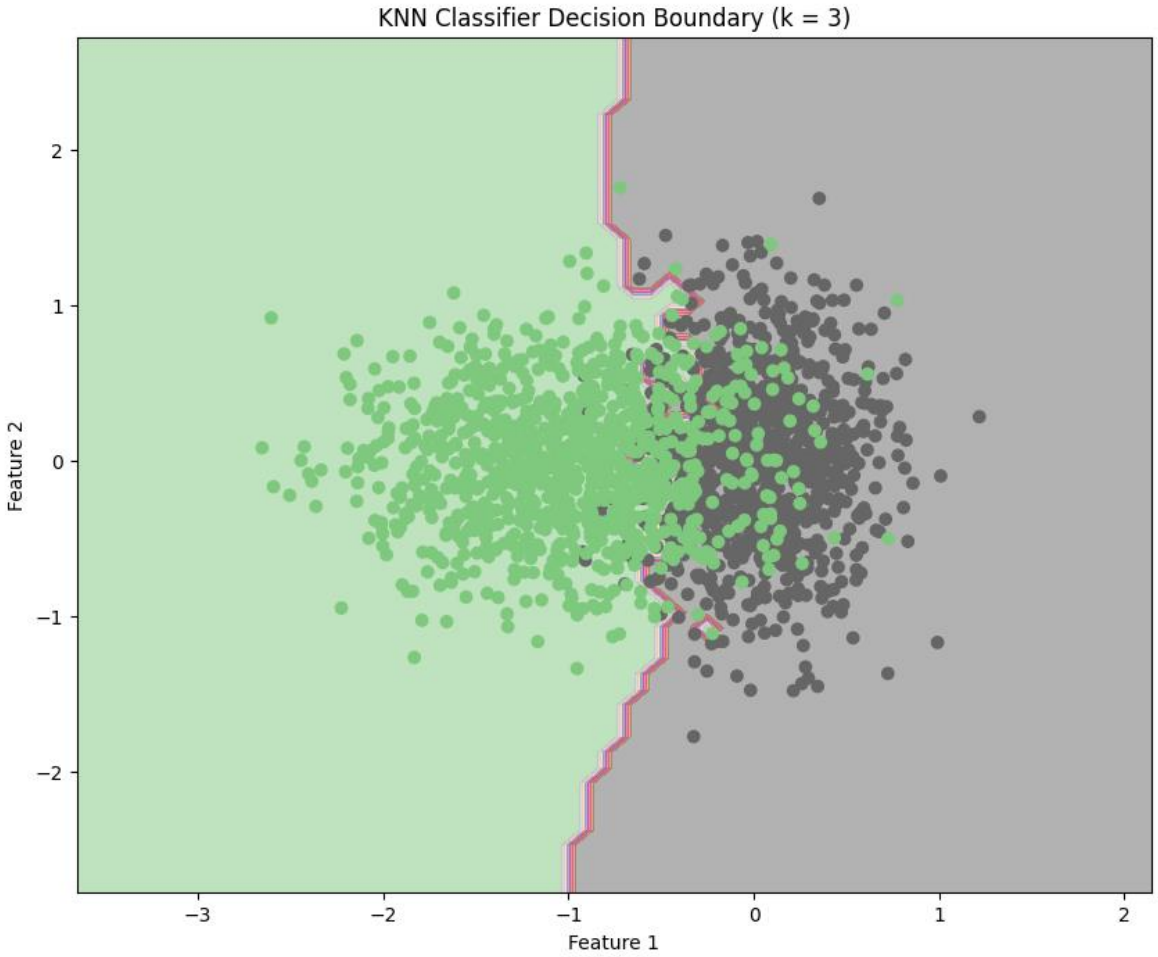
حال با روش Knn داده ها را بررسی می کنیم.

```python
from collections import Counter
from scipy.spatial import cKDTree

class KNNClassifier:
    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y
        self.tree = cKDTree(X)

    def predict(self, X):
        _, indices = self.tree.query(X, k=self.k)
        k_labels = self.y_train[indices]
        y_pred = np.array([Counter(labels).most_common()[0][0] for labels in k_labels.squeeze()])
        return y_pred
```

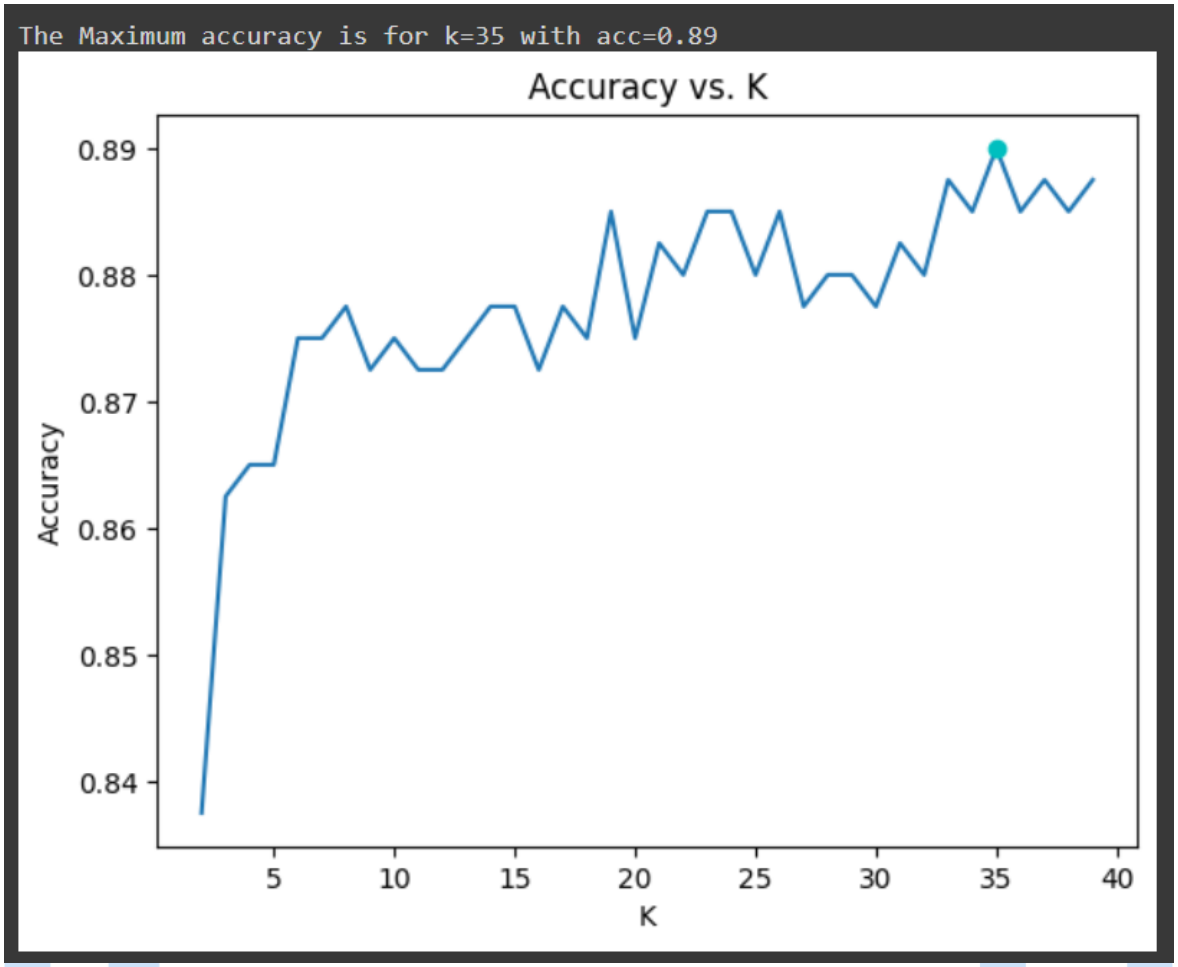
با مقدار k=3 خواهیم داشت.

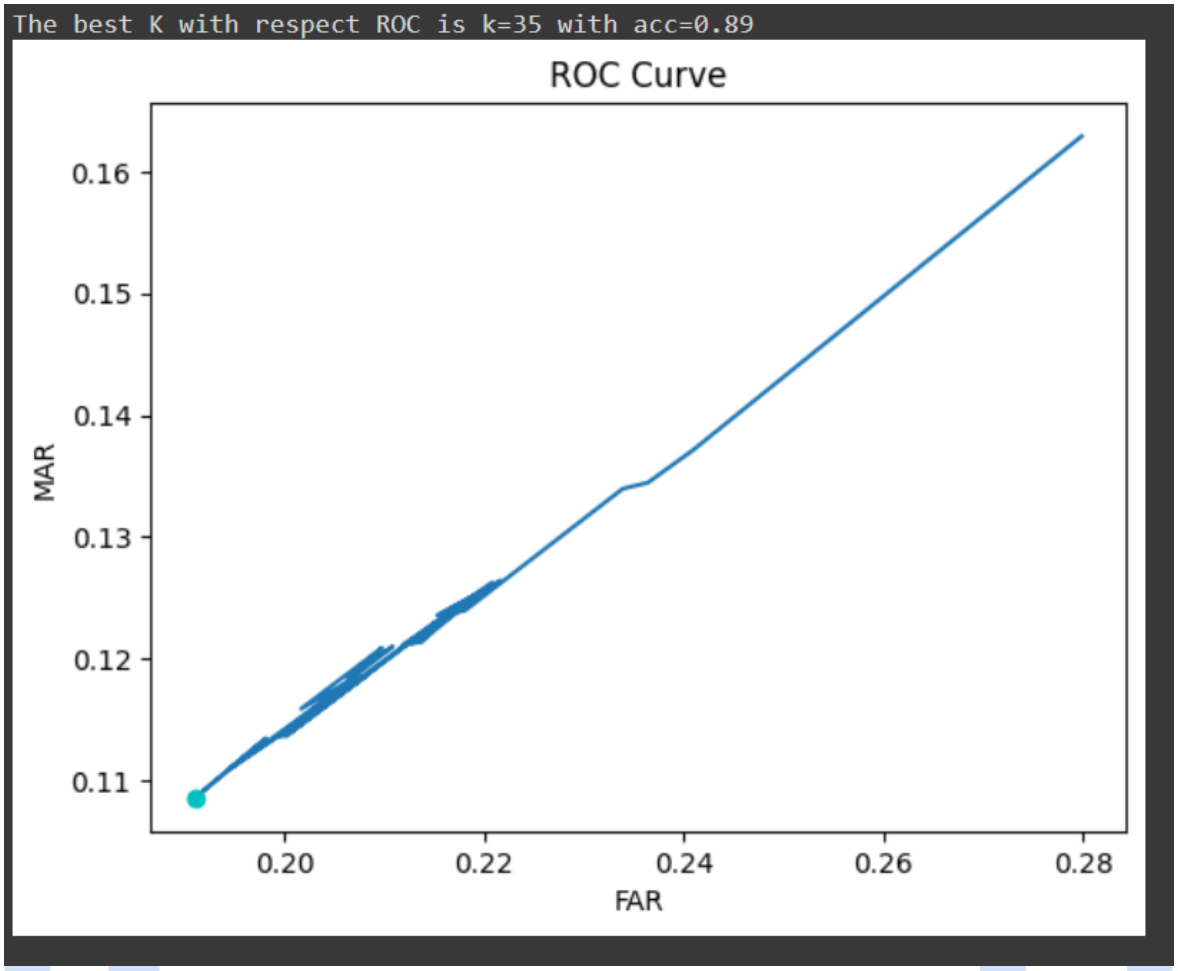



```
Acc= 0.8625
MAR= 0.13714821763602253
FAR= 0.24094497329928025
Precision= [0.87878788 0.84653465]
Recall= [0.84878049 0.87692308]
```

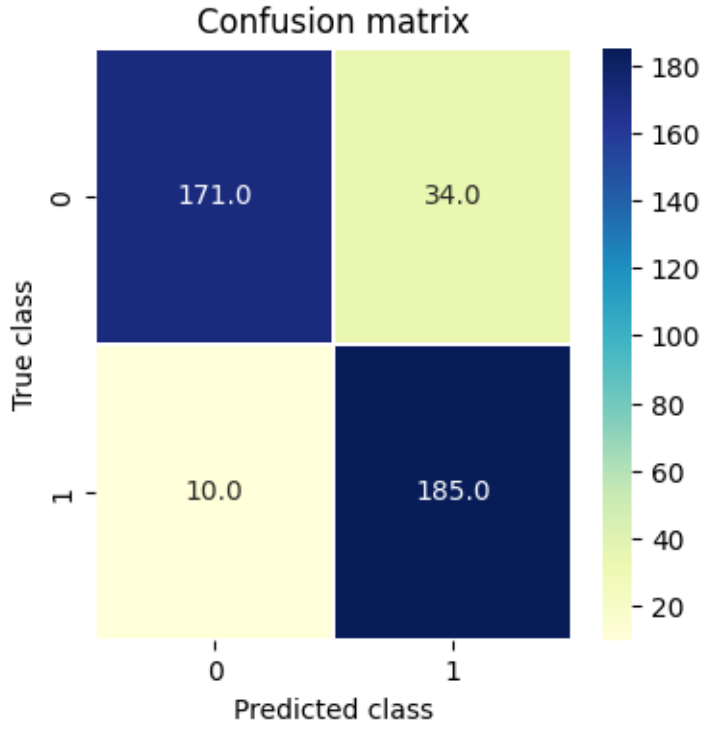


حال سعی می‌کنیم بهترین k را پیدا کنیم. نتایج زیر بدست آمده اند.

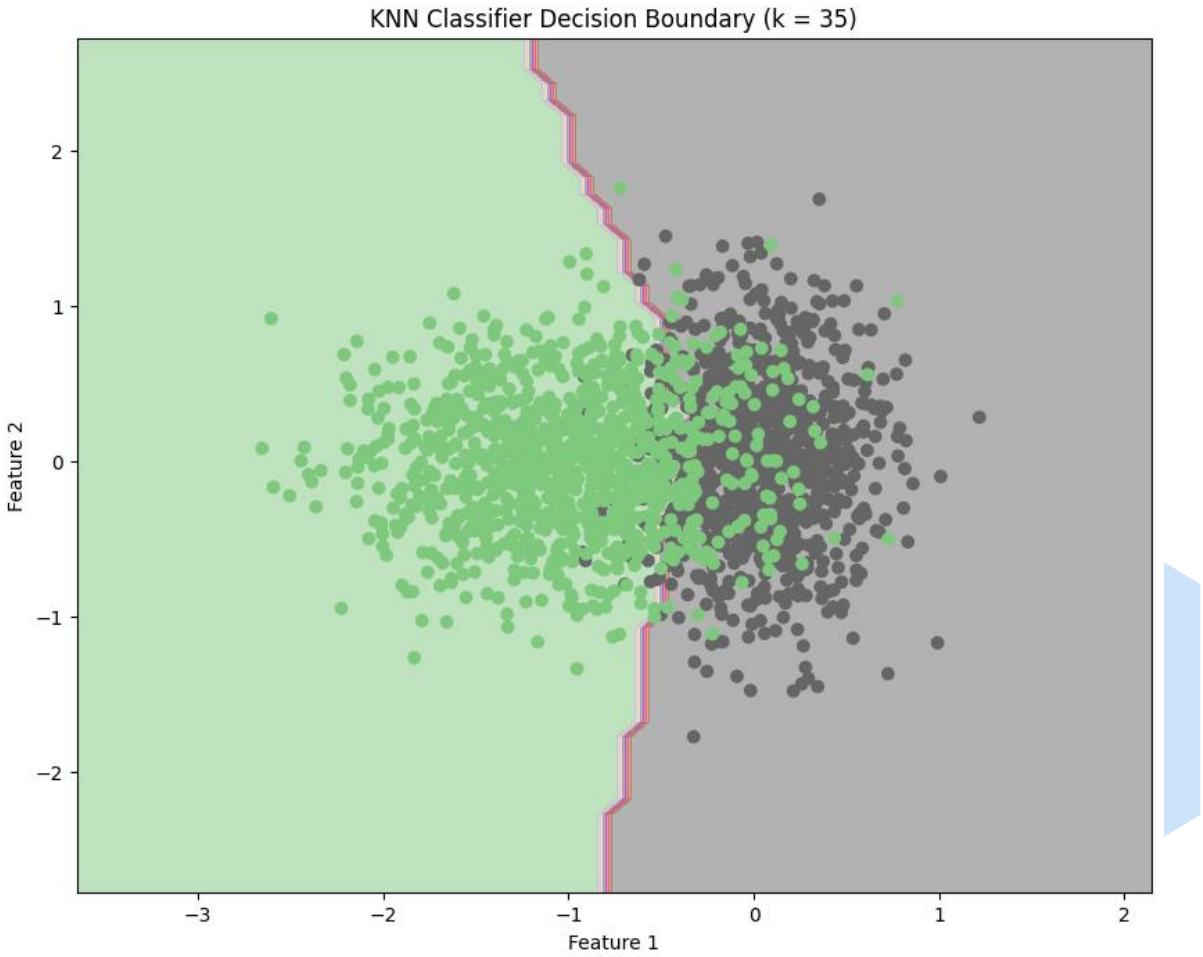

The Maximum accuracy is for k=35 with acc=0.89
Accuracy vs. K
Accuracy
K
0.89
0.88
0.87
0.86
0.85
0.84
5
10
15
20
25
30
35
40



بهترین k بدست آمده را به KNN اعمال می‌کنیم.



```
Acc= 0.89
MAR= 0.10856785490931833
FAR= 0.1910399020308195
Precision= [0.94475138 0.84474886]
Recall= [0.83414634 0.94871795]
```



مشاهده می‌شود که هم مقدار accuracy و هم مقادیر FAR و MAR بهبود یافته اند.

لازم به ذکر است که در هر دو روش بهترین حالت با اولویت بر نمودار ROC انتخاب شده است؛ چراکه کاهش مقادیر FAR و MAR برای ما از افزایش accuracy بیشتر اولویت دارد.