7th Homework

Chalkiopoulos Georgios | *p*3352124

March 13, 2022

Exercise 1.

It is:

$$\begin{aligned}\prod_{n=1}^{N} P(y_n|x_n;\boldsymbol{\theta}) &= \prod_{n=1}^{N} P(0|x_n;\boldsymbol{\theta})^{y_n} P(1|x_n;\boldsymbol{\theta})^{y_n-1} \\ &= \prod_{n=1}^{N} P(\sigma(\boldsymbol{\theta}^T x_n))^{y_n} P(1-\sigma(\boldsymbol{\theta}^T x_n))^{y_n-1} \\ &= \prod_{n=1}^{N} P(s_n)^{y_n} P(1-s_n)^{y_n-1} \qquad (1.1)\end{aligned}$$

The negative log-likelihood is:

$$L(\boldsymbol{\theta}) = -\sum_{n=1}^{N} y_n \left(\ln s_n + (1-y_n)\ln(1-s_n)\right) \qquad (1.2)$$

The gradient of (1.2) is:

$$\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\frac{\partial \sum_{n=1}^{N} y_n \left(\ln s_n + (1-y_n)\ln(1-s_n)\right)}{\partial \boldsymbol{\theta}} \qquad (1.3)$$

,where $s_n = \sigma(\boldsymbol{\theta}^T \boldsymbol{x})$ thus $\dfrac{\partial \sigma(\boldsymbol{\theta}^T \boldsymbol{x})}{\partial \boldsymbol{\theta}} = \boldsymbol{x}$.

Using $\dfrac{\partial \sigma(t)}{\partial t} = \sigma(t)(1-\sigma(t)) \Rightarrow \dfrac{\partial s_n}{\partial \theta} = s_n(1-s_n)x_n$ it is:

$$\begin{aligned}\frac{\partial L(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} &= -\sum_{n=1}^{N} \frac{y_n}{s_n}\frac{\partial s_n}{\partial \theta} + \frac{1}{(1-s_n)}\frac{\partial 1-s_n}{\partial \theta} - \frac{y_n}{(1-s_n)}\frac{\partial 1-s_n}{\partial \theta} \\ &= -\sum_{n=1}^{N} \frac{y_n}{s_n} s_n(1-s_n)x_n - \frac{1}{(1-s_n)} s_n(1-s_n)x_n + \frac{y_n}{(1-s_n)} s_n(1-s_n)x_n \\ &= -\sum_{n=1}^{N} y_n x_n - y_n s_n x_n - s_n x_n + y_n s_n x_n = \sum_{n=1}^{N} (s_n - y_n)x_n \qquad (1.4)\end{aligned}$$

Using simple equations that have been calculated in previous assignments it i easy to show that (1.4) can be written as:

$$\nabla L(\boldsymbol{\theta}) = X^T(\boldsymbol{s} - \boldsymbol{y})$$

and the gradient descent:

$$\boldsymbol{\theta}^i = \boldsymbol{\theta}^{i-1} - \mu_i \nabla L(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{i-1}} = \boldsymbol{\theta}^{i-1} - \mu_i X^T(\boldsymbol{s}^{(i-1)} - \boldsymbol{y})$$

Exercise 2.

Exercise 3.

The points are presented in Figure 1.

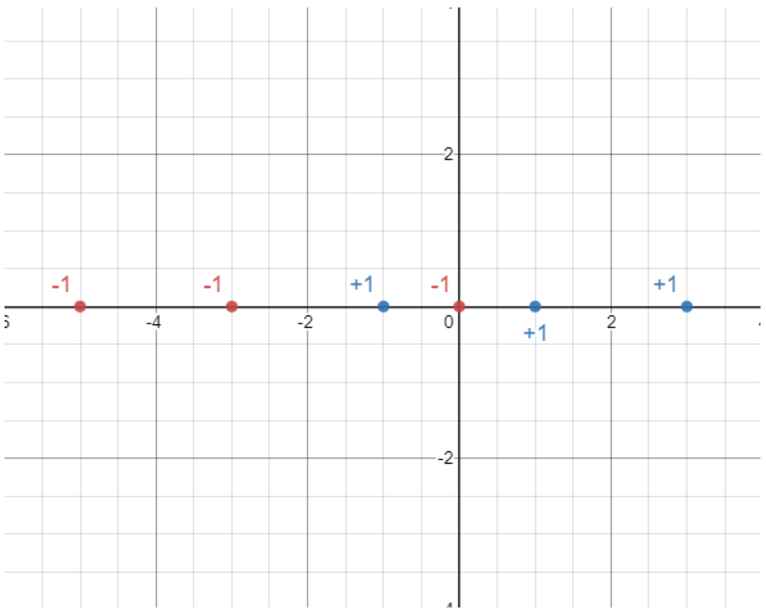


Figure 1: Dataset points.

As mentioned in the theory problems of this kind can be solved using decision trees.

Exercise 4.

Using the data that is shown in the lecture's slides we have:

t_Y: x_2, x_3, x_4

t_N: x_1, x_5

Where $x_1, x_3 \in \omega_1$ and $x_2, x_4, x_5 \in \omega_1$. Using the computation for the entropy reductions, knowing that the entropy of the root node is $I = 0.9710$ it is:

$PY(\omega_1) = 1/3 \quad PY(\omega_2) = 2/3 \quad IY = 0.918$

$PN(\omega_1) = 1/2 \quad PN(\omega_2) = 2/3 \quad IN = 1$

ΔI = I - (3/5) 0.918 - 2/5 = 0.02

Exercise 5.

Considering the SVM problem we will derive the equations of the Karush-Kuhn-Tacker for the conditions (1) and (2) of the slides.
It is:

$$L(\boldsymbol{\theta}, \theta_0, \lambda) = \frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} - \sum_{i=1}^{N} \lambda_i [y_i(\boldsymbol{\theta}^T \boldsymbol{x}_i + \theta_0) - 1] \tag{5.1}$$

$$\frac{\partial L(\boldsymbol{\theta}, \theta_0, \lambda)}{\partial \boldsymbol{\theta}} = 0 \tag{5.2}$$

$$\frac{\partial L(\boldsymbol{\theta}, \theta_0, \lambda)}{\partial \theta_0} = 0 \tag{5.3}$$

Moreover:

$$\frac{\partial L(\boldsymbol{\theta}, \theta_0, \lambda)}{\partial \boldsymbol{\theta}} = \frac{\partial \left(\frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} - \sum_{i=1}^{N} \lambda_i [y_i(\boldsymbol{\theta}^T \boldsymbol{x}_i + \theta_0) - 1]\right)}{\partial \boldsymbol{\theta}}$$

$$= \boldsymbol{\theta} - \sum_{i=1}^{N} \lambda_i \ y_i \ \boldsymbol{x_i} \overset{5.2}{\Rightarrow}$$

$$\boldsymbol{\theta} = \sum_{i=1}^{N} \lambda_i \ y_i \ \boldsymbol{x_i} \tag{5.4}$$

Similarly:

$$\frac{\partial L(\boldsymbol{\theta}, \theta_0, \lambda)}{\partial \theta_0} = \frac{\partial \left(\frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} - \sum_{i=1}^N \lambda_i [y_i(\boldsymbol{\theta}^T \boldsymbol{x}_i + \theta_0) - 1] \right)}{\partial \boldsymbol{\theta_0}}$$

$$= -\sum_{i=1}^N \lambda_i \ y_i = 0 \overset{5.3}{\Rightarrow}$$

$$\sum_{i=1}^N \lambda_i \ y_i = 0 \tag{5.5}$$

By substituting (5.4) and (5.5) to (5.1) we have that:

$$L(\boldsymbol{\theta}, \theta_0, \lambda) = \frac{1}{2}\boldsymbol{\theta}^T\boldsymbol{\theta} - \sum_{i=1}^N \lambda_i [y_i(\boldsymbol{\theta}^T \boldsymbol{x}_i + \theta_0) - 1] \Rightarrow$$

$$= \frac{1}{2} \sum_{i=1}^N \lambda_i y_i \boldsymbol{x}_i^T \sum_{j=1}^N \lambda_j y_j \boldsymbol{x}_j - \sum_{i=1}^N \lambda_i y_i \boldsymbol{\theta}^T \boldsymbol{x}_i - \theta_0 \sum_{i=1}^N \lambda_i y_i + \sum_{i=1}^N \lambda_i \Rightarrow$$

$$= \frac{1}{2} \sum_{ij} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j - \sum_{ij} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j + \sum_{i=1}^N \lambda_i \Rightarrow$$

$$= \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{ij} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j$$

Exercise 6.

With ω_1 having $x_1 = [-1, 1]^T$ and $x_2 = [-1, -1]^T$ and ω_2 having $x_3 = [1, -1]^T$ and $x_4 = [1, 1]^T$ the poitns are shown in Figure 2. Intuitively we would consider the line $x_1 = 0$ and the one the SVM would return. Using the dual representation of the SVM

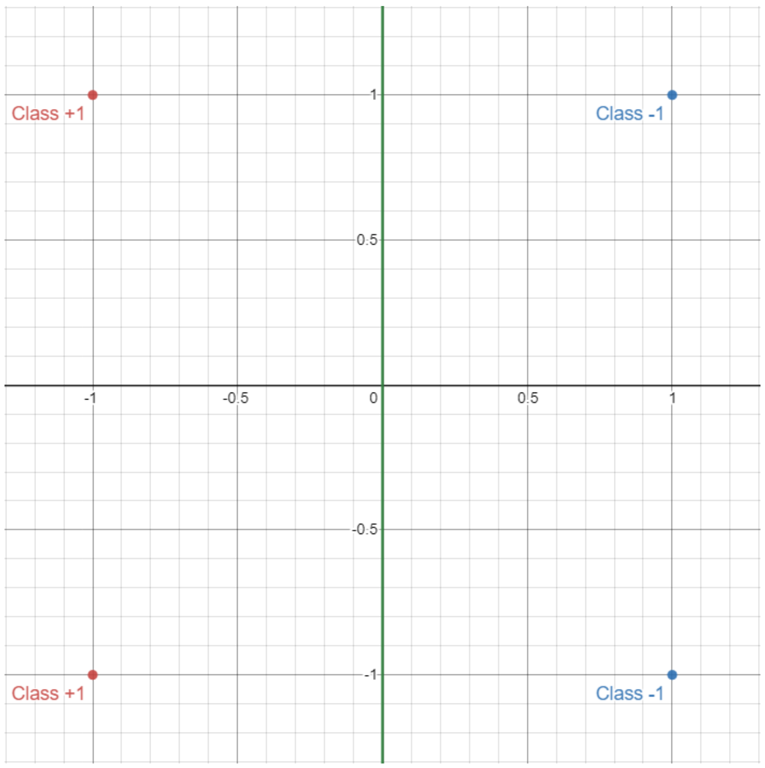


Figure 2: Dataset points and suspected separation line.

problem, with $y_1 = y_2 = 1$ and $y_3 = y_4 = -1$ it is:

$$J_i^*(\lambda) = \sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{ij} \lambda_i \lambda_j y_i y_j \boldsymbol{x_i}^T \boldsymbol{x_j}$$

$$\begin{aligned}
&= \sum_{i=1}^{N} \lambda_i \\
&- \frac{1}{2}\lambda_1\lambda_2 \begin{bmatrix}-1 & 1\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix} + \frac{1}{2}\lambda_1\lambda_3 \begin{bmatrix}-1 & 1\end{bmatrix}\begin{bmatrix}1\\-1\end{bmatrix} + \frac{1}{2}\lambda_1\lambda_4 \begin{bmatrix}-1 & 1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} \\
&- \frac{1}{2}\lambda_2\lambda_1 \begin{bmatrix}-1 & -1\end{bmatrix}\begin{bmatrix}-1\\1\end{bmatrix} + \frac{1}{2}\lambda_2\lambda_3 \begin{bmatrix}-1 & -1\end{bmatrix}\begin{bmatrix}1\\-1\end{bmatrix} + \frac{1}{2}\lambda_2\lambda_4 \begin{bmatrix}-1 & -1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} \\
&+ \frac{1}{2}\lambda_3\lambda_1 \begin{bmatrix}1 & -1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} + \frac{1}{2}\lambda_3\lambda_2 \begin{bmatrix}1 & -1\end{bmatrix}\begin{bmatrix}-1\\1\end{bmatrix} - \frac{1}{2}\lambda_3\lambda_4 \begin{bmatrix}1 & -1\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix} \\
&+ \frac{1}{2}\lambda_4\lambda_1 \begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} + \frac{1}{2}\lambda_4\lambda_2 \begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix} - \frac{1}{2}\lambda_4\lambda_3 \begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}1\\-1\end{bmatrix} \\
&- \frac{1}{2}\lambda_1^2 \begin{bmatrix}-1 & 1\end{bmatrix}\begin{bmatrix}-1\\1\end{bmatrix} - \frac{1}{2}\lambda_2^2 \begin{bmatrix}-1 & -1\end{bmatrix}\begin{bmatrix}-1\\-1\end{bmatrix} - \frac{1}{2}\lambda_3^2 \begin{bmatrix}1 & -1\end{bmatrix}\begin{bmatrix}1\\-1\end{bmatrix} - \frac{1}{2}\lambda_4^2 \begin{bmatrix}1 & 1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} \\
&= \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 \\
&- \frac{1}{2}\lambda_1\lambda_2(1-1) + \frac{1}{2}\lambda_1\lambda_3(-1-1) + \frac{1}{2}\lambda_1\lambda_4(-1+1) \\
&- \frac{1}{2}\lambda_2\lambda_1(1-1) + \frac{1}{2}\lambda_2\lambda_3(1-1) + \frac{1}{2}\lambda_2\lambda_4(-1-1) \\
&+ \frac{1}{2}\lambda_3\lambda_1(-1-1) + \frac{1}{2}\lambda_3\lambda_2(1-1) - \frac{1}{2}\lambda_3\lambda_4(1-1) \\
&+ \frac{1}{2}\lambda_4\lambda_1(1-1) + \frac{1}{2}\lambda_4\lambda_2(-1-1) - \frac{1}{2}\lambda_4\lambda_3(1-1) \\
&- \frac{1}{2}\lambda_1^2(1+1) - \frac{1}{2}\lambda_2^2(1+1) - \frac{1}{2}\lambda_3^2(1+1) - \frac{1}{2}\lambda_4^2(1+1) \\
&= \lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 - \lambda_1^2 - \lambda_2^2 - \lambda_3^2 - \lambda_4^2 - 2\lambda_1\lambda_3 - 2\lambda_2\lambda_4
\end{aligned}$$

Using the partial derivative for every λ_i it is:

$$\frac{\partial J_i^*(\lambda)}{\partial \lambda_1} = 1 - 2\lambda_1 - 2\lambda_3 = 0 \Rightarrow$$

$$\lambda_1 + \lambda_3 = \frac{1}{2} \tag{6.1}$$

Similarly:

$$\lambda_2 + \lambda_4 = \frac{1}{2} \tag{6.3}$$

Moreover, using $\sum_{i=1}^{N} \lambda_i y_i = 0$ we have:

$$\lambda_1 + \lambda_2 = \lambda_3 + \lambda_4 \tag{6.3}$$

Using $\boldsymbol{\theta} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i$ and (6.1), (6.3) and (6.3) we finally get that:

$$\boldsymbol{\theta} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

Finally using the 4th Karush-Kuhn-Tucker condition from the lectures for λ_1 it is:

$$\begin{aligned} \lambda_1[y_1(\boldsymbol{\theta}^T \boldsymbol{x}_i + \theta_0) - 1] = 0 \Rightarrow \\ \theta_0 = 0 \end{aligned} \tag{1}$$

This confirm our original hypothesis, since the line separating the two classes will be

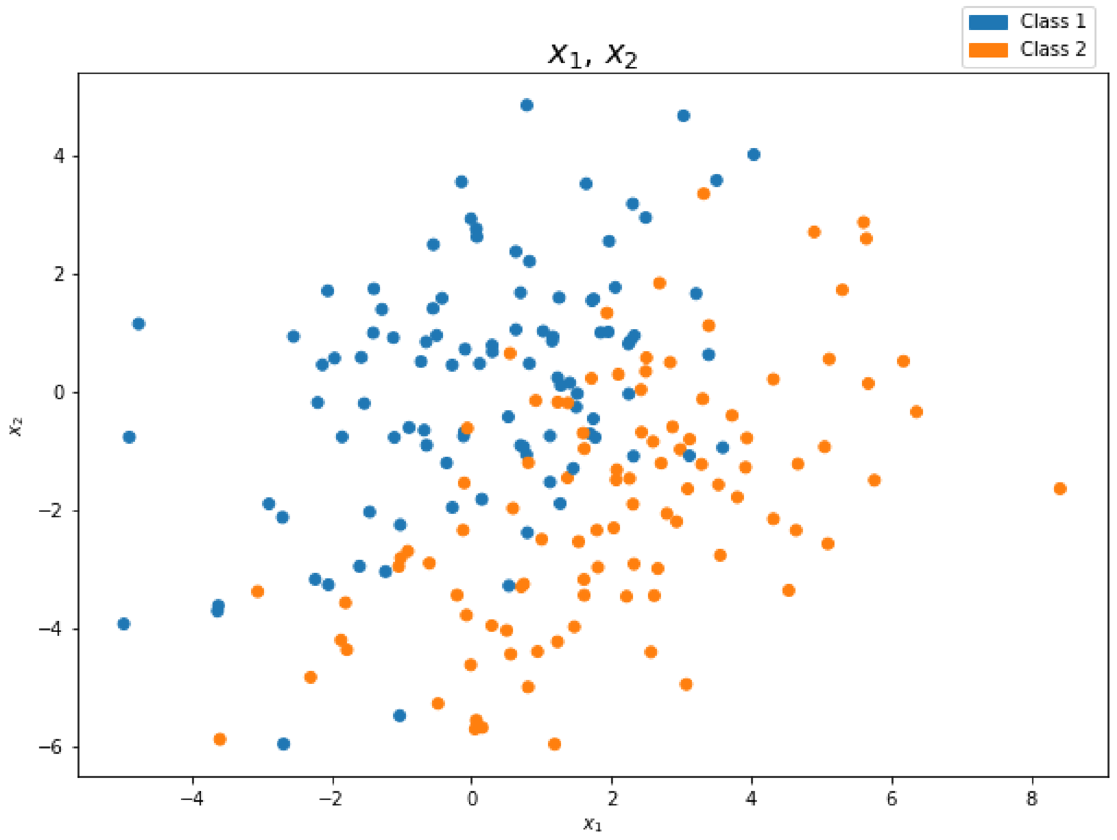
$$\begin{aligned} g(\boldsymbol{x}) = \boldsymbol{\theta}^T \boldsymbol{x} + \theta_0 = 0 \Rightarrow \\ x_1 = 0 \end{aligned}$$

Excercise 7

[1]:
```
# Imports
import scipy.io as sio
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import pandas as pd
from scipy.stats import norm

from sklearn.neighbors import KNeighborsClassifier
from sklearn.mixture import GaussianMixture
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

from sklearn.metrics import accuracy_score, fbeta_score, make_scorer,␣
 ↪roc_auc_score
```

[2]:
```
Dataset = sio.loadmat('HW8.mat')
train_x = Dataset['train_x']
train_y= Dataset['train_y']

test_x = Dataset['test_x']
test_y = Dataset['test_y']

df = pd.DataFrame(np.hstack((train_x, train_y)), columns = ['X1', 'X2', 'y'])
y_1 = df[['X1', 'X2']].loc[ df.y == 1].to_numpy()
y_2 = df[['X1', 'X2']].loc[ df.y == 2].to_numpy()

df_test = pd.DataFrame(np.hstack((test_x, test_y)), columns = ['X1', 'X2', 'y'])
y_1_test = df_test[['X1', 'X2']].loc[ df_test.y == 1].to_numpy()
y_2_test = df_test[['X1', 'X2']].loc[ df_test.y == 2].to_numpy()
```

[3]:
```
# define color pattern
color = pd.Series(train_y.flatten()).apply(lambda x: 'C0' if x == 1 else 'C1')
# Create patches for the legend
patches = [ mpatches.Patch(color = 'C0', label = 'Class 1'), mpatches.
 ↪Patch(color = 'C1', label = 'Class 2') ]



#plot X data
fig = plt.figure(figsize=(10,7))
ax = fig.add_subplot(111)
ax.scatter(train_x[:,0], train_x[:,1], c=color)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_title('$x_1$, $x_2$', fontsize=18)
_ = ax.legend(handles = patches, \
            loc = 'upper left', bbox_to_anchor=(0.85, 0, 0, 1.11))
```



7.a. Adopt the Bayes classifier.

7.a.i. Use the training sete to estimate $P(\omega_1)$ etc…

- the $p(x|\omega)$ is calculated in ii

[4]:
```
# Parametric approach
gm_1 = GaussianMixture(n_components=1, random_state=0).fit(y_1)
mean_1 = gm_1.means_[0]
cov_1 = gm_1.covariances_[0]
print(mean_1, end='\n'*2)
print(cov_1, end='\n'*2)

# Parametric approach
gm_1 = GaussianMixture(n_components=1, random_state=0).fit(y_2)
mean_2 = gm_1.means_[0]
cov_2 = gm_1.covariances_[0]
```

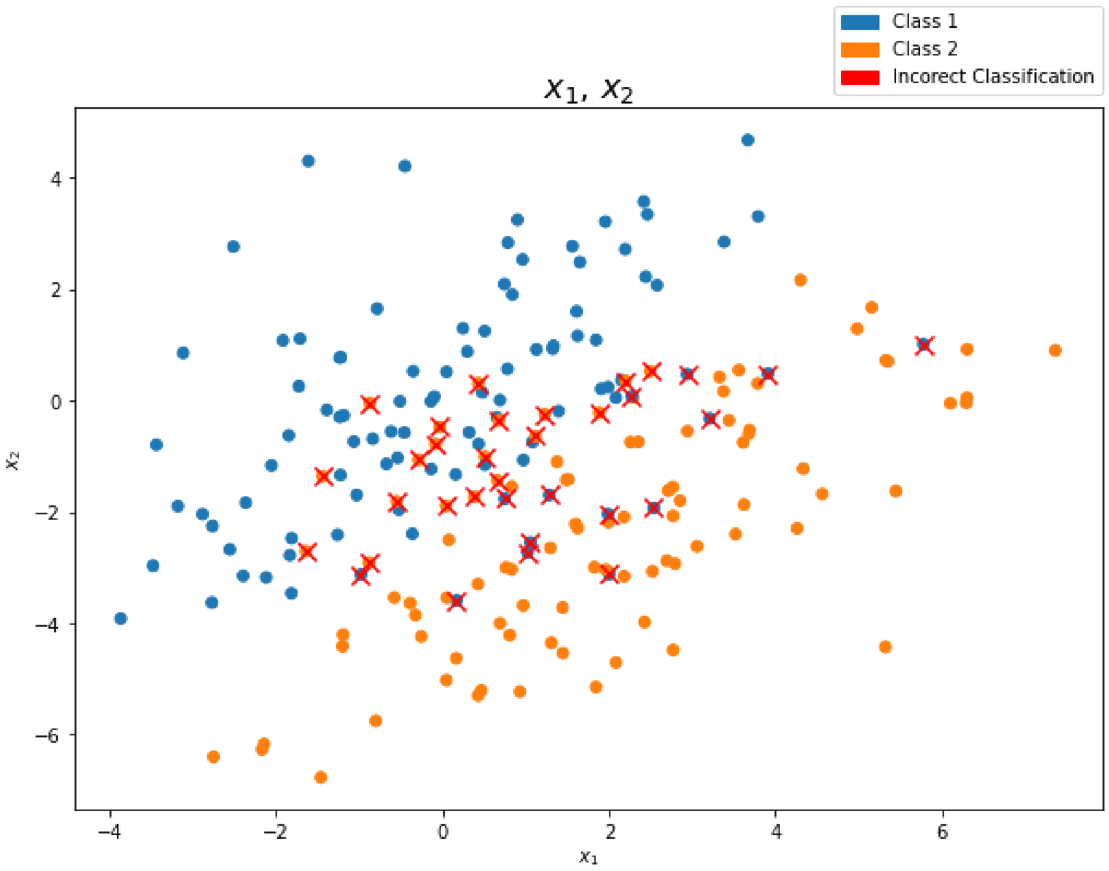
```
print(mean_2, end='\n'*2)
print(cov_2, end='\n'*2)
```

```
[0.14549472 0.11840199]

[[3.60099744 1.72386737]
 [1.72386737 4.17836281]]

[ 2.07024339 -1.89136529]

[[4.67059811 2.5746834 ]
 [2.5746834  4.33386385]]
```

7.a.ii. Classify the $x_i s$ of the test set

```
[5]: p1 = np.count_nonzero(train_y == 1)/len(train_y)
p2 = 1 - p1

Btest_y = []
for x in test_x:
    cov_norm_1, cov_norm_2 = np.linalg.norm(cov_1)**1/2, np.linalg.
 ↪norm(cov_2)**1/2
    cov_inv_1, cov_inv_2 = np.linalg.inv(cov_1), np.linalg.inv(cov_2)
    x_mu_1, x_mu_2 = x - mean_1, x - mean_2

    p_x_1 = 1/ ( (2*np.pi) * cov_norm_1 ) * np.exp(-0.5 * (x_mu_1).
 ↪dot(cov_inv_1).dot((x_mu_1.T)))
    p_x_2 = 1/ ( (2*np.pi) * cov_norm_2 ) * np.exp(-0.5 * (x_mu_2).
 ↪dot(cov_inv_2).dot((x_mu_2.T)))

    P1_x = p1 * p_x_1 / (p1 * p_x_1 + p2 *p_x_2)
    P2_x = p2 * p_x_2 / (p1 * p_x_1 + p2 *p_x_2)

    if P1_x > P2_x:
        Btest_y.append(1)
    else:
        Btest_y.append(2)

Btest_y = np.array(Btest_y)
df_result = pd.DataFrame(np.hstack((test_x, (Btest_y.reshape(-1,1) - test_y))),␣
 ↪columns = ['X1', 'X2', 'y'])
df_result_wrong = df_result[['X1', 'X2']].loc[ df_result.y != 0].to_numpy()
```

```
[6]: def plot_results(test_y, test_x, df_result_wrong):
    # define color pattern
    color = pd.Series( test_y.flatten()).apply(lambda x: 'C0' if x == 1 else␣
 ↪'C1')
    # Create patches for the legend
    patches = [ mpatches.Patch(color = 'C0', label = 'Class 1'), mpatches.
 ↪Patch(color = 'C1', label = 'Class 2'),
                mpatches.Patch(color = 'r', label = 'Incorect Classification')
                ]


    #plot X data
    fig = plt.figure(figsize=(10,7))
    ax = fig.add_subplot(111)
    ax.scatter(test_x[:,0], test_x[:,1], c=color)
    ax.scatter(df_result_wrong[:,0], df_result_wrong[:,1], c='r', marker = 'x',␣
 ↪s=100)
    ax.set_xlabel('$x_1$')
    ax.set_ylabel('$x_2$')
    ax.set_title('$x_1$, $x_2$', fontsize=18)
    _ = ax.legend(handles = patches, \
                loc = 'upper left', bbox_to_anchor=(0.73, 0, 0, 1.16))

plot_results(test_y, test_x, df_result_wrong)
```



7.a.iii. Estimate the error classification probability

[7]:
```
print(f"The ratio of incorectly classified points is:\t {len(df_result_wrong)/
 ↪len(df_result)}")
print(f"The error classification probability is:\t {1- len(df_result_wrong)/
 ↪len(df_result)}")
```

```
The ratio of incorectly classified points is:    0.165
The error classification probability is:         0.835
```

7.b. Adopt the naïve Bayes classifier

7.b.i. Use the training set to estimate $P(\omega_1)$ etc…

- the $p(x|\omega)$ is calculated in ii

[8]:
```
# Clalculate means and variances

## Class 1
mean_11 = y_1[:,0].mean()
```

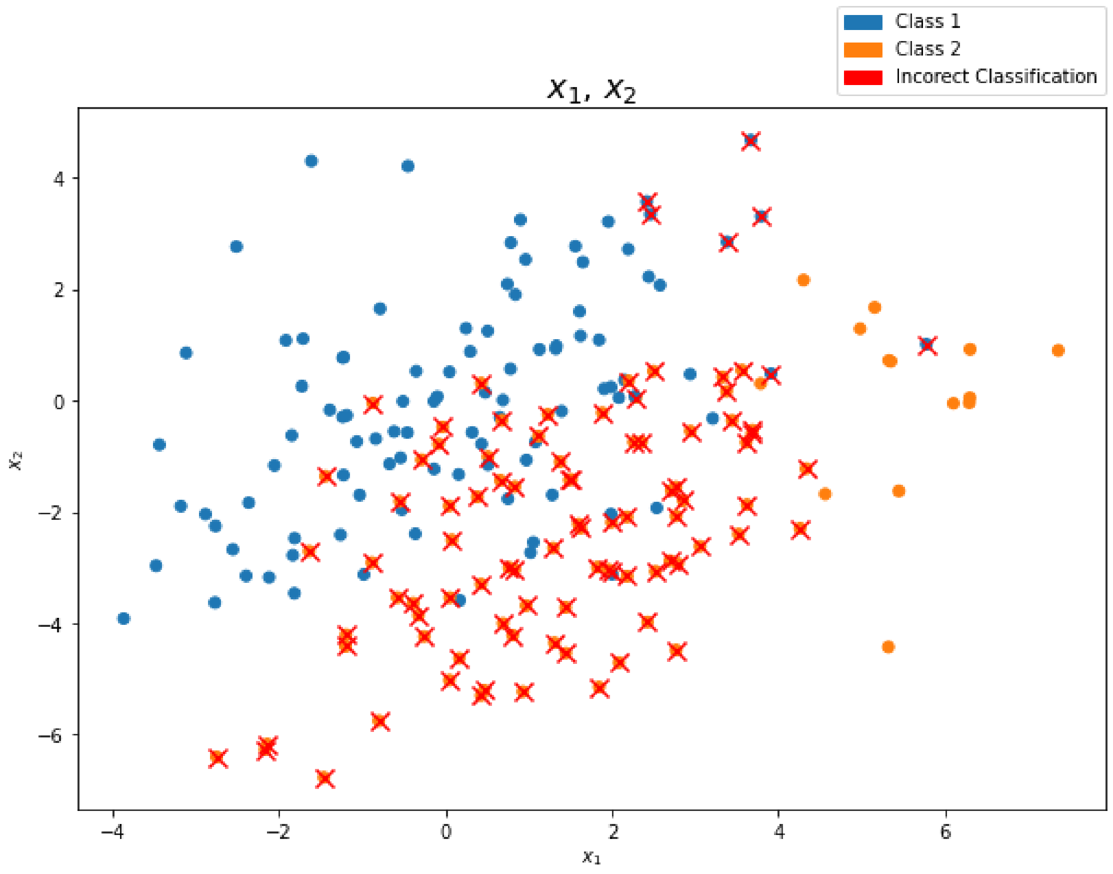
```
mean_12 = y_1[:,1].mean()
s11 = np.power((y_1[:,0]-mean_11), 2).mean()
s12 = np.power((y_1[:,1]-mean_12), 2).mean()
print(mean_11, mean_12)
print(s11, s12, end='\n'*2)

## Class 2
mean_21 = y_2[:,0].mean()
mean_22 = y_2[:,1].mean()
s21 = np.power((y_2[:,0]-mean_21), 2).mean()
s22 = np.power((y_2[:,1]-mean_22), 2).mean()

print(mean_21, mean_22)
print(s21, s22, end='\n'*2)
```

```
0.1454947206275725 0.11840199475343569
3.600996439417043 4.17836180522307

2.0702433928758794 -1.8913652876020373
4.670597106503913 4.333862851556329
```

- knowing that $P(\omega_1) = P(\omega_2) = 0.5$ we can just calculate $g_i(x)$ as $\hat{p}(x|\omega_i)$

[9]:
```
NBtest_y = []
for x in test_x:
    g_1 = norm.pdf(x[0], loc=mean_11, scale=s11) * norm.pdf(x[1], loc=mean_12,␣
 ↪scale=s12)
    g_2 = norm.pdf(x[0], loc=mean_21, scale=s21) * norm.pdf(x[1], loc=mean_21,␣
 ↪scale=s21)

    if g_1 > g_2:
        NBtest_y.append(1)
    else:
        NBtest_y.append(2)
NBtest_y = np.array(NBtest_y)
df_result = pd.DataFrame(np.hstack((test_x, (NBtest_y.reshape(-1,1) - test_y))),␣
 ↪columns = ['X1', 'X2', 'y'])
df_result_wrong = df_result[['X1', 'X2']].loc[ df_result.y != 0].to_numpy()
```

[10]:
```
plot_results(test_y, test_x, df_result_wrong)
```



7.b.iii. Estimate the error classification probability

```
[11]: print(f"The ratio of incorectly classified points is:\t {len(df_result_wrong)/
       ↪len(df_result)}")
      print(f"The error classification probability is:\t {1- len(df_result_wrong)/
       ↪len(df_result):.3f}")
```

```
The ratio of incorectly classified points is:    0.465
The error classification probability is:         0.535
```

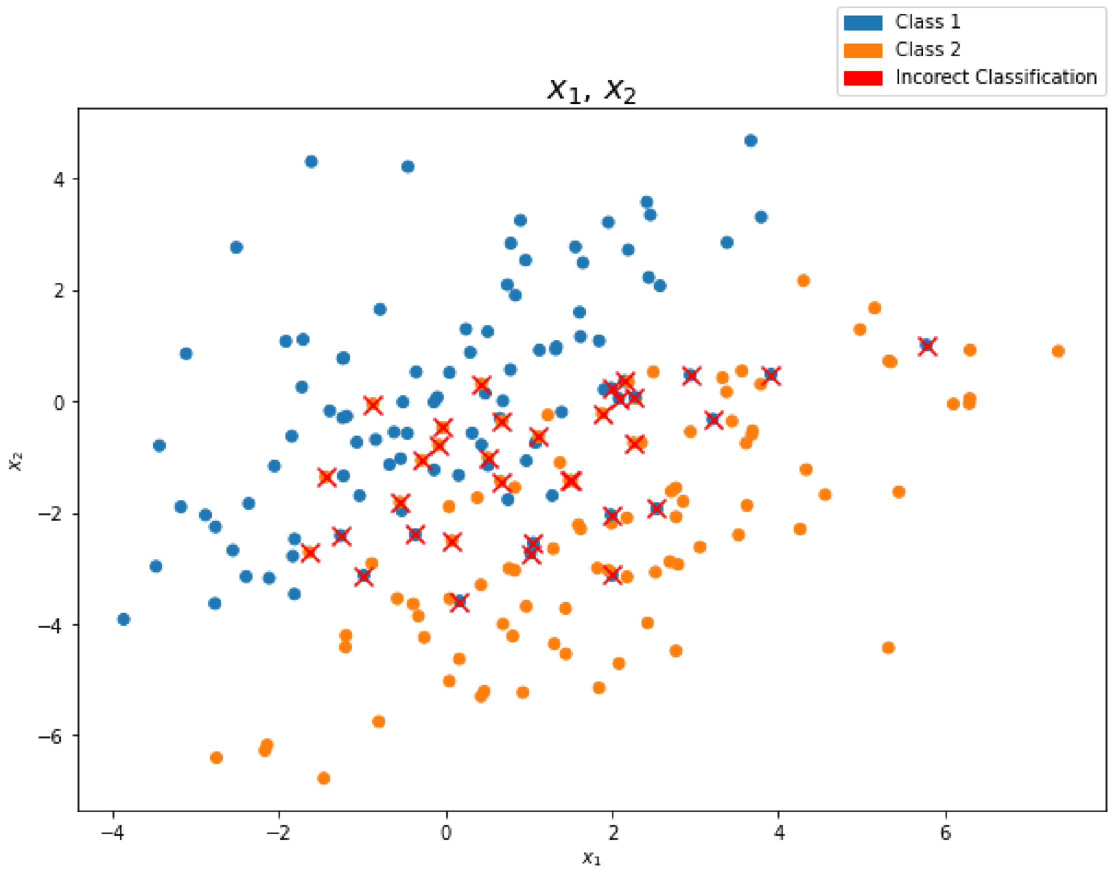
7.c. Adopt the k-nearest neighbor classifier

For $k = 5$ and estimate the classification error probability.

```
[12]: # We will use the default python library
      neigh = KNeighborsClassifier(n_neighbors=5)
      neigh.fit(train_x, train_y.ravel())
      knntest_y = np.array(neigh.predict(test_x), dtype=int)
```

```
df_result = pd.DataFrame(np.hstack((test_x, (knntest_y.reshape(-1,1) -␣
 ↪test_y))), columns = ['X1', 'X2', 'y'])
df_result_wrong = df_result[['X1', 'X2']].loc[ df_result.y != 0].to_numpy()

plot_results(test_y, test_x, df_result_wrong)

print(f"The ratio of incorectly classified points is:\t {len(df_result_wrong)/
 ↪len(df_result)}")
print(f"The error classification probability is:\t {1- len(df_result_wrong)/
 ↪len(df_result)}")
```

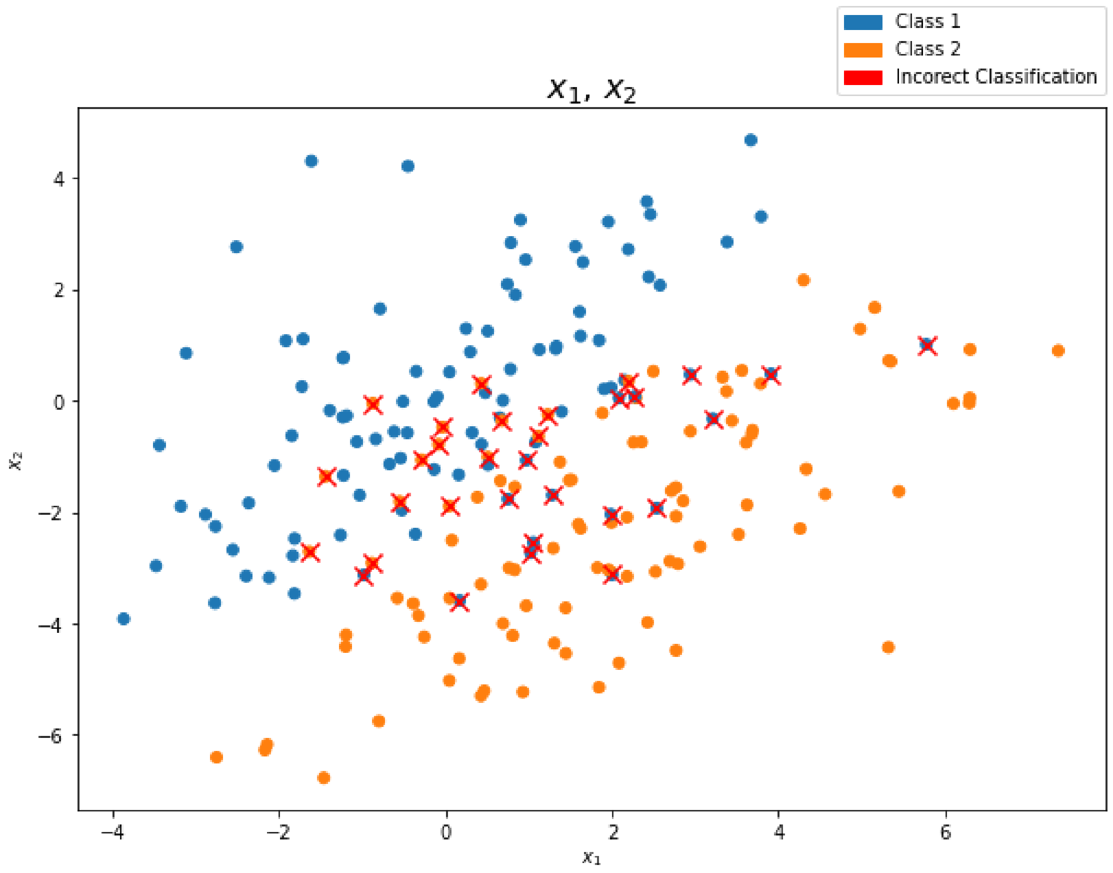
```
The ratio of incorectly classified points is:     0.17
The error classification probability is:          0.83
```



7.d. Adopt the logistic regression classifier

```
[13]: clf = LogisticRegression(random_state=0).fit(train_x, train_y.ravel())
logittest_y = np.array(clf.predict(test_x), dtype=int)

df_result = pd.DataFrame(np.hstack((test_x, (logittest_y.reshape(-1,1) -␣
 ↪test_y))), columns = ['X1', 'X2', 'y'])
df_result_wrong = df_result[['X1', 'X2']].loc[ df_result.y != 0].to_numpy()

print(f"The ratio of incorectly classified points is:\t {len(df_result_wrong)/
 ↪len(df_result)}")
print(f"The error classification probability is:\t {1- len(df_result_wrong)/
 ↪len(df_result)}")


plot_results(test_y, test_x, df_result_wrong)
```

```
The ratio of incorectly classified points is:     0.155
The error classification probability is:          0.845
```



7.e. Depict graphically the training set, using different colors for points from different classes.

- This graph has been presented previously

Report the classification results obtained by the four classifiers and comment on them.

Under what conditions, the first two classifiers would exhibit the same performance?

The results of each classifier are shown in the corresponding graphs, and the misclassified points are highlighted. Looking at the reults we can see that the logistic regression classifier has the best performance, followed by the Bayes classifier and the K-nearest neighbour. Finally we see that the Naive bayes has a sub-par performance.

The first two classifiers would have the same results if the assumption, that the two samples x_1 and x_2 are independent was true. In this case the covariance matrix of the Bayes classifier would be diagonal and the expression of the pdf could be expressed in the same manner for both the Naive and typical Bayes classifier.

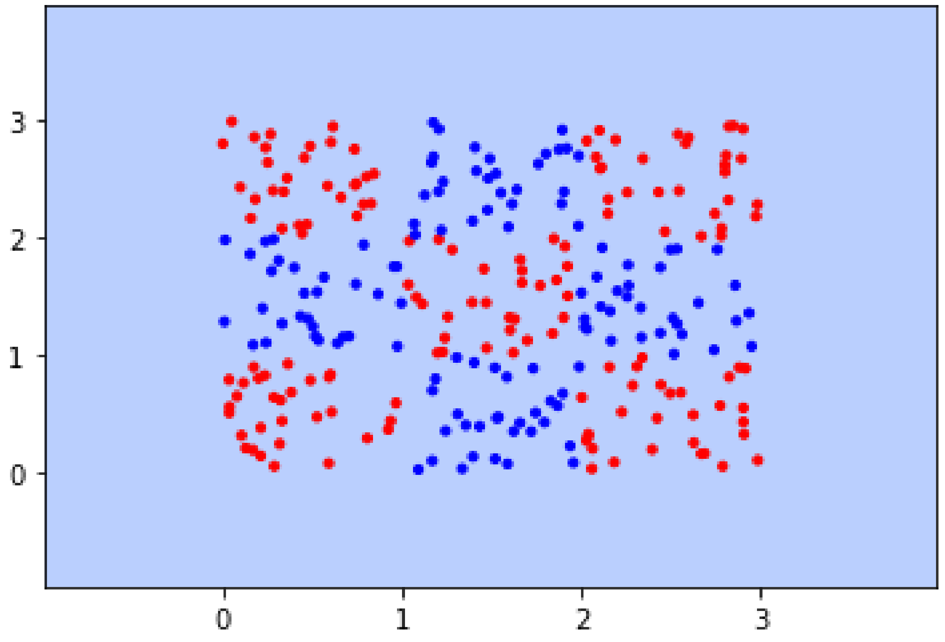
Excercise 8

[14]:
```
Dataset_a = sio.loadmat('HW9a.mat')

train_x_a = Dataset_a['train_X']
train_y_a = Dataset_a['train_y']
test_x_a = Dataset_a['test_X']
test_y_a = Dataset_a['test_y']




from sklearn import svm


def make_meshgrid(x, y, h=.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy


def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out
```

Part 1

Train the SVM classifier using the training set given above and measure its performance using the test set, using the linear kernel (this corresponds to the linear case presented in the lecture).

```
[15]: #select kernel function:{'rbf','poly'}
clf = svm.SVC(kernel='linear',C=10,gamma=1,degree=2)
clf.fit(train_x_a, train_y_a.reshape(270))

X00, X11 = test_x_a[:,0], test_x_a[:,1]
xx, yy = make_meshgrid(X00, X11)
fig, ax= plt.subplots(1, 1)
color= ['red' if l == 1 else 'blue' for l in test_y_a.reshape(270)]
plot_contours(ax, clf, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.6)
ax.scatter(X00, X11,c=color, cmap=plt.cm.coolwarm, s=10, edgecolors='face')
plt.show()

print(f"roc_au Score: {roc_auc_score(test_y_a, clf.predict(test_x_a).
 ↪flatten())*100:.2f}%")
```



```
roc_au Score: 50.00%
```

- We can see that the linear kernel doesn't have good results. All points are classified as 1 (blue). This is expected, since the data separating our points in not linear.

Part 2

RandomizedSearch for the 'rbf' kernel

- We will explore various parameters for the 'rbf' kernel

```
[16]: #select kernel function:{'rbf','poly'}
clf = svm.SVC(random_state=42)
auc = make_scorer(roc_auc_score)

kernel_choice = ['rbf']
C_range = np.logspace(-2, 10, 13)
gamma_range = np.logspace(-9, 3, 13)
degree_range = range(0,10)
param_grid = dict(gamma=gamma_range, C=C_range, degree=degree_range,␣
 ↪kernel=kernel_choice)
rand_search = RandomizedSearchCV(clf, param_distributions = param_grid, n_iter =␣
 ↪1000,  scoring = auc, n_jobs=-1, verbose=1)
rand_search.fit(train_x_a, train_y_a.reshape(270))
rand_search.best_estimator_
```

```
Fitting 5 folds for each of 1000 candidates, totalling 5000 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    2.9s
[Parallel(n_jobs=-1)]: Done 1057 tasks      | elapsed:   20.0s
[Parallel(n_jobs=-1)]: Done 1718 tasks      | elapsed:   46.9s
[Parallel(n_jobs=-1)]: Done 2448 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 3553 tasks      | elapsed:  2.1min
[Parallel(n_jobs=-1)]: Done 5000 out of 5000 | elapsed:  2.7min finished
```

```
[16]: SVC(C=10000000.0, degree=4, gamma=0.1, random_state=42)
```

Explore the values of C

```
[17]: roc = 0
for C in np.logspace(-2, 10, 13):
    clf = svm.SVC(degree=7, gamma=0.1, random_state=42, C=C)
    clf.fit(train_x_a, train_y_a.reshape(270))
    roc_current = roc_auc_score(test_y_a, clf.predict(test_x_a).flatten())*100
    if roc_current > roc:
        best_clf = clf
```

```
[18]: fig, ax= plt.subplots(1, 1)
color= ['red' if l == 1 else 'blue' for l in test_y_a.reshape(270)]
plot_contours(ax, best_clf, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.6)
ax.scatter(X00, X11,c=color, cmap=plt.cm.coolwarm, s=10, edgecolors='face')
plt.show()

print(f"roc_au Score: {roc_auc_score(test_y_a, best_clf.predict(test_x_a).
 ↪flatten())*100:.2f}%")
```

output_34_0.png

```
roc_au Score: 96.08%
```

RandomizedSearch for the 'poly' kernel

- We will explore various parameters for the 'poly' kernel

[19]:
```
#select kernel function:{'rbf','poly'}
clf = svm.SVC(random_state=42)
auc = make_scorer(roc_auc_score)

kernel_choice = ['poly']
C_range = np.logspace(-2, 10, 13)
gamma_range = np.logspace(-9, 3, 13)
degree_range = range(0,3)
param_grid = dict(gamma=gamma_range, C=C_range, degree=degree_range,␣
 ↪kernel=kernel_choice)
rand_search = RandomizedSearchCV(clf, param_distributions = param_grid, n_iter =␣
 ↪5,  scoring = auc, n_jobs=-1, verbose=1)
rand_search.fit(train_x_a, train_y_a.reshape(270))
rand_search.best_estimator_
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 out of  25 | elapsed:    0.0s finished
```

[19]:
```
SVC(C=100000000.0, degree=0, gamma=1000.0, kernel='poly', random_state=42)
```

[20]:
```
fig, ax= plt.subplots(1, 1)
color= ['red' if l == 1 else 'blue' for l in test_y_a.reshape(270)]
plot_contours(ax, rand_search, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.6)
ax.scatter(X00, X11,c=color, cmap=plt.cm.coolwarm, s=10, edgecolors='face')
```

```
plt.show()

print(f"roc_au Score: {roc_auc_score(test_y_a, rand_search.predict(test_x_a).
 ↪flatten())*100:.2f}%")
```

output_37_0.png

```
roc_au Score: 50.00%
```

Commenting

- As shown above the best results are achieved using the 'rbf' kernel, having succesfully separated the majority of the points. On the other hand the 'poly' and 'linear' methods did not fetch satisfying results. While it makes sense for the linear kernel not to fetch satisfying results, we can assume that the reason the 'poly' one did not have good results is due to the complexity. Unfortunately the training time for this kernel is very high, which make it hard to explore the variables (given the limited time we have for each assignment).

```
[ ]:
```