

# Introduction to Machine Learning Assignment 1

Question 1:

(A) Given that : Part 1

The class priors are  $P(L = 0) = 0.65$  and  $P(L = 1) = 0.35$ . The class class-conditional pdfs are  $p(x|L=0) = w_1 g(x|m_0, C_0) + w_2 g(x|m_2, C_2)$  and  $p(x|L=1) = g(x|m_1, C_1)$ , where  $g(x|m, C)$  is a multivariate Gaussian probability density function with mean vector  $m$  and covariance matrix  $C$ . The parameters of the class-conditional Gaussian pdfs are:  $w_1 = w_2 = 1/2$ , and

$$m_0 = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \quad C_0 = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \quad m_2 = \begin{bmatrix} 0 \\ 3 \end{bmatrix} \quad C_2 = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \quad m_1 = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \quad C_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

(2.1)  
(a) consider the ERM of 2-classes.

deciding on the basis of the ratio of the class posteriors, by considering the likelihoods under class and pdfs.

$$\frac{p(x|L=1)}{p(x|L=0)} > \gamma$$

$$\frac{p(x|L=1)}{p(x|L=0)} > \frac{P(L=0)}{P(L=1)} \quad * \quad \frac{101-100}{110-111}$$

$$\frac{g(x|m_1, C_1)}{g(x|m_0, C_0)} > \frac{P(L=0)}{P(L=1)} \quad * \quad \frac{101-100}{110-111}$$

100 = True negative detection  
101 = Risk of a false negative detection  
110 = Risk of a false positive detection  
111 = Risk of a true positive detection

of 0-1 loss threshold = 1, decide based on the ratio of class posteriors by

$$\frac{p(L=1|x)}{p(L=0|x)} > \frac{110-100}{101-111} \quad D(x)=1$$

$$\frac{p(L=1|x)}{p(L=0|x)} < \frac{110-100}{101-111} \quad D(x)=0$$

On case of a 0-1 loss the function simplifies to:

$$\frac{g(x|m_1, C_1)}{g(x|m_0, C_0)} > \frac{P(L=0)}{P(L=1)}$$

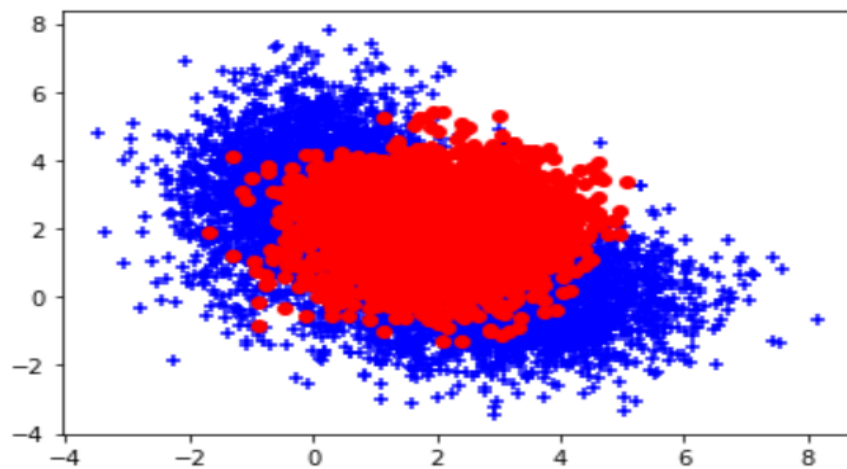
$$\frac{g(x|m_1, C_1)}{g(x|m_0, C_0)} > \frac{0.65}{0.35}$$

(D=1)  $\frac{g(x|m_1, C_1)}{g(x|m_0, C_0)} > 1.857 \quad (D=0)$

here the approximate threshold gamma to use the minimum loss is 1.857.

Part 2)

The given output is the true labels of Class 0 and Class 1. Here the Class 0 is the mixture of the two weighted means and weighted covariances



(2) The class priors and conditional pdfs are below:

class 0:

$$P(L=0) = 0.65$$

$$m_{01} = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \quad m_{02} = \begin{bmatrix} 0 \\ 3 \end{bmatrix}$$

weight is divided equally  
 $w_1 = w_2 = 0.5$

$$P(X|L=0) = w_1 g(x|m_{01}, C_{01}) + w_2 g(x|m_{02}, C_{02})$$

$$C_{01} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \quad C_{02} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

$$P(X|L=0) = 0.5 \times \begin{bmatrix} 3 & 2 \\ 0 & 0 \end{bmatrix} + 0.5 \times \begin{bmatrix} 0 & 1 \\ 3 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1.5 & 2 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 \\ 1.5 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 0 \\ 0 & 1.5 \end{bmatrix} + \begin{bmatrix} 1.5 & 0 \\ 0 & 3 \end{bmatrix}$$

$$P(X|L=0) = \begin{bmatrix} 4.5 & 0 \\ 0 & 4.5 \end{bmatrix}$$

class 1:

$$P(X|L=1) = 0.35$$

$$m_{11} = \begin{bmatrix} 3 \\ 2 \end{bmatrix} \quad C_{11} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The graph <sup>1</sup> shows the distribution of true class 0 and class 1 labels:

Part 3

(3) Expected Risk (Loss / cost) minimization:

For minimum probability error:

$$E_x[\text{Risk}] = \int_{-\infty}^{\infty} \text{Risk}(\text{Decide}/x) P_x(x) dx$$

Decide is the decision matrix.

here ~~loss~~ <sup>Risk</sup> matrix is taken out as

$$\text{Risk}(D(x) = d/x) = \sum_{L=1}^L d_L P(L=L/x)$$

where

$\sum$  produces the average loss of deciding a given  $x$ , considering the class posterior probabilities for the given  $x$ .

$d_L$  = loss we would incur by deciding the given sample from class

$P(L=L/x)$  = Probability of class label using  $L$  given by  $x$ .

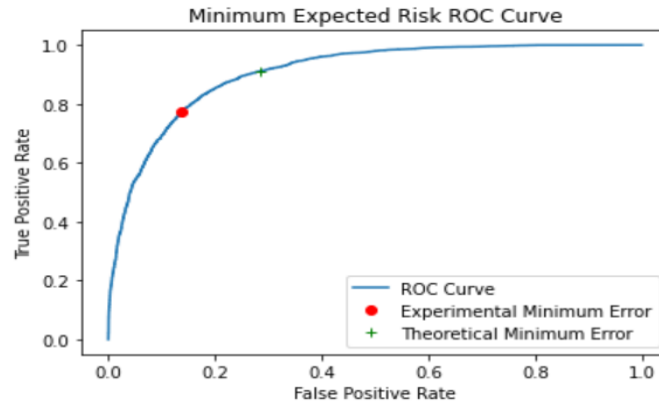
$$D(x) = \text{argmin Risk}(D(x) = d/x)$$

hence total theoretical minimum probability of error is obtained from

$$P_e = 1 - P(D=0 | L=0) P(L=0) - P(D=1 | L=1) P(L=1) =$$

Plot of the ROC curve for the minimum expected risk classifier applied on the 10,000 samples with gamma varied from 0 to 1:

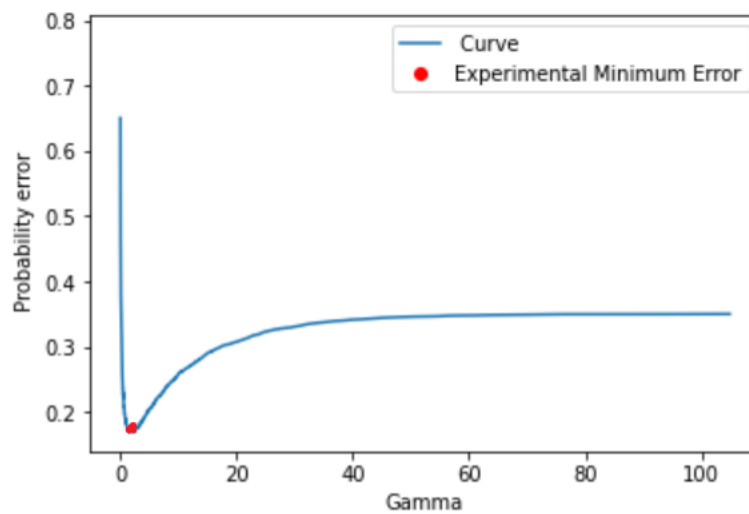




Experimental Gamma  
1.596296836403239  
Experimental minerror  
0.16777374269360185  
Theoretical Error  
0.21545820287024148

The experimental gamma to provide the minimum error is 1.596 and the theoretical gamma value . And the experimental minimum error was 0.167. This number is close to the theoretical obtained gamma value which corresponded to a probability of error of 0.2154 and we would expect the experimental gamma to approach the theoretical gamma as  $n$  approaches infinity.

Also as shown below The location of the minimum error is marked. Additionally as the gamma parameter approached its limits at 0 and  $+\infty$  the probability of error asymptotes to the priors for the two distributions. That is when the gamma is set to its minimum value all of the data points will be classified as class 1 so the overall error will be the proportion of data in class 0 which is equivalent to its prior. Similarly when gamma is at  $+\infty$  all of the data points will be classified as class 0 and so all of the class 1 data will be misclassified and the probability of error is equivalent to the prior for class 1.



## PART B : Fisher LDA ROC CURVE

This minimizes the overlap between the two datasets

aiding the classifications process. In this implementation the analysis was performed using the sample mean and covariances of the two sets of data.

Fisher LDA curve and ROC.

LDA is a method to project data from 2 categories to a 1-D space where they are maximally separated. Fisher LDA measures separability of the projection using  $\frac{(u_1 - u_2)^2}{(\sigma_1^2 + \sigma_2^2)}$  where  $u$  is mean and  $\sigma$  is covariances.

calculated weighted mean of class 1 as it is a mixture of two means gives us

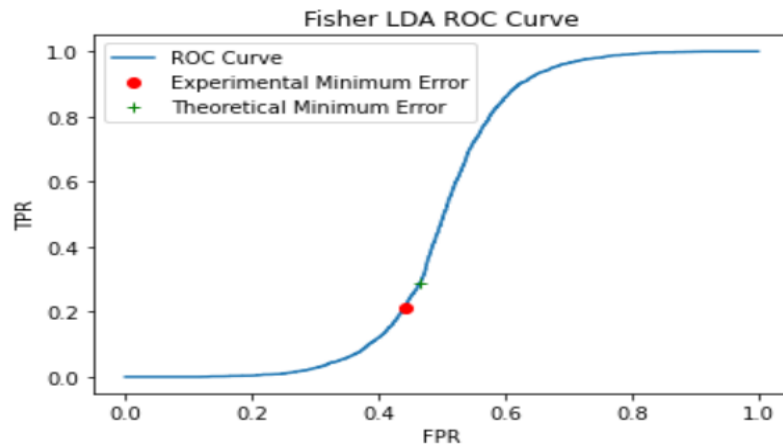
$$u_0 = w_1 u_1 + w_2 u_2$$
$$w_1 = w_2 = 0.5$$

$u_1$  is mean of  $m_01$   
 $u_2$  is mean  $m_02$ .

covariance  $\Sigma_0$  is.

$$\begin{aligned}\Sigma_0 &= \int (x - u_0)(x - u_0)^T \text{pdf}(x) dx \\ &= w_1 \int (x - u_1)(x - u_1)^T g(x | u_1, \Sigma_1) dx \\ &\quad + w_2 \int (x - u_2)(x - u_2)^T g(x | u_2, \Sigma_2) dx \\ &= w_1 (\Sigma_1 + u_1 u_1^T - u_0 u_1^T - u_1 u_0^T + u_0 u_0^T) \\ &\quad + w_2 (\Sigma_2 + u_2 u_2^T - u_0 u_2^T - u_2 u_0^T + u_0 u_0^T)\end{aligned}$$
$$S_B = (u_1 - u_0)(u_1 - u_0)^T$$
$$S_W = \Sigma_0 + \Sigma_1$$

sign vector is given by  $(S_B, S_W)$



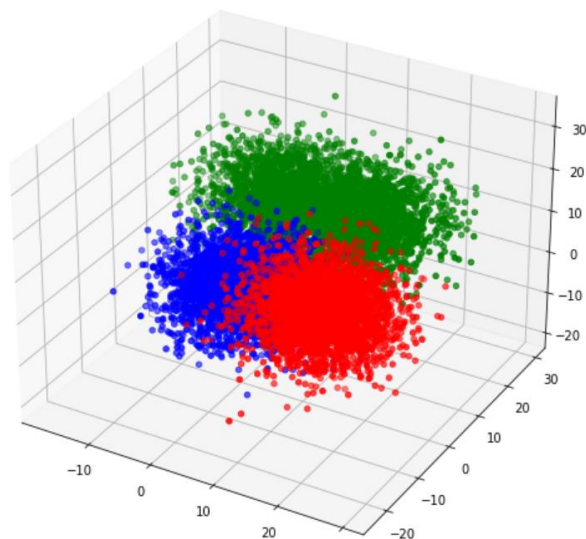
Experimental Gamma  
0.7851069764811559  
Experimental minerror  
0.3501000461751577  
Theoretical Error  
0.4897722025550254

As observed from the tau reduces the minimum probability is also marked on the graph. ROC curve generated after implementing the Fisher LDA from above, applying it to the 10,000 generated samples, and varying the threshold  $\tau$  from  $-\infty$  to  $+\infty$ . Based on the 10,000 samples generated, the minimum probability of error was calculated to be  $p_e = 0.35$  at the threshold value of  $\tau = 0.785$ . This error and threshold value were calculated by finding the minimum error as the value of  $\tau$  was changed from  $-\infty$  to  $+\infty$ .

Question no 2 :

PART A

a)



The above figure shows the class 1, 2 and 3. The third class is the combination of equally weighted means. Given A 3-dimensional random vector  $X$  takes values from a mixture of four Gaussians. One of these Gaussians represent the class-conditional pdf for class 1, and another Gaussian represents the class-conditional pdf for class 2. Class 3 data originates from a mixture of the remaining 2 Gaussian components with equal weights. Here the value of mean and covariance are assumed to set such that the distances between means of pairs of Gaussians to twice the average standard deviation of the Gaussian components, so that there is some significant overlap between class-conditional pdfs. class priors are set to 0.3, 0.3, 0.4.

Q.2  
(a)  
part

Given is a 3-D random vector  $X$  from a mixture of four Gaussians

class 0  
 $P(L=0) = 0.3 =$

mean 0 =  $\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$

covariance 0 =  $\begin{bmatrix} 27 & 0 & 0 \\ 0 & 27 & 0 \\ 0 & 0 & 27 \end{bmatrix}$

$P(L=1) = 0.3$   
 mean 1 =  $\begin{bmatrix} 13 \\ 0 \\ 0 \end{bmatrix}$

covariance 1 =  $\begin{bmatrix} 27 & 0 & 0 \\ 0 & 27 & 0 \\ 0 & 0 & 27 \end{bmatrix}$

$P(L=2) = 0.4$   
 mean 2 =  $\begin{bmatrix} 13 \\ 13 \\ 13 \end{bmatrix}$

covariances =  $\begin{bmatrix} 27 & 0 & 0 \\ 0 & 27 & 0 \\ 0 & 0 & 27 \end{bmatrix}$

Mean 3  $\begin{bmatrix} 0 \\ 13 \\ 13 \end{bmatrix}$  Covariance  $\begin{bmatrix} 27 & 0 & 0 \\ 0 & 27 & 0 \\ 0 & 0 & 27 \end{bmatrix}$

$\Rightarrow 0.5 \text{ Mean 2} + 0.5 \text{ Mean 3} = P(L=2)$

$0.5 \text{ covariance 2} + 0.5 \text{ covariance 3}$   
 $= \text{Weighted covariance}$   
 for  $P(L=2) \rightarrow 2^{\text{nd}} \text{ class.}$

$= 0.5 \begin{bmatrix} 13 \\ 13 \\ 13 \end{bmatrix} + 0.5 \begin{bmatrix} 0 \\ 13 \\ 13 \end{bmatrix} = \text{weighted mean}$

$= 0.5 \begin{bmatrix} 27 & 0 & 0 \\ 0 & 27 & 0 \\ 0 & 0 & 27 \end{bmatrix} \times 2 = \text{Weighted covariance}$

Lambda matrix is given as.

$\Lambda = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$

which is a 0-1 loss matrix, which indicates our design choice that all incorrect decisions are equally bad and they are worse than all correct decisions which are equally good.

Also mean and covariances are set such that

$\sigma^2 + \sigma^2 \sim 2 (u_1 - u_2)$   
 covariance  $\sim 2 (\text{distance between means})$

(b) To determine the decision to be made about each point we used

$$\alpha_* = \operatorname{argmin}_{\alpha_i} R(\alpha_i | x)$$

$$R(\alpha_i | x) = \sum_{j=1}^C (\alpha_i | w_j) p(w_j | x)$$

entries for the confusion matrix can be taken out as

$P(D=i | L=j)$  for  $i, j \in \{1, 2, 3\}$

$D_{\text{map}} = \operatorname{argmax}_d P(L=d | x)$

$= \operatorname{argmax}_d P(x | L=d)$

If the class priors are equal, then MAP classifier simplifies to ML classifier

ML = Maximum Likelihood.

The decision rule to be made about each point for achieving minimum probability error is given as.

$D(x) = \operatorname{argmin}_{d \in \{1, \dots, C\}} R(D=d | x)$

$= \operatorname{argmin}_{d \in \{1, \dots, C\}} 1 - P(L=d | x)$

here  $R(D=d | x) = \sum_{j=1}^C (\alpha_i | w_j) p(w_j | x)$



The confusion matrix obtained here by using the decision rule above is

the confusion matrix as obtained

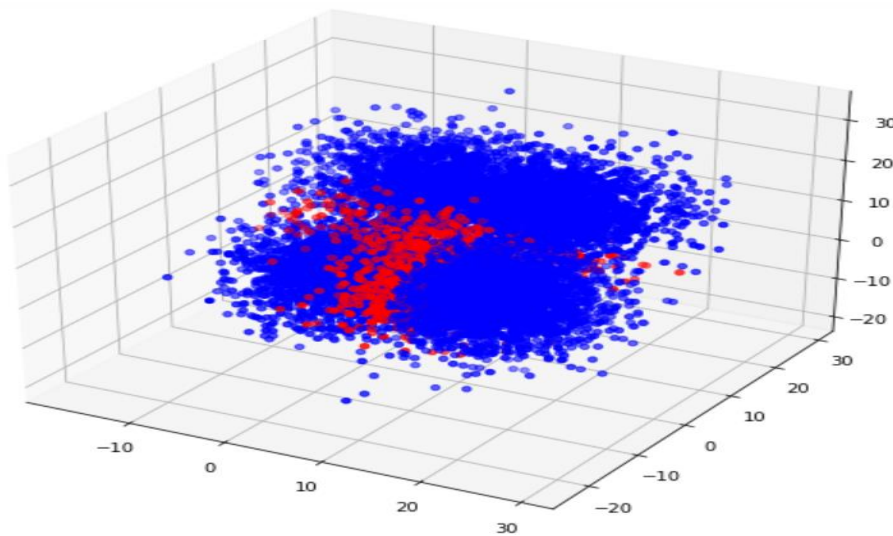
0.17	0.09	0.39
1.009	1.11	1.26
0.15	0.023	0.89

The total expected risk of 0.7134

Mean is taken as 13. Covariance is taken as 27 .

Based on this decision boundary, the minimum probability of error was calculated by counting the number of missed detections and false alarms and multiplying their probabilities by the class priors.

(c)

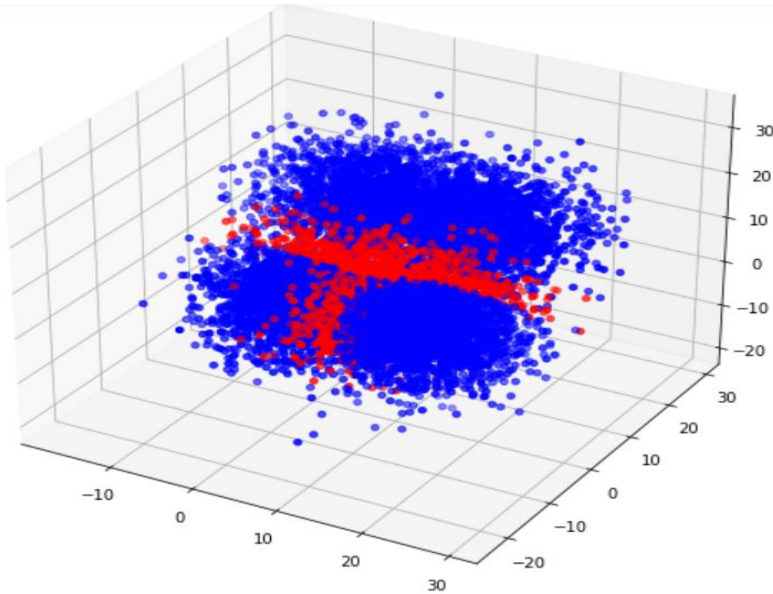


Here the class 3 will try to avoid making mistakes and is having the larger value . Decision is in the favor of class 3. Here the data distributions that are overlapping having new lambda matrix will have a lambda of 1. Interestingly, its is observed that probability of class here is showing less which clearly illustrates that it is trying to avoid mistakes. Probability of Error lies between 5 to 15 percentage only .

## PART B

Case 1 : lamda matrix is 10 upon the class 3 only

$$\Lambda_{10} = \begin{bmatrix} 0 & 1 & 10 \\ 1 & 0 & 10 \\ 1 & 1 & 0 \end{bmatrix}$$

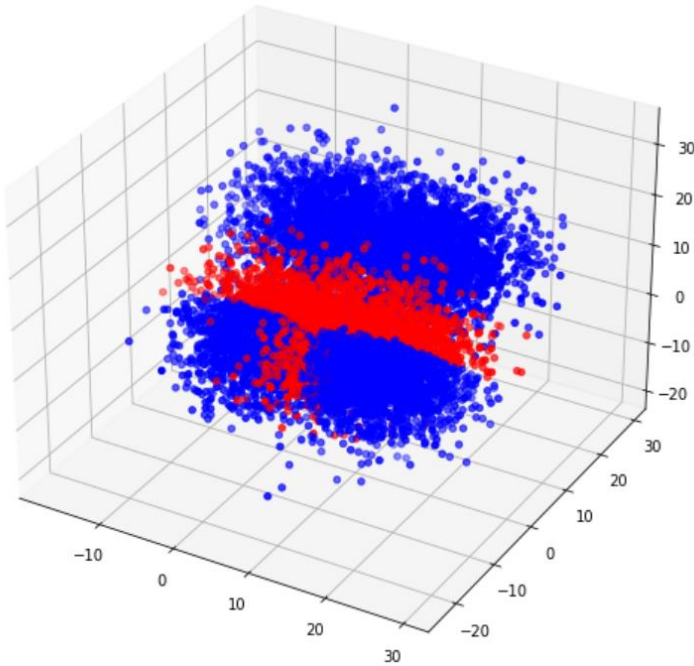


The minimum expected risk is : 0.732

It is observed that as we increase the lambda range the minimum expected risk starts to increase .

Case 2 : lamda matrix is 100 upon class 3 only

$$\Lambda_{100} = \begin{bmatrix} 0 & 1 & 100 \\ 1 & 0 & 100 \\ 1 & 1 & 0 \end{bmatrix}$$



The minimum expected risk is given as : 1.5199

It is observed that as we increase the lambda range the minimum expected risk starts to increase.

As the size of the matrix lambda starts to increase the value of the error also starts to increase here for the class 3 .

Code in python

Question number 1:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
import random
samples = 10000
p1 = 0.65
p2 = 0.35
samples_0 = 0
samples_1 = 0
mean01 = np.array([3, 0])
mean02 = np.array([0, 3])
cov01 = np.array([[2, 0],[0, 1]])
cov02 = np.array([[1, 0],[0, 2]])
mean1 = np.array([2,2])
cov1 = np.array([[1, 0],[0, 1]])
for i in range(0, samples):
    if random.random()<p1:
```

```

        samples_0 = samples_0 + 1
    else:
        samples_1 = samples_1 + 1
samples_0, samples_1
class01 = []
class02 = []
count01 = 0
count02 = 0
for i in range(0, samples_0):
    class0_random = np.random.uniform(0,1)
    if(class0_random>0.5):
        count01 = count01 + 1
        #label01 = np.random.multivariate_normal(mu01, cov01, 1)
    else:
        count02 = count02 + 1
        #label02 = np.random.multivariate_normal(mu02, cov02, 1)
class1 = np.random.multivariate_normal(mean1, cov1, samples_1)
class01 = np.random.multivariate_normal(mean01, cov01, count01)
class02 = np.random.multivariate_normal(mu02, cov02, count02)
len(class01), len(class02), len(class1)
plt.scatter(class01[:,0], class01[:,1], marker='+', c='blue')
plt.scatter(class02[:,0], class02[:,1], marker='+', c='blue')
plt.scatter(class1[:,0], class1[:,1], marker='o', c='red')
N01discriminant = []
N02discriminant = []
N1discriminant = []
for j in range (count01):
    N01discriminant.append(multivariate_normal.pdf(class01[j], mean1,
cov1)/(0.5*multivariate_normal.pdf(class01[j], mean01, cov01)+0.5*multivariate_normal.pdf(class01[j],
mean02, cov02)))
for j in range (count02):
    N02discriminant.append(multivariate_normal.pdf(class02[j], mean1,
cov1)/(0.5*multivariate_normal.pdf(class02[j], mean01, cov01)+0.5*multivariate_normal.pdf(class02[j],
mean02, cov02)))
for j in range (samples_1):
    N1discriminant.append(multivariate_normal.pdf(class1[j], mean1,
cov1)/(0.5*multivariate_normal.pdf(class1[j], mean01, cov01)+0.5*multivariate_normal.pdf(class1[j],
mean02, cov02)))
fpr = []
tpr = []
p_error = []
gamma_vals = []
NO_discriminant = N01discriminant + N02discriminant
fulldiscriminant = N01discriminant + N02discriminant + N1discriminant

```



```

#fulldiscriminant.append(0)
len(fulldiscriminant)
for i in sorted(fulldiscriminant):
    fp = len([j for j in NO_discriminant if j >= i ]) / samples_0
    tp = len([j for j in N1discriminant if j >= i ]) / samples_1
    fpr.append(fp)
    tpr.append(tp)
    gamma_vals.append(i)
    p_error.append(fp*p1+(1-tp)*p2)
minimum_error = min(p_error)
minimum_index = 0
for i in range(0,len(p_error)):
    if p_error[i] == minimum_error:
        minimum_index = i
        break
print('Part A')
print ('Experimental Gamma')
print(gamma_vals[minimum_index])
print('Experimental minerror')
print(min(p_error) )
print('Theoretical Error')
theofp = len ([j for j in NO_discriminant if j >= (p2/p1)]) / samples_0
theotp = len ([j for j in N1discriminant if j >= (p2/p1)]) / samples_1
print(theofp*p1+(1-theotp)*p2)
plt.figure(1)
plt.plot(fpr,tpr, label='ROC Curve')
plt.plot(fpr[minimum_index], tpr[minimum_index], 'ro', label='Experimental Minimum Error')
plt.plot(theofp , theotp , 'g+', label = 'Theoretical Minimum Error')
plt.title('Minimum Expected Risk ROC Curve')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend()
plt.show()
plt.figure(2)
plt.plot(gamma_vals,p_error,label='Curve')
plt.ylabel(" Probability error")
plt.xlabel("Gamma")
plt.plot(tpr[minimum_index],fpr[minimum_index], 'ro', label='Experimental Minimum Error')
plt.legend()
plt.show()
class0 = np.concatenate((class01, class02))
class0.shape
class02.shape
class1

```

```

mean0 = 0.5*mean01+0.5*mean02
cov0 = 0.5*(cov01+mu01*mu01.T - mu0*mu01.T - mu01*mu0.T + mu0*mu0.T) +
0.5*(cov02+mu02*mu02.T - mu0*mu02.T - mu02*mu0.T +mu0*mu0.T)
#cov0=np.cov(class0,rowvar=False)
#cov1=np.cov(class1,rowvar=False)
Sb = np.dot((mean0 - mean1), (mean0 - mean1).T)
Sw = cov0 + cov1
w, v = np.linalg.eig(np.linalg.inv(Sw)*Sb)
max_eigen_index = 0
# f i n d i n d e x o f m a x i m u m e i g e n v a l u e
for i in range(0, len(w)):
    if w[i] == max(w):
        max_eigen_index = i
wLDA = v[:, max_eigen_index]
yLDA01 = np.matmul(wLDA, class01.T)
yLDA02 = np.matmul(wLDA, class02.T)
yLDA0 = list(yLDA01)+list(yLDA02)
yLDA1 = np.matmul(wLDA, label1.T)
totalLDA = list(yLDA02)+list(yLDA01)+list(yLDA1)
false_positive = []
true_positive= []
prob_error= []
gamma_values = []
# u s e s a m e l o g i c a s a b o v e t o m a k e d e c i s i o n s
for i in sorted(totalLDA):
    fp = len([j for j in yLDA0 if j >= i])/samples_0
    tp = len([j for j in yLDA1 if j >= i])/samples_1
    false_positive.append(fp)
    true_positive.append(tp)
    gamma_values.append(i)
    prob_error.append(fp*p1 + (1 - tp)* p2 )
print('Experimental Gamma')
print(gamma_values[minimum_index])
print('Experimental minerror')
print(min(prob_error))
print('Theoretical Error')
theofp = len([j for j in yLDA0 if j >= (p2/p1)])/samples_0
theotp = len([j for j in yLDA1 if j >= (p2/p1)])/samples_1
print(theofp*p1 + (1- theofp)*p2)
plt.figure(3)
plt.plot(false_positive, true_positive, label = 'ROC Curve')
plt.plot(false_positive[minimum_index], true_positive[minimum_index], 'ro', label = 'Experimental
Minimum Error')
plt.plot(theofp, theotp, 'g+', label='Theoretical Minimum Error')

```

```

plt.title('Fisher LDA ROC Curve')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.legend()
plt.show()

```

Question number 2 code :

```

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
plt.rcParams['figure.figsize'] = [9,9]
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import multivariate_normal
data = 10000
m = 13
m1 = np.array([0,0,0])
m2 = np.array([m,0,0])
m3 = np.array([m,m,m])
m4 = np.array([0,m,m])
var = 27
c1 = np.array([[var,0,0], [0, var,0], [0,0,var]])
c2 = np.array([[var,0,0], [0, var,0], [0,0,var]])
c3 = np.array([[var,0,0], [0, var,0], [0,0,var]])
c4 = np.array([[var,0,0], [0, var,0], [0,0,var]])
samp = list(np.random.uniform(0, 1, data))
ndata1 = 0
ndata2 = 0
ndata3 = 0
for i in range(len(samp)):
    if samp[i] < 0.3:
        ndata1 += 1
    elif samp[i] < 0.6:
        ndata2 += 1
    else:
        ndata3 += 1
ndata1, ndata2, ndata3
N_data = [ndata1, ndata2, ndata3]
N_data
ndata30 = 0
ndata31 = 0

```

```

for i in range(nsamples3):
    label0_random = np.random.uniform(0,1)
    if(label0_random>=0.5):
        ndata30+=1
    else:
        ndata31+=1
labels30 = np.random.multivariate_normal(m3, c3,ndata30)
labels31 = np.random.multivariate_normal(m4, c4,ndata31)
label3 = np.concatenate((labels30, labels31))
label3.shape
label1=np.random.multivariate_normal(m1, c1, ndata1)
label2=np.random.multivariate_normal(m2, c2, ndata2)
label1.shape
r = [label1, label2, label3]
label1[:,0]
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.scatter(label1[:,0], label1[:,1], label1[:,2], '+',color = 'blue')
ax.scatter(label2[:,0], label2[:,1], label2[:,2], '+',color = 'red')
ax.scatter(label3[:,0], label3[:,1], label3[:,2], '+',color = 'green')
plt.show()
m3 = 0.5*m3 + 0.5*m4
cov3 = np.cov(label3, rowvar=False)
cov3
Mean = [m1,m2, m3]
Covariance = [c1, c2, cov3]
lambda_matrix = [[0,1,1],[1,0,1],[1,1,0]]
def get_risk(i,x, lambda_matrix):
    total=0
    for j in range(0,3):
        total = total+lambda_matrix[i][j]*p[j]*multivariate_normal.pdf(x, Mean[j], Covariance[j])
    return total
loss_matrix = [[0,0,0],[0,0,0],[0,0,0]]
def get_map(gauss, lambda_matrix):
    correct = []
    incorrect = []
    for i in r[gauss]:
        choice = np.argmin([get_risk(0,i,lambda_matrix), get_risk(1,i,lambda_matrix),
get_risk(2,i,lambda_matrix)])
        if choice == gauss:
            correct.append(i)
        else:
            incorrect.append(i)
    loss_matrix[1][choice] +=1

```



```

correct_xs = []
correct_ys = []
correct_zs = []
for i in correct:
    correct_xs.append(i[0])
    correct_ys.append(i[1])
    correct_zs.append(i[2])
incorrect_xs = []
incorrect_ys = []
incorrect_zs = []
for i in incorrect:
    incorrect_xs.append(i[0])
    incorrect_ys.append(i[1])
    incorrect_zs.append(i[2])

return [correct_xs, correct_ys, correct_zs],[incorrect_xs, incorrect_ys, incorrect_zs]

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
correct, incorrect = get_map(0, lambda_matrix)
ax.scatter(correct[0],correct[1],correct[2],'+',color='blue')
ax.scatter(incorrect[0],incorrect[1],incorrect[2],'+',color='red')
correct, incorrect = get_map(1, lambda_matrix)
ax.scatter(correct[0],correct[1],correct[2],'+',color='blue')
ax.scatter(incorrect[0],incorrect[1],incorrect[2],'+',color='red')
correct, incorrect = get_map(2, lambda_matrix)
ax.scatter(correct[0],correct[1],correct[2],'+',color='blue')
ax.scatter(incorrect[0],incorrect[1],incorrect[2],'+',color='red')
plt.show()
total_loss = 0
for i in range(0,3):
    for j in range(0,3):
        if i!=j:
            total_loss = total_loss + loss_matrix[i][j]
print(total_loss/10000)
loss_matrix
for i in range(0,3):
    loss_matrix[i] = [x/N_samples[i] for x in loss_matrix[i]]
print(loss_matrix)
lambda_matrix = [[0,1,10],[1,0,10],[1,1,0]]
def get_risk(i,x, lambda_matrix):
    total=0
    for j in range(0,3):
        total = total+lambda_matrix[i][j]*p[j]*multivariate_normal.pdf(x, Mean[j], Covariance[j])
    return total

```

```

loss_matrix = [[0,0,0],[0,0,0],[0,0,0]]
def get_map(gauss, lambda_matrix):
    correct = []
    incorrect = []
    for i in r[gauss]:
        choice = np.argmin([get_risk(0,i,lambda_matrix), get_risk(1,i,lambda_matrix),
get_risk(2,i,lambda_matrix)])
        if choice == gauss:
            correct.append(i)
        else:
            incorrect.append(i)
        loss_matrix[1][choice] +=1
    correct_xs = []
    correct_ys = []
    correct_zs = []
    for i in correct:
        correct_xs.append(i[0])
        correct_ys.append(i[1])
        correct_zs.append(i[2])
    incorrect_xs = []
    incorrect_ys = []
    incorrect_zs = []
    for i in incorrect:
        incorrect_xs.append(i[0])
        incorrect_ys.append(i[1])
        incorrect_zs.append(i[2])

    return [correct_xs, correct_ys, correct_zs],[incorrect_xs, incorrect_ys, incorrect_zs]
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
correct, incorrect = get_map(0, lambda_matrix)
ax.scatter(correct[0],correct[1],correct[2],'+',color='blue')
ax.scatter(incorrect[0],incorrect[1],incorrect[2],'+',color='red')
correct, incorrect = get_map(1, lambda_matrix)
ax.scatter(correct[0],correct[1],correct[2],'+',color='blue')
ax.scatter(incorrect[0],incorrect[1],incorrect[2],'+',color='red')
correct, incorrect = get_map(2, lambda_matrix)
ax.scatter(correct[0],correct[1],correct[2],'+',color='blue')
ax.scatter(incorrect[0],incorrect[1],incorrect[2],'+',color='red')
plt.show()
total_loss = 0
for i in range(0,3):
    for j in range(0,3):
        if i!=j:

```

```

        total_loss = total_loss + loss_matrix[i][j]
print(total_loss/10000)
lambda_matrix = [[0,1,100],[1,0,100],[1,1,0]]
def get_risk(i,x, lambda_matrix):
    total=0
    for j in range(0,3):
        total = total+lambda_matrix[i][j]*p[j]*multivariate_normal.pdf(x, Mean[j], Covariance[j])
    return total

fig = plt.figure()
ax = fig.add_subplot(projection='3d')
correct, incorrect = get_map(0, lambda_matrix)
ax.scatter(correct[0],correct[1],correct[2],'+',color='blue')
ax.scatter(incorrect[0],incorrect[1],incorrect[2],'+',color='red')
correct, incorrect = get_map(1, lambda_matrix)
ax.scatter(correct[0],correct[1],correct[2],'+',color='blue')
ax.scatter(incorrect[0],incorrect[1],incorrect[2],'+',color='red')
correct, incorrect = get_map(2, lambda_matrix)
ax.scatter(correct[0],correct[1],correct[2],'+',color='blue')
ax.scatter(incorrect[0],incorrect[1],incorrect[2],'+',color='red')
plt.show()
total_loss = 0
for i in range(0,3):
    for j in range(0,3):
        if i!=j:
            total_loss = total_loss + loss_matrix[i][j]
print(total_loss/10000)

```