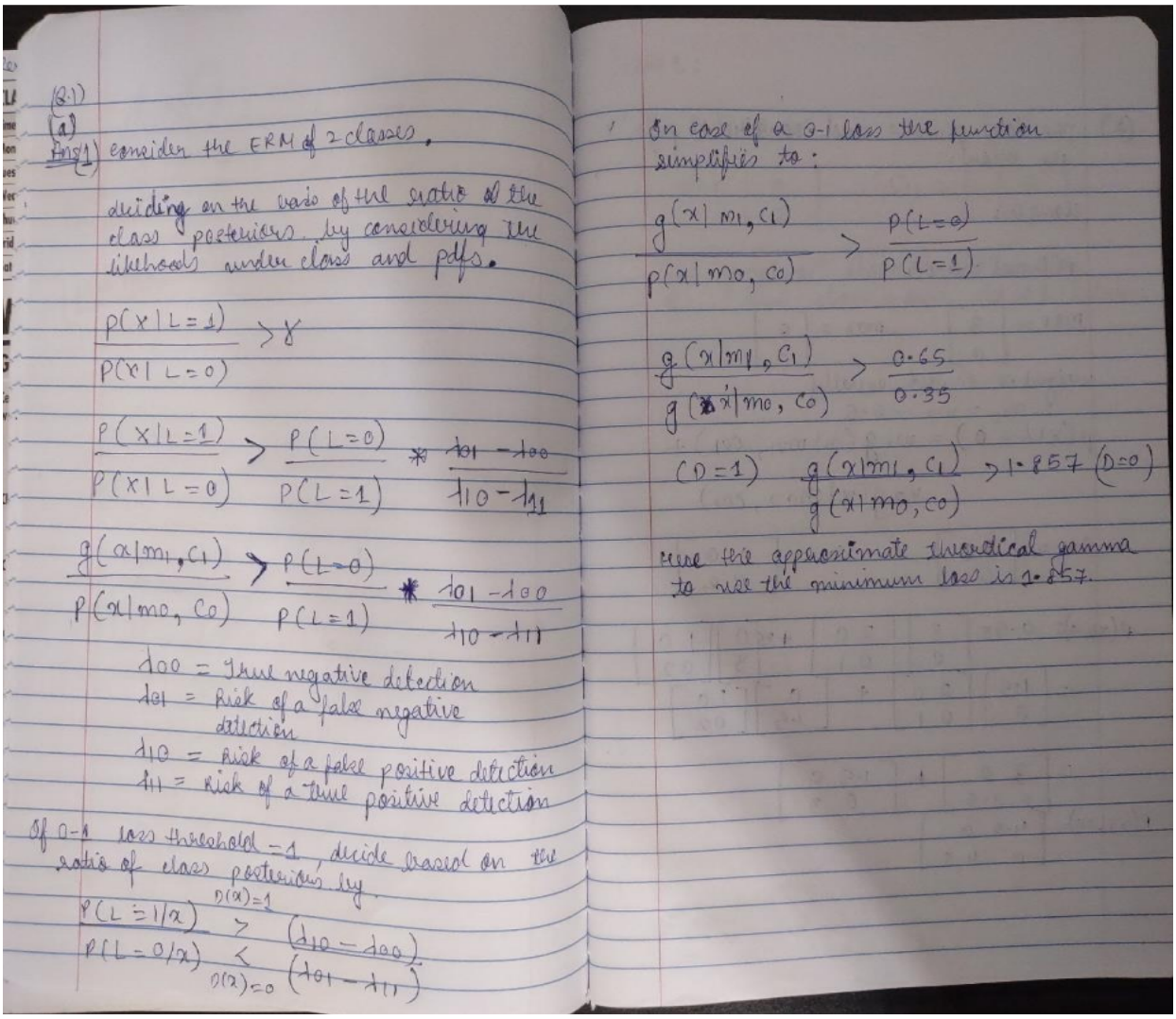
Introduction to Machine Learning Assignment 1

Question 1:

(A)Given that ` : Part 1

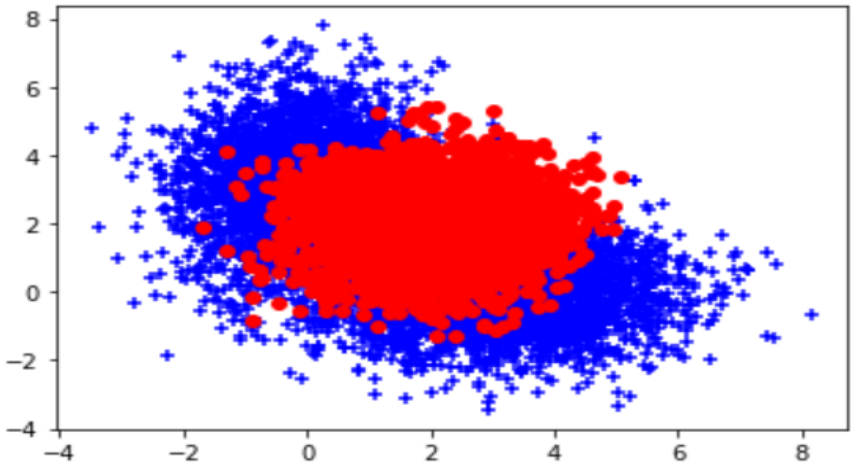
The class priors are P(L = 0) = 0.65 and P(L = 1) = 0.35. The class class-conditional pdfs are $p(x|L=0)=w_1 g(x|m_{01},C_{01})+w_2 g(x|m_{02},C_{02})$ and $p(x|L=1)=g(x|m_1,C_1)$, where g(x|m,C) is a multivariate Gaussian probability density function with mean vector m and covariance matrix C. The parameters of the class-conditional Gaussian pdfs are: $w_1 = w_2 = 1/2$, and

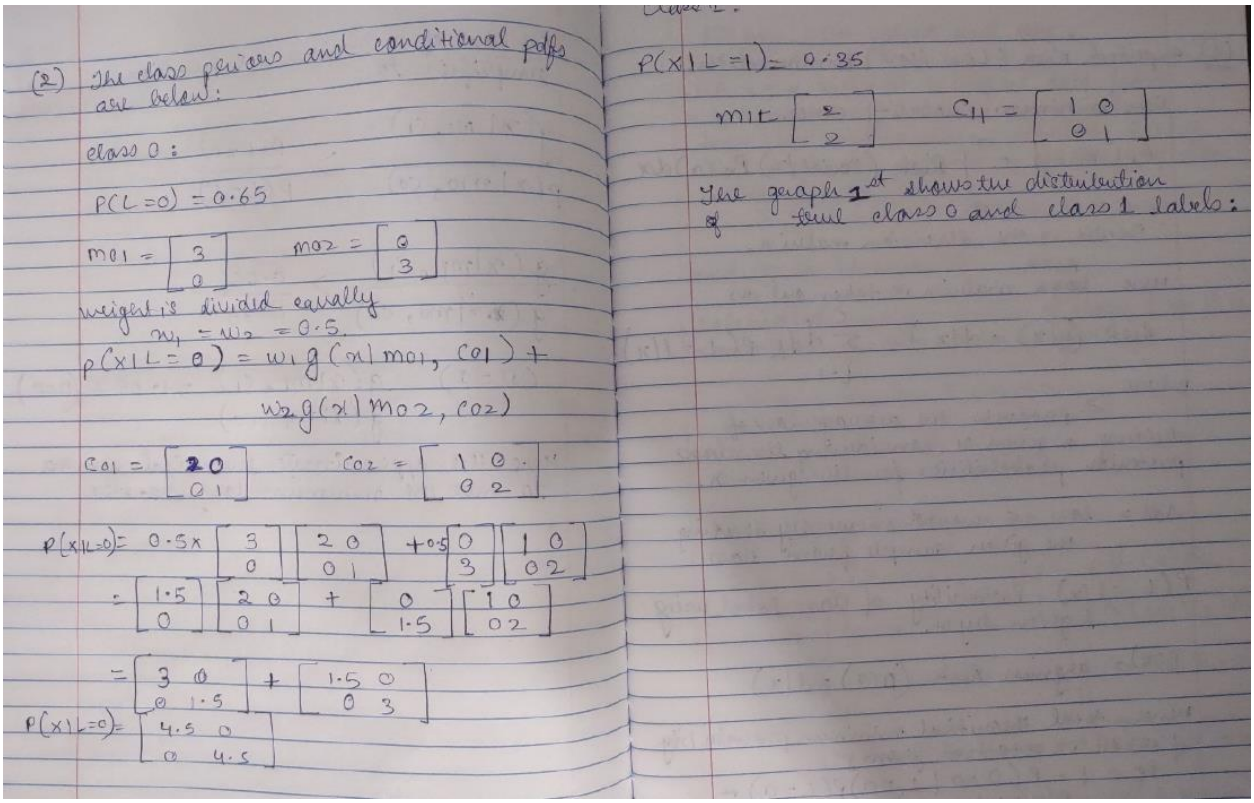
$$\mathbf{m}_{01} = \begin{bmatrix}3\\0\end{bmatrix} \quad \mathbf{C}_{01} = \begin{bmatrix}2 & 0\\0 & 1\end{bmatrix} \quad \mathbf{m}_{02} = \begin{bmatrix}0\\3\end{bmatrix} \quad \mathbf{C}_{02} = \begin{bmatrix}1 & 0\\0 & 2\end{bmatrix} \quad \mathbf{m}_{1} = \begin{bmatrix}2\\2\end{bmatrix} \quad \mathbf{C}_{1} = \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$$



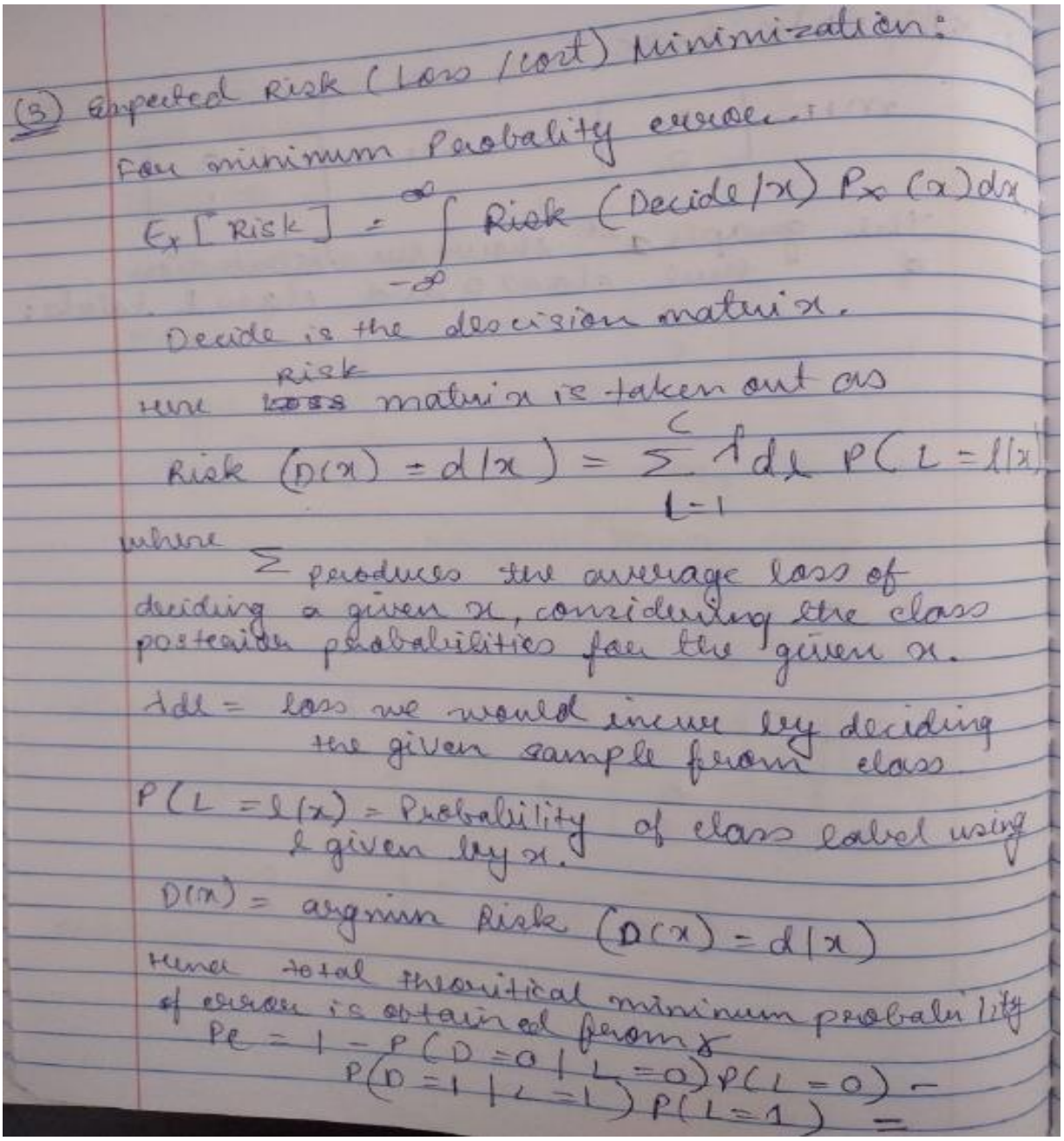
Part 2)

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

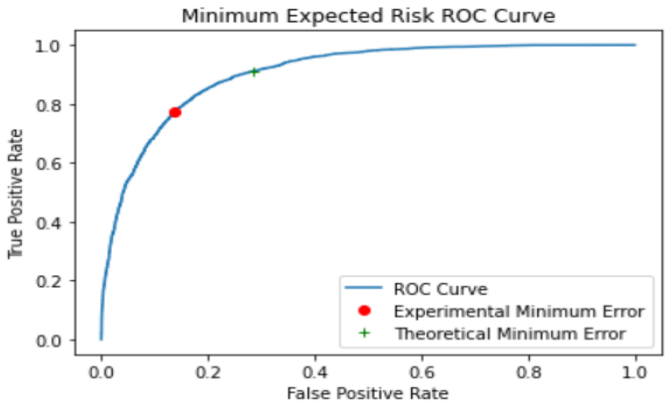




Part 3



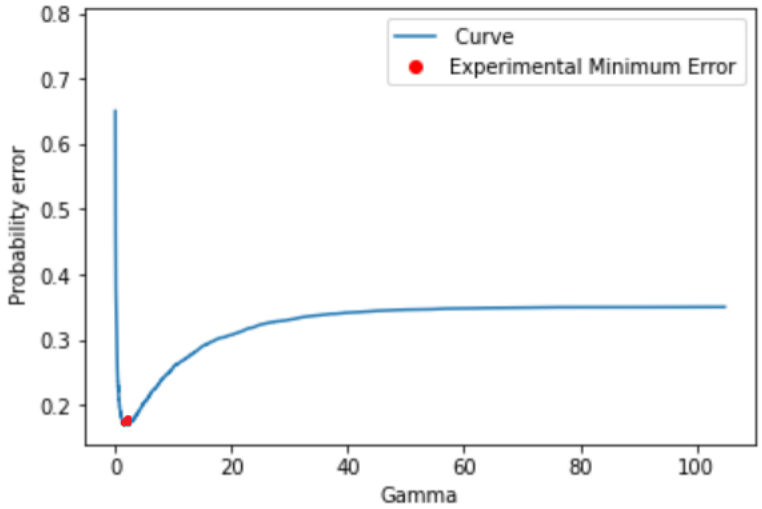
Plot of the ROC curve for the minimum expected risk classifier applied on the 10,000 samples with gamma varied from 0 to 1:




```
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.

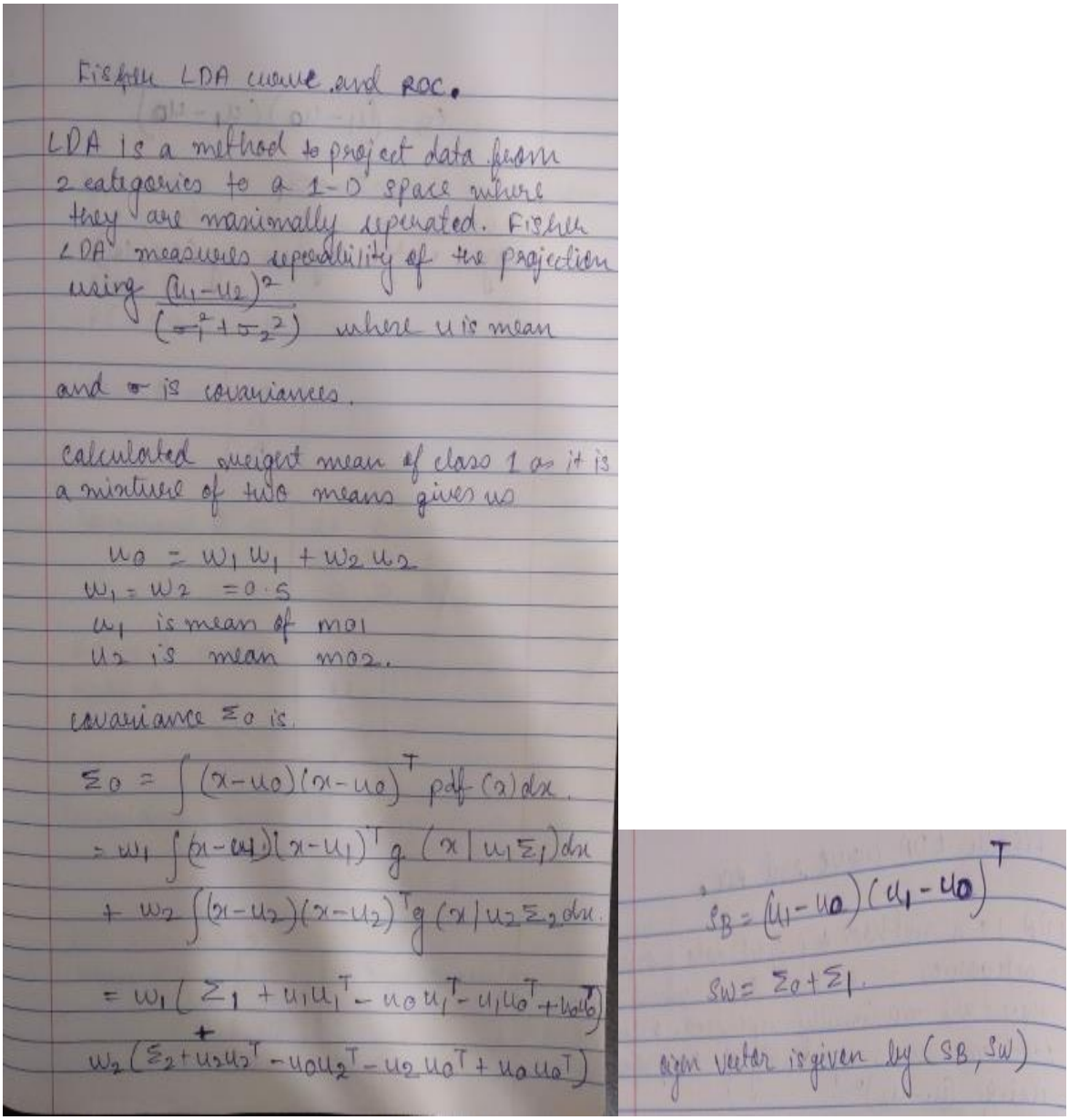
Also as shown below The location of the minimum error is marked. Additionally as the gamma parameter approached its limits at 0 and +∞ the probability of error asymptotes to the priors for the two distributions. That is when the gamma is set to its minimum value all of the data points will be classified 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 +∞ 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 σ is covariances.

calculated weight 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 m01

u_2 is mean m02.

covariance Σ_0 is

$$\Sigma_0 = \int (x-u_0)(x-u_0)^T \, pdf(x)\, dx.$$

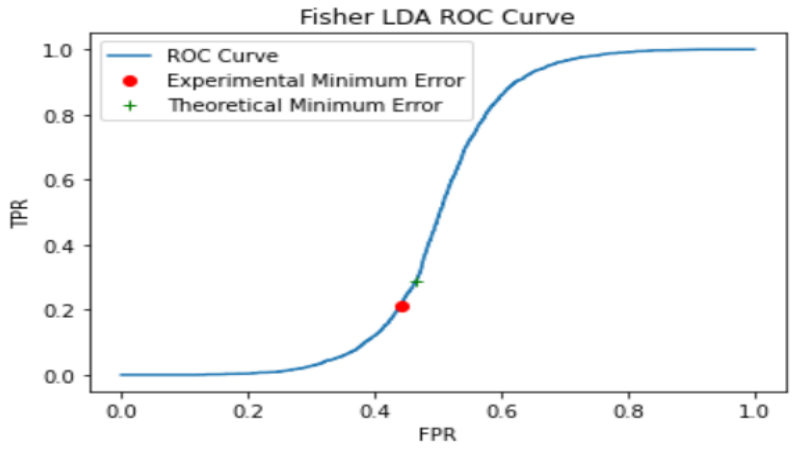
$$= w_1 \int (x-u_1)(x-u_1)^T g(x|u_1\Sigma_1)dx + w_2 \int (x-u_2)(x-u_2)^T g(x|u_2\Sigma_2)dx.$$

$$= w_1(\Sigma_1 + u_1u_1^T - u_0u_1^T - u_1u_0^T + u_0u_0^T) + w_2(\Sigma_2 + u_2u_2^T - u_0u_2^T - u_2u_0^T + u_0u_0^T)$$

$S_B = (u_1-u_0)(u_1-u_0)^T$

$S_W = \Sigma_0 + \Sigma_1$

eigen vector is given by (S_B, S_W)



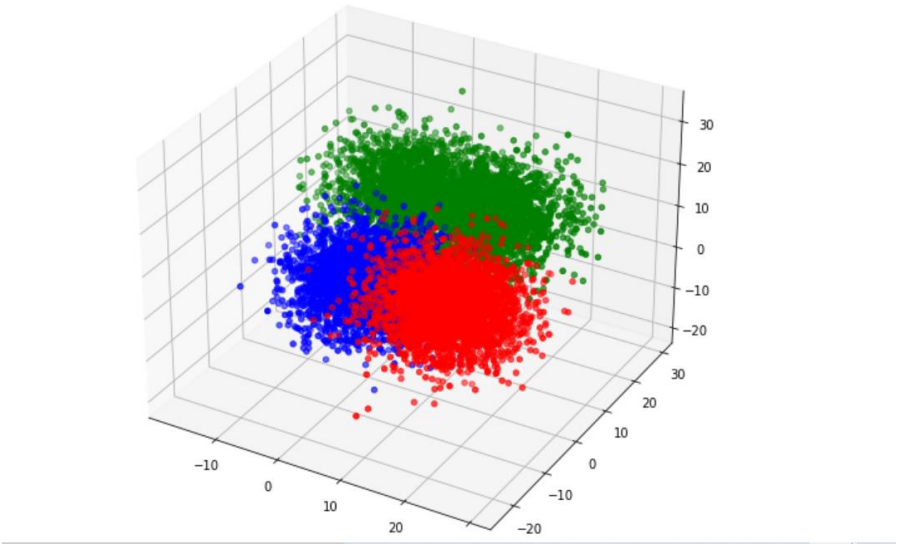

```
Experimental Gamma
0.7851069764811559
Experimental minerror
0.3501000461751577
Theoretical Error
0.4897722025550254
```

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 _ from -infinity to +infinity . Based on the 10,000 samples generated, the minimum probability of error was calculated to be pe = 0:35
at the threshold value of of tau = 0.785. This error and threshold value were calculated by finding the minimum error as the value of tau was changed from -infinity to +infinity

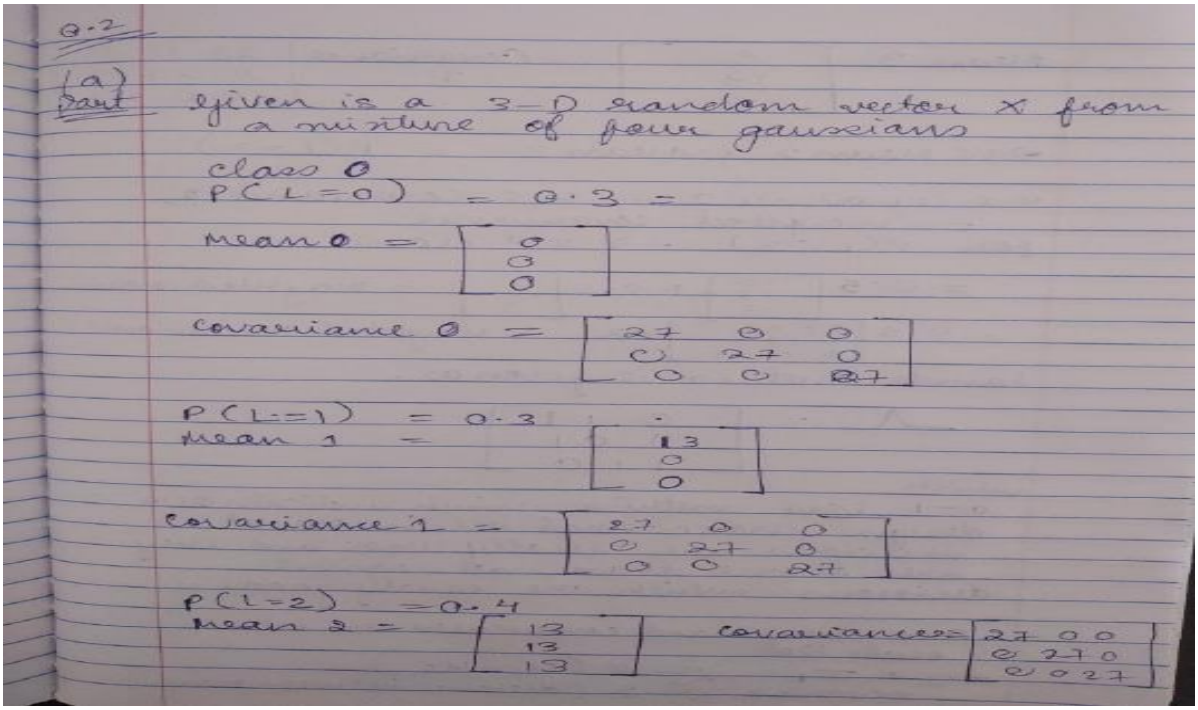
Question no 2 :

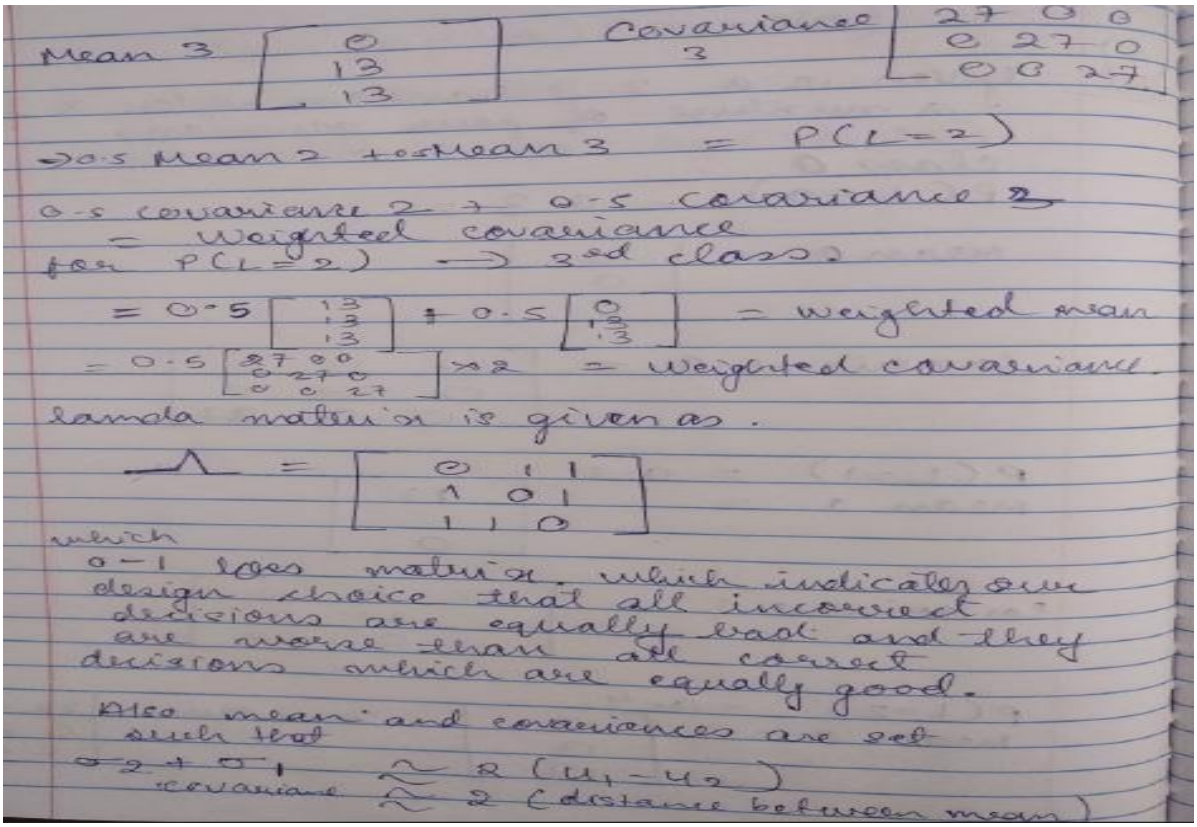
PART A

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.

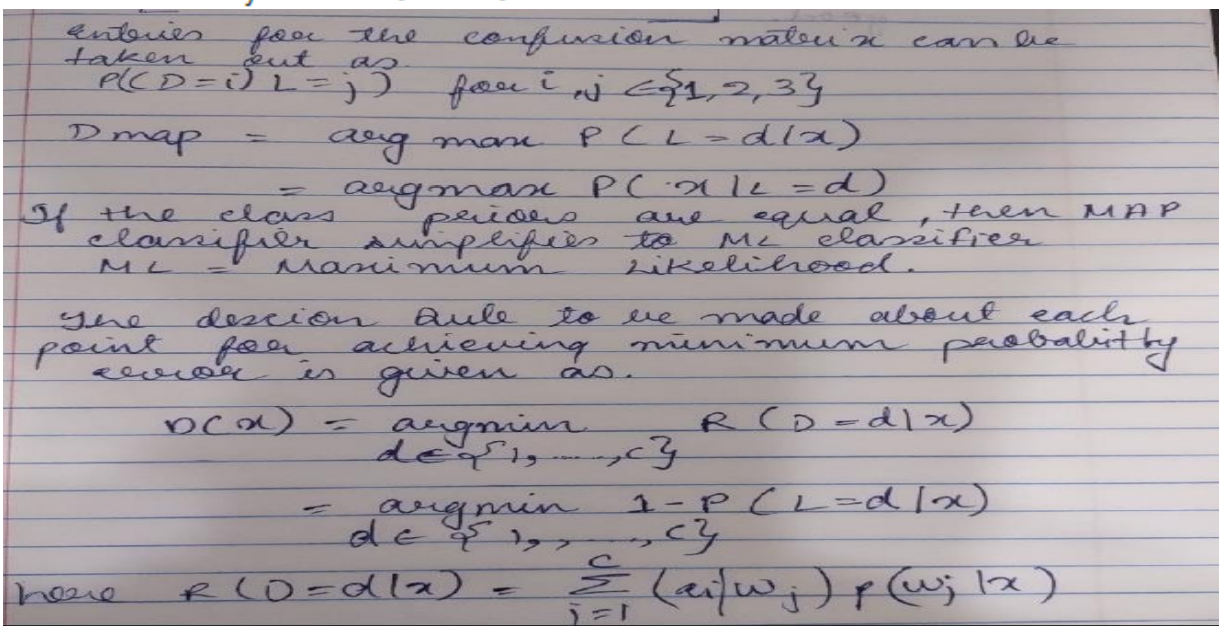




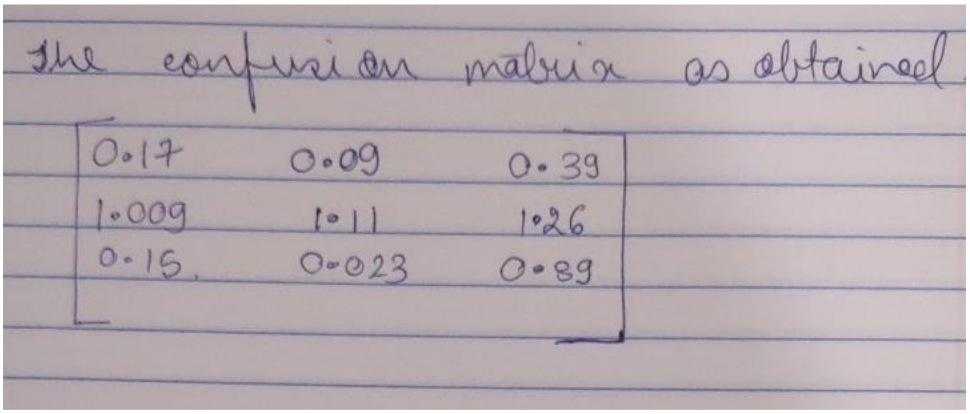
(b) To determine the decision to be made about each point we used

$$\alpha_* = argmin_{\alpha_i} R(\alpha_i|x)$$

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



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

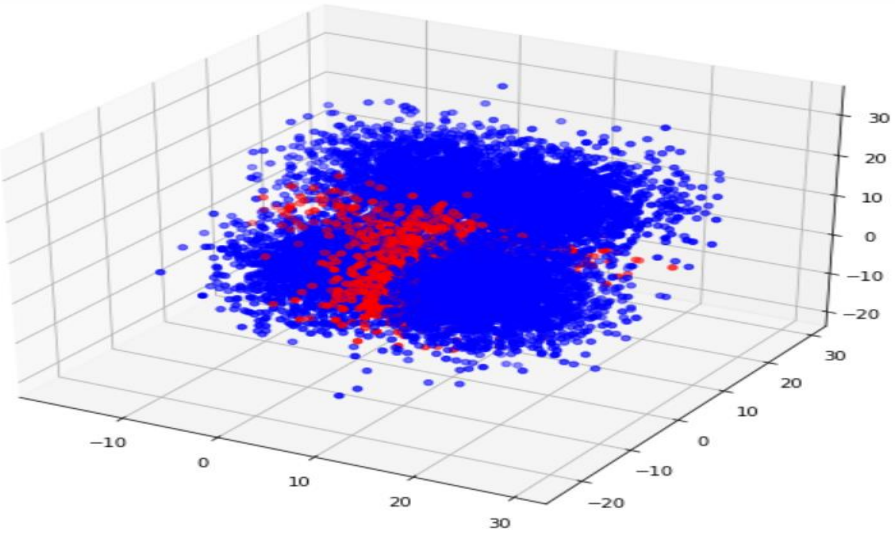


The total expected risk of 0.7134

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)

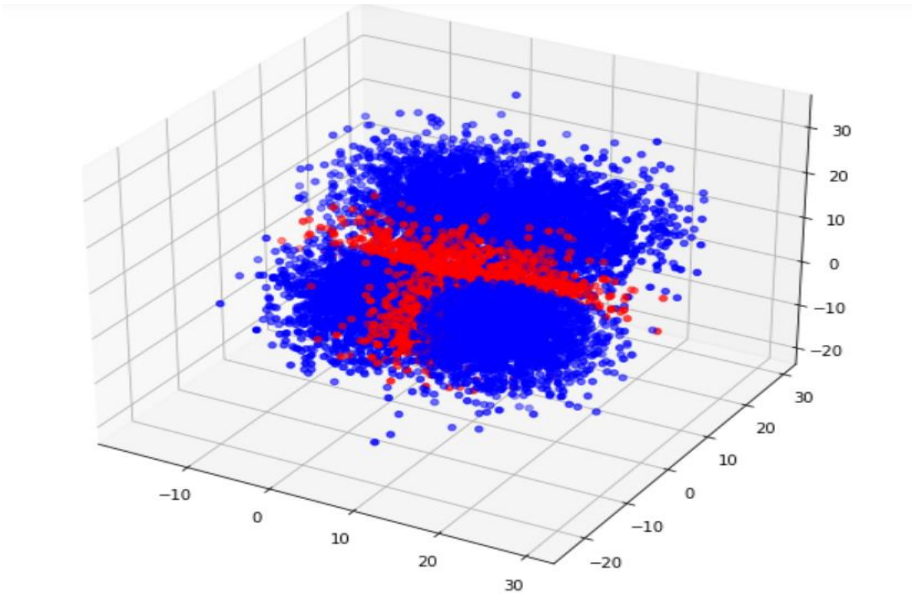


Here the class 3 will try to avoid making mistakes and is having the larger value . Decision is in the favor of class 3. Here the data distributions that are overlapping having new lambda matrix will have a lambda of 1. Interestingly, its is observed that probability of class here is showing less which 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

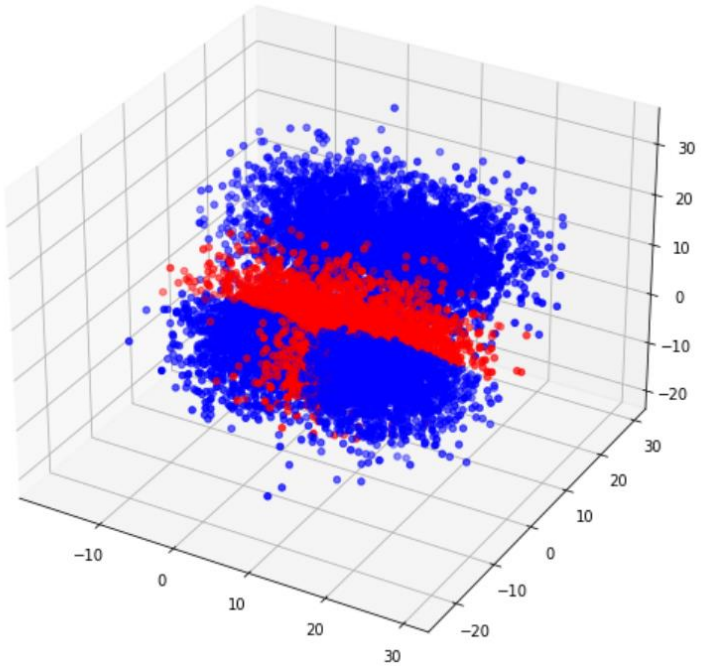
$$\Lambda_{10} = \begin{bmatrix} 0 & 1 & 10 \\ 1 & 0 & 10 \\ 1 & 1 & 0 \end{bmatrix}$$



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 = []
N0_discriminant = N01discriminant + N02discriminant
fulldiscriminant = N01discriminant + N02discriminant + N1discriminant
```

```
#fulldiscriminant.append(0)
len(fulldiscriminant)
for i in sorted(fulldiscriminant):
   fp = len([j for j in N0_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 N0_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 maximum 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 same l o g i c a s a b o v e t o make 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)
```