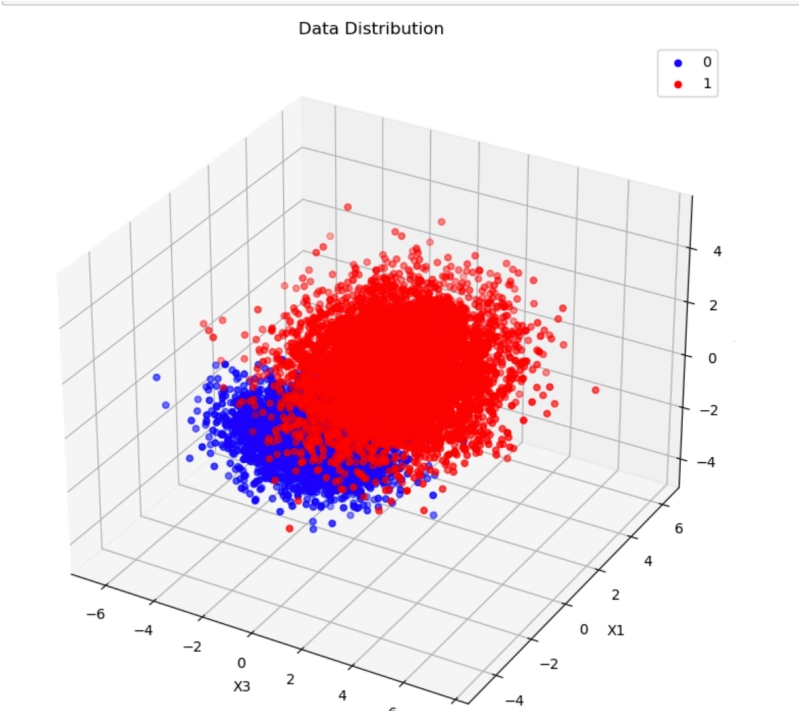
Introduction to Machine Learning and Pattern Recognition
EECE5644
Assignment 1

Name: Vaibhav Kejriwal
NUID: 002201423

Q1.
Part A

Part A – Minimum Expected Risk Classification

Using the mean and covariance matrices given, 10000 samples are generated having multivariate gaussian distribution.

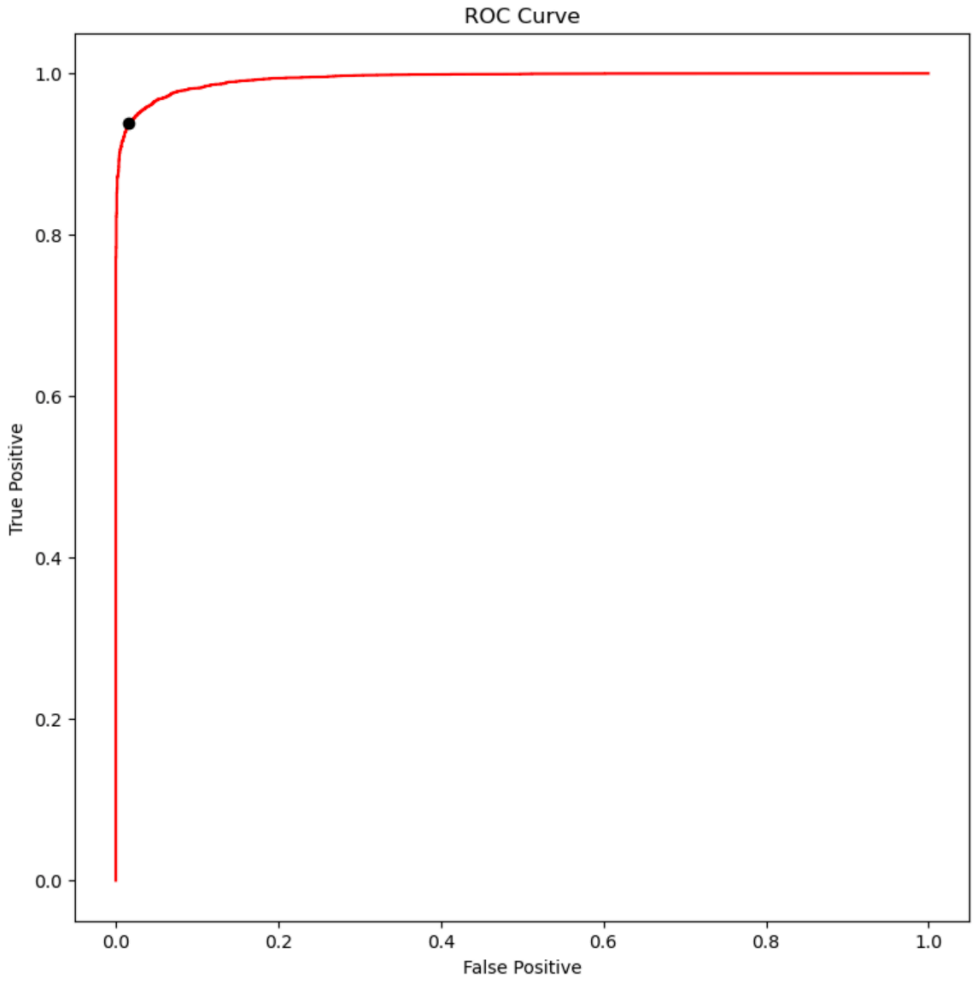


Minimum Expected Risk Classification Rule is as follows:

$$\frac{g(x|m_0, C_0)}{g(x|m_1, C_1)} \underset{<}{>} \frac{P(L=0)\,(\lambda_{10}-\lambda_{00})}{P(L=1)\,(\lambda_{01}-\lambda_{11})} = \frac{(0.65)\,(\lambda_{10}-\lambda_{00})}{(0.35)\,(\lambda_{01}-\lambda_{11})}$$

$$\frac{g(x|m_0, C_0)}{g(x|m_1, C_1)} \underset{(D=0)}{\overset{(D=1)}{>}} (6.5)\,\frac{(\lambda_{10}-\lambda_{00})}{(\lambda_{01}-\lambda_{11})} = \gamma$$

2. The ROC curve is generated by approximating the variation of threshold value γ from 0 to ∞. Practically, γ values are sorted and the mid-points between consecutive two values are considered as threshold points ranging from minimum to maximum.



3. Theoretically, the value of γ is found out by dividing the class priors (0.7/0.3) which is equal to 2.33. With this threshold, false positives and true positives are computed which help in estimating the Minimum P(error). It can be observed that there is negligible difference in the theoretical and experimental values.

	γ	Minimum Error
Theoretical	1.857143	0.031358
Experimental	1.800646	0.031138

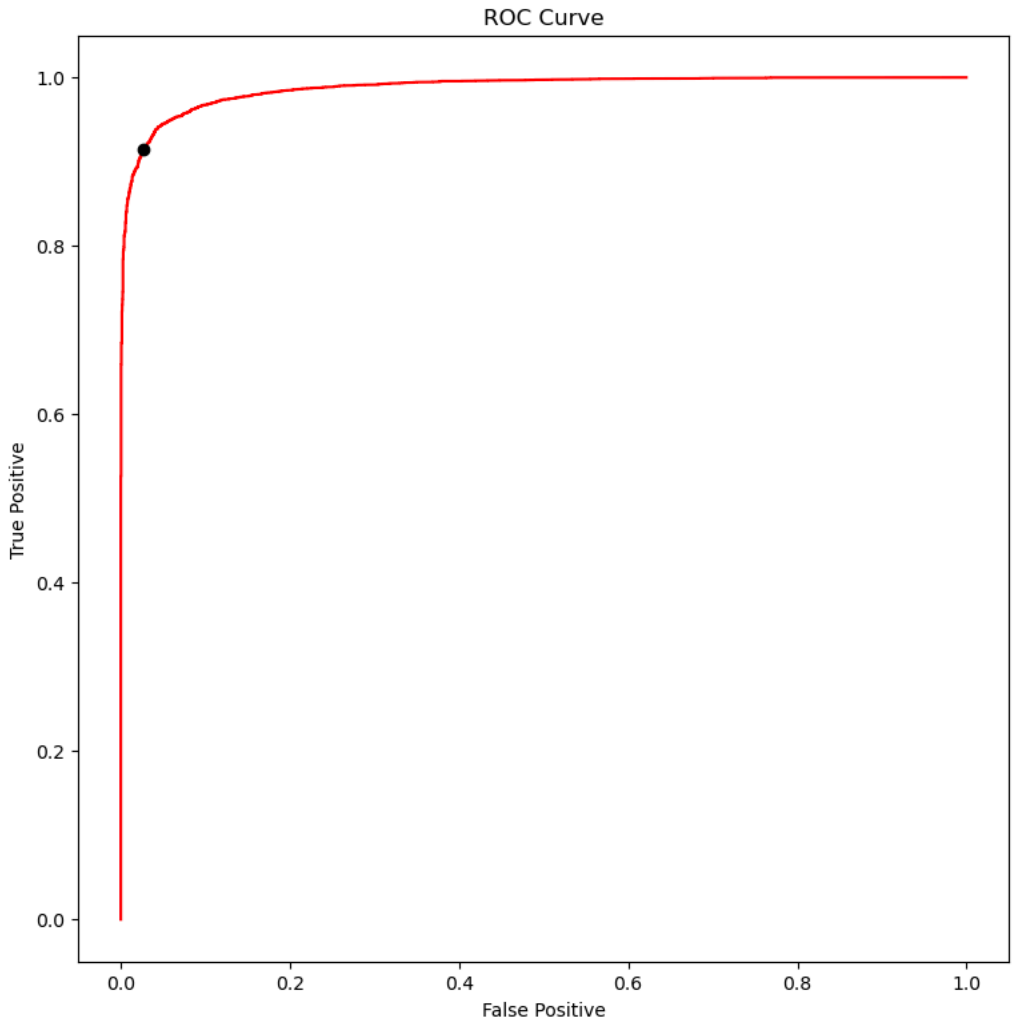
Part B – Naive Bayes Classification

1. Minimum Expected Risk Classification Rule is as follows:

$$\frac{g_{NB}(x|m_0, I)}{g_{NB}(x|m_1, I)} > \frac{P(L=0)(\lambda_{10}-\lambda_{00})}{P(L=1)(\lambda_{01}-\lambda_{11})} = \frac{(0.65)(\lambda_{10}-\lambda_{00})}{(0.35)(\lambda_{01}-\lambda_{11})}$$

$$\frac{g_{NB}(x|m_0, I)}{g_{NB}(x|m_1, I)} \underset{(D=0)}{\overset{(D=1)}{>}} (6.5)\frac{(\lambda_{10}-\lambda_{00})}{(\lambda_{01}-\lambda_{11})} = \gamma$$

2. The ROC curve for Naive Bayes Classifier is not as sharp as that of the ERM classifier implying a slight decrease in performance. The original covariance values for different combination of features were inappreciable. Thus, assuming that the features are independent did not significantly deviate the results. Nevertheless, the shape implies that the classification model is reasonably good.



3. There is an infinitesimal difference between the theoretical and practical values of error despite gamma having considerable variation. The minimum P error has increased compared to the previous ERM classifier.

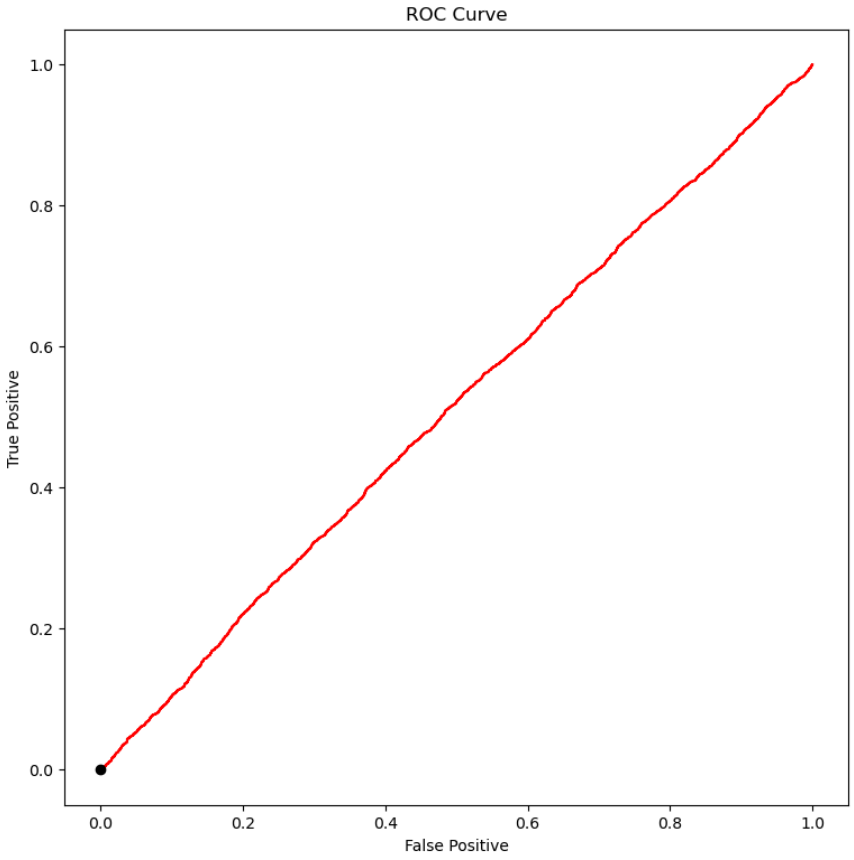
	γ	Minimum Error
Theoretical	1.857143	0.046865
Experimental	1.568752	0.045701

Part C – LDA Classification

1. LDA Classification Rule is as follows:

(D=1) $w^T_{LDA}x > \tau$ (D=0)

2. ROC curve is plotted by ranging τ from minimum to maximum value taking mid-points of consecutive values as thresholds. The shape of the ROC curve implies that the classification model is not up to the mark.



3. It can be observed that there is a notable difference between the theoretical and practical values of γ. The Minimum P error is considerably larger than the previous two Bayesian classifiers possibly due to the overlapping nature of the data distribution.

	γ	Minimum Error
Theoretical	1.8572143	0.376001
Experimental	4.607466	0.350179

Problem 2 Part A

1. Generating Data: P (x | L = 0) = 0.3

P (x | L = 2) = 0.4

P (x | L = 1) = 0.3

Mean Vectors for Gaussian Mixtures-

Class 0 - μ_0 = [0, 0, 45] Gaussian Mixture 1

Class 1 - μ_1 = [0, 45, 0] Gaussian Mixture 2

Class 2 – μ_2 = [45, 0, 0] with P (Gauss 3 | L = 2) = 0.5

Class 3 μ_3 = [45, 0, 45] with P (Gauss 4 | L = 2) = 0.5

Here, we assume a cube with edge length as 15 units. The 4 corners of the cube are taken as mean vectors for the gaussian distributions to compute the class conditional PDF. The covariance matrices are obtained using the formula given below-

C = (s^2) (I + aA) (I+aA) where |a| << 1

A is a random square matrix with size equal to the number of features, i.e., 3 and s is around 0.4 E. By adding the term aA to the identity matrix, it can be ensured that the distribution is elliptical gaussian with eccentricity a.

2. Decision Rule –

$$\begin{matrix} R(D=0|x) \\ R(D=1|x) \\ R(D=2|x) \end{matrix} = A * \begin{bmatrix} P(L=0|x) \\ P(L=1|x) \\ P(L=2|x) \end{bmatrix}$$

Risk Loss Matrix Class Posteriors

$$\text{Decision}(x) = \underset{d \in \{0,1,2\}}{\text{argmin}} \text{ Risk (Decision}(x) = d \mid x)$$

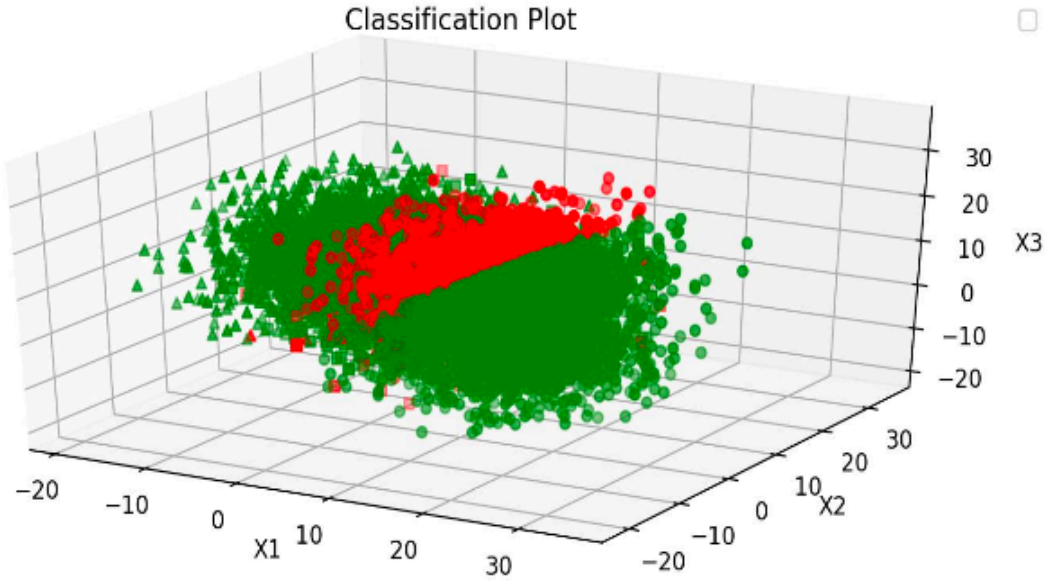
Here, Loss Matrix $A = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$

The Confusion Matrix for this classifier results as –

$$C = \begin{bmatrix} 0.92852564 & 0.0340526 & 0.22406745 \\ 0.03333333 & 0.92582603 & 0.0183955 \\ 0.03814103 & 0.04012138 & 0.75753705 \end{bmatrix}$$

Minimum Expected Risk is 0.08645208097757304

3. Visualization 3-D Scatter Plot-



Part B

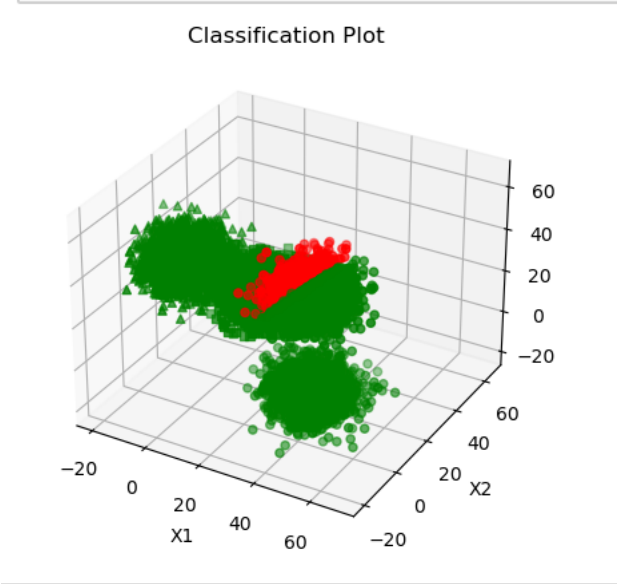
$$\Lambda_{10} = \begin{bmatrix} 0 & 10 & 10 \\ 1 & 0 & 10 \\ 1 & 1 & 0 \end{bmatrix} \quad and \quad \Lambda_{100} = \begin{bmatrix} 0 & 100 & 100 \\ 1 & 0 & 100 \\ 1 & 1 & 0 \end{bmatrix}$$

For A10 Loss Matrix:

Confusion Matrix:

```
[[1.         0.         0.14822335]
 [0.         1.         0.        ]
 [0.         0.         0.85177665]]
```

Average Expected Risk 0.0227455909276464 45

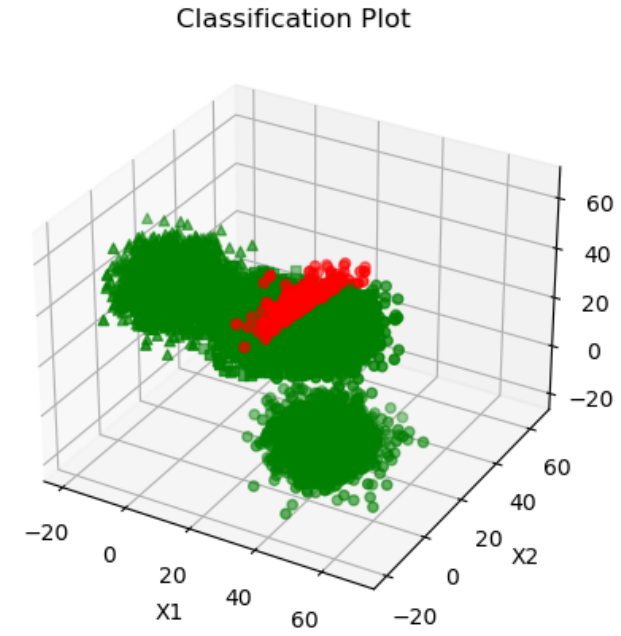


For A100 Loss Matrix:

Confusion Matrix:

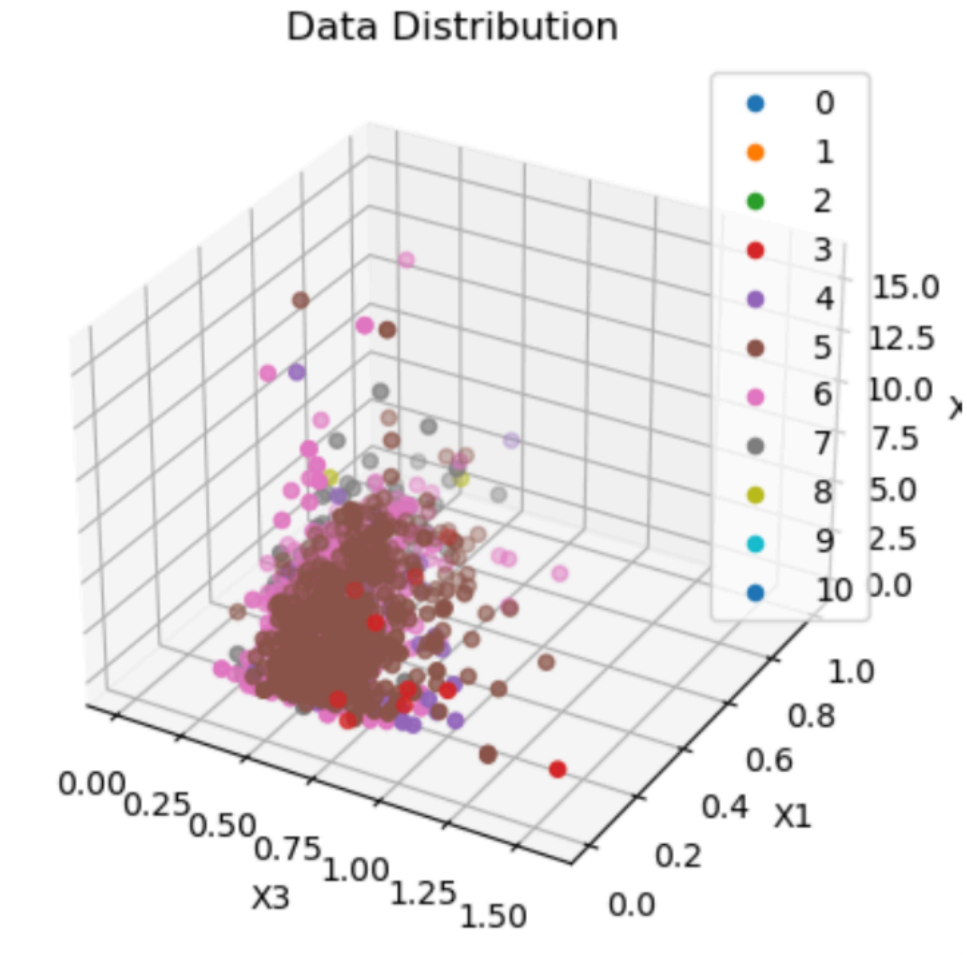
```
[[1.         0.         0.11649746]
 [0.         1.         0.        ]
 [0.         0.         0.88350254]]
```

```
Average Expected Risk
0.034364805418552934
```



As penalty for label 3 is increased, the percentage of misclassifications decreases for this class. With higher loss for a label, the model gets effectively trained to avoid making wrong decisions for this label by compromising other decisions.

Problem 3
Part A – Wine Quality Classification



\

From the distribution it is evident that samples 5 and 6 tend to dominate the dataset whereas samples 0,1,2,9 and 10 are not present in the training set. The dataset consists of 11 features which is a reasonable number to train a computationally efficient model. Any kind of dimensionality reduction technique may result in loss of significant information. Therefore, it is considered all features are relevant to build the classification model.

For the class conditional PDF to be gaussian for the given features, it is assumed that the samples are independent and identically distributed along with 1600 samples to be a sufficiently large number to apply the central limit theorem. This theorem states that a sample distribution approximates a normal distribution if the number of samples is large enough.

Using the sample count, the class priors are computed using the following formula:

P(L) = Numbers of samples belonging to class L / Total Number of samples

On testing the conditionality of the co-variance matrices, it is identified that majority of them yield a very large conditional number. Therefore, it is essential to add a small regularization value to broaden the distribution.

CRegularized = CSampleAverage+λI

Here, **λ**is a hyper-parameter which decides the amount of regularization added to the original variance values. To calculate **λ,** I considered finding out the arithmetic average of

the non zero eigen values of the matrix. The trace (sum of diagonal elements) is equal to the sum of eigen values of a matrix and rank is number of non-zero eigen values. Hence,

Arithmetic Average = trace (CSampleAverage) / rank (CSampleAverage)

λ = α (Arithmetic Average)

where α is a small real number between 0 and 1. This is a hyperparameter controlling the main hyperparameter **λ,** which can be tuned in the range of 10^{-3} to 10^{-9} to check for maximum

accuracy of the model. The loss function selected was 0-1 loss in order to allot equal penalty for all the incorrect decisions and 0 for correct decisions. Thus, a MAP classifier is designed to solve this classification problem.

Confusion Matrix is :

```
Average Expected Risk 0.3244031178791402
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         1.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.18867925 0.01908957
  0.01567398 0.00502513 0.         0.         0.        ]
 [0.         0.         0.         0.         0.43396226 0.6328928
  0.22100313 0.02512563 0.         0.         0.        ]
 [0.         0.         0.         0.         0.33962264 0.32599119
  0.64733542 0.44723618 0.16666667 0.         0.        ]
 [0.         0.         0.         0.         0.03773585 0.02202643
  0.09717868 0.45226131 0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.01880878 0.07035176 0.83333333 0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.        ]]
```

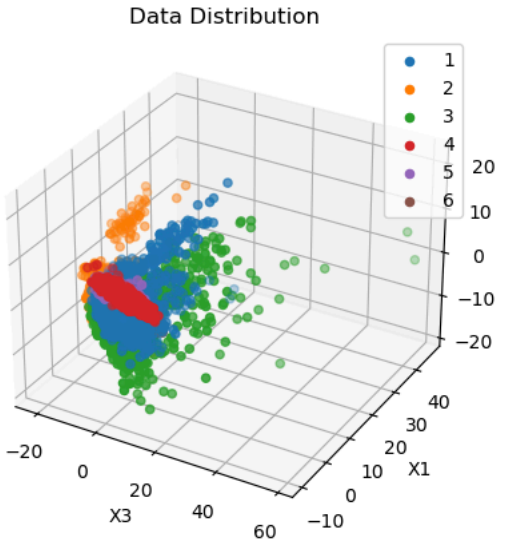
From the results, it can be inferred that Gaussian models may not always be the advisable choice for solving classification models where we do not possess domain knowledge of the problem. There is a possibility that the class posteriors were highly influenced by the priors selected based on the sample count, resulting in misleading results on new samples. Thus, in a practical scenario, prior beliefs need to be judiciously selected in order to strike a balance between priors and role of the features, i.e., class-conditional PDF. A very small $\boldsymbol{\alpha}$ of range 10^{-9} results in the best performance of the model proving that excess of this hyper-parameter will prevent the model from learning essential details of the distributions.

Part B – Human Activity Classification

Here, it is given that there are 561 features which is a large number having a few disadvantages like redundant features not contributing in classification, correlated features providing similar information and extra computation time. Therefore, PCA is used as a dimensionality reduction method to transform these features into few relevant ones based on maximizing the variance.

Data Visualization after applying PCA-

```
Average Expected Risk 0.09130586093998115
[[0.91272431 0.02329916 0.02839757 0.         0.0014556  0.        ]
 [0.05546493 0.91985089 0.12778905 0.00155521 0.         0.        ]
 [0.03181077 0.05684995 0.84381339 0.         0.         0.00071073]
 [0.         0.         0.         0.60186625 0.08151383 0.03411514]
 [0.         0.         0.         0.35692068 0.91411936 0.        ]
 [0.         0.         0.         0.03965785 0.00291121 0.96517413]]
```



Here, a Gaussian classifier tends to yield better results as compared to the Wine Quality Classification problem. This can be attributed to the fact that the dataset is balanced, giving rise to class priors in the similar range. Thus, the class posteriors mainly rely on the class- conditional distributions and the model is able to identify relationships between the reduced features and class labels. PCA does help as an effective tool to identify meaningful information from the abundant feature set. However, this model still needs to be tested on new samples to check for overfitting of the training set.

Code:

Q1 A

jupyter Assignment1_Q1_EECE5644_Vaibhav_Kejriwal Last Checkpoint: 4 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [43]:
```
from turtle import color
```

In [44]:
```
import numpy as np
```

In [45]:
```
import matplotlib.pyplot as plt
```

In [46]:
```
from scipy.stats import multivariate_normal
```

In [47]:
```
from mpl_toolkits.mplot3d import Axes3D
```

In [48]:
```
np.set_printoptions(threshold=np.inf)
```

In [49]:
```
plt.rcParams['figure.figsize'] = [9,9]
```

In [50]:
```
N_features = 4            # Number of features
N_Samples = 10000         # Number of Samples
N_labels = 2
```

In [51]:
```
# Mean vectors
mean_matrix = np.ones(shape=[N_labels, N_features])
mean_matrix [0, :] = [-1,-1,-1,-1]
```

In [52]:
```
# Covariance matrices
covariance_matrix = np.ones(shape=[N_labels, N_features, N_features])
covariance_matrix [0, :, :] = [[2, -0.5, 0.3, 0],[-0.5, 1, -0.5, 0], [0.3, -0.5, 1, 0],
covariance_matrix [1, :, :] = [[1, 0.3, -0.2, 0], [0.3, 2, 0.3, 0], [-0.2, 0.3, 1, 0],
```

In [53]:
```
#Seed to obtain same results for random numbers
np.random.seed(10)
```

In [54]:
```
# Class Priors and assigning labels
priors = [0.65, 0.35]
label = (np.random.rand(N_Samples) >= priors[1]).astype(int)
```

In [55]:
```
# Generate gaussian distribution for 10000 samples using mean and covariance matrices f
X = np.zeros(shape = [N_Samples, N_features])
for i in range(N_Samples):
        if (label[i] == 0):
                X[i, :] = np.random.multivariate_normal(mean_matrix[0, :], covariance_m
        elif (label[i] == 1):
                X[i, :] = np.random.multivariate_normal(mean_matrix[1, :], covariance_m
```

In [56]:
```
# Compute discriminant score using class conditional PDF
GaussPDF0 = np.log(multivariate_normal.pdf(X,mean = mean_matrix[0, :], cov = covariance
GaussPDF1 = np.log(multivariate_normal.pdf(X,mean = mean_matrix[1, :], cov = covariance
discrim_score = GaussPDF1 - GaussPDF0
```

In [57]:
```
# Sort tau values to navigate from minimum to maximum value
sorted_tau = np.sort(discrim_score)
tau_sweep = []
```

In [58]:
```
# Calculate mid-points which will be used as threshold values
for i in range(0,9999):
        tau_sweep.append((sorted_tau[i] + sorted_tau[i+1])/2.0)
```

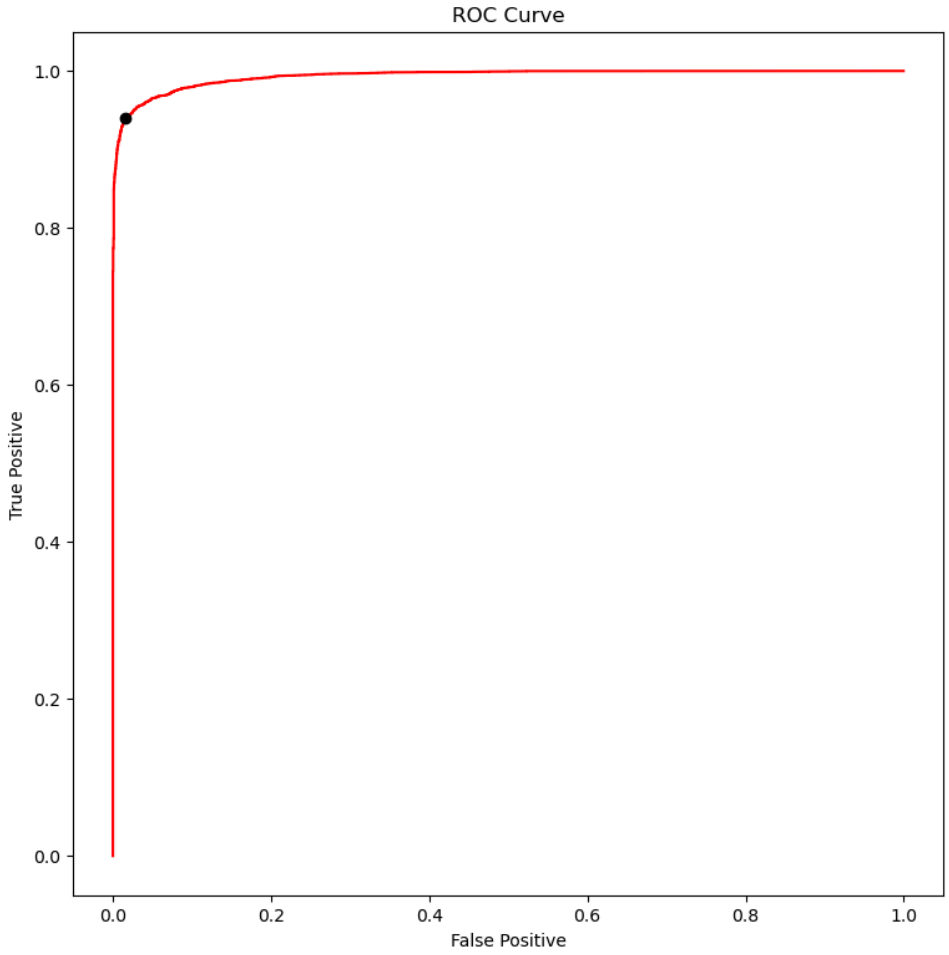
In [59]:
```
# Array initialization for results
decision = []
TP = [None] * len(tau_sweep)
FP = [None] * len(tau_sweep)
minPerror = [None] * len(tau_sweep)
```

In [60]:
```
# Classify for each threshold and compute error and evaluation metrics
for (index, tau) in enumerate(tau_sweep):
        decision = (discrim_score >= tau)
        TP[index] = (np.size(np.where((decision == 1) & (label == 1))))/np.size(np.wher
        FP[index] = (np.size(np.where((decision == 1) & (label == 0))))/np.size(np.wher
        minPerror[index] = (priors[0] * FP[index]) + (priors[1] * (1 - TP[index]))
```

In [61]:
```
# Theoretical classification based on class priors
loggamma_ideal = np.log(priors[0] / priors[1])
ideal_decision = (discrim_score >= loggamma_ideal)
TP_ideal = (np.size(np.where((ideal_decision == 1) & (label == 1))))/np.size(np.where(l
FP_ideal = (np.size(np.where((ideal_decision == 1) & (label == 0))))/np.size(np.where(l
minPerror_ideal = (priors[0] * FP_ideal) + (priors[1] * (1 - TP_ideal))
print("Gamma Ideal - %f and corresponding minimum error %f" %(np.exp(loggamma_ideal), m
```

```
Gamma Ideal - 1.857143 and corresponding minimum error 0.031358
```

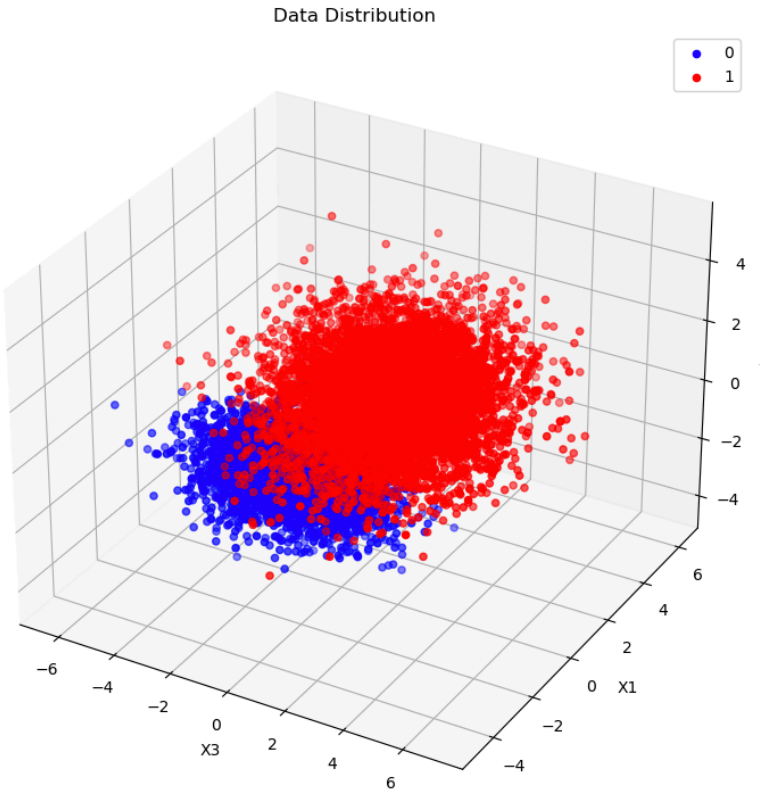
In [62]:
```
# Plot ROC curve
plt.plot(FP, TP, color = 'red')
plt.xlabel('False Positive')
plt.ylabel('True Positive')
plt.title('ROC Curve')
plt.plot(FP[np.argmin(minPerror)], TP[np.argmin(minPerror)],'o',color = 'black')
plt.show()
```



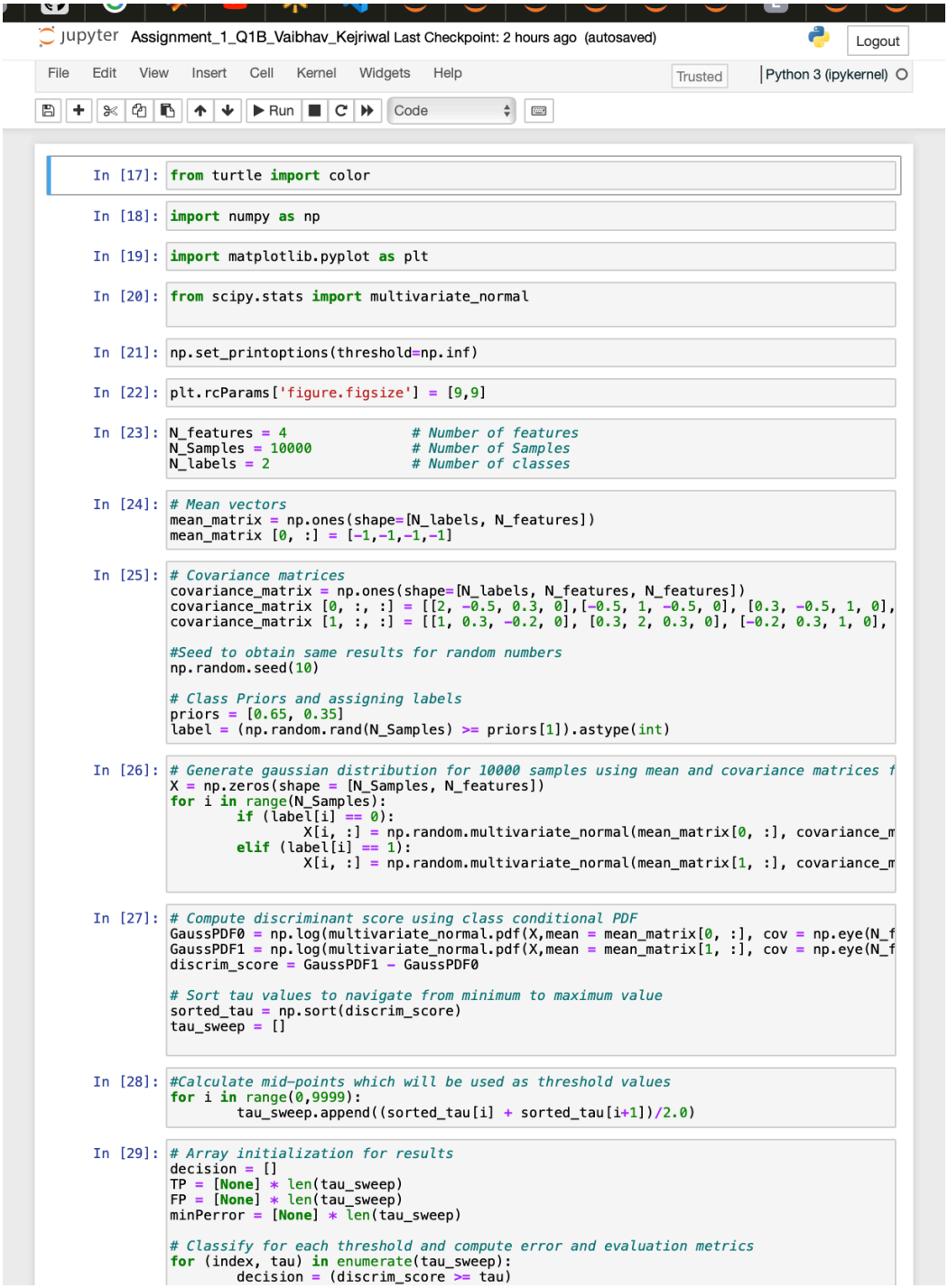
In [63]:
```
print("Gamma Practical - %f and corresponding minimum error %f" %(np.exp(tau_sweep[np.a
```

```
Gamma Practical - 1.800646 and corresponding minimum error 0.031138
```

In [64]:
```
#Plot Data Distribution
fig = plt.figure()
ax = plt.axes(projection = "3d")
Class0 = ax.scatter(X[(label==0),3],X[(label==0),1],X[(label==0),2],'+',color ='blue',
Class1 = ax.scatter(X[(label==1),3],X[label==1,1],X[label==1,2],'.',color = 'red', labe
plt.xlabel('X3')
plt.ylabel('X1')
ax.set_zlabel('X2')
ax.legend()
plt.title('Data Distribution')
plt.show()
```



Q1 B



jupyter Assignment_1_Q1B_Vaibhav_Kejriwal Last Checkpoint: 2 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted

Python 3 (ipykernel)

Run Code

In [17]:
```
from turtle import color
```

In [18]:
```
import numpy as np
```

In [19]:
```
import matplotlib.pyplot as plt
```

In [20]:
```
from scipy.stats import multivariate_normal
```

In [21]:
```
np.set_printoptions(threshold=np.inf)
```

In [22]:
```
plt.rcParams['figure.figsize'] = [9,9]
```

In [23]:
```
N_features = 4              # Number of features
N_Samples = 10000           # Number of Samples
N_labels = 2                # Number of classes
```

In [24]:
```
# Mean vectors
mean_matrix = np.ones(shape=[N_labels, N_features])
mean_matrix [0, :] = [-1,-1,-1,-1]
```

In [25]:
```
# Covariance matrices
covariance_matrix = np.ones(shape=[N_labels, N_features, N_features])
covariance_matrix [0, :, :] = [[2, -0.5, 0.3, 0],[-0.5, 1, -0.5, 0], [0.3, -0.5, 1, 0],
covariance_matrix [1, :, :] = [[1, 0.3, -0.2, 0], [0.3, 2, 0.3, 0], [-0.2, 0.3, 1, 0],

#Seed to obtain same results for random numbers
np.random.seed(10)

# Class Priors and assigning labels
priors = [0.65, 0.35]
label = (np.random.rand(N_Samples) >= priors[1]).astype(int)
```

In [26]:
```
# Generate gaussian distribution for 10000 samples using mean and covariance matrices f
X = np.zeros(shape = [N_Samples, N_features])
for i in range(N_Samples):
        if (label[i] == 0):
                X[i, :] = np.random.multivariate_normal(mean_matrix[0, :], covariance_m
        elif (label[i] == 1):
                X[i, :] = np.random.multivariate_normal(mean_matrix[1, :], covariance_m
```

In [27]:
```
# Compute discriminant score using class conditional PDF
GaussPDF0 = np.log(multivariate_normal.pdf(X,mean = mean_matrix[0, :], cov = np.eye(N_f
GaussPDF1 = np.log(multivariate_normal.pdf(X,mean = mean_matrix[1, :], cov = np.eye(N_f
discrim_score = GaussPDF1 - GaussPDF0

# Sort tau values to navigate from minimum to maximum value
sorted_tau = np.sort(discrim_score)
tau_sweep = []
```

In [28]:
```
#Calculate mid-points which will be used as threshold values
for i in range(0,9999):
        tau_sweep.append((sorted_tau[i] + sorted_tau[i+1])/2.0)
```

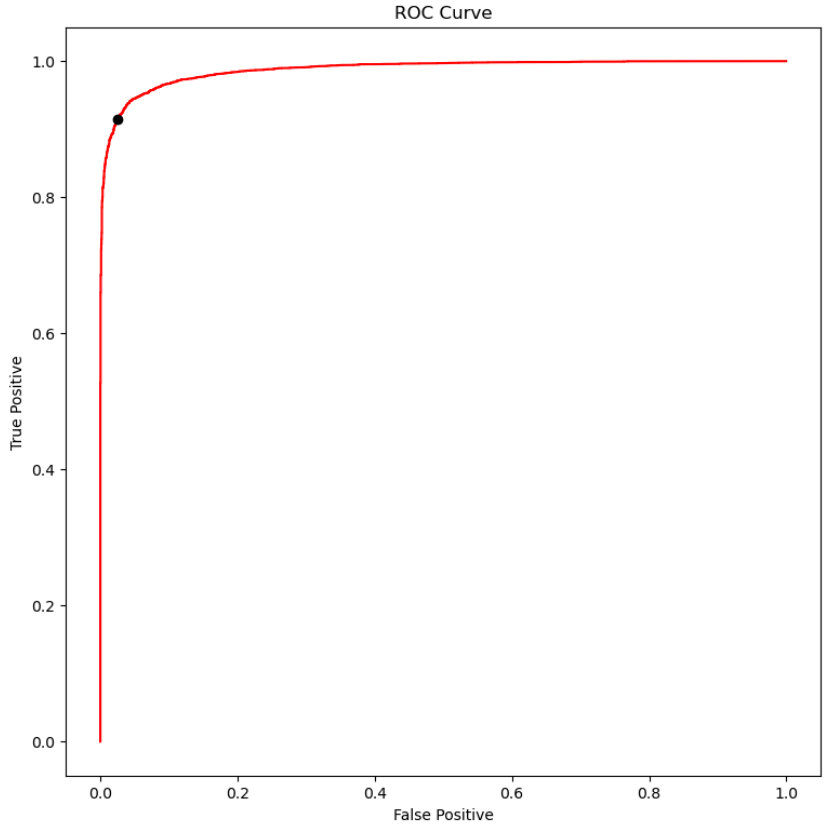
In [29]:
```
# Array initialization for results
decision = []
TP = [None] * len(tau_sweep)
FP = [None] * len(tau_sweep)
minPerror = [None] * len(tau_sweep)

# Classify for each threshold and compute error and evaluation metrics
for (index, tau) in enumerate(tau_sweep):
        decision = (discrim_score >= tau)
```

```
minPerror = [None] * len(tau_sweep)

# Classify for each threshold and compute error and evaluation metrics
for (index, tau) in enumerate(tau_sweep):
        decision = (discrim_score >= tau)
        TP[index] = (np.size(np.where((decision == 1) & (label == 1))))/np.size(np.wher
        FP[index] = (np.size(np.where((decision == 1) & (label == 0))))/np.size(np.wher
        minPerror[index] = (priors[0] * FP[index]) + (priors[1] * (1 - TP[index]))
```

In [30]:
```
# Theoretical classification based on class priors
loggamma_ideal = np.log(priors[0] / priors[1])
ideal_decision = (discrim_score >= loggamma_ideal)
TP_ideal = (np.size(np.where((ideal_decision == 1) & (label == 1))))/np.size(np.where(l
FP_ideal = (np.size(np.where((ideal_decision == 1) & (label == 0))))/np.size(np.where(l
minPerror_ideal = (priors[0] * FP_ideal) + (priors[1] * (1 - TP_ideal))
print("Gamma Ideal - %f and corresponding minimum error %f" %(np.exp(loggamma_ideal), m
```

```
Gamma Ideal - 1.857143 and corresponding minimum error 0.046865
```

In [31]:
```
# Plot ROC curve
plt.plot(FP, TP, color = 'red')
plt.xlabel('False Positive')
plt.ylabel('True Positive')
plt.title('ROC Curve')
plt.plot(FP[np.argmin(minPerror)], TP[np.argmin(minPerror)],'o',color = 'black')
plt.show()
```



In [33]:
```
print("Gamma Practical - %f and corresponding minimum error %f" %(np.exp(tau_sweep[np.a
```

```
Gamma Practical - 1.568752 and corresponding minimum error 0.045701
```

Q1 C

jupyter Assignment_1_Q1C_Vaibhav_Kejriwal Last Checkpoint: 2 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
from turtle import color
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
np.set_printoptions(threshold=np.inf)
plt.rcParams['figure.figsize'] = [9,9]
from numpy import linalg as LA
```

In [2]:
```
N_features = 4                  # Number of features
N_Samples = 10000               # Number of Samples
N_labels = 2                    # Number of classes

# Mean vectors
mean_matrix = np.ones(shape=[N_labels, N_features])
mean_matrix [0, :] = [-1,-1,-1,-1]

# Covariance matrices
covariance_matrix = np.ones(shape=[N_labels, N_features, N_features])
covariance_matrix [0, :, :] = [[2, -0.5, 0.3, 0],[-0.5, 1, -0.5, 0], [0.3, -0.5, 1, 0],
covariance_matrix [1, :, :] = [[1, 0.3, -0.2, 0], [0.3, 2, 0.3, 0], [-0.2, 0.3, 1, 0],
```

In [4]:
```
#Seed to obtain same results for random numbers
np.random.seed(10)

# Class Priors and assigning labels
priors = [0.65, 0.35]
label = (np.random.rand(N_Samples) >= priors[1]).astype(int)
```

In [5]:
```
# Generate gaussian distribution for 10000 samples using mean and covariance matrices f
X = np.zeros(shape = [N_Samples, N_features])
for i in range(N_Samples):
        if (label[i] == 0):
                X[i, :] = np.random.multivariate_normal(mean_matrix[0, :], covariance_m
        elif (label[i] == 1):
                X[i, :] = np.random.multivariate_normal(mean_matrix[1, :], covariance_m

# Compute between-class and within-class scatter matrices
Sb = (mean_matrix[0, :] - mean_matrix[1, :]) * np.transpose (mean_matrix[0, :] - mean_m
Sw = covariance_matrix[0, :, :] + covariance_matrix[1, :, :]

# Eigenvalues and eigen vectors of inverse(Sw.Sb)
V, W = LA.eig(LA.inv(Sw) * Sb)
```

In [6]:
```
# Eigen vector with maximizing optimization objective
W_LDA = W[np.argmax(V)]
X0 = X[np.where(label == 0)]
X1 = X[np.where(label == 1)]

# Data projection using wLDA
Y0 = np.zeros(len(X0))
Y1 = np.zeros(len(X1))
Y0 = np.dot(np.transpose(W_LDA), np.transpose(X0))
Y1 = np.dot(np.transpose(W_LDA), np.transpose(X1))

# Ranging threshold from minimum to maximum
Y = np.concatenate([Y0, Y1])
sort_Y = np.sort(Y)
tau_sweep = []
```

In [7]:
```
# Calculate mid-points which will be used as threshold values
for i in range(0,9999):
        tau_sweep.append((sort_Y[i] + sort_Y[i+1])/2.0)

# Array initialization for results
decision = []
TP = [None] * len(tau_sweep)
FP = [None] * len(tau_sweep)
minPerror = [None] * len(tau_sweep)
```

```
FP = [None] * len(tau_sweep)
minPerror = [None] * len(tau_sweep)

# Classify for each threshold and compute error and evaluation metrics
for (index, tau) in enumerate(tau_sweep):
        decision = (Y >= tau)
        TP[index] = (np.size(np.where((decision == 1) & (label == 1))))/np.size(np.wher
        FP[index] = (np.size(np.where((decision == 1) & (label == 0))))/np.size(np.wher
        minPerror[index] = (priors[0] * FP[index]) + (priors[1] * (1 - TP[index]))
```

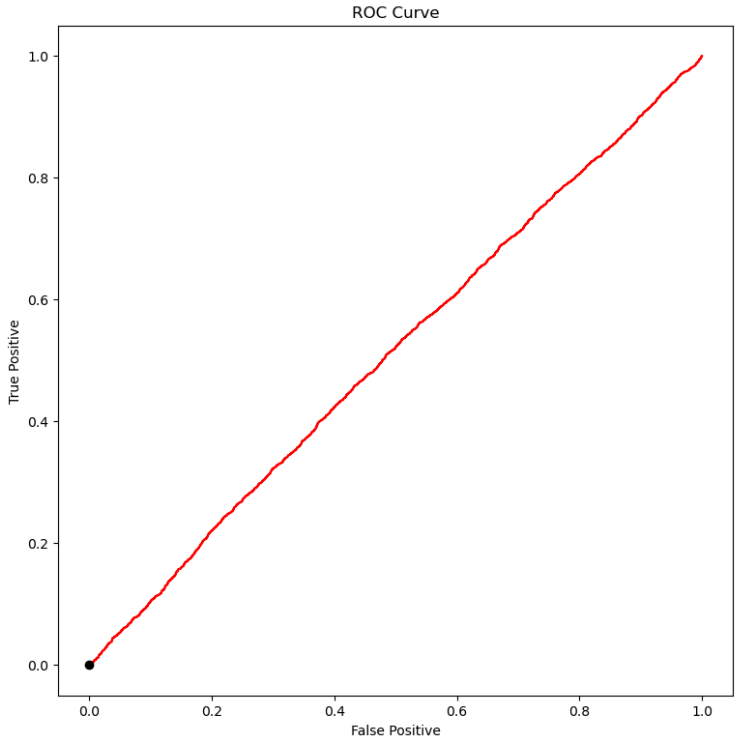
In [8]:

```
# Theoretical classification based on class priors
gamma_ideal = priors[0] / priors[1]
ideal_decision = (Y >= gamma_ideal)
TP_ideal = (np.size(np.where((ideal_decision == 1) & (label == 1))))/np.size(np.where(l
FP_ideal = (np.size(np.where((ideal_decision == 1) & (label == 0))))/np.size(np.where(l
minPerror_ideal = (priors[0] * FP_ideal) + (priors[1] * (1 - TP_ideal))
print("Tau Ideal - %f and corresponding minimum error %f" %(gamma_ideal, minPerror_idea
```

```
Tau Ideal - 1.857143 and corresponding minimum error 0.376001
```

In [9]:

```
# Plot ROC curve
plt.plot(FP, TP, color = 'red')
plt.xlabel('False Positive')
plt.ylabel('True Positive')
plt.title('ROC Curve')
plt.plot(FP[np.argmin(minPerror)], TP[np.argmin(minPerror)],'o',color = 'black')
plt.show()
```



In [10]:

```
print("Tau Practical - %f and corresponding minimum error %f"%(tau_sweep[np.argmin(minP
```

```
Tau Practical - 4.607466 and corresponding minimum error 0.350179
```

Q2

jupyter Q2_Assignmetn1_EECE5644_Vaibhav_Kejriwal Last Checkpoint: an hour ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [1]:
```
import random
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from mpl_toolkits.mplot3d import Axes3D
np.set_printoptions(threshold=np.inf)
```

In [2]:
```
N = 10000               # Number of Samples
N_features = 3          # Number of features
N_labels = 3            # Number of classes
N_mixtures = 4          # Number of gaussian distributions

#Seed to obtain same results for random numbers
np.random.seed(10)

# Class Priors
priors = np.array([[0.3, 0.3, 0.4]])

# Mean vectors
mean_matrix = np.zeros(shape=[N_mixtures, N_features])
mean_matrix [0, :] = [0, 0, 45]
mean_matrix [1, :] = [0, 45, 0]
mean_matrix [2, :] = [45, 0, 0]
mean_matrix [3, :] = [45, 0, 45]
```

In [6]:
```
# Covariance matrices
covariance_matrix = np.zeros(shape=[N_mixtures, N_features, N_features])
covariance_matrix[0, :, :] = 36 * np.linalg.matrix_power((np.eye(N_features)) + (0.01 *
covariance_matrix[1, :, :] = 36 * np.linalg.matrix_power((np.eye(N_features)) + (0.02 *
covariance_matrix[2, :, :] = 36 * np.linalg.matrix_power((np.eye(N_features)) + (0.03 *
covariance_matrix[3, :, :] = 36 * np.linalg.matrix_power((np.eye(N_features)) + (0.04 *
```

In [7]:
```
# Prior weights for gaussian components of class 2 and assigning labels
prior_gmm_label3 = [0.5,0.5]
cumsum = np.cumsum(priors)
randomlabels = np.random.rand(N)
label = np.zeros(shape = [10000])
for i in range(0,N-1):
    if randomlabels[i] <= cumsum[0]:
        label[i] = 0
    elif randomlabels[i] <= cumsum[1]:
        label[i] = 1
    else:
        label[i] = 2
```

In [8]:
```
# Generate gaussian distribution for 10000 samples using mean and covariance matrices f
X = np.zeros(shape = [N, N_features])
for i in range(N):
    if (label[i] == 0):
        X[i, :] = np.random.multivariate_normal(mean_matrix[0, :], covariance_matrix[0,
    elif (label[i] == 1):
        X[i, :] = np.random.multivariate_normal(mean_matrix[1, :], covariance_matrix[1,
    elif (label[i] == 2):
    # Split samples based on mixture weights
        if (np.random.rand(1,1) >= prior_gmm_label3[1]):
            X[i, :] = np.random.multivariate_normal(mean_matrix[2, :], covariance_matri
        else:
            X[i, :] = np.random.multivariate_normal(mean_matrix[3, :], covariance_matri
```

```
/tmp/ipykernel_36009/710216468.py:11: RuntimeWarning: covariance is not symmetric posi
tive-semidefinite.
  X[i, :] = np.random.multivariate_normal(mean_matrix[2, :], covariance_matrix[2, :,
:])
/tmp/ipykernel_36009/710216468.py:7: RuntimeWarning: covariance is not symmetric posit
ive-semidefinite.
  X[i, :] = np.random.multivariate_normal(mean_matrix[1, :], covariance_matrix[1, :,
:])
/tmp/ipykernel_36009/710216468.py:5: RuntimeWarning: covariance is not symmetric posit
ive-semidefinite.
```

jupyter Q2_Assignmetn1_EECE5644_Vaibhav_Kejriwal Last Checkpoint: an hour ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Not Trusted | Python 3 (ipykernel)

Run Code

:])

In [29]:
```
#Select appropriate loss matrix and comment other two
#loss_matrix = np.ones(shape = [N_labels, N_labels]) - np.eye(N_labels)
#loss_matrix = np.array([[0, 10, 10], [1, 0, 10], [1, 1, 0]])
loss_matrix = np.array([[0, 100, 100], [1, 0, 100], [1, 1, 0]])

print(loss_matrix)

# Compute Class conditional PDF
P_x_given_L = np.zeros(shape = [N_labels, N])
for i in range(N_labels):
    P_x_given_L[i, :] = multivariate_normal.pdf(X,mean = mean_matrix[i, :], cov = covar
```

```
[[  0 100 100]
 [  1   0 100]
 [  1   1   0]]
```

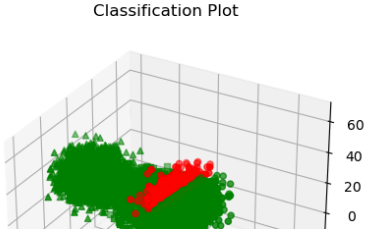
In [30]:
```
# Compute Class Posteriors using priors and class conditional PDF
P_x = np.matmul(priors, P_x_given_L)
ClassPosteriors = (P_x_given_L * (np.matlib.repmat(np.transpose(priors), 1, N))) / np.m

# Evaluate Expected risk and decisions based on minimum risk
ExpectedRisk = np.matmul(loss_matrix, ClassPosteriors)
Decision = np.argmin(ExpectedRisk, axis = 0)
print("Average Expected Risk", np.sum(np.min(ExpectedRisk, axis = 0)) / N)
```

```
Average Expected Risk 0.034364805418552934
```

In [31]:
```
# Estimate Confusion Matrix
ConfusionMatrix = np.zeros(shape = [N_labels, N_labels])

for d in range(N_labels):
    for l in range(N_labels):
        ConfusionMatrix[d, l] = (np.size(np.where((d == Decision) & (l == label)))) / n

print(ConfusionMatrix)
```

```
[[1.         0.         0.11649746]
 [0.         1.         0.        ]
 [0.         0.         0.88350254]]
```

In [32]:
```
# Plot Classification results
fig = plt.figure()
ax = plt.axes(projection = "3d")
ax.scatter(X[(label==2) & (Decision == 1),0],X[(label==2) & (Decision == 1),1],X[(label
ax.scatter(X[(label==2) & (Decision == 2),0],X[(label==2) & (Decision == 2),1],X[(label
ax.scatter(X[(label==2) & (Decision == 0),0],X[(label==2) & (Decision == 0),1],X[(label
ax.scatter(X[(label==1) & (Decision == 1),0],X[(label==1) & (Decision == 1),1],X[(label
ax.scatter(X[(label==1) & (Decision == 2),0],X[(label==1) & (Decision == 2),1],X[(label
ax.scatter(X[(label==1) & (Decision == 0),0],X[(label==1) & (Decision == 0),1],X[(label
ax.scatter(X[(label==0) & (Decision == 0),0],X[(label==0) & (Decision == 0),1],X[(label
ax.scatter(X[(label==0) & (Decision == 2),0],X[(label==0) & (Decision == 2),1],X[(label
ax.scatter(X[(label==0) & (Decision == 1),0],X[(label==0) & (Decision == 1),1],X[(label
plt.xlabel('X1')
plt.ylabel('X2')
ax.set_zlabel('X3')
plt.title('Classification Plot')
plt.show()
```



Q3 A

jupyter Assignment1_Q3A_EECE5644_Vaibhav_Kejriwal Last Checkpoint: 2 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Run Code

In [90]:

```
import random
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
from numpy import linalg as LA

# Import Dataset
df = pd.read_csv('/Users/vaibhavkejriwal/Desktop/Python/winequality-red.csv')
Data = df.to_numpy()

N = Data.shape[0]       # Number of Samples
label = Data[:, 11]     # Seperate column containing class labels
Data = Data[:, 0:11]    # Feature set
N_labels = 11           # Number of classes
N_features = 11         # Number of features
mean_matrix = np.zeros(shape = [N_labels, N_features])
covariance_matrix = np.zeros(shape = [N_labels, N_features, N_features])

# Compute Mean Vectors and Covariance matrices
for i in range(0, N_labels):
    mean_matrix[i, :] = np.mean(Data[(label == i)], axis = 0)
    # Identity covariance matrix for labels not in dataset
    if (i not in label):
        covariance_matrix[i, :, :] = np.eye(N_features)
    else:
        covariance_matrix[i, :, :] = np.cov(Data[(label == i), :], rowvar = False)
        covariance_matrix[i, :, :] += (0.000000005) * ((np.trace(covariance_matrix[i, :
    #Check if covariance matrices are ill-conditioned
    #print(LA.cond(covariance_matrix[i,:,:]))

# Assign 0-1 loss matrix
loss_matrix = np.ones(shape = [N_labels, N_labels]) - np.eye(N_labels)

# Compute class conditional PDF
P_x_given_L = np.zeros(shape = [N_labels, N])
for i in range(0, N_labels):
    if i in label:
        P_x_given_L[i, :] = multivariate_normal.pdf(Data, mean = mean_matrix[i, :], cov

# Estimate class priors based on sample count
priors = np.zeros(shape = [11, 1])
for i in range(0, N_labels):
    priors[i] = (np.size(label[np.where((label == i))])) / N

# Compute Class Posteriors using priors and class conditional PDF
P_x = np.matmul(np.transpose(priors), P_x_given_L)
ClassPosteriors = (P_x_given_L * (np.matlib.repmat(priors, 1, N))) / np.matlib.repmat(P

# Evaluate Expected risk and decisions based on minimum risk
ExpectedRisk = np.matmul(loss_matrix, ClassPosteriors)
Decision = np.argmin(ExpectedRisk, axis = 0)
print("Average Expected Risk", np.sum(np.min(ExpectedRisk, axis = 0)) / N)

# Estimate Confusion Matrix
ConfusionMatrix = np.zeros(shape = [N_labels, N_labels])
for d in range(N_labels):
    for l in range(N_labels):
        if l in label and d in label:
            ConfusionMatrix[d, l] = (np.size(np.where((d == Decision) & (l == label))))

print(ConfusionMatrix)

# Plot Data Distribution
fig = plt.figure()
ax = plt.axes(projection = "3d")
for i in range(N_labels):
    ax.scatter(Data[(label==i),1],Data[(label==i),2],Data[(label==i),3], label=i)
plt.xlabel('X3')
plt.ylabel('X1')
ax.set_zlabel('X2')
ax.legend()
```

jupyter Assignment1_Q3A_EECE5644_Vaibhav_Kejriwal Last Checkpoint: 2 hours ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

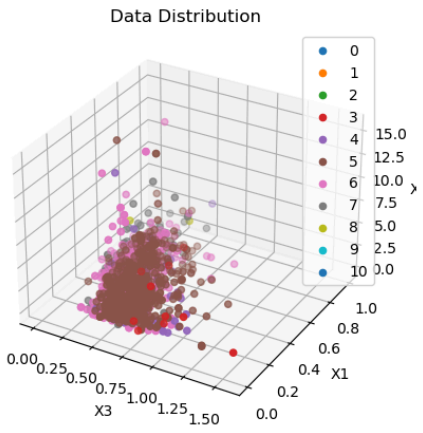
Not Trusted | Python 3 (ipykernel)

Run Code

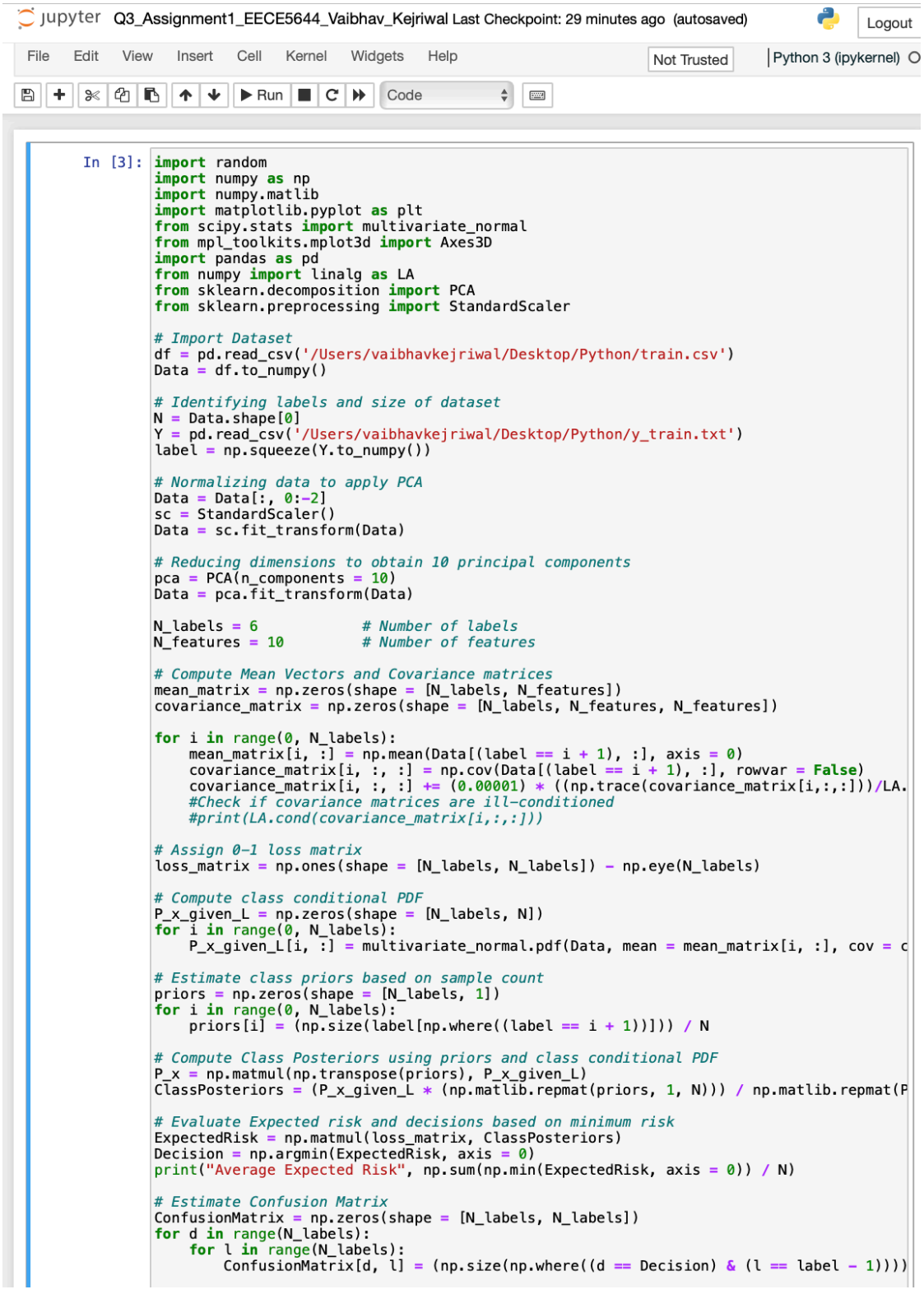
```
            ConfusionMatrix[d, l] = (np.size(np.where((d == Decision) & (l == label))))

print(ConfusionMatrix)

# Plot Data Distribution
fig = plt.figure()
ax = plt.axes(projection = "3d")
for i in range(N_labels):
    ax.scatter(Data[(label==i),1],Data[(label==i),2],Data[(label==i),3], label=i)
plt.xlabel('X3')
plt.ylabel('X1')
ax.set_zlabel('X2')
ax.legend()
plt.title('Data Distribution')
plt.show()
```

```
/Users/vaibhavkejriwal/anaconda3/lib/python3.11/site-packages/numpy/core/fromnumeric.py:3464: RuntimeWarning: Mean of empty slice.
  return _methods._mean(a, axis=axis, dtype=dtype,
/Users/vaibhavkejriwal/anaconda3/lib/python3.11/site-packages/numpy/core/_methods.py:184: RuntimeWarning: invalid value encountered in divide
  ret = um.true_divide(
```

```
Average Expected Risk 0.3244031178791402
[[0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         1.         0.         0.        ]
 [0.         0.         0.         0.         0.18867925 0.01908957]
 [0.01567398 0.00502513 0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.43396226 0.6328928 ]
 [0.22100313 0.02512563 0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.33962264 0.32599119]
 [0.64733542 0.44723618 0.16666667 0.         0.         0.        ]
 [0.         0.         0.         0.         0.03773585 0.02202643]
 [0.09717868 0.45226131 0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.        ]
 [0.01880878 0.07035176 0.83333333 0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.        ]]
```



Q3 B


jupyter Q3_Assignment1_EECE5644_Vaibhav_Kejriwal Last Checkpoint: 29 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3 (ipykernel)
Run
Code
In [3]:
import random
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
from mpl_toolkits.mplot3d import Axes3D
import pandas as pd
from numpy import linalg as LA
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
# Import Dataset
df = pd.read_csv('/Users/vaibhavkejriwal/Desktop/Python/train.csv')
Data = df.to_numpy()
# Identifying labels and size of dataset
N = Data.shape[0]
Y = pd.read_csv('/Users/vaibhavkejriwal/Desktop/Python/y_train.txt')
label = np.squeeze(Y.to_numpy())
# Normalizing data to apply PCA
Data = Data[:, 0:-2]
sc = StandardScaler()
Data = sc.fit_transform(Data)
# Reducing dimensions to obtain 10 principal components
pca = PCA(n_components = 10)
Data = pca.fit_transform(Data)
N_labels = 6 # Number of labels
N_features = 10 # Number of features
# Compute Mean Vectors and Covariance matrices
mean_matrix = np.zeros(shape = [N_labels, N_features])
covariance_matrix = np.zeros(shape = [N_labels, N_features, N_features])
for i in range(0, N_labels):
mean_matrix[i, :] = np.mean(Data[(label == i + 1), :], axis = 0)
covariance_matrix[i, :, :] = np.cov(Data[(label == i + 1), :], rowvar = False)
covariance_matrix[i, :, :] += (0.00001) * ((np.trace(covariance_matrix[i,:,:]))/LA.
#Check if covariance matrices are ill-conditioned
#print(LA.cond(covariance_matrix[i,:,:]))
# Assign 0-1 loss matrix
loss_matrix = np.ones(shape = [N_labels, N_labels]) - np.eye(N_labels)
# Compute class conditional PDF
P_x_given_L = np.zeros(shape = [N_labels, N])
for i in range(0, N_labels):
P_x_given_L[i, :] = multivariate_normal.pdf(Data, mean = mean_matrix[i, :], cov = c
# Estimate class priors based on sample count
priors = np.zeros(shape = [N_labels, 1])
for i in range(0, N_labels):
priors[i] = (np.size(label[np.where((label == i + 1))])) / N
# Compute Class Posteriors using priors and class conditional PDF
P_x = np.matmul(np.transpose(priors), P_x_given_L)
ClassPosteriors = (P_x_given_L * (np.matlib.repmat(priors, 1, N))) / np.matlib.repmat(P
# Evaluate Expected risk and decisions based on minimum risk
ExpectedRisk = np.matmul(loss_matrix, ClassPosteriors)
Decision = np.argmin(ExpectedRisk, axis = 0)
print("Average Expected Risk", np.sum(np.min(ExpectedRisk, axis = 0)) / N)
# Estimate Confusion Matrix
ConfusionMatrix = np.zeros(shape = [N_labels, N_labels])
for d in range(N_labels):
for l in range(N_labels):
ConfusionMatrix[d, l] = (np.size(np.where((d == Decision) & (l == label - 1))))

```
np.set_printoptions(suppress=True)
print(ConfusionMatrix)

# Plot Data Distribution
fig = plt.figure()
ax = plt.axes(projection = "3d")
for i in range(1, N_labels + 1):
    ax.scatter(Data[(label==i),1],Data[(label==i),2],Data[(label==i),3], label=i)
plt.xlabel('X3')
plt.ylabel('X1')
ax.set_zlabel('X2')
ax.legend()
plt.title('Data Distribution')
plt.show()
```

```
Average Expected Risk 0.09130586093998115
[[0.91272431 0.02329916 0.02839757 0.         0.0014556  0.        ]
 [0.05546493 0.91985089 0.12778905 0.00155521 0.         0.        ]
 [0.03181077 0.05684995 0.84381339 0.         0.         0.00071073]
 [0.         0.         0.         0.60186625 0.08151383 0.03411514]
 [0.         0.         0.         0.35692068 0.91411936 0.        ]
 [0.         0.         0.         0.03965785 0.00291121 0.96517413]]
```