EECE 5644: Machine Learning Project 1 Report

1. Given the following data:

$$p(\mathbf{x}) = p(\mathbf{x}|L=0)p(L=0) + p(\mathbf{x}|L=1)p(L=1)$$
$$p(L=0) = 0.7$$
$$p(L=1) = 0.3$$

$$p(\mathbf{x}|L=0) = g(\mathbf{x}|m_0, C_0)$$
$$p(\mathbf{x}|L=1) = g(\mathbf{x}c_1, C_1)$$

class conditional parameters:

$$m_0 = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} C_0 = \begin{bmatrix} 2 & -0.5 & 0.3 & 0 \\ -0.5 & 1 & -0.5 & 0 \\ 0.3 & -0.5 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} m_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} C_1 = \begin{bmatrix} 1 & 0.3 & -0.2 & 0 \\ 0.3 & 2 & 0.3 & 0 \\ -0.2 & 0.3 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

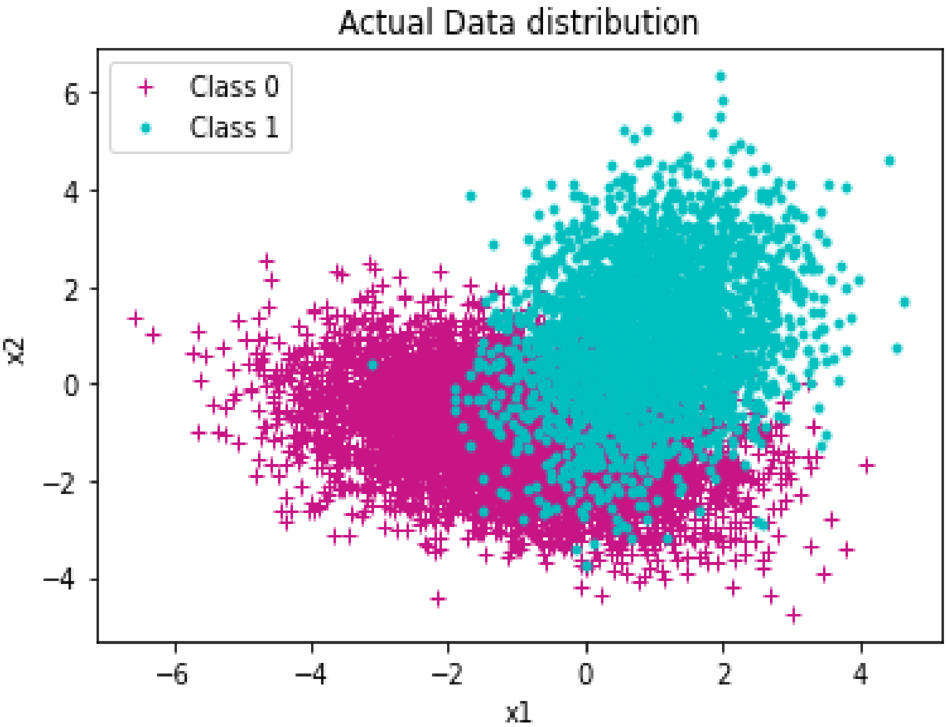

Figure 1: True data distribution

A ERM classification using the knowledge of true data pdf (Bayes Classifier)

i Minimum expected risk classification rule
Likelihood ratio test:

$$\frac{p(x|L=1)}{p(x|L=0} \underset{L=0}{\overset{L=1}{\gtrless}} \frac{p(L=0)}{p(L=1} * \frac{(\lambda_{10}-\lambda_{00})}{(\lambda_{01}-\lambda_{11})} = \gamma$$

For $0-1$ loss classification, the above reduces to:

$$\frac{p(x|L=1)}{p(x|L=0} \underset{L=0}{\overset{L=1}{\gtrless}} = \frac{0.7}{0.3} * \frac{(1-0)}{(1-0)} = \frac{7}{3}$$

In the above formula, $\gamma = 2.33$ is the theoretical threshold. Classify $L = 1$ if

$$\frac{p(x|L=1)}{p(x|L=0} > 2.33$$

and classify $L = 0$ otherwise.

ii ROC curve
The ERM classifier is implemented on the 10K samples and the ROC curve using true positive and false positive rates are plotted below.

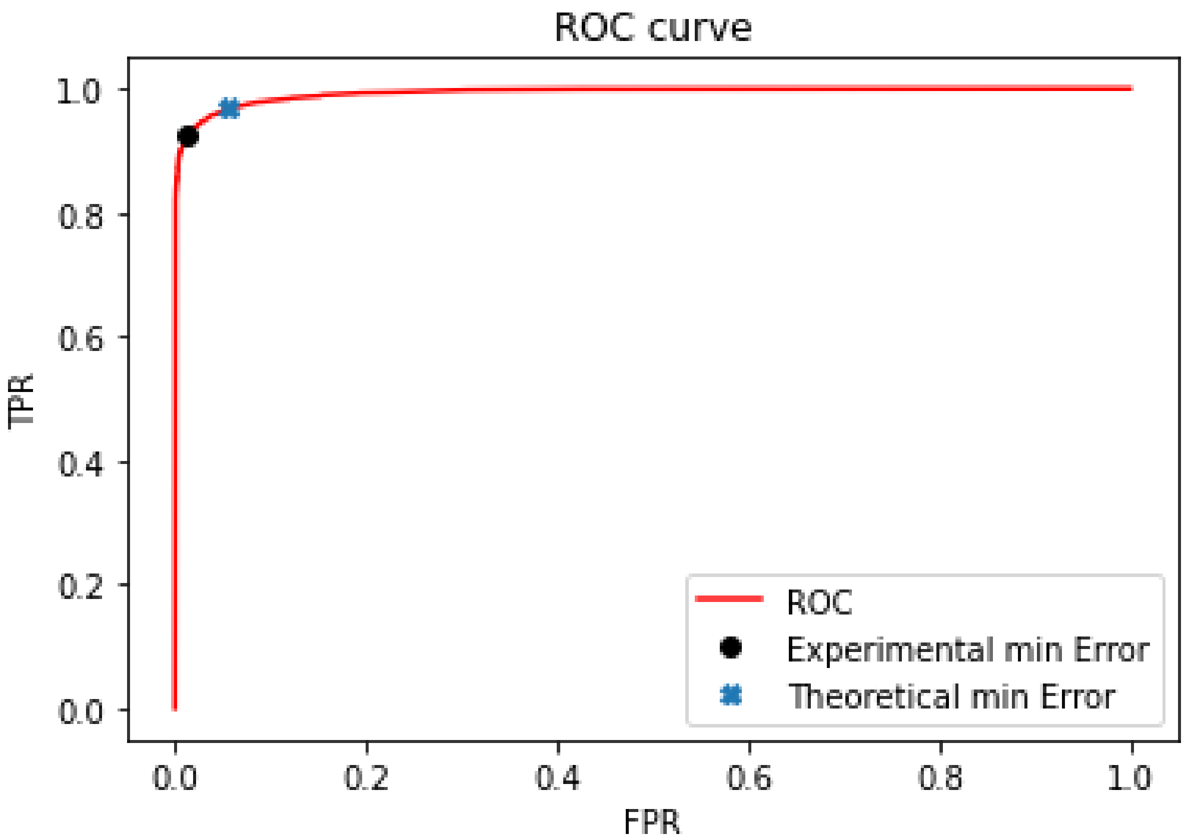

Figure 2: ROC curve with minimum probability error

In the above figure, the value of TP rate and FP rate at the optimal threshold (where the minimum error probability occur) is 0.9287 *and* 0.011

respectively. The minimum probability of error was calculated to be $P_e = 0.0294$.

Since the true data distributions has a little overlapping due to their different means, the classifier is predicting the actual classes in most cases.

types	γ	minimum probability error
Theoretical	2.333	0.0482
Experimental	0.981	0.0294

B Naive Bayesian Classifier

ERM classification using incorrect knowledge of data distribution, which assumes independent features.

i Minimum expected risk classification rule

The classification rule is the same as with part A. That is, the likelihood ratio will be:

$$\frac{p(x|L=1)}{p(x|L=0} \underset{L=0}{\overset{L=1}{\gtrless}} \frac{p(L=0)}{p(L=1} * \frac{(\lambda_{10}-\lambda_{00})}{(\lambda_{01}-\lambda_{11})} = 2.333$$

ii ROC curve:

The ROC curve for this classifier is plotted below.

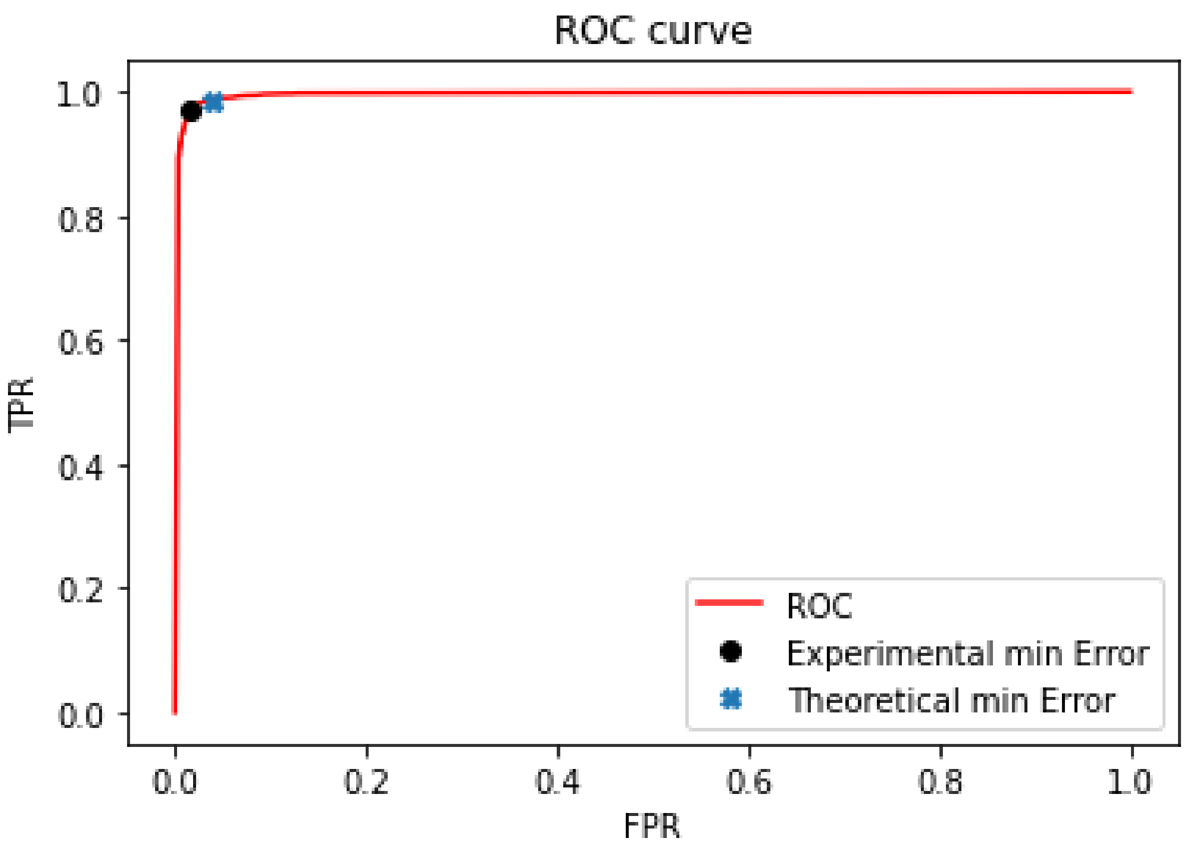

Figure 3: ROC curve with minimum probability error

In the above figure, the value of TP rate and FP rate at the optimal threshold (where the minimum error probability occur) is 0.9732 *and* 0.0159 respectively. The minimum probability of error was calculated to be $P_e = 0.0192$. Since the values of covariance matrix used (identity matrix) did not vary by much from the one we used above, the results are close to the previous classifier.

types	γ	minimum probability error
Theoretical	2.333	0.0297
Experimental	0.4712	0.0192

C LDA based classifier:
Increasing the distance between the means of the pdfs and reducing the variances of each of the pdfs helps in classifying these two distributions. In this implementation, class conditional mean and variance were used from the data samples provided.

i FLDA classification rule:
The classification rule applied on LDA is given below:

$$w_{LDA}^T x \underset{L=0}{\overset{L=1}{\gtrless}} \tau = 2.33$$

ii The projected data

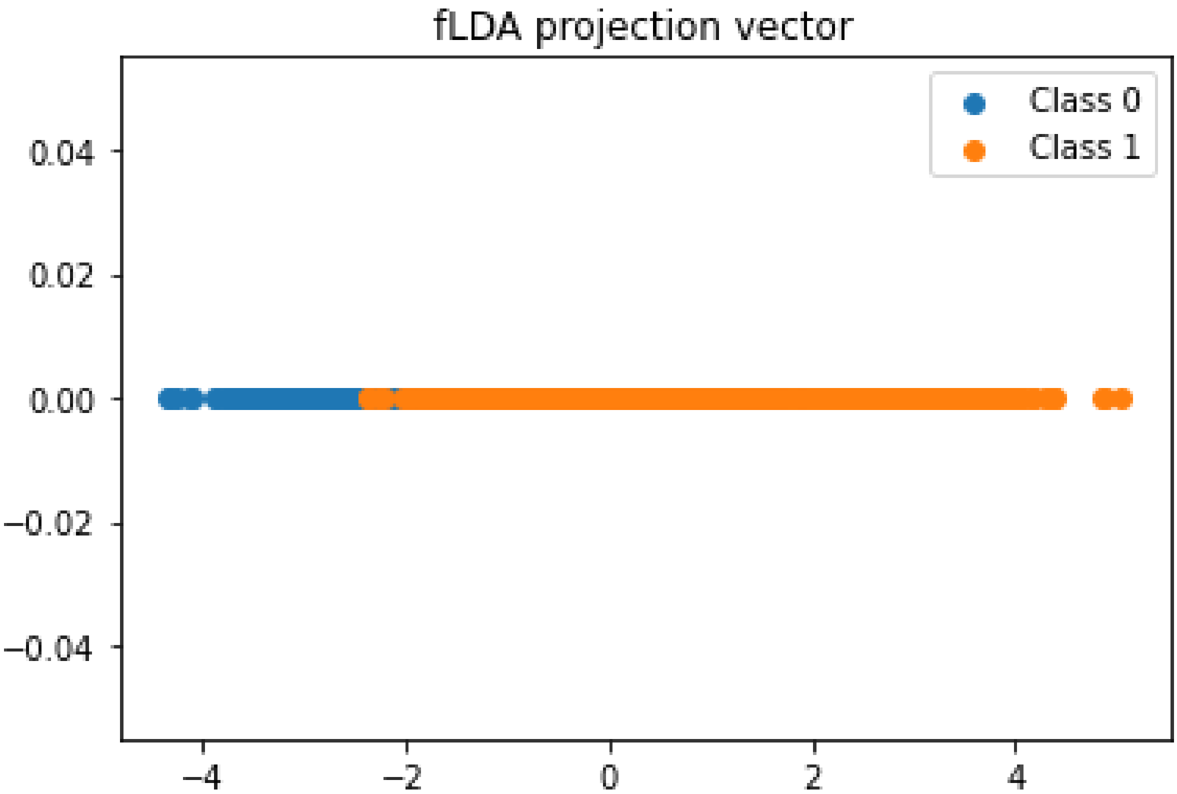

Figure 4: Projected data using LDA classifier

iii ROC curve

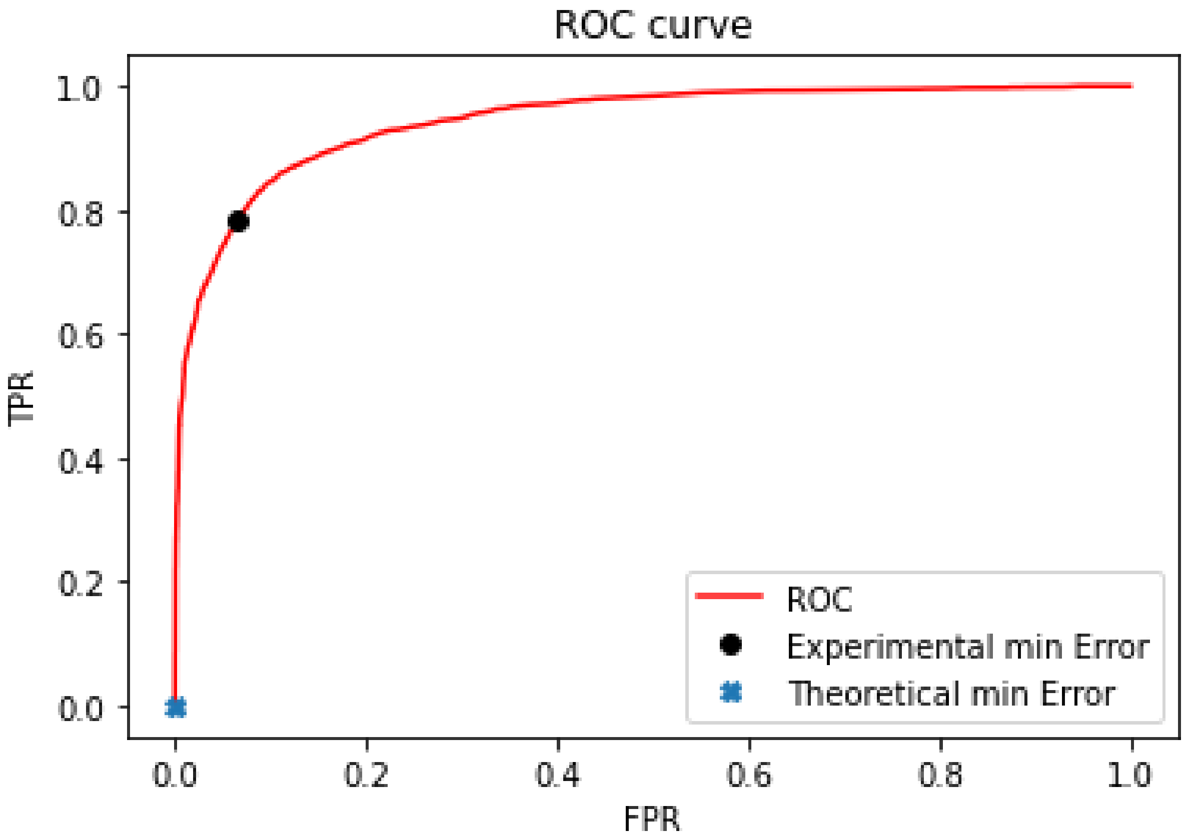

Figure 5: ROC curve with minimum probability error for LDA classifier

types	τ	minimum probability error
Theoretical	2.333	0.0278
Experimental	0.257	0.1095

The probability of error for this classifier is higher than the ERM classifiers above, because there's an overlap of the projected distributions.

2. For this question, I selected the mean and covariance matrices using the formula provided by Prof.Deniz. The following values were used to generate the parameters.

$$C = S^2 * I \ \ , \ C = S^2(I + aA)(I + aA)$$

, used to generate the covariance matrices. Where $s = 0.1 * E$,$E = 7$is the edge length of a cube,$a = 0.07$ and $A = randn(3, 3)$

$$m_1 = \begin{bmatrix}-1\\-1\\-1\end{bmatrix} m_2 = \begin{bmatrix}-1\\1\\-1\end{bmatrix} m_3 = \begin{bmatrix}1\\1\\-1\end{bmatrix} m_4 = \begin{bmatrix}1\\-1\\-1\end{bmatrix}$$

$$C_1 = \begin{bmatrix}0.45 & -0.04 & -0.07\\-0.02 & 0.51 & -0.01\\-0.07 & 0.09 & 0.47\end{bmatrix} C_2 = \begin{bmatrix}0.5 & 0 & 0\\0 & 0.5 & 0\\0 & 0 & 0.5\end{bmatrix} C_3 = \begin{bmatrix}0.5 & 0 & 0\\0 & 0.5 & 0\\0 & 0 & 0.5\end{bmatrix} C_4 = \begin{bmatrix}0.5 & 0 & 0\\0 & 0.5 & 0\\0 & 0 & 0.5\end{bmatrix}$$

Priors:

$$p(L = 1) = 0.3p(L = 2) = 0.3p(L = 3) = 0.4$$

The third Gaussian distribution is selected from the mixture of the 3^{rd} and4^{th} matrices with equal probability.

A MAP classifier:
The cost matrix is:

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

The classification rule for this classifier is:

$$\alpha^* = argmin_{\alpha_i} R(\alpha_i|x)$$
$$R(\alpha_i|x) = \Sigma_{j=1}^{3} \lambda(\alpha_i|l_j)p(l_j|x)$$

where, α^* is the optimal action or decision made and R is the risk associated with choosing action i given that the true label is j.

i True Data Visualization:

Figure 6: True data distribution

ii Confusion matrix(row is the predicted class and column indicates the true class):

$$\begin{bmatrix} 2507 & 200 & 123 \\ 226 & 2655 & 998 \\ 264 & 42 & 2985 \end{bmatrix}, Normalized\ \ to: \begin{bmatrix} 0.836 & 0.07 & 0.03 \\ 0.075 & 0.91 & 0.24 \\ 0.088 & 0.02 & 0.73 \end{bmatrix}$$

From the confusion matrix, we can see that the classifier is predicting accurately most of the time. Compared to the other labels, class 3 is less accurately predicted because of its overlap with class 2 in the true data distribution.

iii Predicted data plots:

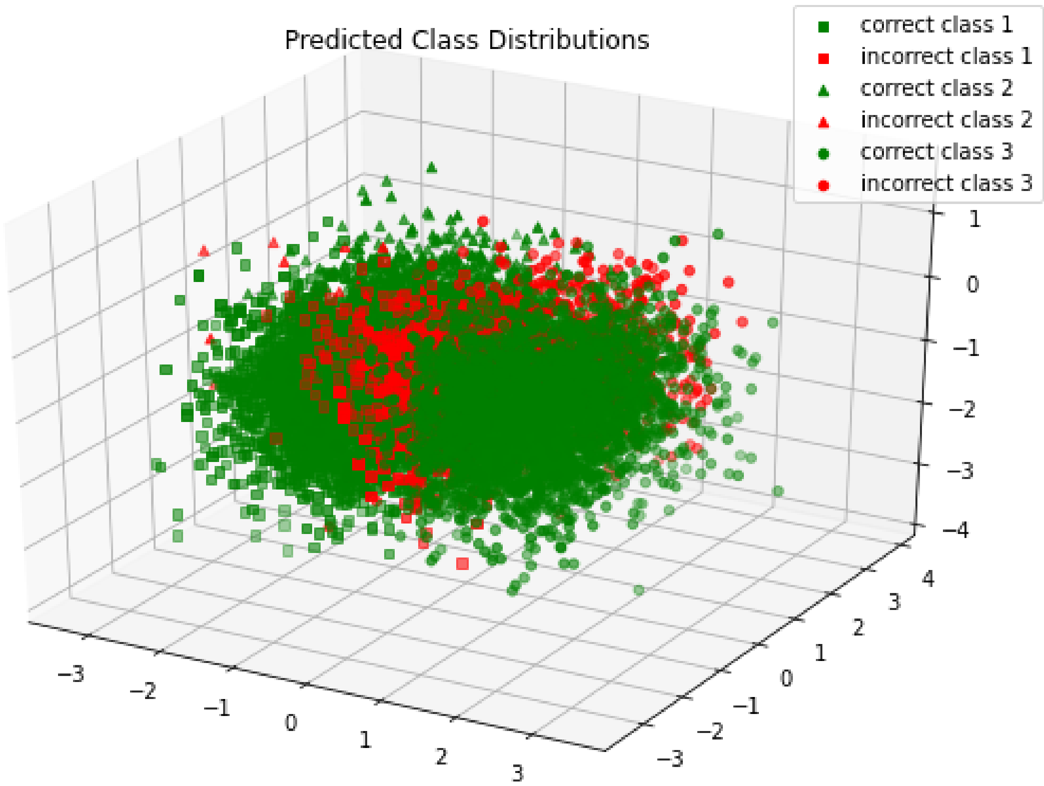

Figure 7: Predicted class results

As we can see from the plots, the classifier is predicting the true classes most of the time. This is due to the fact that the selected covariance for all the distributions is small and the probability of the data points being classified as their true labels is high in this case.

B i The cost matrix used:

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

- Confusion Matrix:

$$\begin{bmatrix} 0.66 & 0.054 & 0.004 \\ 0.06 & 0.88 & 0.13 \\ 0.28 & 0.062 & 0.87 \end{bmatrix}$$

- Data Visualizations:

The red dots in the above plots shows data points misclassfied as class 3, and their number decreases as compared to the previous plot. This is due to the increase in the value of the cost matrix for class 3.

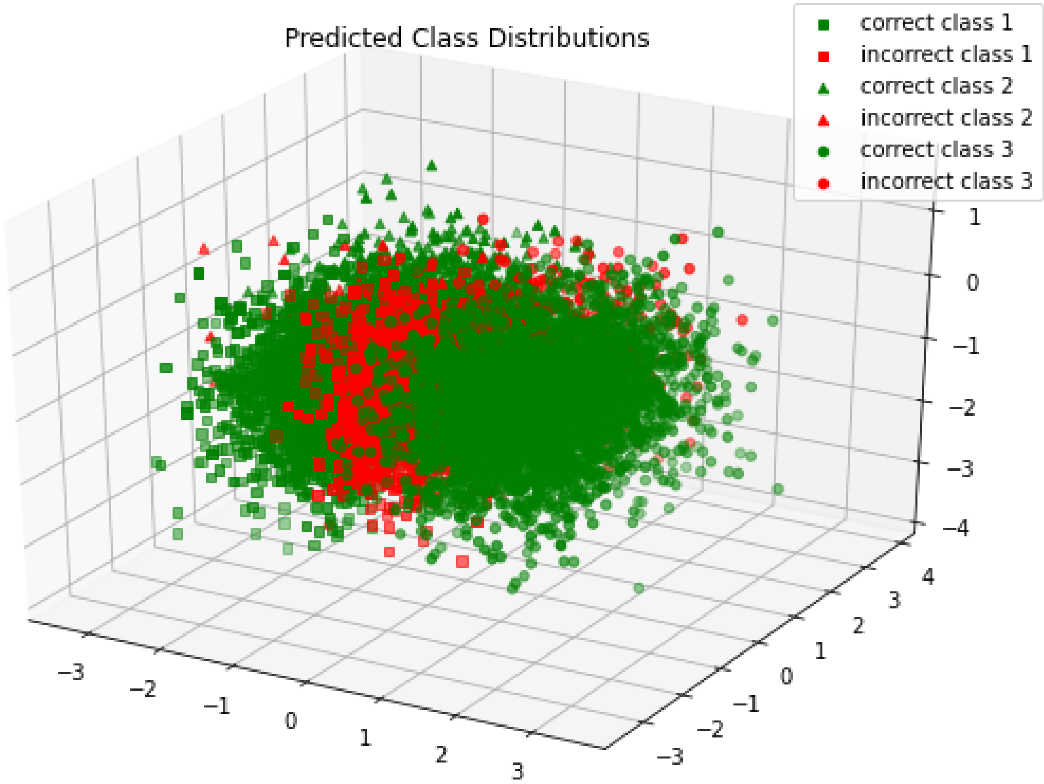

Figure 8: Predicted class results

ii The cost matrix used:

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

- Confusion Matrix:

$$\begin{bmatrix} 0.034 & 0.03 & 0 \\ 0.05 & 0.78 & 0.05 \\ 0.61 & 0.18 & 0.95 \end{bmatrix}$$

- Data Visualizations:

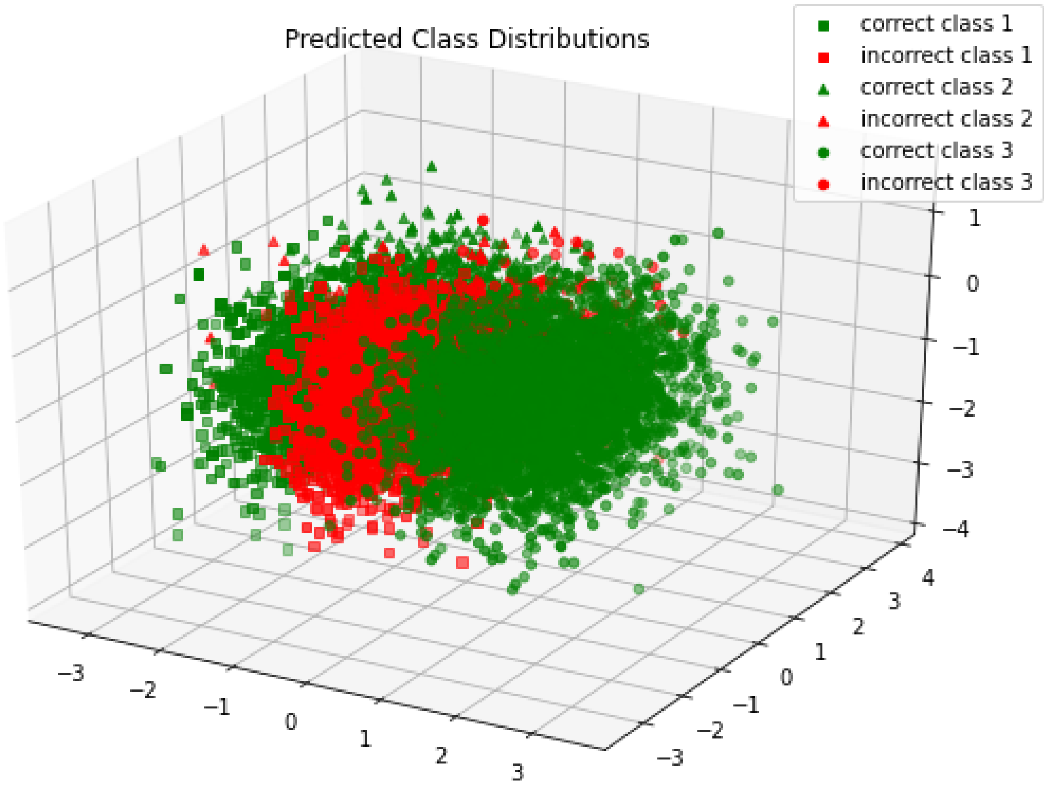

Figure 9: Predicted class results

iii Conclusion :
In conclusion, as the cost of selecting wrong classes for true class 3 increases (by 10 and 100) , we can see from the plots that there is a decrease in the number of incorrect prediction for class 3. Also, we can tell this information from the confusion matrix, the number of correctly classified labels increases for class 3.

3. A Wine Dataset: This dataset has 11 features and labels, and 4898 total samples. For those labels which has not samples, I estimate the mean to zero and the covariance matrices to identity matrix. The priors for this dataset were calculated to be:

$$p = [0, 0, 0, 0.0041, 0.0333, 0.2975, 0.4488, 0.1796, 0.0357, 0.001, 0]$$

To avoid the ill-conditioned covariance,I use λ =0.1 and the cost matrix used for this classification is 0-1 loss.

i True Class Distribution:
I applied PCA on the the datasets by reducing the dimensions to 2.

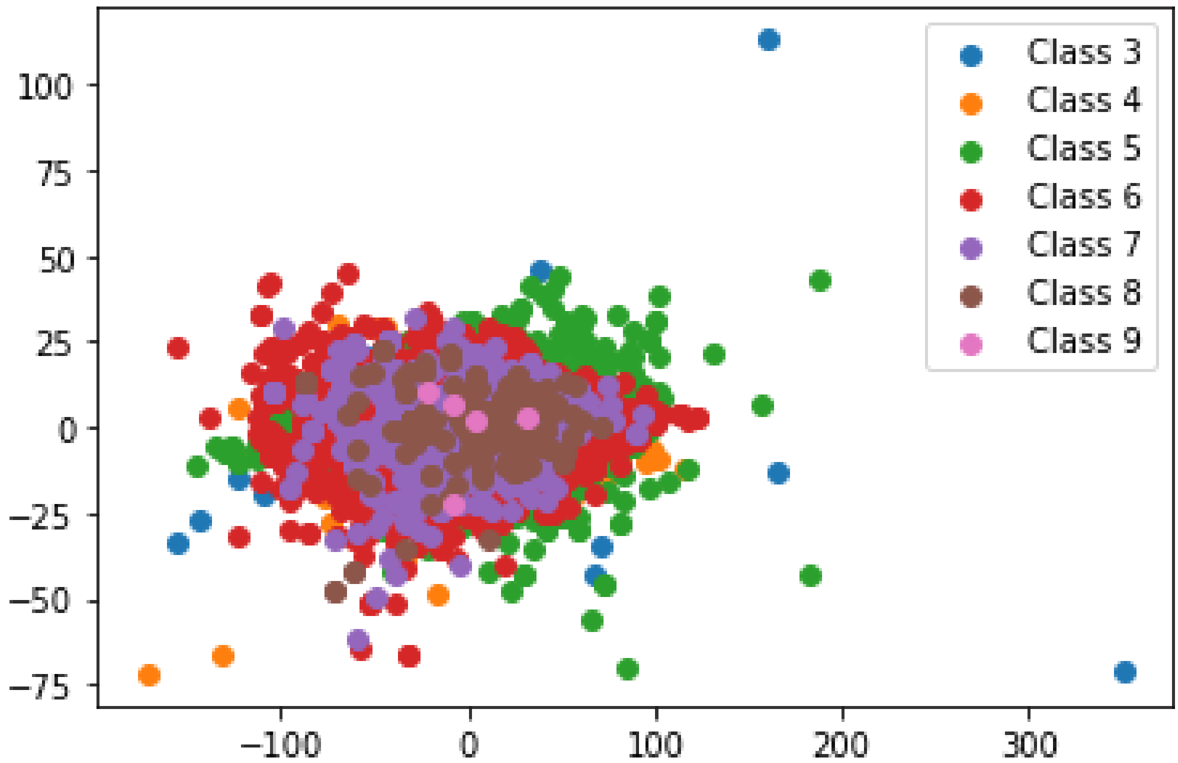

Figure 10: True data distribution

From the true distribution, class 6 has the highest distributions and class 0,1,2 and 10 has no samples.

ii Confusion Matrix:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \color{red}{0.35} & 0.012 & 0.0034 & 0.005 & 0.002 & 0.02 & 0 & 0 \\ 0 & 0 & 0 & 0.05 & \color{red}{0.018} & 0.005 & 0.003 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.25 & 0.65 & \color{red}{0.637} & 0.28 & 0.12 & 0.09 & 0 & 0 \\ 0 & 0 & 0 & 0.35 & 0.31 & 0.34 & \color{red}{0.61} & 0.535 & 0.451 & 0.4 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0.013 & 0.13 & \color{red}{0.343} & 0.42 & 0.6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.001 & 0 & 0.0034 & \color{red}{0.017} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The red values indicates the true positive rate for that specific true class, class 5 and 6 are better predicted than the other labels (which has smaller true positive rate).

iii Predicted Class Distributions

The 2D projected visualization of the predicted class is shown below:

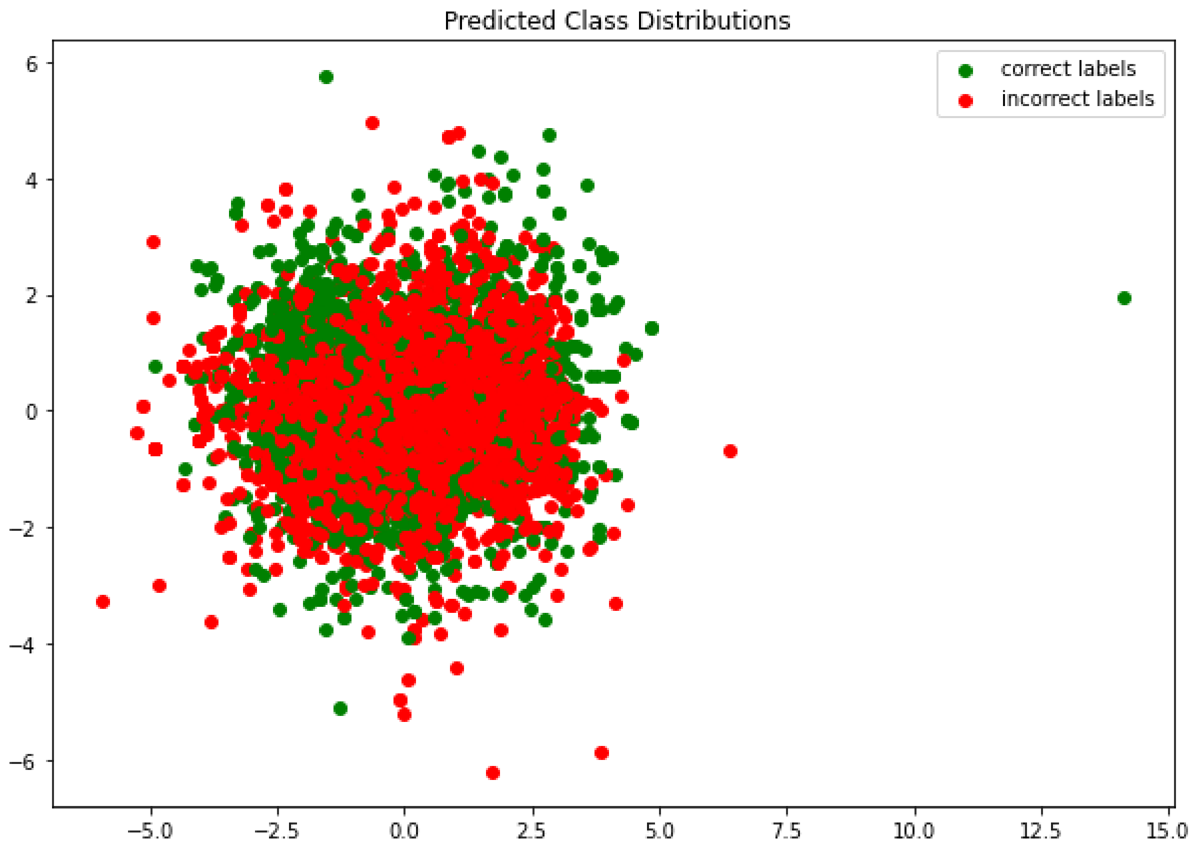

Figure 11: Predicted class distribution

From the figure, the red shows incorrect labels by the classifier which implies the gaussian distribution we initially assume is wrong for this dataset.

B HAR Dataset:

I use the training HAR dataset for the training and later check the classifier with both train and test data. To avoid the ill-conditoined covariance matrices, I use λ=0.01.

The priors for this dataset (training data) we calculated to be:

$$p = [0.167, 0.146, 0.134, 0.175, 0.187, 0.192]$$

i True Class Distribution:

I applied PCA on the the training HAR datasets by reducing the dimensions to 2.

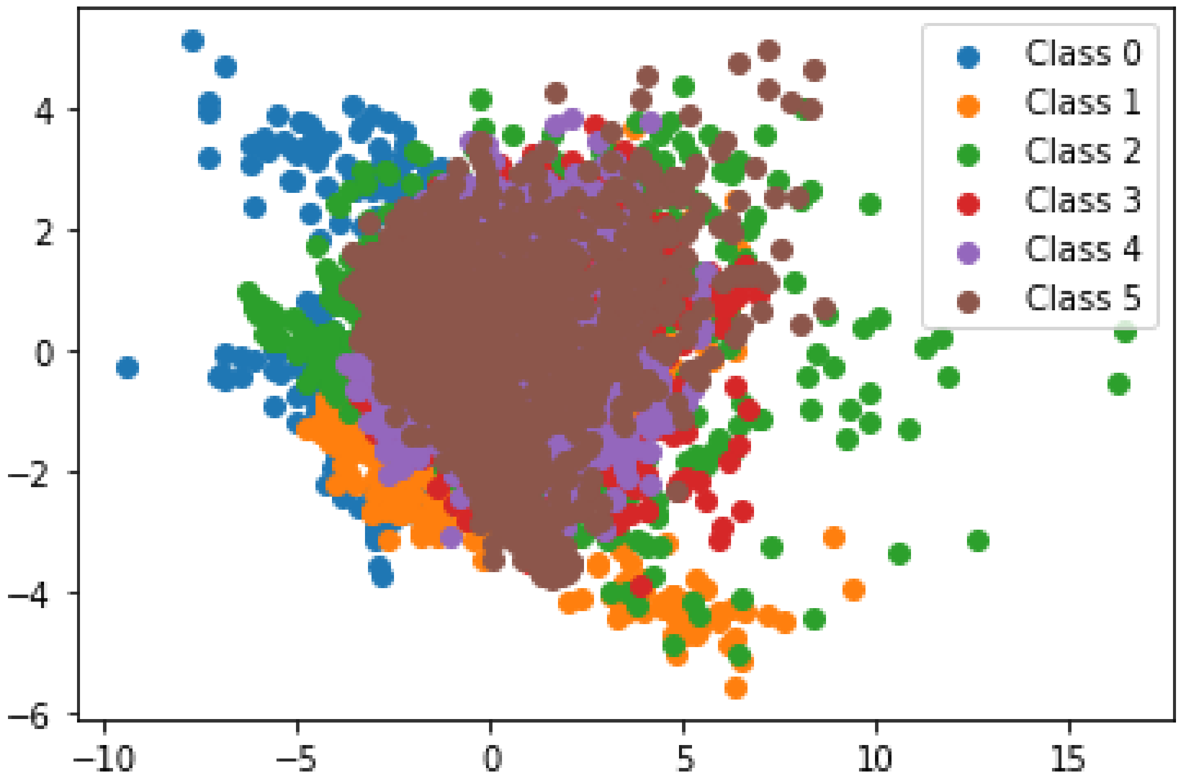

Figure 12: True class data distribution

ii Confusion Matrix:

Confusion matrix of train data

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.93 & 0 & 0 \\ 0 & 0 & 0 & 0.07 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Confusion matrix of test data:

$$\begin{bmatrix} 0.97 & 0 & 0.01 & 0 & 0 & 0 \\ 0 & 1 & 0.1 & 0 & 0 & 0 \\ 0.03 & 0 & 0.89 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.79 & 0.01 & 0 \\ 0 & 0 & 0 & 0.21 & 0.99 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The left matrix is the confusion matrix of the train data and as expected, it accurately predicts the true label in almost all cases because it's the same data that train the classifier. The right matrix is the confusion matrix of a test data applied to a trained classifier, here the classifier is also good on the newly unseen data.

iii Predicted Class Distributions

The following plots are the predicted class distributions for the test data.

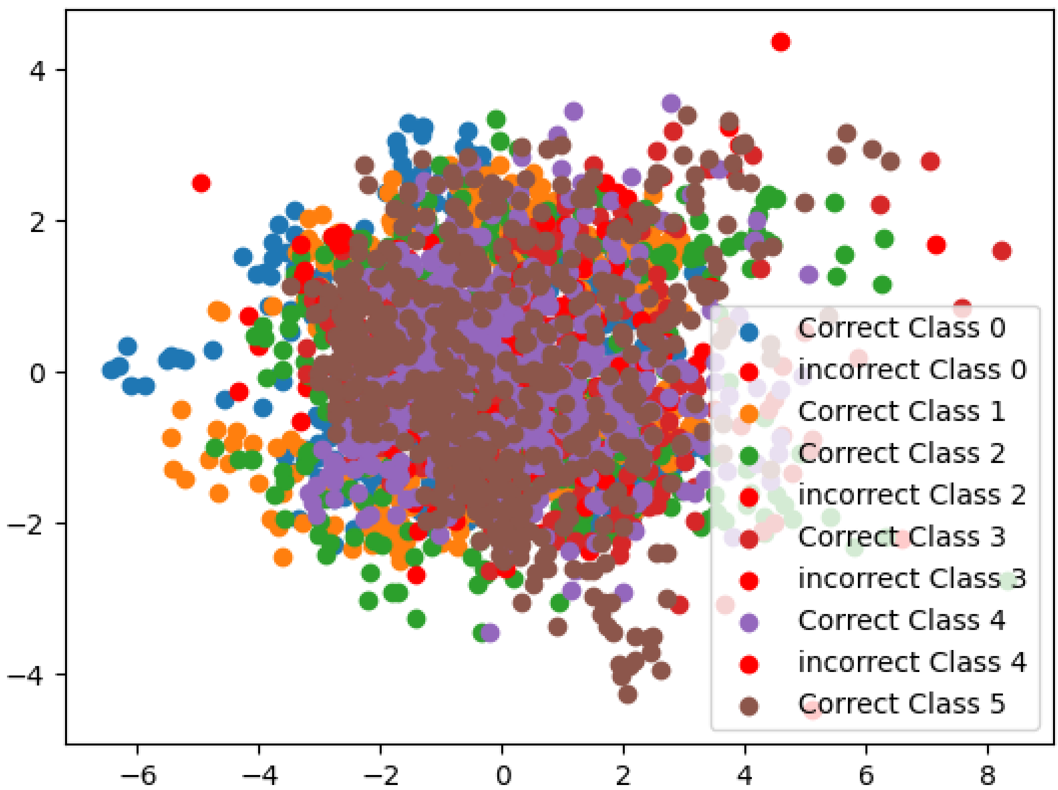

Figure 13: 2D representation of predicted class distributions

In order to visualize the correct and incorrect labels after classification regardless of the class names, I used the plot below. It shows the number of correctly predicted labels is high compared to the incorrect labels.

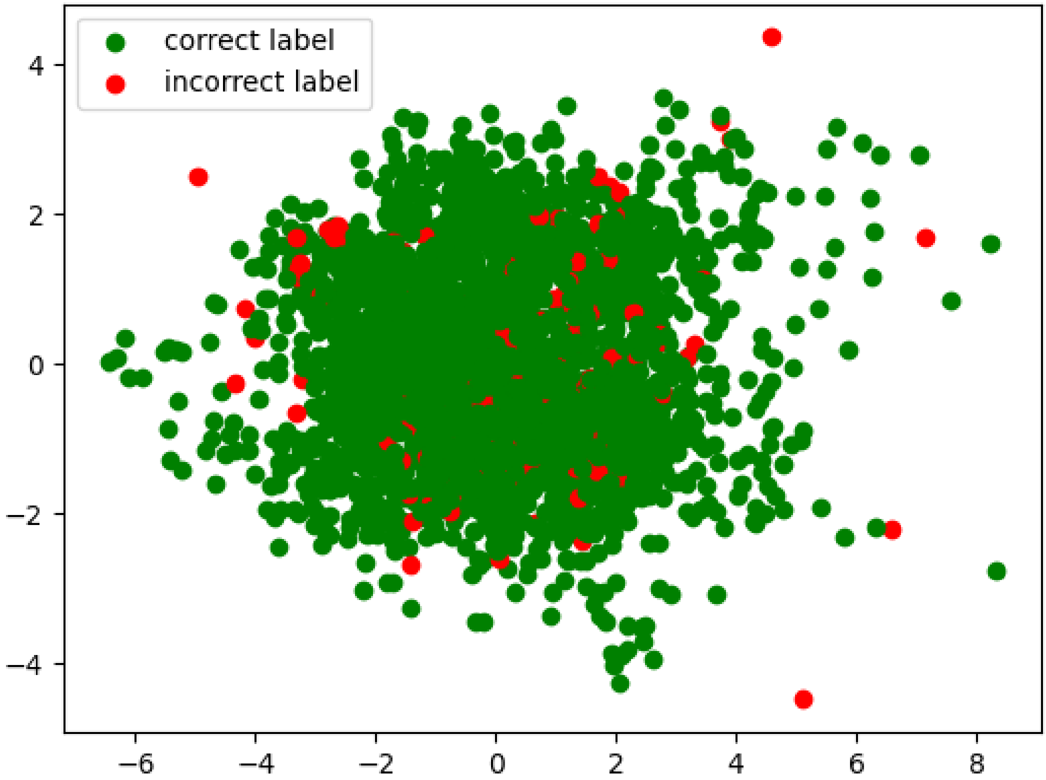

Figure 14: 2D representation of correct and incorrect class distributions

A Appendix

```python

#I use the following references:
#EECE5644_2020Spring_TakeHome1Solutions
#EECE5644_2020summer2_TakeHome1Solution_Python
# SciPy: Open source scientific tools for Python
#Professor Deniz code for the threshold parameter
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import multivariate_normal as mvn
from typing import Iterable
#ERM classifier

features = 4
samples = 10000

mean_0 = np.array([-1, -1, -1, -1])
mean_1 = np.array([1, 1, 1, 1])

cov_0 = np.array([[2, -0.5, 0.3, 0], [-0.5, 1, -0.5, 0], [0.3, -0.5, 1, 0], [0, 0, 0, 2]])
cov_1 = np.array([[1, 0.3, -0.2, 0], [0.3, 2, 0.3, 0], [-0.2, 0.3, 1, 0], [0, 0, 0, 3]])

# cov_0=np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]])# covariance matrix for Naive-Bayes classifier
# cov_1=np.array([[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1]])#covariance matrix for Naive-Bayes classifier

prior = [0.7, 0.3]

fpr = []  # false positive rate array
tpr = []  # true positive rate array

fpr_theory = []  # theoretical false positive rate
tpr_theory = []  # theoretical true positive rate

P_error = []
gamma_list = []

label = np.zeros((3, samples))
label[0, :] = (np.random.uniform(0, 1, samples) >= prior[0]).astype(int)
dataset = np.zeros((features, samples))
for index in range(samples):
    if label[0, index] == 0:
        dataset[:, index] = mvn(mean=mean_0.reshape(4, ), cov=cov_0).rvs(1)
    else:
        dataset[:, index] = mvn(mean=mean_1.reshape(4, ), cov=cov_1).rvs(1)

class0_count = float(list(label[0, :]).count(0))  # number of samples for class 0
class1_count = float(list(label[0, :]).count(1))  # number of samples for class 1


```

```python
# Calculate the discriminant score
logValpdf1 = np.log(mvn.pdf(dataset.T, mean=mean_1, cov=cov_1))
logValpdf0 = np.log(mvn.pdf(dataset.T, mean=mean_0, cov=cov_0))
discriminant_score = logValpdf1 - logValpdf0


# Create list of threshold values

tau = np.log(sorted(discriminant_score[np.array(discriminant_score[:].
    astype(float) >= 0)]))
mid_tau = np.array([tau[0] - 100, (tau[0:(len(tau) - 1)] + (np.diff(tau)
    ) / 2).tolist(), tau[len(tau) - 1] + 100])


def flatten(lis):
    for item in lis:
        if isinstance(item, Iterable) and not isinstance(item, str):
            for x in flatten(item):
                yield x
        else:
            yield item

mid_tau = list(flatten(mid_tau.tolist()))

for gamma in mid_tau:
    label[1, :] = (discriminant_score >= gamma)
    x10 = [i for i in range(label.shape[1]) if (label[1, i] == 1 and
    label[0, i] == 0)]
    x11 = [i for i in range(label.shape[1]) if (label[1, i] == 1 and
    label[0, i] == 1)]
    fpr.append(len(x10) / class0_count)
    tpr.append(len(x11) / class1_count)
    P_error.append((len(x10) / class0_count) * prior[0] + (1 - len(x11)
    / class1_count) * prior[1])

# theoretical minimum error
label[2, :] = (discriminant_score >= np.log(prior[1] / prior[0])).astype
    (int)
x10_theory = [i for i in range(label.shape[1]) if (label[2, i] == 1 and
    label[0, i] == 0)]
x11_theory = [i for i in range(label.shape[1]) if (label[2, i] == 1 and
    label[0, i] == 1)]
fpr_theory.append(len(x10_theory) / class0_count)
tpr_theory.append(len(x11_theory) / class1_count)
min_p_error_theory = (len(x10_theory) / class0_count) * prior[0] + (1 -
    len(x11_theory) / class1_count) * prior[1]

minimum_error = min(P_error)
min_idx = np.argmin(P_error)


print('Optimal threshold {}'.format(mid_tau[min_idx]))
print('TPR at minimum probability:{}'.format(tpr[min_idx]))
print('FPR at minimum probability:{}'.format(fpr[min_idx]))

# Plot the actual data distribution
x0 = [i for i in range(label.shape[1]) if (label[0, i] == 0)]
x1 = [i for i in range(label.shape[1]) if (label[0, i] == 1)]
```

```

plt.plot(dataset[0, x0], dataset[1, x0], '+', color='mediumvioletred')
plt.plot(dataset[0, x1], dataset[1, x1], '.', color='c')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title("Actual Data distribution")
plt.legend(['Class 0', 'Class 1'])
plt.show()

# Plot the ROC curve
plt.plot(fpr, tpr, color='red')
plt.plot(fpr[min_idx], tpr[min_idx], 'o', color='k')
plt.plot(fpr_theory, tpr_theory, 'X')
plt.xlabel('P_False Alarm')
plt.ylabel('P_Correct Detection')
plt.title('ROC Curve')
plt.legend(['ROC', 'Experimental min Error', 'Theoretical min Error'])
plt.show()

# LDA classifier

Sb = np.dot((mean_0 - mean_1), (mean_0 - mean_1).T)
Sw = cov_0 + cov_1

A = (np.linalg.inv(Sw)).dot(Sb)
eigenvalues, eigenvectors = np.linalg.eig(A)
eigenvectors = eigenvectors.T

w = np.array(eigenvectors[np.argmax(eigenvalues)])
y0 = np.zeros((2, len(x0)))
y1 = np.zeros((2, len(x1)))
y0[0, :] = np.dot(w.T, dataset[:, x0])
y1[0, :] = np.dot(w.T, dataset[:, x1])
y = np.sort(np.hstack((y0[0], y1[0])))
a = []

fpr = []
tpr = []
Perror = []
p_thr = []
thery_mid_tau = []

for threshold in mid_tau:
    x00 = list((y0[0, :] >= threshold).astype(int)).count(0)
    x01 = list((y1[0, :] >= threshold).astype(int)).count(0)
    x10 = list((y0[0, :] >= threshold).astype(int)).count(1)
    x11 = list((y1[0, :] >= threshold).astype(int)).count(1)
    fpr.append(float(x10) / y0.shape[1])
    tpr.append(float(x11) / y1.shape[1])
    Perror.append((x10 / class0_count) * prior[0] + (1 - x11 /
    class1_count) * prior[1])

for lists in range(len(mid_tau)):
    thery_mid_tau.append(np.log(prior[1] / prior[0]))
for threshold in thery_mid_tau:
    x10_thr = list((y0[0, :] >= threshold).astype(int)).count(1)
    x11_thr = list((y1[0, :] >= threshold).astype(int)).count(1)
    p_thr.append((x10_thr / class0_count) * prior[0] + (1 - x11_thr /
```

```
    class1_count) * prior[1])
idx=np.argmin(Perror)
print('Minimum probability error:{}'.format(min(Perror)))
print('TPR at min error:{}'.format(tpr[idx]))
print('FPR at min error:{}'.format(fpr[idx]))

# projected data distribution
plt.scatter(y0[0, :], np.zeros((y0.shape[1])))
plt.scatter(y1[0, :], np.zeros((y1.shape[1])))
plt.legend(['Class 0', 'Class 1'])
plt.title('fLDA projection ')
plt.show()

# Plot the ROC curve
plt.plot(fpr, tpr, color='red')
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.plot(fpr[np.argmin(Perror)], tpr[np.argmin(Perror)], 'o', color='k')
plt.title("ROC curve")
plt.legend(['ROC', 'Experimental min Error'])
plt.show()
```

Listing 1: Question 1

B Appendix

```

import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import multivariate_normal as mvn
from numpy.random.mtrand import sample

E = 7
s = 0.1 * E
A = np.random.randn(3, 3)
a = 0.07

temp = np.eye(3) + a * A
temp2 = np.matmul(temp, temp)
C1 = pow(s, 2) * temp2
C2 = pow(s, 2) * np.eye(3)
C3 = pow(s, 2) * np.eye(3)  # mixture for 3
C4 = pow(s, 2) * np.eye(3)  # mixture for 3

vertices = [(-1.0, -1.0, -1.0),
            (-1.0, 1.0, -1.0),
            (1.0, 1.0, -1.0),
            (1.0, -1.0, -1.0),
            (-1.0, -1.0, 1.0),
            (-1.0, 1.0, 1.0),
            (1.0, 1.0, 1.0),
            (1.0, -1.0, 1.0)]

mean_1 = np.array([vertices[0]])
mean_2 = np.array([vertices[1]])
mean_3 = np.array([vertices[2]])  # mixture for third label
```

```python
mean_4 = np.array([vertices[3]])  # mixture for third label

prior = [0.3, 0.3, 0.4]

features = 3
samples = 10000

label = np.zeros((3, samples))
for i in range(samples):
    p = np.random.uniform(0, 1)
    if p >= 0.6:
        label[0, i] = 3
    else:  # sample from label 1
        w = np.random.uniform(0, 1)
        if w >= 0.5:
            label[0, i] = 2
        else:
            label[0, i] = 1

dataset = np.zeros((features, samples))
for index in range(samples):
    if label[0, index] == 1:
        dataset[:, index] = np.random.multivariate_normal(mean_1.reshape
    (3, ), C1, 1)
    elif label[0, index] == 2:
        dataset[:, index] = np.random.multivariate_normal(mean_2.reshape
    (3, ), C2, 1)
    else:  # mixture sampling
        idd = np.random.uniform(0, 1)
        if idd >= 0.5:
            dataset[:, index] = np.random.multivariate_normal(mean_3.
    reshape(3, ), C3, 1)
        else:
            dataset[:, index] = np.random.multivariate_normal(mean_4.
    reshape(3, ), C4, 1)

x1 = [i for i in range(label.shape[1]) if (label[0, i] == 1)]
x2 = [i for i in range(label.shape[1]) if (label[0, i] == 2)]
x3 = [i for i in range(label.shape[1]) if (label[0, i] == 3)]

# Bayes Classifier

class_1_data = dataset[:, x1]
class_2_data = dataset[:, x2]
class_3_data = dataset[:, x3]
mean_list = [mean_1, mean_2, mean_3]
cov_list = [C1, C2, C3]
data_all = [class_1_data, class_2_data, class_3_data]

lambda_matrix = [[0, 1, 1], [1, 0, 1], [1, 1, 0]]
#lambda_matrix=[[0,1,10],[1,0,10],[1,1,0]] # for part B
#lambda_matrix=[[0,1,100],[1,0,100],[1,1,0]]# for part B


def risk(i, x, lambda_matrix):
    tot_risk = 0
    for j in range(3):
        pp = np.random.uniform(0, 1)
```

```python
        if p >= 0.5:
            mean_mixture = mean_3
            cov_mixture = C3
        else:
            mean_mixture = mean_4
            cov_mixture = C4
        mean_list = [mean_1, mean_2, mean_mixture]
        cov_list = [C1, C2, cov_mixture]
        tot_risk = tot_risk + lambda_matrix[i][j] * prior[j] * mvn.pdf(x
    , mean_list[j][0], cov_list[j])
    return tot_risk


def MAP(true_clas, lambda_matrix):
    predicted_correct = []
    predicted_incorrect = []
    confusion_matrix = np.zeros((3, 3))  # assuming the rows are actual
    and the columns as predicted
    conf_list = [[[] for _ in range(3)] for _ in range(3)]
    for i in (data_all[true_clas].T):
        choice = np.argmin([risk(0, i, lambda_matrix), risk(1, i,
    lambda_matrix), risk(2, i, lambda_matrix)])
        if choice == true_clas:
            predicted_correct.append(i)
        else:
            predicted_incorrect.append(i)
        conf_list[choice][true_clas].append(i)
    for i in range(3):
        for j in range(3):
            confusion_matrix[i][j] = len(conf_list[i][j])
    return predicted_correct, predicted_incorrect, confusion_matrix


fig = plt.figure(figsize=(10, 7.5))
ax = plt.axes(projection="3d")

# True data distribution
ax.scatter3D(dataset[0, x1], dataset[1, x1], dataset[2, x1], marker='p',
     label='class 1')
ax.scatter3D(dataset[0, x2], dataset[1, x2], dataset[2, x2], marker='x',
     label='class 2')
ax.scatter3D(dataset[0, x3], dataset[1, x3], dataset[2, x3], marker='d',
     label='class 3')
plt.title("True Class Distributions")
plt.legend()
plt.show()

confusion_mat = np.zeros((3, 3))

fig = plt.figure(figsize=(10, 7.5))
ax = plt.axes(projection="3d")
for i in range(3):
    predicted_correct, predicted_incorrect, confusion_matrix = MAP(i,
    lambda_matrix)
    confusion_mat[:, i] = confusion_matrix[:, i]

    correct_array = np.array(predicted_correct)
    incorrect_array = np.array(predicted_incorrect)
```

```
    labels = [['correct class 1', 'incorrect class 1'], ['correct class 2', 'incorrect class 2'],
              ['correct class 3', 'incorrect class 3']]
    markers = ['s', '^', 'o']
    colors = ['g', 'b', 'o']
    ax.scatter3D(correct_array[:, 0], correct_array[:, 1], correct_array[:, 2], c='g', marker=markers[i],
                 label=labels[i][0])
    ax.scatter3D(incorrect_array[:, 0], incorrect_array[:, 1], incorrect_array[:, 2], c='r', marker=markers[i],
                 label=labels[i][1])

plt.title(" Predicted Class Distributions")
plt.legend()
plt.show()

print('confusion matrix:{}'.format(confusion_mat))
```

Listing 2: Question 2

C Appendix

```

#Wine Dataset


import pandas as pd
import numpy as np
from scipy.stats import multivariate_normal as mvn
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

wine_white = pd.read_csv('white.csv', sep=';')
samples = wine_white.shape[0]

dataset = []
for i in range(11):
    temp = wine_white.loc[wine_white['quality'] == i].to_numpy()
    dataset.append(np.delete(temp, -1, axis=1))
muVector = []
sigmaVector = []
lamdba_const = 0.1
for j in range(11):
    muVector.append(np.mean(dataset[j], axis=0))
    sigmaVector.append(np.cov(dataset[j], rowvar=False) + lamdba_const * np.eye(11))
for i in range(11):
    if i == 0 or i == 1 or i == 2 or i == 10:
        muVector[i] = np.zeros(11)
        sigmaVector[i] = np.eye(11)

# (prior for a given class) = (number of samples in the class) / (total number of samples)).

priors = []
for i in dataset:
```

```
    temp = (i.shape[0]) / samples
    priors.append(temp)

lambda_matrix = (np.full((11, 11), 1))  # check the actual label
np.fill_diagonal(lambda_matrix, [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])


def risk(i, x, lambda_matrix):
    tot = 0
    for j in range(11):
        tot = tot + lambda_matrix[i][j] * priors[j] * mvn.pdf(x,
   muVector[j], sigmaVector[j])
    return tot


def MAP(true_label, lambda_matrix):
    predicted_correct = []
    predicted_incorrect = []
    confusion_matrix = np.zeros((11, 11))  # assuming the cols are
   actual and the rows as predicted
    conf_list = [[[] for _ in range(11)] for _ in range(11)]
    for i in dataset[true_label]:
        choice = np.argmin([risk(k, i, lambda_matrix) for k in range(11)
   ])
        if choice == true_label:
            predicted_correct.append(i)
        else:
            predicted_incorrect.append(i)

        conf_list[choice][true_label].append(i)

    for i in range(11):
        for j in range(11):
            confusion_matrix[i][j] = len(conf_list[i][j])
    return predicted_correct, predicted_incorrect, confusion_matrix


def PCA(X, n):
    mean_x = X - np.mean(X, axis=0)

    cov_mat = np.cov(mean_x, rowvar=False)

    eigen_values, eigen_vectors = np.linalg.eigh(cov_mat)

    sorted_index = np.argsort(eigen_values)[::-1]
    sorted_eigenvalue = eigen_values[sorted_index]
    sorted_eigenvectors = eigen_vectors[:, sorted_index]

    eigenvector_subset = sorted_eigenvectors[:, 0:n]

    x_pca = np.dot(eigenvector_subset.transpose(), mean_x.transpose()).
   transpose()

    return x_pca


fig = plt.figure(figsize=(10, 7))
ax = plt.axes()  # projection ="2d"
```

```
scale = StandardScaler()

confusion_mat = np.zeros((11, 11))

for i in range(11):

    predicted_correct, predicted_incorrect, confusion_matrix = MAP(i,
    lambda_matrix)  # calling the function
    confusion_mat[:, i] = confusion_matrix[:, i]  # cols wise actual
    values
    # scale the data before pca

    correct_stacked = np.zeros((0, 11))
    for j in predicted_correct:
        correct_stacked = np.vstack((correct_stacked, j))
    incorrect_stacked = np.zeros((0, 11))
    for k in predicted_incorrect:
        incorrect_stacked = np.vstack((incorrect_stacked, k))

    if correct_stacked.size == 0 or incorrect_stacked.size == 0:
        continue

    correct = scale.fit_transform(correct_stacked)
    incorrect = scale.fit_transform(incorrect_stacked)
    # apply PCA
    correct_reduced = PCA(correct, 2)
    incorrect_reduced = PCA(incorrect, 2)

    label = 'correct class {}'.format(i)
    ax.scatter(correct_reduced[:, 0], correct_reduced[:, 1], c='g')
    ax.scatter(incorrect_reduced[:, 0], incorrect_reduced[:, 1], c='r')

plt.title(" Predicted Class Distributions")
plt.legend(['correct labels', 'incorrect labels'])
plt.show()

# True class distribution in 2D
fig = plt.figure(figsize=(10, 7))
ax = plt.axes()
for i in dataset:
    if i.size == 0:
        continue
    dt = PCA(i, 2)
    ax.scatter(dt[:, 0], dt[:, 1], label='Class {}'.format(dataset.index
    (i)))
plt.legend()
plt.show()

print(confusion_mat)

#HAR Dataset

import pandas as pd
import numpy as np
from scipy.stats import multivariate_normal as mvn

train_data = (np.genfromtxt('X_train.txt', delimiter=''))
```

```
train_label = (np.genfromtxt('y_train.txt', delimiter=''))

test_data=(np.genfromtxt('X_test.txt', delimiter=''))
test_label = (np.genfromtxt('y_test.txt', delimiter=''))


nn = train_label.tolist()
train_df = pd.DataFrame(train_data)
train_df['Labels'] = nn

mm = test_label.tolist()
test_df = pd.DataFrame(test_data)
test_df['Labels'] = mm


dataset = []
for i in range(1, 7, 1):
    temp = train_df.loc[train_df['Labels'] == i].to_numpy()
    dataset.append(np.delete(temp, -1, axis=1))
samples = train_df.shape[0]
dataset2=[]
for i in range(1, 7, 1):
    temp = test_df.loc[test_df['Labels'] == i].to_numpy()
    dataset2.append(np.delete(temp, -1, axis=1))

muVector = []
sigmaVector = []
lambda_const = 0.01
for j in range(6):
    muVector.append(np.mean(dataset[j], axis=0))
    sigmaVector.append(np.cov(dataset[j], rowvar=False) + lambda_const * np.eye(561))

priors = []
for i in dataset:
    temp = (i.shape[0]) / samples
    priors.append(temp)

lambda_matrix = (np.full((6, 6), 1))
np.fill_diagonal(lambda_matrix, [0, 0, 0, 0, 0, 0])



def risk(ii, x, cost_matrix):
    tot_risk = 0
    for j in range(6):
        tot_risk = tot_risk + cost_matrix[ii][j] * priors[j] * mvn.pdf(x, muVector[j], sigmaVector[j])
    return tot_risk


def MAP(true_label, cost_matrix):
    predicted_correct = []
    predicted_incorrect = []
    confusion_matrix = np.zeros((6, 6))  # assuming the rows are actual and the columns as predicted
    conf_list = [[[] for _ in range(6)] for _ in range(6)]

```

```
    for data in dataset2[true_label]:  # data is the row vector

        predicted_class = np.argmin([risk(k, data, cost_matrix) for k in
     range(6)])

        if predicted_class == true_label:
            predicted_correct.append(data)
        else:
            predicted_incorrect.append(data)

        conf_list[predicted_class][true_label].append(data)

    for rows in range(6):
        for cols in range(6):
            confusion_matrix[rows][cols] = len(conf_list[rows][cols])
    return predicted_correct, predicted_incorrect, confusion_matrix


confusion_mat = np.zeros((6, 6))
corr=[]
incorr=[]
for i in range(6):
    predicted_correct, predicted_incorrect, confusion_matrix = MAP(i,
    lambda_matrix)  # calling the function
    confusion_mat[:,i] = confusion_matrix[:,i]  # col wise actual values
    corr.append(predicted_correct)
    incorr.append(predicted_incorrect)

print('confusion matrx:{}'.format(confusion_mat),'correct matrx:{}'.
    format(corr),'incorrect matrx:{}'.format(incorr))
```

Listing 3: Question 3