Assignment 1 - Mahdiar Edraki

Friday, October 2, 2020 6:57 PM

Note: All codes used in this assignment were written from scratch by Mahdiar Edraki using Python.

Question 1 (60%)

The probability density function (pdf) for a 4-dimensional real-valued random vector $\mathbf{X}$ is as follows: $p(\mathbf{x}) = p(\mathbf{x}|L=0)P(L=0) + p(\mathbf{x}|L=1)P(L=1)$. Here L is the true class label that indicates which class-label-conditioned pdf generates the data.

The class priors are $P(L=0) = 0.7$ and $P(L=1) = 0.3$. The class class-conditional pdfs are $p(\mathbf{x}|L=0) = g(\mathbf{x}|\mathbf{m}_0, \mathbf{C}_0)$ and $p(\mathbf{x}|L=1) = g(\mathbf{x}|\mathbf{m}_1, \mathbf{C}_1)$, where $g(\mathbf{x}|\mathbf{m}, \mathbf{C})$ is a multivariate Gaussian probability density function with mean vector $\mathbf{m}$ and covariance matrix $\mathbf{C}$. The parameters of the class-conditional Gaussian pdfs are:

$$\mathbf{m}_0 = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \quad \mathbf{C}_0 = \begin{bmatrix} 2 & -0.5 & 0.3 & 0 \\ -0.5 & 1 & -0.5 & 0 \\ 0.3 & -0.5 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \quad \mathbf{m}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{C}_1 = \begin{bmatrix} 1 & 0.3 & -0.2 & 0 \\ 0.3 & 2 & 0.3 & 0 \\ -0.2 & 0.3 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

For numerical results requested below, generate 10000 samples according to this data distribution, keep track of the true class labels for each sample. Save the data and use the same data set in all cases.

Given: $X \in \mathbb{R}^4$

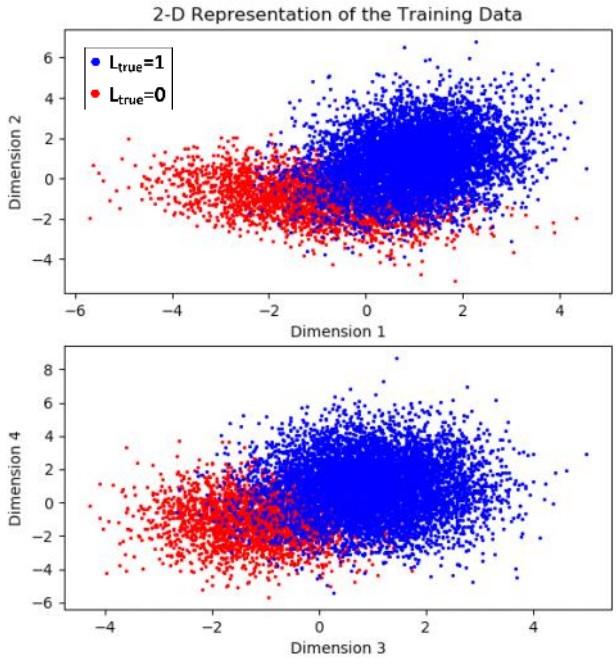
$p(x) = p(x|L=0)P(L=0) + p(x|L=1)P$

where L is the true class label

- Class priors: $P(L=0) = 0.7$ and $P(L=1) = 0.3$
- Class-conditional pdfs:

$p(x|L=0) = g(x|m_0, C_0)$ $\quad$ $p(x|L=1) = g(x|m_1, C_1)$

- Since x is 4-dimensional, we can visualize it by two 2-D plots:



1. Specify the minimum expected risk classification rule in the form of a likelihood-ratio test: $\frac{p(\mathbf{x}|L=1)}{p(\mathbf{x}|L=0)} \overset{?}{>} \gamma$, where the threshold γ is a function of class priors and fixed (nonnegative) loss values for each of the four cases $D = i|L = j$ where D is the decision label that is either 0 or 1, like L.

We have 2 classes to choose from. So our classifier decision is either $D=0$ or $D=1$

• Let • $L_0 = 0$ • $L_1 = 1$ • $D_0 = 0$ • $D_1 = 1$

We will assign a risk to each decision using $\lambda(D_i \mid C_j)$

Therefore, **expected risk** is: $R(D_i \mid x) = \sum_{j=0}^{1} \lambda(D_i \mid L_j)\, p(L_j \mid x)$ (since we have 2 classes)

Objective find $D_* = \underset{D_i}{\operatorname{argmin}}\ R(D_i \mid x)$

$\Rightarrow$ • $R(D_0 \mid x) = \lambda(D_0 \mid L_0)\, p(L_0 \mid x) + \lambda(D_0 \mid L_1)\, p(L_1 \mid x)$

• $R(D_1 \mid x) = \lambda(D_1 \mid L_1)\, p(L_1 \mid x) + \lambda(D_1 \mid L_0)\, p(L_0 \mid x)$

let $\lambda = \begin{bmatrix} \lambda(D_0 \mid L_0) & \lambda(D_0 \mid L_1) \\ \lambda(D_1 \mid L_0) & \lambda(D_1 \mid L_1) \end{bmatrix} = \begin{bmatrix} \lambda_{00} & \lambda_{01} \\ \lambda_{10} & \lambda_{11} \end{bmatrix}$

$R(D_0 \mid x) \underset{D_0}{\overset{D_1}{\gtrless}} R(D_1 \mid x)$ → **Decision Rule**

$\lambda_{00}\, p(L_0 \mid x) + \lambda_{01}\, p(L_1 \mid x) \underset{D_0}{\overset{D_1}{\gtrless}} \lambda_{11}\, p(L_1 \mid x) + \lambda_{10}\, p(L_0 \mid x)$

$\Rightarrow (\lambda_{01} - \lambda_{11})\, p(L_1 \mid x) \underset{D_0}{\overset{D_1}{\gtrless}} (\lambda_{10} - \lambda_{00})\, p(L_0 \mid x)$

Assuming $\lambda_{10} > \lambda_{00}$ and $\lambda_{01} > \lambda_{11}$

$\Rightarrow \frac{p(L_1 \mid x)}{p(L_0 \mid x)} \underset{D_0}{\overset{D_1}{\gtrless}} \frac{\lambda_{10} - \lambda_{00}}{\lambda_{01} - \lambda_{11}}$

Using Bayes Rule: $p(L_1 \mid x) = \frac{p(x \mid L_1)\, p(L_1)}{p(x)}$

$\Rightarrow \frac{\left(\frac{p(x \mid L_1)\, p(L_1)}{p(x)}\right)}{\left(\frac{p(x \mid L_0)\, p(L_0)}{p(x)}\right)} \underset{D_0}{\overset{D_1}{\gtrless}} \frac{\lambda_{10} - \lambda_{00}}{\lambda_{01} - \lambda_{11}}$

$$\Rightarrow \quad \frac{p(x|L_1)}{p(x|L_0)} \underset{D_0}{\overset{D_1}{\gtrless}} \frac{(\lambda_{10}-\lambda_{00})}{(\lambda_{01}-\lambda_{11})}\left(\frac{p(L_0)}{p(L_1)}\right) \rightarrow \text{let } \gamma = \frac{(\lambda_{10}-\lambda_{00})}{(\lambda_{01}-\lambda_{11})}\left(\frac{p(L_0)}{p(L_1)}\right)$$

$$\Rightarrow \quad \boxed{\frac{p(x|L_1)}{p(x|L_0)} \underset{D_0}{\overset{D_1}{\gtrless}} \gamma} \leftarrow \text{Minimum expected risk classification rule}$$

---

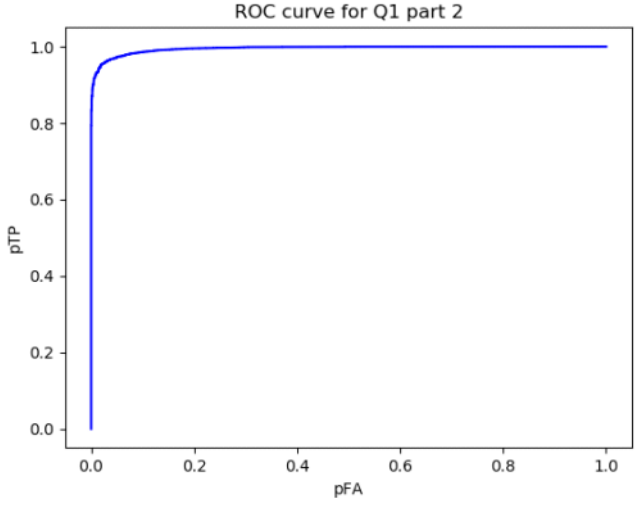
2. Implement this classifier and apply it on the 10K samples you generated. Vary the threshold γ gradually from 0 to ∞, and for each value of the threshold compute the true positive (detection) probability $P(D=1|L=1;\gamma)$ and the false positive (false alarm) probability $P(D=1|L=0;\gamma)$. Using these paired values, trace/plot an approximation of the ROC curve of the minimum expected risk classifier. Note that at $\gamma=0$ The ROC curve should be at $\binom{1}{1}$, and as *gamma* increases it should traverse towards $\binom{0}{0}$. Due to the finite number of samples used to estimate probabilities, your ROC curve approximation should reach this destination value for a finite threshold value. Keep track of $(D=0|L=1;\gamma)$ and $P(D=1|L=0;\gamma)$ values for each *gamma* value for use in the next section.

$$\frac{p(x|L_1)}{p(x|L_0)} \underset{D_0}{\overset{D_1}{\gtrless}} \gamma$$

$$p(x|L_1) = \frac{1}{(2\pi)^{n/2}|\Sigma_1|^{1/2}} e^{-\frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1)}$$

Using this equation, we can obtain the **discriminant_score** for each random variable X.

→ we can plot the ROC by **sorting** the discriminant scores and varying γ across the range of those values.



→ where $pTP = p(D=1|L=1;\gamma)$
↳ probability of true positive

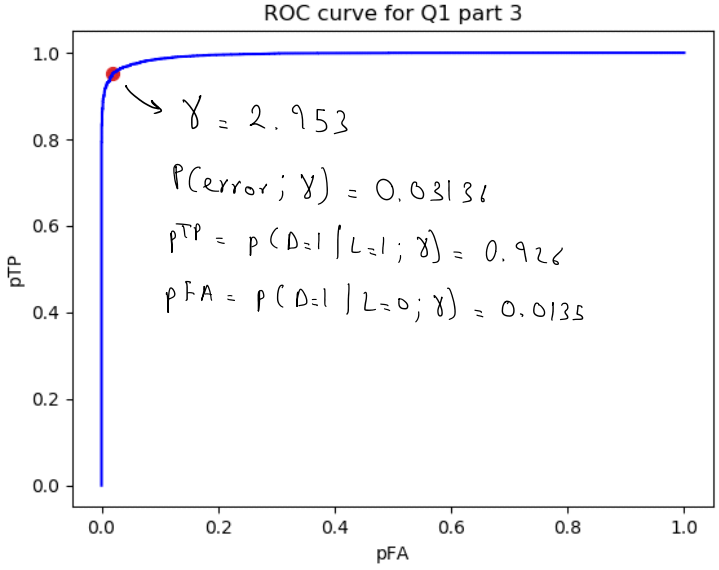
$pFA = p(D=1|L=0;\gamma)$
↳ probability of false alarm

3. Determine the threshold value that achieves minimum probability of error, and on the ROC curce, superimpose clearly (using a different color/shape marker) the true positive and false positive values attained by this minimum-P(error) classifier. Calculate and report an estimate of the minimum probability of error that is achievable for this data distribution. Note that $P(error;\gamma) = P(D=1|L=0;\gamma)P(L=0) + P(D=0|L=1;\gamma)P(L=1)$. How does your empirically selected γ value that minimizes P(error) compare with the theoretically optimal threshold you compute from priors and loss values?

$P(error, \gamma) = \underbrace{P(D_1 | L_0; \gamma)}_{pFP} \underbrace{P(L_0)}_{0.7} + \underbrace{P(D_0 | L_1; \gamma)}_{pFN} \underbrace{P(L_1)}_{0.3}$

↳ From python we get:
- Minimum probability of error = 0.03136
- Threshold value: γ = 2.953

$N = 10,000 \rightarrow$
- When $p(D=1 | L=1; \gamma) = 0.926$ and $p(D=1 | L=0; \gamma) = 0.0135$



Theoretically optimal threshold:

Recall our decision rule:

$$\frac{p(x|L_1)}{p(x|L_0)} \underset{D_0}{\overset{D_1}{\gtrless}} \gamma \quad \text{where} \quad \gamma = \frac{(\lambda_{10} - \lambda_{00})}{(\lambda_{01} - \lambda_{11})} \left(\frac{P(L_0)}{P(L_1)} \right)$$

↓ if we assume 0-1 loss

The theoretically optimal threshold $\gamma = \frac{P(L_0)}{} = \frac{0.7}{} = 2.33$

$P(L_1)$ 0.7

The theoretically optimal threshold is 2.53

⇒ Empirically selected γ = 2.953

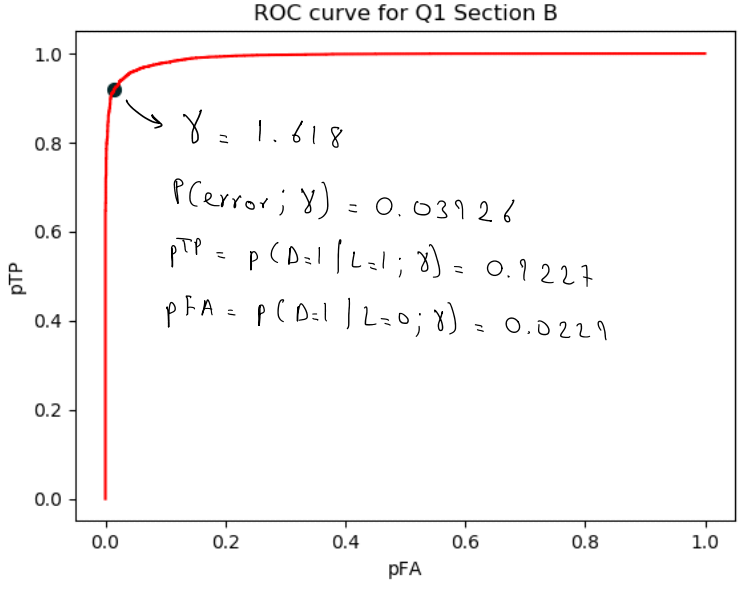
Theoretically selected γ = 2.53 → The reason for the difference is that we are only using 10,000 data samples to obtain the empirical threshold.

To make our empirically obtained γ closer to the theoretical one, we must increase our data sample size, but that is not the goal of this question.

Given that our empirically identified γ is close to the theoretical one, this shows that the algorithm I developed is functioning correctly.

Part B: ERM classification attempt using incorrect knowledge of data distribution (Naive Bayesian Classifier, which assumes features are independent given each class label)... For this part, assume that you know the true class prior probabilities, but for some reason you think that the class conditional pdfs are both Gaussian with the true means, but (incorrectly) with covariance ma[redacted] (assuming that off-diagonal values are all zeros). Analyze the impact of this model mismatch in this Naive Bayesian (NB) approach to classifier design by repeating the same steps in Part A on the same 10K sample data set you generated earlier. Report the same results, answer the same questions. Did this model mismatch negatively impact your ROC curve and minimum achievable probability of error?

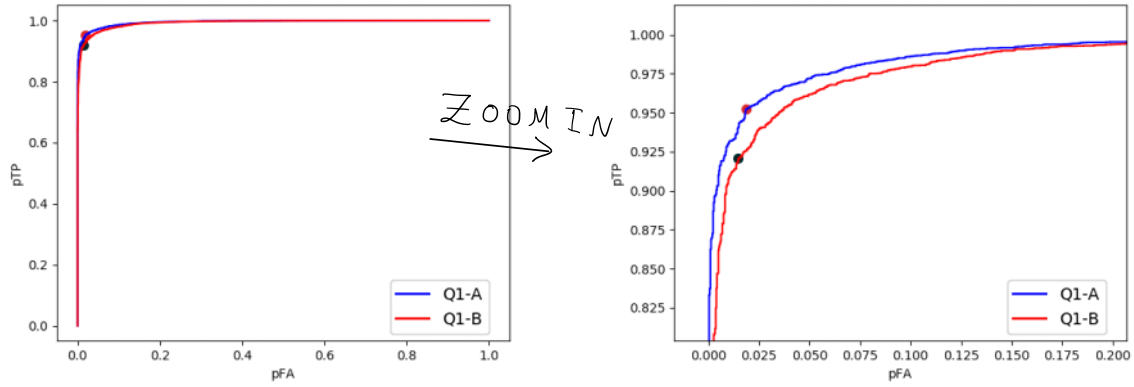
Naive Bayesian: $\Sigma_0 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$, $\Sigma_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}$



- From part A: minimum achievable probability of error = 0.03136
- From part B: minimum achievable probability of error: 0.03926

- From part A: minimum achievable probability of error = 0.03136
- From part B: minimum achievable probability of error: 0.03926

> Using a Naive Bayesian approach negatively impacted the minimum achievable probability of error.




```
Area under ROC curve for Q1 Section A is: 0.9949429313260083
Area under ROC curve for Q1 Section B is: 0.9923460964945879
```

From Python we get:

The area under ROC curve for Q1-A $\approx$ 0.9949429

The area under ROC curve for Q1-B $\approx$ 0.9923461

* Note: The **Best** classifier would have an area of 1.0 which would mean that there exists a γ where $pTP = 1$ and $pFA = 0$

↳ This means that a classifier with **higher** area under its ROC curve is **better**.

Since $\text{Area}_{Q1\text{-}A} > \text{Area}_{Q1\text{-}B}$, we conclude that the classifier used in **Section B** performs **worse** than the one in **Section A**, thus:

> Looking at the ROC curve from part A and B, we see that the Naive Bayesian approach negatively impacted the ROC curve as well.

Part C: In the third part of this exercise, repeat the same steps as in the previous two cases, but this time using a Fisher Linear Discriminant Analysis (LDA) based classifier. Using the 10K available samples, estimate the class conditional pdf mean and covariance matrices using sample average estimators for mean and covariance. From these estimated mean vectors and covariance matrices, determine the Fisher LDA projection weight vector (via the generalized eigendecomposition of within and between class scatter matrices): $\mathbf{w}_{LDA}$. For the classification rule $\mathbf{w}_{LDA}^T\mathbf{x}$ compared to a threshold τ, which takes values from $-\infty$ to ∞, trace the ROC curve. Identify the threshold at which the probability of error (based on sample count estimates) is minimized, and clearly mark that operating point on the ROC curve estimate. Discuss how this LDA classifier performs relative to the previous two classifiers.

Note: When finding the Fisher LDA projection matrix, do not be concerned about the difference in the class priors. When determining the between-class and within-class scatter matrices, use equal weights for the class means and covariances, like we did in class.

Using LDA, the goal is to find a vector w_{LDA} such that $y = w_{LDA}^T x$ makes the y's as separate from each other as possible.

2-Class Fisher LDA: Objective: $\max_w \dfrac{(m_0 - m_1)^2}{\text{var}(y_0) + \text{var}(y_1)}$

- $m_0 = E[y_0] = E[w^T x_0] = w^T(\mu_0) = m_0$, $w^T(\mu_1) = m_1$
- $\text{var}(y_0) = E[(y_0 - m_0)^2] = w^T \Sigma_0 w = \text{var}(y_0)$, $w^T \Sigma_1 w = \text{var}(y_1)$

$$\max_w \frac{(w^T\mu_0 - w^T\mu_1)^2}{w^T\Sigma_0 w + w^T\Sigma_1 w} = \max_w \frac{w^T(\mu_0 - \mu_1)(\mu_0 - \mu_1)^T w}{w^T(\Sigma_0 + \Sigma_1) w}$$

- let $S_B = (\mu_0 - \mu_1)(\mu_0 - \mu_1)^T$ → Between Class Scatter
- $S_w = \Sigma_1 + \Sigma_2$ → Within Class Scatter

(Both are symmetric)

$\Rightarrow$ Objective: $\max_w \dfrac{w^T S_B w}{w^T S_w w}$

Take derivative to maximize

$$\frac{\partial f}{\partial w^T} = \frac{(2S_B w)(w^T S_w w) - (2S_w w)(w^T S_B w)}{(w^T S_w w)^2} \overset{\text{at solution}}{=} 0$$

Assume $S_w > 0$

$$\frac{\partial f}{\partial w^T} = 0 \Rightarrow (2S_B w)(w^T S_w w) - (2S_w w)(w^T S_B w) = 0$$

$$\Rightarrow S_B w (w^T S_w w) = (S_w w)(w^T S_B w)$$

$$\Rightarrow S_B w = \left(\frac{w^T S_B w}{w^T S_w w}\right) S_w w$$

w: Generalized eigenvector of (S_B, S_w)

$\left(\frac{w^T S_B w}{w^T S_w w}\right)$: Generalized eigenvalue of (S_B, S_w)

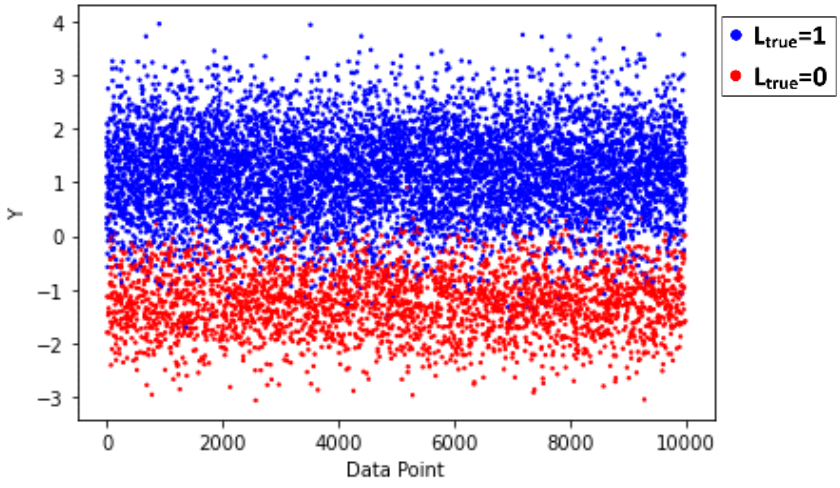
To optimize the problem: w must be the generalized eigenvector with the largest generalized eigenvalue

To optimize the problem: w must be the generalized eigenvector with the **largest generalized eigenvalue**

Once we have found w, we can find

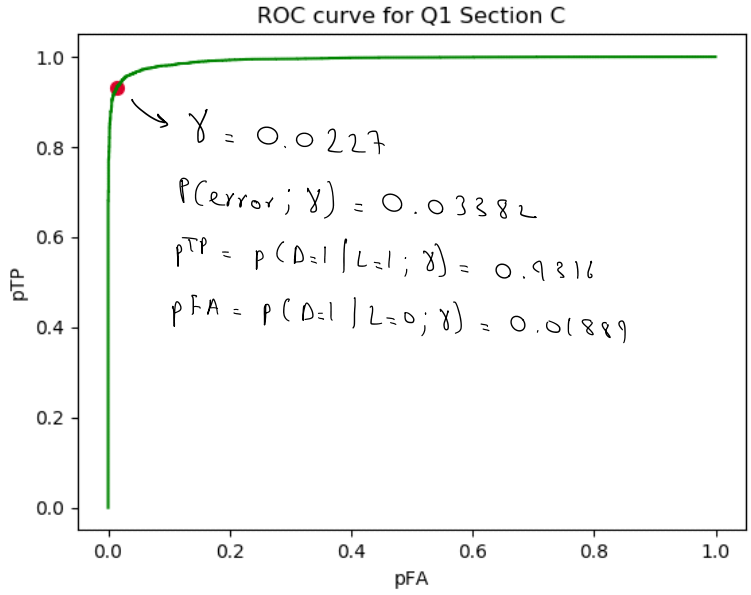
$$y = w^T x$$

plotting y and color coding based on $L=0$ and $L=1$ gives:



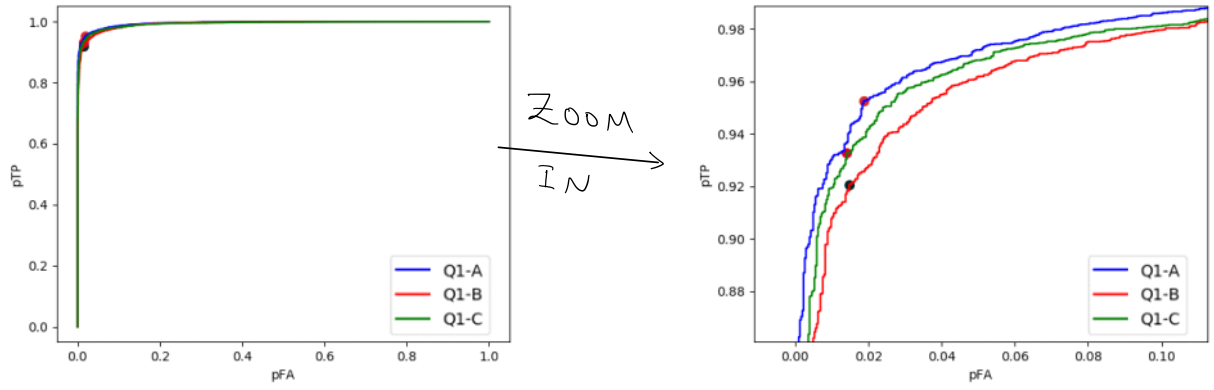
Now we can define the **Decision Rule:** $y \underset{L=0}{\overset{L=1}{\gtrless}} \gamma$

- We can vary γ from $-\infty$ to ∞ and plot the ROC curve.



Comparing the performance of classifiers developed in part A, B, and C:

- Compare ROC curves:



From Python we get:

```
Area under ROC curve for Q1 Section A is: 0.9949429313260083
Area under ROC curve for Q1 Section B is: 0.9923460964945879
Area under ROC curve for Q1 Section C is: 0.9926965672417254
```

The area under ROC curve for Q1-A ≈ 0.9949429
The area under ROC curve for Q1-B ≈ 0.9923461
The area under ROC curve for Q1-C ≈ 0.9926966

Therefore, the 3 classifiers' performance can be ranked as:
1-A > 1-C > 1-B

Looking at the ROC curves, we see that the classifier from part A performs best, followed by the LDA classifier of part C, and the worst performing classifier among the 3, is the Naive Bayesian classifier.

Comparing minimum probability of error:

Classifier	Threshold (γ)	Minimum probability of error **min P(error;γ)**	Probability of true positive **P(D=1|L=1;γ)**	Probability of false positive **P(D=1|L=0;γ)**
ERM	2.953	0.03136	0.9269	0.0135
Naïve Bayesian	1.618	0.03926	0.9227	0.0229
LDA	0.0227	0.03382	0.9316	0.0189

Similar to the observations from the ROC curves, we see that the ERM classifier developed in part A performs best.
The LDA classifier performed better than the Naive Bayesian classifier.

Question 2 (40%)

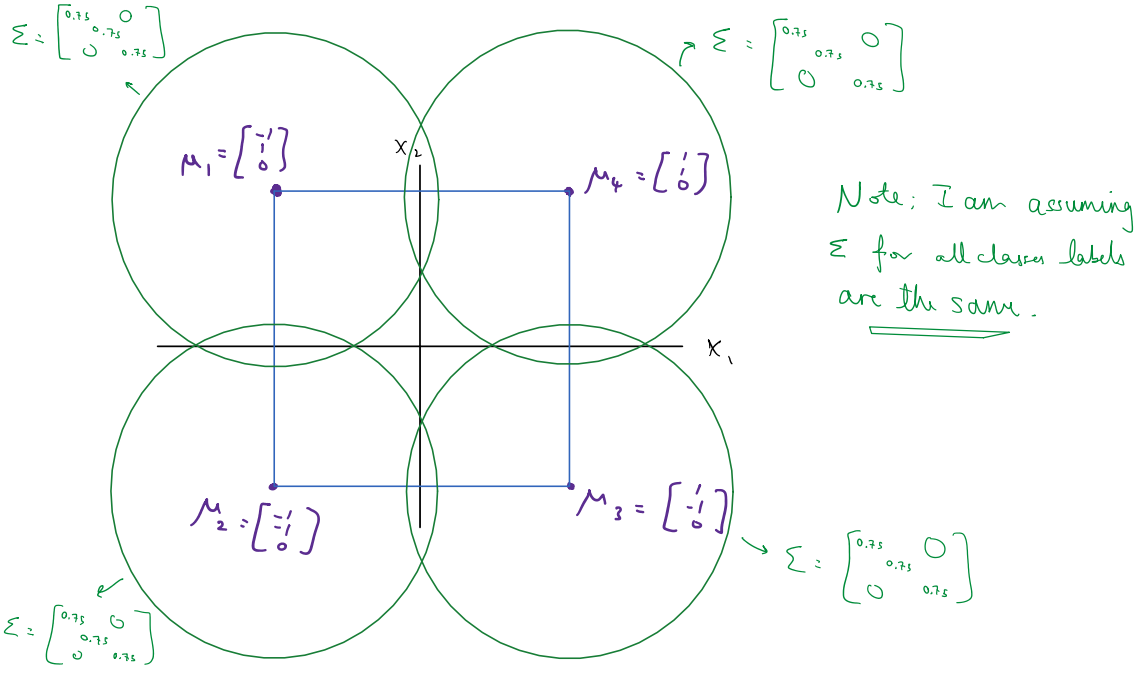
A 3-dimensional random vector $\mathbf{X}$ takes values from a mixture of four Gaussians. Each Gaussian pdf is the class-conditional pdf for one of four class labels $L \in \{1,2,3,4\}$. For this problem, pick your own 4 distinct Gaussian class conditional pdfs $p(\mathbf{x}|L=j)$, $j \in \{1,2,3,4\}$. Set class priors to $0.2, 0.25, 0.25, 0.3$. Select your Gaussian class conditional pdfs to have mean vectors located at the corners of a square. Choose covariance matrices such that each Gaussian class conditional pdf is spherically symmetric. Adjust the eigenvalues of the covariance matrices such that they are comparable to approximately 10-20% of the separation between the mean vectors for two classes (in order to have some significant level of overlap between class distributions to make the numerical and visual analyses more instructive).

Given;

Random vector $X \in \mathbb{R}^3$

4 class labels, $L \in \{1, 2, 3, 4\}$

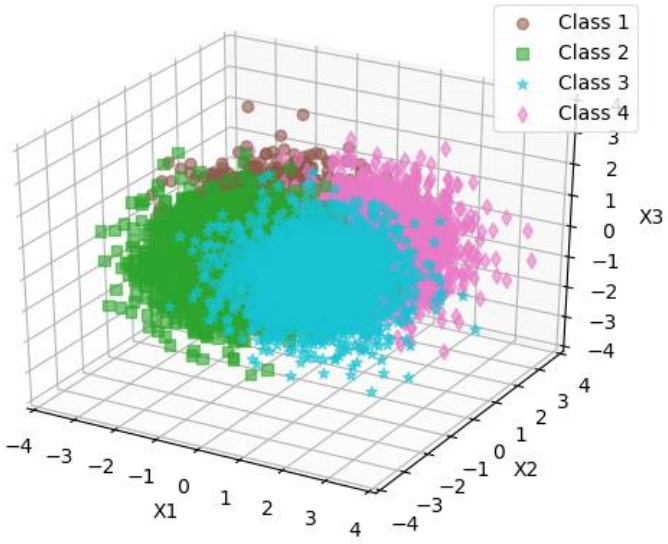
Let's visualize the Gaussian class conditional pdfs:



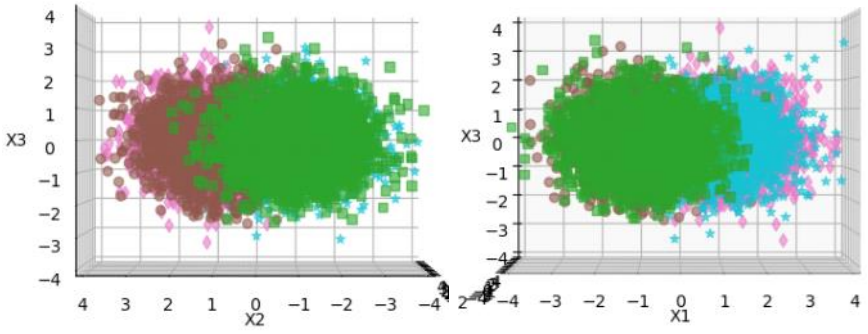
Note: I am assuming Σ for all classes labels are the same.

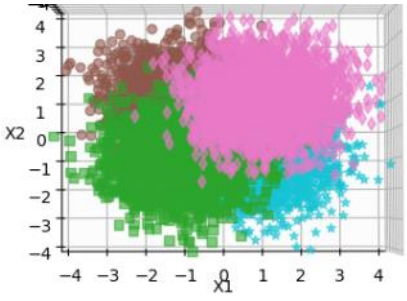
Part A: Minimum probability of error classification (0-1 loss, MAP classification rule).

1. Generate 10000 samples from this data distribution and keep track of the true labels of each sample.



Same View





---

2. Specify the decision rule that achieves minimum probability of error (i.e., use 0-1 loss), implement this classifier with the true data distribution knowledge, classify the 10K samples and count the samples corresponding to each decision-label pair to empirically estimate the confusion matrix whose entries are $P(D = i|L = j)$ for $i, j \in \{1,2,3,4\}$.

Our goal is to find the following:

Expected Risk

$$\begin{bmatrix} R(D=1|x) \\ \vdots \\ R(D=c|x) \end{bmatrix} = \Lambda \begin{bmatrix} P(L=1|x) \\ \vdots \\ P(L=c|x) \end{bmatrix}$$

since our decisions = labels

loss matrix

↳ class posteriors

Once we find the Expected Risks, we pick the one with the lowest risk.

- First we need to find class posteriors:

Using Bayes Rule: $P(L=i \mid x) = \dfrac{p(x \mid L=i)\, p(L=i)}{p(x)}$

- $p(x) = \sum_{i=1}^{C} p(x \mid L=j)\, p(L=j)$ → equal scaling factor across all risks. Can be ignored for our purposes.
- $p(L=i)$ = class prior = given ✓
- $p(x \mid L=i)$ = gaussian pdf with μ_i and Σ ✓

↓ Rewritten as

$$\begin{bmatrix} R(D=1 \mid x) \\ \vdots \\ R(D=c \mid x) \end{bmatrix} = \Lambda \begin{bmatrix} p(x \mid L=1)\, p(L=1) \\ \vdots \\ p(x \mid L=C)\, p(L=C) \end{bmatrix}$$

(ignoring $\frac{1}{p(x)}$)

- Assume Λ is a 0-1 loss matrix.

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

From the code, we obtain the following confusion matrix:

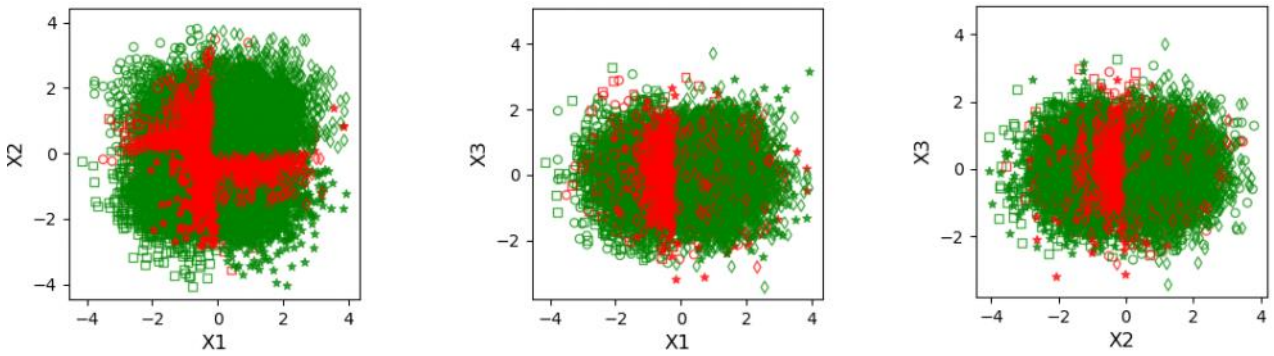
```
Q2 Part 2 - The Confusion Matrix is:
[[0.7389313  0.09701203 0.01145792 0.07277076]
 [0.11704835 0.77532014 0.11615962 0.0204988 ]
 [0.01526718 0.10710128 0.75345713 0.09873591]
 [0.12875318 0.02056655 0.11892533 0.80799453]]
```

---

3. Provide a visualization of the data (scatter-plot in 2-dimensional space), and for each sample indicate the true class label with a different marker shape (dot, circle, triangle, square) and whether it was correctly (green) or incorrectly (red) classifier with a different marker color as indicated in parantheses.

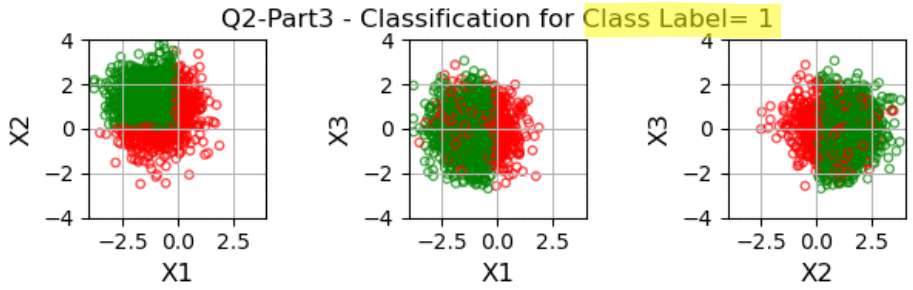
Looking at all 4 classes at once:

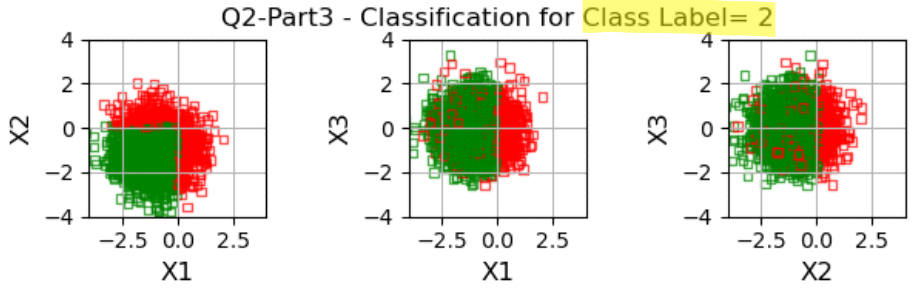
Q2-Part3 - Classification of all 4 classes

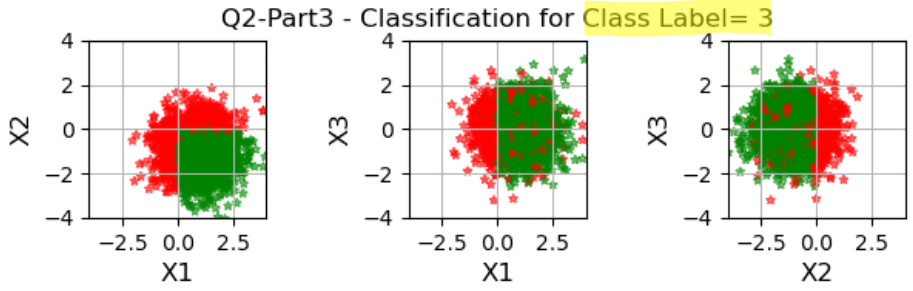


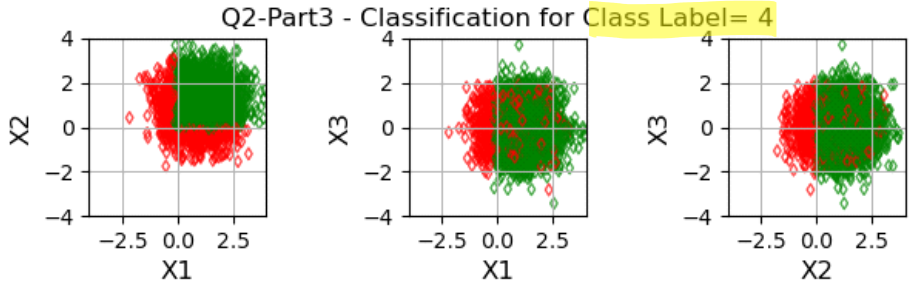
Looking at the 4 classes individually:

* Note that the limits of the axes are the same across plots.









Part B: Repeat the exercise for the ERM classification rule with the following loss values (errors between Gaussian pairs that have higher separation in their means will be penalized more):

$$\Lambda = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 10 & 0 & 5 & 10 \\ 20 & 10 & 0 & 1 \\ 30 & 20 & 1 & 0 \end{bmatrix} \quad (1)$$

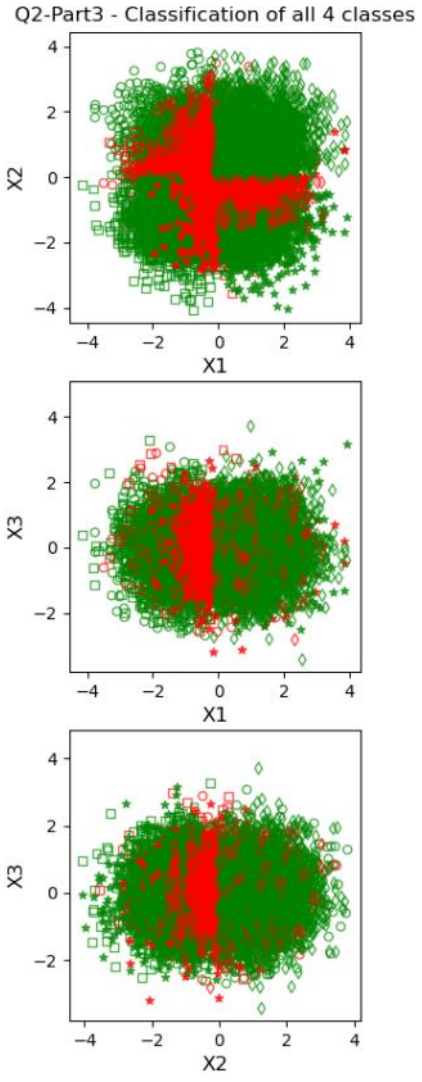
For this part, using the 10K samples, estimate the minimum expected risk that this optimal ERM classification rule will achieve. Present your results with visual and numerical reprentations. Briefly discuss interesting insights, if any.

Hint: For each sample, determine the loss matrix entry corresponding to the decision-label pair that this sample falls into, and add this loss to an estimate of cumulative loss. Divide cumulative loss by the number of samples to get average loss as an estimate for expected loss.

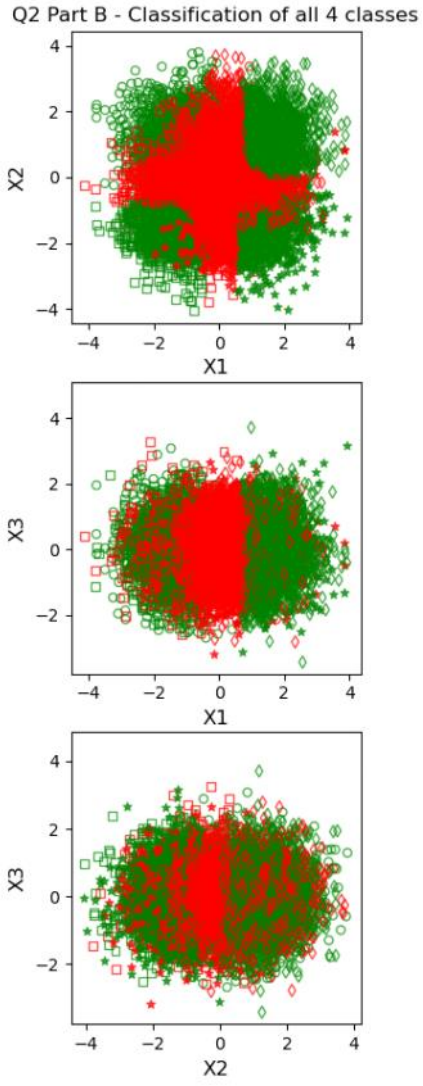
1. Visual Representation:

Similar to section A, let us look at the classification performance of Part A and Part B on all 4 classes at once:

Results from Part A



Results from part B

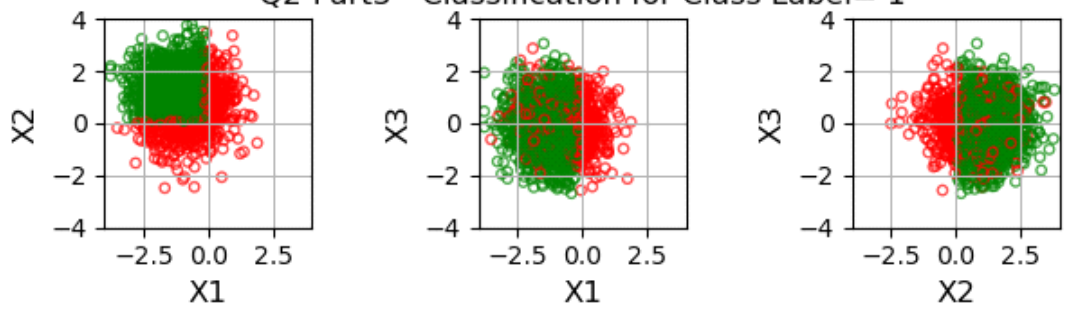


Quick comparison: We see that with the loss matrix introduced in part B, we seem to be getting many more misclassified data points. In other words, **visually**, we see more red points on the right compared to the left.

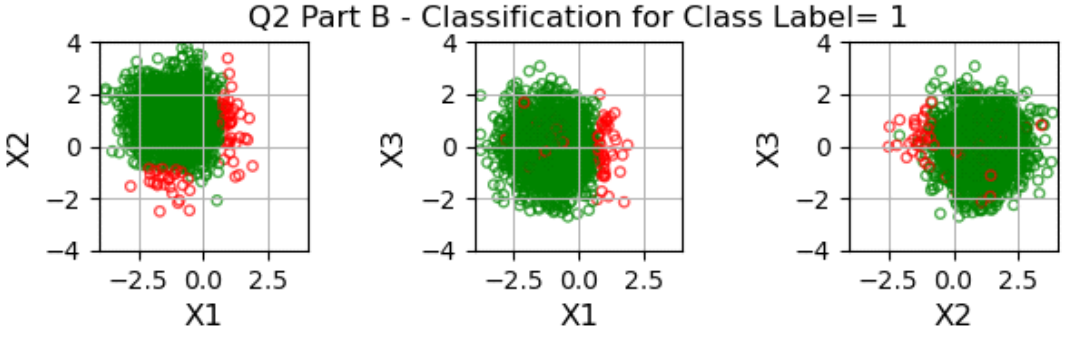
- Now let's look at each class separately:

Class Label 1

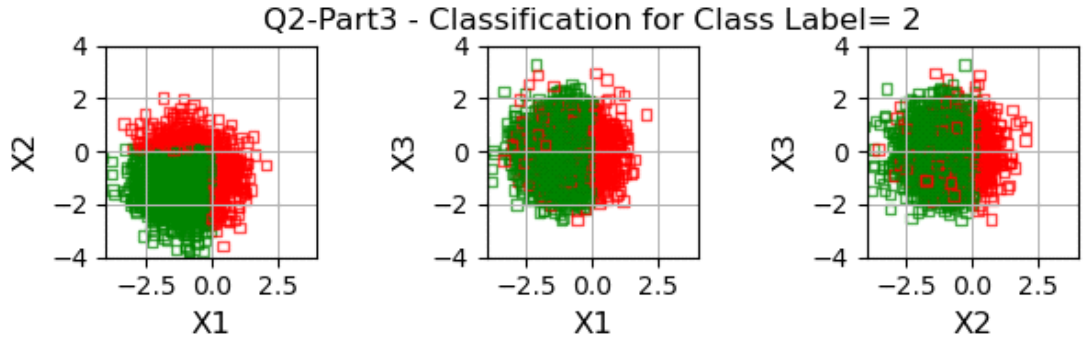
Part A



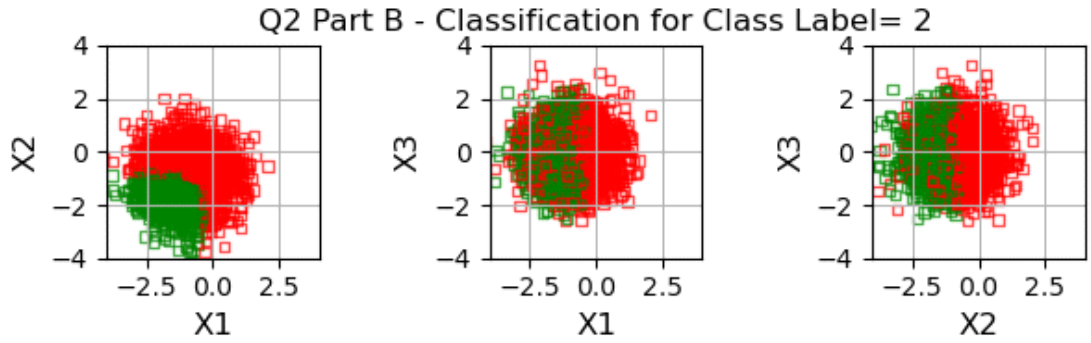
Part B



Class Label 2

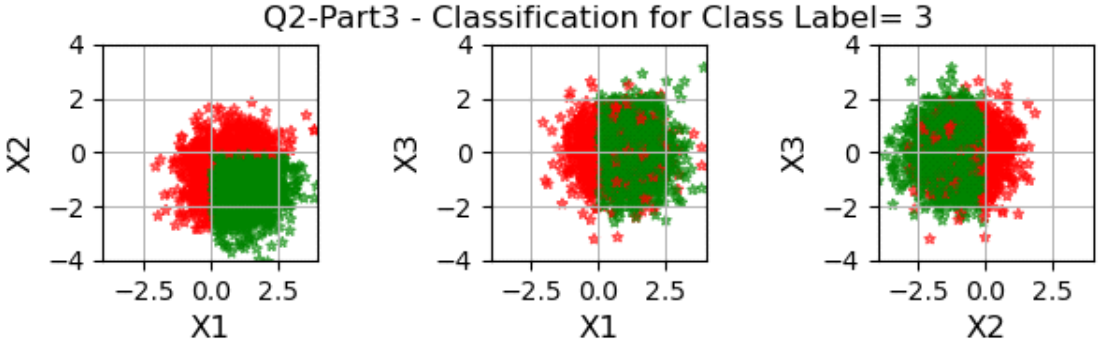


Part A

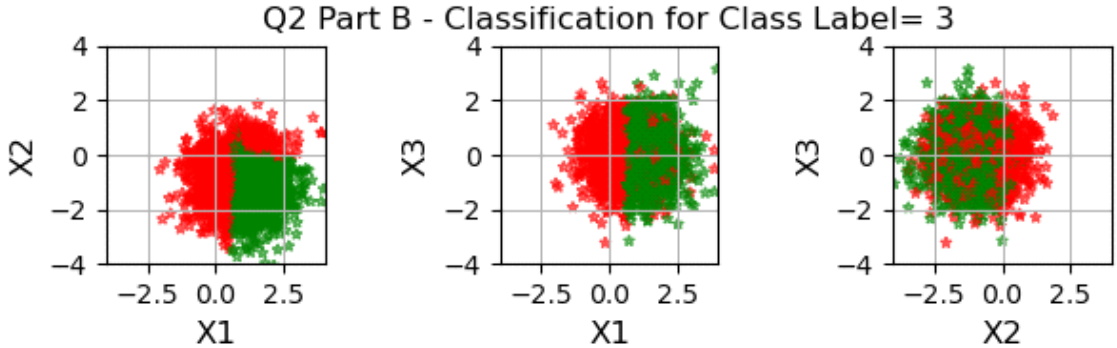


Part B

Class Label 3

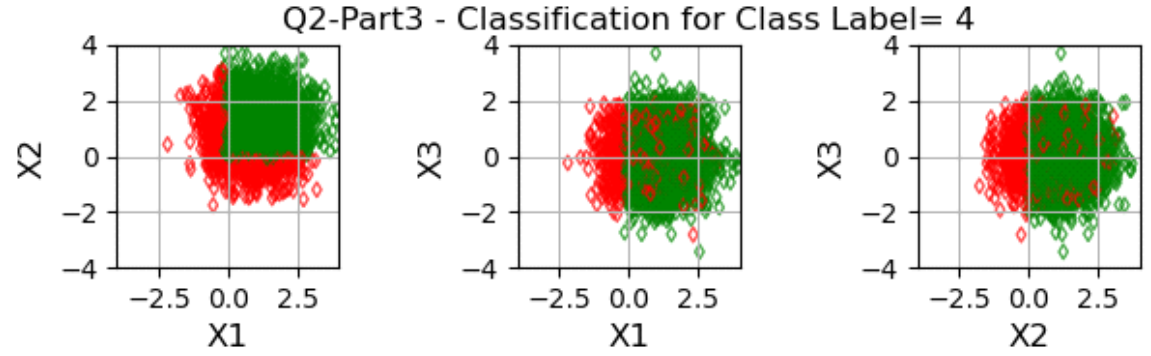


Part A

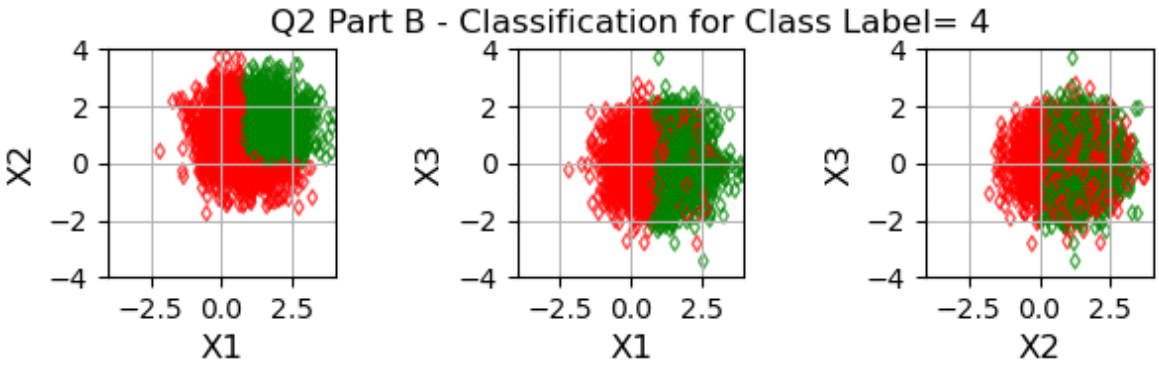


Part B

Class Label 4



Part A



Part B

Visual Observations: let N_{MC} represent the number of misclassified data points per plot

- Class Label 1: N_{MC} from part A $>$ N_{MC} from part B

This would suggest that the risk of misclassifying a data from class label 1 as belonging to a different class label is high in part A

- Class Label 2: N_{MC} from part A $<$ N_{MC} from part B

This would suggest that misclassifying a data from class label 2 has a lower risk in part B, compared to the other class labels.

- Class Label 3: N_{MC} from part A $<$ N_{MC} from part B

This would suggest that misclassifying a data from class label 3 has a lower risk in part B, compared to other class labels.

- Class Label 4: N_{MC} from part A $<$ N_{MC} from part B

This would suggest that misclassifying a data from class label 4 has a lower risk in part B, compared to other class labels.

=> Based on these visual observations, we would expect the risk of misclassifying a data from class label 1 to carry the highest risk among all 4 class labels.

2. Numerical Representation:

One way to look at the loss matrix is as follows:

$$\text{Decision}\left[\Lambda = \overbrace{\begin{bmatrix} 0 & 1 & 2 & 3 \\ 10 & 0 & 5 & 10 \\ 20 & 10 & 0 & 1 \\ 30 & 20 & 1 & 0 \end{bmatrix}}^{\text{Class label}}\right.$$

So the entry of the loss matrix for deciding $D=2$, when the real class label $L=1$, is 20

Knowing the above, we can compare the number of correct classification and misclassification for each class label from part A and part B.

The results below are from calculating the percentage of correctly and incorrectly classifying each class label based on loss matrices of parts A and B, respectively.

```
Numerical Representation Comparison for Part A & Part B:

Class Label 1:
Correct Label   - Part A: 73.9%   Part B: 96.4%
Incorrect Label - Part A: 26.1%   Part B: 3.6%

Class Label 2:
Correct Label   - Part A: 77.5%   Part B: 39.9%
Incorrect Label - Part A: 22.5%   Part B: 60.1%

Class Label 3:
Correct Label   - Part A: 75.3%   Part B: 62.9%
Incorrect Label - Part A: 24.7%   Part B: 37.1%

Class Label 4:
Correct Label   - Part A: 80.8%   Part B: 52.4%
Incorrect Label - Part A: 19.2%   Part B:47.6%
```

From these results, we see that using the loss matrix provided in part B, increases the percentage of correctly classified class label 1 data by over 20%

From these results, we see that using the loss matrix provided in part B, increases the percentage of correctly classified class label 1 data by over 20%. This, of course, comes at the cost (risk) of misclassifying the other 3 class labels. As we see from the table the percentage of misclassified data points for class labels 2, 3, and 4 are higher using the loss matrix from part B.

Begin Appendix on next page:

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Oct  2 18:56:24 2020

@author: Mahdi
"""
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.stats import multivariate_normal
from scipy.linalg import eigh
######################################################################
#                           Question 1                               #
######################################################################

#------------ Setup the Given vectors and matrices ----------#

## Number of samples and dimensions ##
n= 4; #Number of dimensions
N= 10000; #Number of samples simulated
##---------------------------------##

## Class means and covariance ##
Sigma_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]]);

Sigma_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]]);

Mu_0=np.array([-1, -1, -1, -1]);
Mu_1=np.array([ 1,  1,  1,  1]);
##---------------------------##

## Class priors ##
p=np.array([0.7, 0.3]); #First component for p(L=0), second component for p(L=1)
##--------------##

## Generate samples based on class conditional pdfs ## !!! ONLY RUN ONCE AND THEN COMMENT OUT
label=(np.random.random(N)>=p[1]).astype(int);
Sum_L1=sum(On for On in label if On ==1) ;
Sum_L0=N-Sum_L1 ;
X = np.zeros((n, N))
L0_Counter=0;
L1_Counter=0;
#
Data_Set_L0= np.random.multivariate_normal(Mu_0,Sigma_0,Sum_L0);
Data_Set_L1= np.random.multivariate_normal(Mu_1,Sigma_1,Sum_L1);
for i in range(0,N):
    if label[i]==0:
        X[:,i]= np.transpose(Data_Set_L0[L0_Counter,:]);
        L0_Counter=L0_Counter+1;
    else:
        X[:,i]= np.transpose(Data_Set_L1[L1_Counter,:]);
```

```
        L1_Counter=L1_Counter+1;
#np.save("HW1_1000Samples_X_values",X,allow_pickle=True)
#np.save("HW1_1000Samples_label_values",label,allow_pickle=True)

#X = np.load("HW1_1000Samples_X_values.npy")
#label = np.load("HW1_1000Samples_label_values.npy")
#Sum_L1=sum(On for On in label if On ==1) ;
#Sum_L0=N-Sum_L1 ;
Fig_Num=1;
plt.figure(Fig_Num)

plt.subplot(211)
plt.scatter(Data_Set_L0[:,0],Data_Set_L0[:,1],2,'red')
plt.scatter(Data_Set_L1[:,0],Data_Set_L1[:,1],2,'blue')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.title('2-D Representation of the Training Data')

plt.subplot(212)
plt.scatter(Data_Set_L0[:,2],Data_Set_L0[:,3],2,'red')
plt.scatter(Data_Set_L1[:,2],Data_Set_L1[:,3],2,'blue')
plt.xlabel('Dimension 3')
plt.ylabel('Dimension 4')
print("Finished Q1 Section-A Part-1");
##-------------------------------------------------##
#---------- END Setup the  Given vectors and matrices --------#

##------- Q1 Part 2---------##
#---- Calculate Discriminant Score ----#
def evalGaussian(x, Mu, Sigma):
    g=multivariate_normal.pdf(x,Mu,Sigma);
    return g
discriminantScore=np.zeros(N);

for i in range(0,N):
     discriminantScore[i] = (evalGaussian(X[:,i],Mu_1[:],Sigma_1[:,:])/evalGaussian(X[:,i],Mu_0[:],
     # If discriminantScore > gamma, D=1, otherwise, D=0
#- END Calculate Discriminant Score --#
gamma = np.sort(discriminantScore);
eps=1;
mid_gamma=np.zeros(N);
mid_gamma[0]=gamma[0]-eps;
mid_gamma[N-1]=gamma[N-1]+eps;
for i in range(1,N-1):
    mid_gamma[i]=(gamma[i]+gamma[i+1])/2;

SZ= np.size(mid_gamma);
pTN=np.zeros(SZ);
pFN=np.zeros(SZ);
pTP=np.zeros(SZ);
pFP=np.zeros(SZ);
Area_2=np.zeros(1);


for i in range(0,SZ):
    Decision=np.zeros(N);
    TP = np.zeros(1); #Number of Correct Decisions/ True Positive
```

```
    FP =np.zeros(1); #Number of False Alarms/False Positive
    TN=np.zeros(1); #Number of True Negatives
    FN=np.zeros(1); #Number of False Negatives
    for j in range(0,N):
        Decision[j]=(discriminantScore[j]>=mid_gamma[i]);
        if Decision[j]==True and label[j]==1:
            TP = TP+1;
        elif  Decision[j]==True and label[j]==0:
             FP=FP+1;
        elif  Decision[j]==False and label[j]==0:
             TN=TN+1;
        else:
             FN=FN+1;

    pFP[i]=FP/Sum_L0;
    pTP[i]=TP/Sum_L1;
    pFN[i]=FN/Sum_L1;
    pTN[i]=TN/Sum_L0;

Fig_Num=Fig_Num+1;
plt.figure(Fig_Num)
plt.plot(pFP,pTP,c='blue')
plt.xlabel("pFA")
plt.ylabel("pTP")
plt.title("ROC curve for Q1 part 2")

plt.show()

## Calculate area under ROC curve
Flip_pTP=np.flip(pTP);
Flip_pFP=np.flip(pFP);
Area_Q1_P2=0;
for i in range(1,SZ):
    Area_Q1_P2= Area_Q1_P2 + Flip_pTP[i]*(Flip_pFP[i]-Flip_pFP[i-1]);
print("Area under ROC curve for Q1 Section A is: "+str(Area_Q1_P2))
print("Finished Q1 Section-A Part-2");

print("Finished Q1 Section-A Part-2");
##------ END Q1 Part 2 ------##

##-------- Q1 Part 3 ---------##
#Calculate  the  probability of error as a function of gamma
P_Err=np.zeros(SZ);
for i in range(0,SZ):
    P_Err[i]=pFP[i]*p[0]+pFN[i]*p[1];

Min_P_Err=min(P_Err);

Min_P_Err_ind = [i for i, v in enumerate(P_Err) if v == Min_P_Err];
Best_Gamma=mid_gamma[Min_P_Err_ind];
plt.figure(Fig_Num)
plt.plot(pFP,pTP,c='blue')
plt.xlabel("pFA")
plt.ylabel("pTP")
plt.title("ROC curve for Q1 part 3")
plt.scatter(pFP[Min_P_Err_ind],pTP[Min_P_Err_ind],50,'#d62728')
plt.show()
```

```
print("Estimate of the minimum P_Error= "+str(Min_P_Err)+" with threshold value= "+str(Best_Gamma)+
print("Finished Q1 Section-A Part-3");
##------ END Q1 Part 3 -------##
##-----------------------------END Q1 Section A ------------------------##


##------------------------------- Q1 Section B --------------------------##
## Redefine Covariance Matrix ##
B_Sigma_0=np.array([[2,    0,    0,   0],
                    [0,    1,    0,   0],
                    [0,    0,    1,   0],
                    [0,    0,    0,   2]]);

B_Sigma_1=np.array([[1,    0,    0,   0],
                    [0,    2,    0,   0],
                    [0,    0,    1,   0],
                    [0,    0,    0,   3]]);
##---------------------------##
B_discriminantScore=np.zeros(N);

for i in range(0,N):
     B_discriminantScore[i] = (evalGaussian(X[:,i],Mu_1[:],B_Sigma_1[:,:])/evalGaussian(X[:,i],Mu_0
     # If discriminantScore > gamma, D=1, otherwise, D=0
#- END Calculate Discriminant Score --#
B_gamma = np.sort(B_discriminantScore);
eps=1;
B_mid_gamma=np.zeros(N);
B_mid_gamma[0]=B_gamma[0]-eps;
B_mid_gamma[N-1]=B_gamma[N-1]+eps;
for i in range(0,N-1):
    B_mid_gamma[i]=(B_gamma[i]+B_gamma[i+1])/2;

B_SZ= np.size(B_mid_gamma);
B_pTN=np.zeros(B_SZ);
B_pFN=np.zeros(B_SZ);
B_pTP=np.zeros(B_SZ);
B_pFP=np.zeros(B_SZ);

for i in range(0,B_SZ):
    B_Decision=np.zeros(N);
    B_TP = np.zeros(1); #Number of Correct Decisions/ True Positive
    B_FP =np.zeros(1); #Number of False Alarms/False Positive
    B_TN=np.zeros(1); #Number of True Negatives
    B_FN=np.zeros(1); #Number of False Negatives
    for j in range(0,N):
        B_Decision[j]=(B_discriminantScore[j]>=B_mid_gamma[i]);
        if B_Decision[j]==1 and label[j]==1:
            B_TP = B_TP+1;
        elif  B_Decision[j]==1 and label[j]==0:
             B_FP=B_FP+1;
        elif  B_Decision[j]==0 and label[j]==0:
             B_TN=B_TN+1;
        else:
             B_FN=B_FN+1;
```

```
    B_pFP[i]=B_FP/Sum_L0;
    B_pTP[i]=B_TP/Sum_L1;
    B_pFN[i]=B_FN/Sum_L1;
    B_pTN[i]=B_TN/Sum_L0;

Fig_Num=Fig_Num+1;
plt.figure(Fig_Num)
plt.plot(B_pFP,B_pTP,c='red')
plt.xlabel("pFA")
plt.ylabel("pTP")
plt.title("ROC curve for Q1 Section B")
plt.show()

#Calculate  the  probability of error as a function of gamma
B_P_Err=np.zeros(B_SZ);
for i in range(0,B_SZ):
    B_P_Err[i]=B_pFP[i]*p[0]+B_pFN[i]*p[1];

B_Min_P_Err=min(B_P_Err);

B_Min_P_Err_ind = [i for i, v in enumerate(B_P_Err) if v == B_Min_P_Err];
B_Best_Gamma=B_mid_gamma[B_Min_P_Err_ind];

plt.figure(Fig_Num)
plt.plot(B_pFP,B_pTP,c='red')
plt.xlabel("pFA")
plt.ylabel("pTP")
plt.title("ROC curve for Q1 Section B")
plt.scatter(B_pFP[B_Min_P_Err_ind],B_pTP[B_Min_P_Err_ind],50,'#002728')
plt.show()

## Calculate area under ROC curve
Flip_B_pTP=np.flip(B_pTP);
Flip_B_pFP=np.flip(B_pFP);
Area_Q1_SB=0;
for i in range(1,B_SZ):
    Area_Q1_SB= Area_Q1_SB + Flip_B_pTP[i]*(Flip_B_pFP[i]-Flip_B_pFP[i-1]);
print("Area under ROC curve for Q1 Section B is: "+str(Area_Q1_SB))

print("Estimate of the minimum P_Error= "+str(B_Min_P_Err)+" with threshold value= "+str(B_Best_Gam
print("Finished Q1 Section-B");
##------------------------------ END Q1 Section B ------------------------##

##------------------------------- Q1 Section C --------------------------##
### Estimate Class Conditional pdfs mean and covariance from the sampled data
Est_Mu_0=np.zeros(n);
Est_Sigma_0=np.zeros((n,n));
Est_Mu_1=np.zeros(n);
Est_Sigma_1=np.zeros((n,n));
S_B=np.zeros((n,n));
S_W=np.zeros((n,n));
L0_Index=np.flatnonzero(label == 0);
L1_Index=np.flatnonzero(label == 1);
X_L0=X[:,L0_Index];
X_L1=X[:,L1_Index];
for i in range(0,n):
    Est_Mu_0[i]=np.mean(X_L0[i,:]);
```

```
    Est_Mu_1[i]=np.mean(X_L1[i,:]);
Est_Sigma_0=np.cov(X_L0);
Est_Sigma_1=np.cov(X_L1);
### Calculate Between and Within class scatter
for i in range(len(Est_Mu_0)):
   for j in range(len(Est_Mu_1)):
       S_B[i][j] = (Est_Mu_0[i]-Est_Mu_1[i]) * (Est_Mu_0[j]-Est_Mu_1[j])
S_W=Est_Sigma_0+Est_Sigma_1;
eigvals, eigvecs = eigh(S_B, S_W,eigvals_only=False);

Max_eigval=max(eigvals);
Max_eigval_ind = [i for i, v in enumerate(eigvals) if v == Max_eigval];
Max_eigvec=eigvecs[:,Max_eigval_ind];

### Project X onto w_transpose to obtain y
y = np.zeros(N);
Color = [];
for i in range(0,N):
    y[i]=np.transpose(Max_eigvec)@X[:,i];
    if label[i]==0:
        Color.append('red');
    else:
        Color.append('blue');

Fig_Num=Fig_Num+1;
plt.figure(Fig_Num)
plt.scatter(np.linspace(0, N-1, N),y,c=Color,s=2)
plt.xlabel("Data Point")
plt.ylabel("Y")

##Calculate pTP pFP pTN pFN
C_gamma = np.sort(y);
eps=1;
C_mid_gamma=np.zeros(N);
C_mid_gamma[0]=C_gamma[0]-eps;
C_mid_gamma[N-1]=C_gamma[N-1]+eps;
for i in range(0,N-1):
    C_mid_gamma[i]=(C_gamma[i]+C_gamma[i+1])/2;

C_SZ= np.size(C_mid_gamma);
C_pTN=np.zeros(C_SZ);
C_pFN=np.zeros(C_SZ);
C_pTP=np.zeros(C_SZ);
C_pFP=np.zeros(C_SZ);

for i in range(0,C_SZ):
    C_Decision=np.zeros(N);
    C_TP = np.zeros(1); #Number of Correct Decisions/ True Positive
    C_FP =np.zeros(1); #Number of False Alarms/False Positive
    C_TN=np.zeros(1); #Number of True Negatives
    C_FN=np.zeros(1); #Number of False Negatives
    for j in range(0,N):
        C_Decision[j]=(y[j]>=C_mid_gamma[i]);
        if C_Decision[j]==1 and label[j]==1:
            C_TP = C_TP+1;
        elif  C_Decision[j]==1 and label[j]==0:
             C_FP=C_FP+1;
```

```
        elif  C_Decision[j]==0 and label[j]==0:
             C_TN=C_TN+1;
        else:
             C_FN=C_FN+1;

    C_pFP[i]=C_FP/Sum_L0;
    C_pTP[i]=C_TP/Sum_L1;
    C_pFN[i]=C_FN/Sum_L1;
    C_pTN[i]=C_TN/Sum_L0;

#Calculate  the  probability of error as a function of gamma
C_P_Err=np.zeros(C_SZ);
for i in range(0,C_SZ):
    C_P_Err[i]=C_pFP[i]*p[0]+C_pFN[i]*p[1];

C_Min_P_Err=min(C_P_Err);

C_Min_P_Err_ind = [i for i, v in enumerate(C_P_Err) if v == C_Min_P_Err];
C_Best_Gamma=C_mid_gamma[C_Min_P_Err_ind];

Fig_Num=Fig_Num+1;
plt.figure(Fig_Num)
plt.plot(C_pFP,C_pTP,c='green')
plt.xlabel("pFA")
plt.ylabel("pTP")
plt.title("ROC curve for Q1 Section C")
plt.scatter(C_pFP[C_Min_P_Err_ind],C_pTP[C_Min_P_Err_ind],50,'#d60028')
plt.show()

## Calculate area under ROC curve
Flip_C_pTP=np.flip(C_pTP);
Flip_C_pFP=np.flip(C_pFP);
Area_Q1_SC=0;
for i in range(1,C_SZ):
    Area_Q1_SC= Area_Q1_SC + Flip_C_pTP[i]*(Flip_C_pFP[i]-Flip_C_pFP[i-1]);
print("Area under ROC curve for Q1 Section C is: "+str(Area_Q1_SC))

print("Estimate of the minimum P_Error= "+str(C_Min_P_Err)+" with threshold value= "+str(C_Best_Gam

print("Finished Q1 Section-C");


Fig_Num=Fig_Num+1;
plt.figure(Fig_Num)
plt.plot(pFP,pTP,c='blue')
plt.plot(B_pFP,B_pTP,c='red')
plt.plot(C_pFP,C_pTP,c='green')
#plt.legend(['Q1-A','Q1-B','Q1-C'])
plt.scatter(B_pFP[B_Min_P_Err_ind],B_pTP[B_Min_P_Err_ind],50,'#002728')
plt.scatter(pFP[Min_P_Err_ind],pTP[Min_P_Err_ind],50,'#d62728')
plt.scatter(C_pFP[C_Min_P_Err_ind],C_pTP[C_Min_P_Err_ind],50,'#d60028')

plt.xlabel("pFA")
plt.ylabel("pTP")
```

```
# -*- coding: utf-8 -*-
"""
Created on Sat Oct 10 18:57:04 2020

@author: Mahdi
"""

import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.stats import multivariate_normal
from scipy.linalg import eigh

######################################################################
#                           Question 2                               #
######################################################################

#------------ Setup the Given vectors and matrices ----------#

## Number of samples and dimensions ##
n= 3; #Number of dimensions
N= 10000; #Number of samples simulated
C= 4; #Number of classes
##---------------------------------##

## Class means and covariance ##
Sigma=np.array([[0.75,       0,       0],
                [  0,     0.75,       0],
                [  0,       0,     0.75]]);
Mu=np.zeros((C,n));
Mu[0,:]=np.array([-1,  1, 0]);
Mu[1,:]=np.array([-1, -1, 0]);
Mu[2,:]=np.array([ 1, -1, 0]);
Mu[3,:]=np.array([ 1,  1, 0]);
##---------------------------##

## Class priors ##
p=np.array([0.2, 0.25, 0.25, 0.3]); #p[0]=class 1 prior, p[1]=class 2 prior, p[2]=class 3 prior, p[
##--------------##

## Generate samples based on class conditional pdfs ## !!! ONLY RUN ONCE AND THEN COMMENT OUT
N_random_numbers=np.random.random(N);
label=np.zeros(N);
for i in range(0,N):
    if N_random_numbers[i]<=p[0]:
        label[i]=1;
    elif N_random_numbers[i]<=(p[0]+p[1]):
            label[i]=2;
    elif N_random_numbers[i]<=(p[0]+p[1]+p[2]):
            label[i]=3;
    else:
        label[i]=4;
Sum_CL=np.zeros(C)
for i in range(0,C):
    Sum_CL[i]=sum(label==(i+1)); #Number of data points with the given class label

X = np.zeros((n, N))
```

```
L_Counter=np.zeros(C);

Data_Set_L=np.zeros((C,N,n))
for i in range(0,C):
    temp_Mu=np.zeros(n)
    temp_Mu[:]=Mu[i,:]
    Data_Set_L[i,0:int(Sum_CL[i]),:]= np.random.multivariate_normal(temp_Mu,Sigma,int(Sum_CL[i]));

for i in range(0,N):
    for CLASS in range(0,C):
        if label[i]==CLASS+1:
            X[:,i]= np.transpose(Data_Set_L[CLASS,int(L_Counter[CLASS]),:]);
            L_Counter[CLASS]=L_Counter[CLASS]+1;

Fig_Num=1;
LIM=4;

fig=plt.figure(Fig_Num)
ax = fig.add_subplot(111, projection='3d')

m=['o','s','*','d']

ax.scatter(Data_Set_L[0,:,0], Data_Set_L[0,:,1], Data_Set_L[0,:,2], c='#8c564b', marker=m[0], s=30,
ax.scatter(Data_Set_L[1,:,0], Data_Set_L[1,:,1], Data_Set_L[1,:,2], c='#2ca02c', marker=m[1], s=30,
ax.scatter(Data_Set_L[2,:,0], Data_Set_L[2,:,1], Data_Set_L[2,:,2], c='#17becf', marker=m[2], s=30,
ax.scatter(Data_Set_L[3,:,0], Data_Set_L[3,:,1], Data_Set_L[3,:,2], c='#e377c2', marker=m[3], s=30,
# ax.legend(['Class 1', 'Class 2', 'Class 3', 'Class 4'])
ax.set_xlabel("X1")
ax.set_ylabel("X2")
ax.set_zlabel("X3")
ax.set_xlim((-LIM, LIM))
ax.set_ylim((-LIM, LIM))
ax.set_zlim((-LIM, LIM))

#---------- END Setup the Given vectors and matrices --------#

#Define Loss Matrix
lossMatrix= np.ones(C)-np.identity(C);

#---- Calculate Expected Risk and decide on the minimum one ----#
def evalGaussian(x, Mu, Sigma):
    g=multivariate_normal.pdf(x,Mu,Sigma);
    return g

pxgivenL=np.zeros((C,N))
Class_Posterior=np.zeros((C,N))
Expected_Risk=np.zeros((C,N))
Decision=np.zeros(N)
for i in range(0,C):
    for j in range(0,N):
        pxgivenL[i,j] = evalGaussian(X[:,j],Mu[i,:],Sigma[:,:]); #Evaluate p(x|L=i)         !!DOES
    Class_Posterior[i,:]=pxgivenL[i,:]*p[i]; #This is not really class posterior, we must divide by

for j in range(0,N):
    Expected_Risk[:,j]=lossMatrix@Class_Posterior[:,j];
    Decision[j]= np.argmin(Expected_Risk[:,j])+1;
#- END Calculate Expected Risk --#
```

```
##Calculate Confusion Matrix
Confusion_Matrix=np.zeros((C,C));
Sum_DL=np.zeros((C,C));
for D in range(0,C):
        for L in range(0,C):
            Sum_DL[D,L]=sum(np.logical_and(Decision==D+1, label==L+1)); #Number of data points with
            Confusion_Matrix[D,L]=Sum_DL[D,L]/Sum_CL[L]; #L-1 since python indexing starts at 0

print("Q2 Part 2 - The Confusion Matrix is: ")
print(str(Confusion_Matrix))
print("Finished Q2 part 2")

Color=[]
COLOR_L1=[]
COLOR_L2=[]
COLOR_L3=[]
COLOR_L4=[]

Indeces_L1=np.zeros(int(Sum_CL[0]),dtype=int); #Contains the index numbers of Dataset X that has la
Indeces_L2=np.zeros(int(Sum_CL[1]),dtype=int);
Indeces_L3=np.zeros(int(Sum_CL[2]),dtype=int);
Indeces_L4=np.zeros(int(Sum_CL[3]),dtype=int);
L_Counter=np.zeros(C,dtype=int);
for i in range(0,N):
    if Decision[i]==label[i]:
        Color.append('green');
    else:
        Color.append('red');
    if label[i]==1:
        Indeces_L1[L_Counter[0]]=i;
        L_Counter[0]=L_Counter[0]+1;
        COLOR_L1.append(Color[i]);
    elif label[i]==2:
        Indeces_L2[L_Counter[1]]=i;
        L_Counter[1]=L_Counter[1]+1;
        COLOR_L2.append(Color[i]);
    elif label[i]==3:
        Indeces_L3[L_Counter[2]]=i;
        L_Counter[2]=L_Counter[2]+1;
        COLOR_L3.append(Color[i]);
    else:
        Indeces_L4[L_Counter[3]]=i;
        L_Counter[3]=L_Counter[3]+1;
        COLOR_L4.append(Color[i]);

Fig_Num=Fig_Num+1;
plt.figure(Fig_Num)
plt.subplot(311)
plt.scatter(X[0,Indeces_L1],X[1,Indeces_L1],facecolors='none', edgecolors=COLOR_L1,marker=m[0], s=2
plt.scatter(X[0,Indeces_L2],X[1,Indeces_L2],facecolors='none', edgecolors=COLOR_L2,marker=m[1], s=2
plt.scatter(X[0,Indeces_L3],X[1,Indeces_L3],c=COLOR_L3,marker=m[2], s=25, alpha=0.7)
plt.scatter(X[0,Indeces_L4],X[1,Indeces_L4],facecolors='none', edgecolors=COLOR_L4,marker=m[3], s=2
plt.title('Q2-Part3 - Classification of all 4 classes')
plt.xlabel('X1',fontsize=12)
plt.ylabel('X2',fontsize=12)
plt.xlim((-LIM, LIM))
```

```
plt.ylim((-LIM, LIM))
plt.axis('square')

plt.subplot(312)
plt.scatter(X[0,Indeces_L1],X[2,Indeces_L1],facecolors='none', edgecolors=COLOR_L1,marker=m[0], s=2
plt.scatter(X[0,Indeces_L2],X[2,Indeces_L2],facecolors='none', edgecolors=COLOR_L2,marker=m[1], s=2
plt.scatter(X[0,Indeces_L3],X[2,Indeces_L3],c=COLOR_L3,marker=m[2], s=25, alpha=0.7)
plt.scatter(X[0,Indeces_L4],X[2,Indeces_L4],facecolors='none', edgecolors=COLOR_L4,marker=m[3], s=2
plt.xlabel('X1',fontsize=12)
plt.ylabel('X3',fontsize=12)
plt.xlim((-LIM, LIM))
plt.ylim((-LIM, LIM))
plt.axis('square')

plt.subplot(313)
plt.scatter(X[1,Indeces_L1],X[2,Indeces_L1],facecolors='none', edgecolors=COLOR_L1,marker=m[0], s=2
plt.scatter(X[1,Indeces_L2],X[2,Indeces_L2],facecolors='none', edgecolors=COLOR_L2,marker=m[1], s=2
plt.scatter(X[1,Indeces_L3],X[2,Indeces_L3],c=COLOR_L3,marker=m[2], s=25, alpha=0.7)
plt.scatter(X[1,Indeces_L4],X[2,Indeces_L4],facecolors='none', edgecolors=COLOR_L4,marker=m[3], s=2
plt.xlabel('X2',fontsize=12)
plt.ylabel('X3',fontsize=12)
plt.xlim((-LIM, LIM))
plt.ylim((-LIM, LIM))
plt.axis('square')

Fig_Num=Fig_Num+1;
plt.figure(Fig_Num)
plt.subplots_adjust(hspace=0.8, wspace=0.8)
for i in range(0,C):
    if i==0:
        SUBPLOT=1;
        Scat_Color=COLOR_L1
        Scat_Indeces=Indeces_L1
        Marker=m[0]
    elif i==1:
        SUBPLOT=4;
        Scat_Color=COLOR_L2
        Scat_Indeces=Indeces_L2
        Marker=m[1]
    elif i==2:
        SUBPLOT=7;
        Scat_Color=COLOR_L3
        Scat_Indeces=Indeces_L3
        Marker=m[2]
    else:
        SUBPLOT=10;
        Scat_Color=COLOR_L4
        Scat_Indeces=Indeces_L4
        Marker=m[3]

    plt.subplot(4,3,SUBPLOT)
    plt.scatter(X[0,Scat_Indeces],X[1,Scat_Indeces],facecolors='none', edgecolors=Scat_Color,marker

    plt.xlabel('X1',fontsize=12)
    plt.ylabel('X2',fontsize=12)
    plt.axis('square')
    plt.xlim((-LIM, LIM))
```

```
    plt.ylim((-LIM, LIM))
    plt.grid(b=True,which='both',axis='both')

    plt.subplot(4,3,SUBPLOT+1)
    plt.title("Q2-Part3 - Classification for Class Label= "+str(int(i+1)))
    plt.scatter(X[0,Scat_Indeces],X[2,Scat_Indeces],facecolors='none', edgecolors=Scat_Color,marker
    plt.xlabel('X1',fontsize=12)
    plt.ylabel('X3',fontsize=12)
    plt.axis('square')
    plt.xlim((-LIM, LIM))
    plt.ylim((-LIM, LIM))
    plt.grid(b=True,which='both',axis='both')

    plt.subplot(4,3,SUBPLOT+2)
    plt.scatter(X[1,Scat_Indeces],X[2,Scat_Indeces],facecolors='none', edgecolors=Scat_Color,marker
    plt.xlabel('X2',fontsize=12)
    plt.ylabel('X3',fontsize=12)
    plt.axis('square')
    plt.xlim((-LIM, LIM))
    plt.ylim((-LIM, LIM))
    plt.grid(b=True,which='both',axis='both')


print("Finished Q2 part 3")

#Define Loss Matrix
lossMatrix=np.array([[  0,     1,    2,     3],
                     [ 10,     0,    5,    10],
                     [ 20,    10,    0,     1],
                     [ 30,    20,    1,     0]]);

#---- Calculate Expected Risk and decide on the minimum one ----#
def evalGaussian(x, Mu, Sigma):
    g=multivariate_normal.pdf(x,Mu,Sigma);
    return g

pxgivenL=np.zeros((C,N))
Class_Posterior=np.zeros((C,N))
Expected_Risk=np.zeros((C,N))
Decision=np.zeros(N)
for i in range(0,C):
    for j in range(0,N):
        pxgivenL[i,j] = evalGaussian(X[:,j],Mu[i,:],Sigma[:,:]); #Evaluate p(x|L=i)         !!DOES
    Class_Posterior[i,:]=pxgivenL[i,:]*p[i]; #This is not really class posterior, we must divide by

for j in range(0,N):
    Expected_Risk[:,j]=lossMatrix@Class_Posterior[:,j];
    Decision[j]= np.argmin(Expected_Risk[:,j])+1;
#- END Calculate Expected Risk --#

##Calculate Confusion Matrix
Confusion_Matrix=np.zeros((C,C));
Sum_DL=np.zeros((C,C));
for D in range(0,C):
        for L in range(0,C):
            Sum_DL[D,L]=sum(np.logical_and(Decision==D+1, label==L+1)); #Number of data points with
            Confusion_Matrix[D,L]=Sum_DL[D,L]/Sum_CL[L]; #L-1 since python indexing starts at 0
```

```
print("Q2 Part B - The Confusion Matrix is: ")
print(str(Confusion_Matrix))


B_Color=[]
B_COLOR_L1=[]
B_COLOR_L2=[]
B_COLOR_L3=[]
B_COLOR_L4=[]

Indeces_L1=np.zeros(int(Sum_CL[0]),dtype=int); #Contains the index numbers of Dataset X that has lo
Indeces_L2=np.zeros(int(Sum_CL[1]),dtype=int);
Indeces_L3=np.zeros(int(Sum_CL[2]),dtype=int);
Indeces_L4=np.zeros(int(Sum_CL[3]),dtype=int);
L_Counter=np.zeros(C,dtype=int);
for i in range(0,N):
    if Decision[i]==label[i]:
        B_Color.append('green');
    else:
        B_Color.append('red');
    if label[i]==1:
        Indeces_L1[L_Counter[0]]=i;
        L_Counter[0]=L_Counter[0]+1;
        B_COLOR_L1.append(B_Color[i]);
    elif label[i]==2:
        Indeces_L2[L_Counter[1]]=i;
        L_Counter[1]=L_Counter[1]+1;
        B_COLOR_L2.append(B_Color[i]);
    elif label[i]==3:
        Indeces_L3[L_Counter[2]]=i;
        L_Counter[2]=L_Counter[2]+1;
        B_COLOR_L3.append(B_Color[i]);
    else:
        Indeces_L4[L_Counter[3]]=i;
        L_Counter[3]=L_Counter[3]+1;
        B_COLOR_L4.append(B_Color[i]);

Fig_Num=Fig_Num+1;
plt.figure(Fig_Num)
plt.subplot(311)
plt.scatter(X[0,Indeces_L1],X[1,Indeces_L1],facecolors='none', edgecolors=B_COLOR_L1,marker=m[0], s
plt.scatter(X[0,Indeces_L2],X[1,Indeces_L2],facecolors='none', edgecolors=B_COLOR_L2,marker=m[1], s
plt.scatter(X[0,Indeces_L3],X[1,Indeces_L3],c=B_COLOR_L3,marker=m[2], s=25, alpha=0.7)
plt.scatter(X[0,Indeces_L4],X[1,Indeces_L4],facecolors='none', edgecolors=B_COLOR_L4,marker=m[3], s
plt.title('Q2 Part B - Classification of all 4 classes')
plt.xlabel('X1',fontsize=12)
plt.ylabel('X2',fontsize=12)
plt.xlim((-LIM, LIM))
plt.ylim((-LIM, LIM))
plt.axis('square')

plt.subplot(312)
plt.scatter(X[0,Indeces_L1],X[2,Indeces_L1],facecolors='none', edgecolors=B_COLOR_L1,marker=m[0], s
plt.scatter(X[0,Indeces_L2],X[2,Indeces_L2],facecolors='none', edgecolors=B_COLOR_L2,marker=m[1], s
plt.scatter(X[0,Indeces_L3],X[2,Indeces_L3],c=B_COLOR_L3,marker=m[2], s=25, alpha=0.7)
plt.scatter(X[0,Indeces_L4],X[2,Indeces_L4],facecolors='none', edgecolors=B_COLOR_L4,marker=m[3], s
```

```
plt.xlabel('X1',fontsize=12)
plt.ylabel('X3',fontsize=12)
plt.xlim((-LIM, LIM))
plt.ylim((-LIM, LIM))
plt.axis('square')

plt.subplot(313)
plt.scatter(X[1,Indeces_L1],X[2,Indeces_L1],facecolors='none', edgecolors=B_COLOR_L1,marker=m[0], s
plt.scatter(X[1,Indeces_L2],X[2,Indeces_L2],facecolors='none', edgecolors=B_COLOR_L2,marker=m[1], s
plt.scatter(X[1,Indeces_L3],X[2,Indeces_L3],c=B_COLOR_L3,marker=m[2], s=25, alpha=0.7)
plt.scatter(X[1,Indeces_L4],X[2,Indeces_L4],facecolors='none', edgecolors=B_COLOR_L4,marker=m[3], s
plt.xlabel('X2',fontsize=12)
plt.ylabel('X3',fontsize=12)
plt.xlim((-LIM, LIM))
plt.ylim((-LIM, LIM))
plt.axis('square')

Fig_Num=Fig_Num+1;
plt.figure(Fig_Num)
plt.subplots_adjust(hspace=0.8, wspace=0.8)
for i in range(0,C):
    if i==0:
        SUBPLOT=1;
        Scat_Color=B_COLOR_L1
        Scat_Indeces=Indeces_L1
        Marker=m[0]
    elif i==1:
        SUBPLOT=4;
        Scat_Color=B_COLOR_L2
        Scat_Indeces=Indeces_L2
        Marker=m[1]
    elif i==2:
        SUBPLOT=7;
        Scat_Color=B_COLOR_L3
        Scat_Indeces=Indeces_L3
        Marker=m[2]
    else:
        SUBPLOT=10;
        Scat_Color=B_COLOR_L4
        Scat_Indeces=Indeces_L4
        Marker=m[3]

    plt.subplot(4,3,SUBPLOT)
    plt.scatter(X[0,Scat_Indeces],X[1,Scat_Indeces],facecolors='none', edgecolors=Scat_Color,marker

    plt.xlabel('X1',fontsize=12)
    plt.ylabel('X2',fontsize=12)
    plt.axis('square')
    plt.xlim((-LIM, LIM))
    plt.ylim((-LIM, LIM))
    plt.grid(b=True,which='both',axis='both')

    plt.subplot(4,3,SUBPLOT+1)
    plt.title("Q2 Part B - Classification for Class Label= "+str(int(i+1)))
    plt.scatter(X[0,Scat_Indeces],X[2,Scat_Indeces],facecolors='none', edgecolors=Scat_Color,marker
    plt.xlabel('X1',fontsize=12)
    plt.ylabel('X3',fontsize=12)
```

```
    plt.axis('square')
    plt.xlim((-LIM, LIM))
    plt.ylim((-LIM, LIM))
    plt.grid(b=True,which='both',axis='both')

    plt.subplot(4,3,SUBPLOT+2)
    plt.scatter(X[1,Scat_Indeces],X[2,Scat_Indeces],facecolors='none', edgecolors=Scat_Color,marker
    plt.xlabel('X2',fontsize=12)
    plt.ylabel('X3',fontsize=12)

    plt.axis('square')
    plt.xlim((-LIM, LIM))
    plt.ylim((-LIM, LIM))
    plt.grid(b=True,which='both',axis='both')



#Numerical representation to compare loss matrix of part a and B

Correct_Classify_L1_B=B_COLOR_L1.count('green')/len(B_COLOR_L1)*100
Wrong_Classify_L1_B=B_COLOR_L1.count('red')/len(B_COLOR_L1)*100
Correct_Classify_L2_B=B_COLOR_L2.count('green')/len(B_COLOR_L2)*100
Wrong_Classify_L2_B=B_COLOR_L2.count('red')/len(B_COLOR_L2)*100
Correct_Classify_L3_B=B_COLOR_L3.count('green')/len(B_COLOR_L3)*100
Wrong_Classify_L3_B=B_COLOR_L3.count('red')/len(B_COLOR_L3)*100
Correct_Classify_L4_B=B_COLOR_L4.count('green')/len(B_COLOR_L4)*100
Wrong_Classify_L4_B=B_COLOR_L4.count('red')/len(B_COLOR_L4)*100

Correct_Classify_L1_A=COLOR_L1.count('green')/len(COLOR_L1)*100
Wrong_Classify_L1_A=COLOR_L1.count('red')/len(COLOR_L1)*100
Correct_Classify_L2_A=COLOR_L2.count('green')/len(COLOR_L2)*100
Wrong_Classify_L2_A=COLOR_L2.count('red')/len(COLOR_L2)*100
Correct_Classify_L3_A=COLOR_L3.count('green')/len(COLOR_L3)*100
Wrong_Classify_L3_A=COLOR_L3.count('red')/len(COLOR_L3)*100
Correct_Classify_L4_A=COLOR_L4.count('green')/len(COLOR_L4)*100
Wrong_Classify_L4_A=COLOR_L4.count('red')/len(COLOR_L4)*100
print("")
print("Numerical Representation Comparison for Part A & Part B:")
print("")
print("Class Label 1:")
print("Correct Label   - Part A: "+str(round(Correct_Classify_L1_A,1))+"%"+"   Part B: "+str(round(
print("Incorrect Label - Part A: "+str(round(Wrong_Classify_L1_A,1))+"%"+"   Part B: "+str(round(Wr
print("")
print("Class Label 2:")
print("Correct Label   - Part A: "+str(round(Correct_Classify_L2_A,1))+"%"+"   Part B: "+str(round(
print("Incorrect Label - Part A: "+str(round(Wrong_Classify_L2_A,1))+"%"+"   Part B: "+str(round(Wr
print("")
print("Class Label 3:")
print("Correct Label   - Part A: "+str(round(Correct_Classify_L3_A,1))+"%"+"   Part B: "+str(round(
print("Incorrect Label - Part A: "+str(round(Wrong_Classify_L3_A,1))+"%"+"   Part B: "+str(round(Wr
print("")
print("Class Label 4:")
print("Correct Label   - Part A: "+str(round(Correct_Classify_L4_A,1))+"%"+"   Part B: "+str(round(
print("Incorrect Label - Part A: "+str(round(Wrong_Classify_L4_A,1))+"%"+"   Part B:"+str(round(Wrc
print("")
print("Finished Q2 part B")
```