EECE 5644 Introduction to Machine Learning and Pattern Recognition
Problem Set 1

Problem 1.1

Given,
X: 4-dimensional real-valued random vector
Y: true class label (0 or 1)
PDF for X: $f_X(x) = f_{X|Y}(x|0)P_Y(0) + f_{X|Y}(x|1)P_Y(1)$
Class priors: $P_Y(0) = 0.7, P_Y(1) = 0.3$
Class conditional (Gaussian) PDFs: $f_{X|Y}(x|0) = g(x|\mu_0, \Sigma_0)$, $f_{X|Y}(x|1) = g(x|\mu_1, \Sigma_1)$

$$\mu_0 = \begin{bmatrix} -1 \\ 1 \\ -1 \\ 1 \end{bmatrix} \Sigma_0 = \begin{bmatrix} 2 & -0.5 & 0.3 & 0 \\ -0.5 & 1 & -0.5 & 0 \\ 0.3 & -0.5 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \mu_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \Sigma_1 = \begin{bmatrix} 1 & 0.3 & -0.2 & 0 \\ 0.3 & 2 & 0.3 & 0 \\ -0.2 & 0.3 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

10,000 samples were generated based on the provided PDF using the NumPy library in Python. The complete code is available in Appendix.

Part A: Expected Risk Minimization (ERM) Classification

1. **Minimum Expected Risk Classification Rule**
   The minimum expected risk classification rule in the form of a likelihood-ratio test is given as:

$$\frac{f_{X|Y}(x|1)}{f_{X|Y}(x|0)} \gtrless \gamma = \frac{P_Y(0)}{P_Y(1)} * \frac{L_{1,0} - L_{0,0}}{L_{0,1} - L_{1,1}} = \frac{7}{3} * \frac{L_{1,0} - L_{0,0}}{L_{0,1} - L_{1,1}}$$

   where threshold γ is a function of class priors and fixed loss values for each of the four cases D = i | Y = j where D is the decision label that is either 0 or 1, like Y.

   To minimize the expected risk, the costs for incorrect results, $L_{0,1}$ and $L_{1,0}$ (false positive and false negative) are set to 1 and costs for correct results, $L_{0,0}$ and $L_{1,1}$ (true positive and true negative) are set to 0 i.e. a 0-1 loss function should be used, $L = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

$$\therefore \frac{f_{X|Y}(x|1)}{f_{X|Y}(x|0)} \gtrless \gamma = \frac{7}{3} * \frac{1-0}{1-0} = 2.3333$$

2. **Classifier Implementation and ROC Curve**
   The above classifier was implemented and applied to the 10,000 generated samples. The true and false positive probabilities were computed for each value of threshold, γ from 0 to ∞, and the corresponding ROC curve was plotted in figure 1.

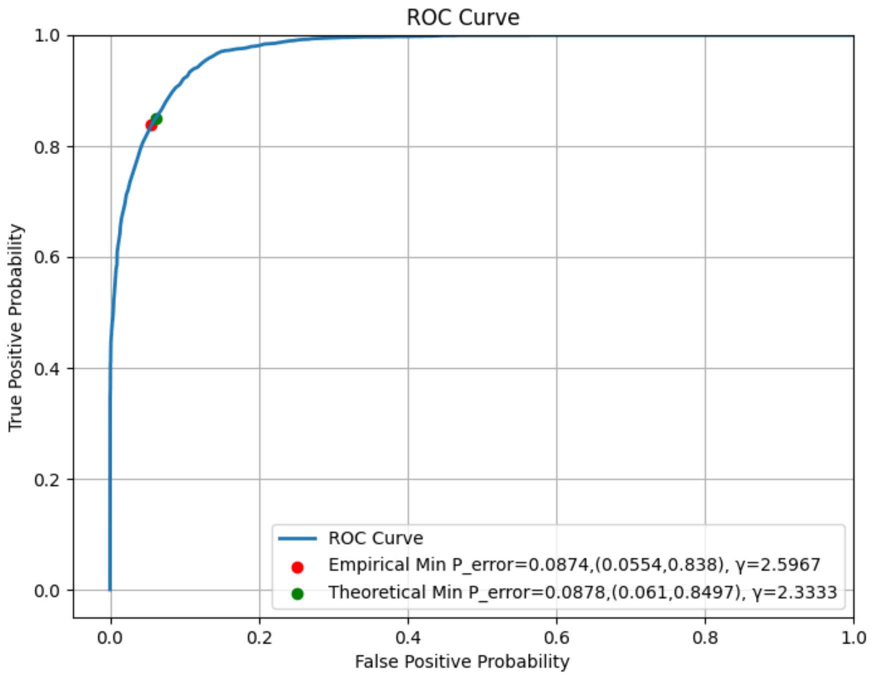


Figure 1: ROC Curve of ERM Classification

3. **Comparison of Theoretical and Empirical Values**
   The probability of error, $P(error;\gamma)$ is calculated using the below equation.
   $$P(error;\gamma) = P(D=1|Y=0;\gamma)P_Y(0) + P(D=0|Y=1;\gamma)P_Y(1)$$

   The empirical minimum probability of error is found by finding the sample within the 10,000 samples that gives the least $P(error;\gamma)$ and its corresponding γ value. The theoretical minimum probability of error is determined by obtaining the P_{error} value for which $\gamma = 2.3333$ (as calculated in 1). Table 1 shows the comparison of the theoretical (green point) and empirical (red point) values. The empirical and theoretical errors differ $\approx$ 0.02, concluding that it is accurate enough.

	Threshold, γ	**Minimum P_{error}**	**$P(D=1\|Y=0;\gamma)$ (False Positive)**	**$P(D=1\|Y=1;\gamma)$ (True Positive)**
Empirical	1.8018	**0.0884**	0.0737	0.8773
Theoretical	**2.3333**	0.0889	0.061	0.846

Table 1: Comparison of Theoretical and Empirical Values

Part B: Naïve Bayesian Classifier

The covariance matrices are updated such that the non-diagonal entries are zero. All other parameters remain the same. The updated matrices are shown below:

$$\Sigma_0 = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 \end{bmatrix} \Sigma_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix}$$

1. **Minimum Expected Risk Classification Rule**
   The classification rule remains unchanged from Part A as the parameters are the same.

$$\frac{f_{X|Y}(x|1)}{f_{X|Y}(x|0)} \gtreqless \gamma = \frac{P_Y(0)}{P_Y(1)} * \frac{L_{1,0} - L_{0,0}}{L_{0,1} - L_{1,1}} = \frac{7}{3} * \frac{1-0}{1-0} = 2.3333$$

2. **Classifier Implementation and ROC Curve**
   The above classifier with the modified Σ_0 *and* Σ_1 values were implemented and applied to the 10,000 generated samples. The true and false positive probabilities were computed for each value of threshold, γ from 0 to ∞, and the corresponding ROC curve was plotted in figure 2.

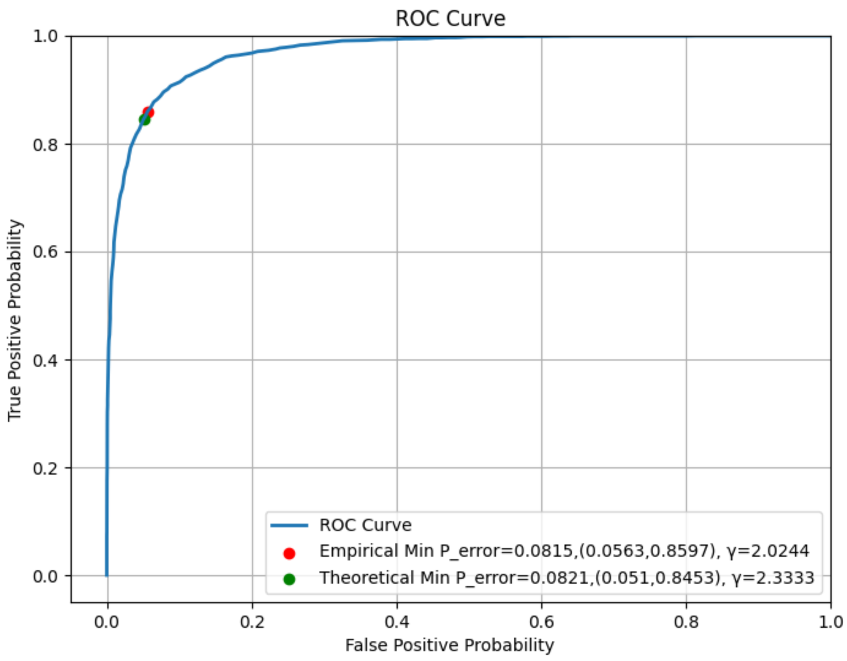


Figure 2: ROC Curve for Naïve Bayesian Classifier

3. **Comparison of Theoretical and Empirical Values**
   The probability of error, $P(error;\gamma)$, empirical and theoretical minimum probability of error can be calculated similar to Part A.

	Threshold, γ	**Minimum P_{error}**	**$P(D=1\|Y=0;\gamma)$ (False Positive)**	**$P(D=1\|Y=1;\gamma)$ (True Positive)**
Empirical	2.0244	**0.0815**	0.0563	0.8597
Theoretical	**2.3333**	0.0821	0.051	0.8453

Table 2: Comparison of Theoretical and Empirical Values

The model mismatch did not lead to any discrepancies in the ROC curve. However, the minimum probability of error values has decreased for both the empirical and theoretical values.

Problem 1.2

Given,
X: 3-dimensional random vector
Class 1: $\mathcal{N}(\mu_1, \Sigma)$
Class 2: $\mathcal{N}(\mu_2, \Sigma)$
Class 3: $\frac{1}{2}\mathcal{N}(\mu_3, \Sigma) + \frac{1}{2}\mathcal{N}(\mu_4, \Sigma)$
Priors: [0.3, 0.3, 0.4]

$\mu_1, \mu_2, \mu_3, \mu_4$ are chosen such that distances between are twice the average standard deviation of the Gaussian components. $\Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4$ are diagonal matrices with the diagonal entries obtained through a random value generator. The values used to generate the below distributions are listed as follows.

$$\Sigma_1 \approx \begin{bmatrix} 0.6982 & 0 & 0 \\ 0 & 0.8611 & 0 \\ 0 & 0 & 0.2299 \end{bmatrix} \Sigma_2 \approx \begin{bmatrix} 0.8978 & 0 & 0 \\ 0 & 0.6535 & 0 \\ 0 & 0 & 0.484 \end{bmatrix}$$
$$\Sigma_3 \approx \begin{bmatrix} 0.3716 & 0 & 0 \\ 0 & 0.071 & 0 \\ 0 & 0 & 0.6715 \end{bmatrix} \Sigma_4 \approx \begin{bmatrix} 0.1026 & 0 & 0 \\ 0 & 0.076 & 0 \\ 0 & 0 & 0.7905 \end{bmatrix}$$

$$\mu_1 \approx \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \mu_2 \approx \begin{bmatrix} 1.3129 \\ 0 \\ 0 \end{bmatrix} \mu_3 \approx \begin{bmatrix} 0 \\ 1.3129 \\ 0 \end{bmatrix} \mu_4 \approx \begin{bmatrix} 1.3129 \\ 1.3129 \\ 0 \end{bmatrix}$$

Part A: Minimum Probability of Error Classification or Bayes Decision Rule or MAP Classifier

1. **Data Generation**
   10,000 samples are generated using NumPy in Python. The code is available in the Appendix. The below figure shows the plot of the distribution.

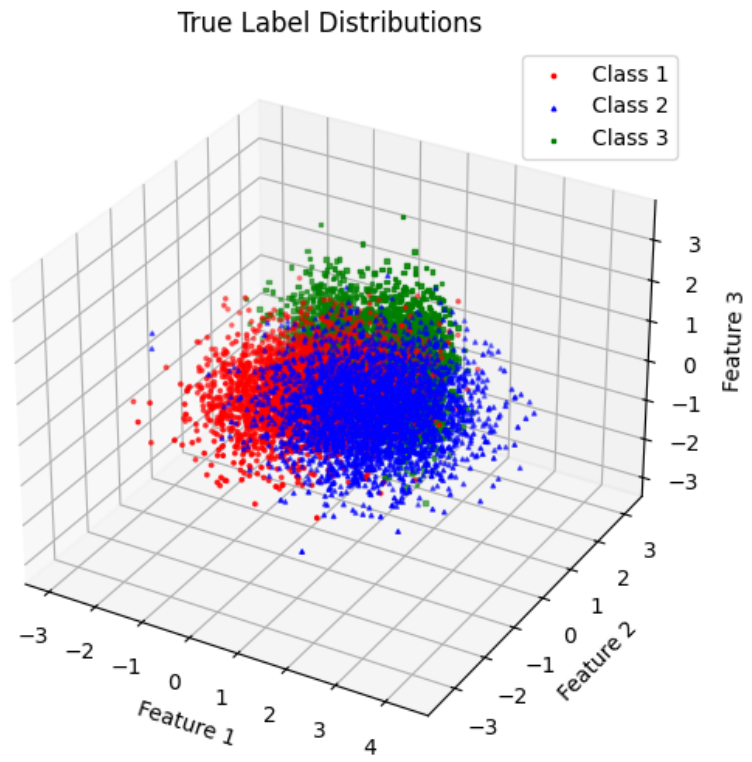


Figure 3: True Data Distributions

2. **Decision Rule**

As the loss function used here is a 0-1 loss function, the risk can be said to be the average probability of error. The conditional risk is given as:

$$R(\alpha_i|x) = \sum_{j=1}^{c} \lambda(\alpha_i|w_j)P(w_j|x) = \sum_{j\neq i} P(w_j|x) = 1 - P(w_i|x)$$

where,

c: Number of classes, 3

α_i: Action taken

w_i: True state

$P(w_i|x)$: Conditional probability that action α_I is correct

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

The Bayes decision rule to minimize risk is done by selecting the action that minimizes the conditional risk. To minimize the average probability of error, the i value that maximizes the posterior probability $P(w_i|x)$ is selected, i.e.

$Decide\ w_i\ if\ P(w_i|x) > P(w_j|x)\ \forall j\ \neq i$

$Decide\ w_i\ if\ P(x|w_i)P(w_i) > P(x|w_j)P(w_j)\ \forall\ j\ \neq i$

$\underset{w_i}{\operatorname{argmax}} P(w_i|x)$

Bayesian Decision Rule:

$$\boldsymbol{D(x) = \underset{i\in\{1,2,3\}}{\operatorname{argmin}} \sum_{j=1}^{c} \lambda_{ij} P(x|Y=j)P(Y=j)}$$

where,

$c: Number\ of\ classes$

$\lambda_{ij}: Loss\ incurred$

$P(x|Y=j): Likelihood\ Probability$

$P(Y=j): Prior\ Probability$

The confusion matrix, whose entries are $P(D=i|Y=j)\ \forall\ i,j \in \{1,2,3\}$ obtained on the implementation of the above decision rule is shown below.

$$Confusion\ Matrix(i,j) = \begin{bmatrix} 0.6933 & 0.2103 & 0.0257 \\ 0.1623 & 0.6673 & 0.0225 \\ 0.1443 & 0.1223 & 0.9517 \end{bmatrix}$$

3. Data Visualization

Figure 4 shows the distribution of the predicted class of the samples and figures 5a, 5b, 5c show the plot for each class respectively. It is observed that at the border areas is where there is more chance for incorrect classification.

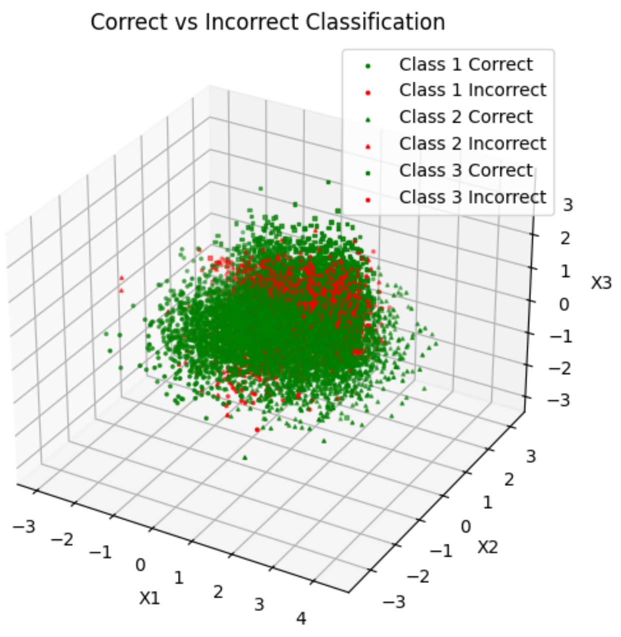
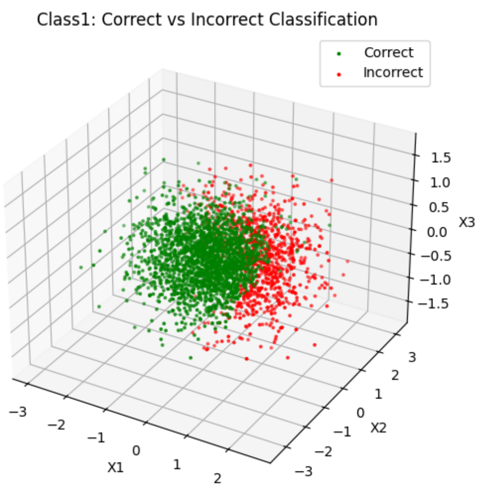
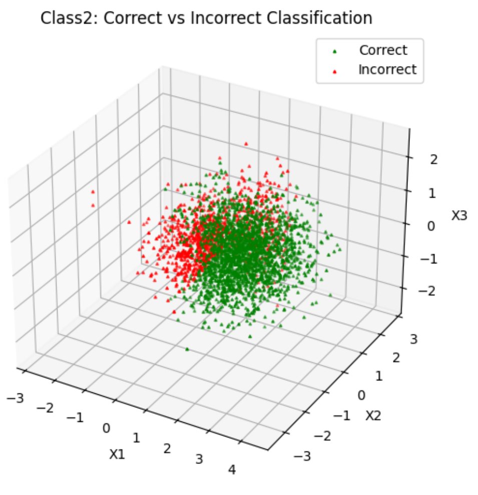


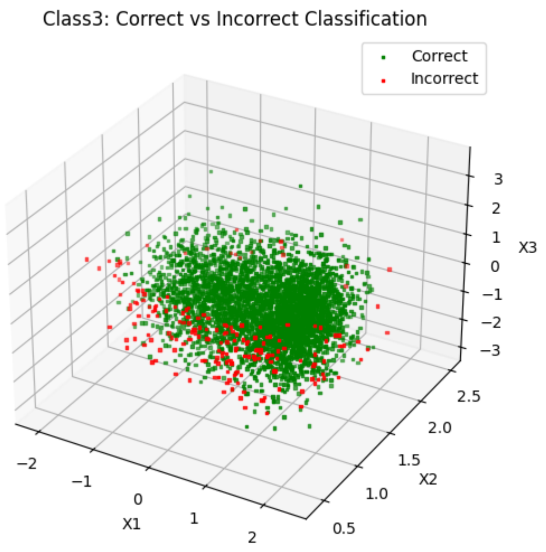
Figure 4: Correct vs Incorrect Classification for 0-1 loss



(a) Class 1



(b) Class 2



(c) Class 3

Figure 5: Correct vs Incorrect Classification for each class

Part B: ERM Classification Rule for Different Loss Matrices

The same data and code used in Part A is used. The loss matrix given as input to the decision function is varied as required. It can be observed that on increasing the loss associated with misclassifying class 3, classes 1 and 2 tend to get misclassified leading to the misclassified area becoming larger. Additionally, the confusion matrix probability entries for misclassification of class 3 decrease from 0-1 loss matrix to Λ_{10}, then Λ_{100}.

1. **For** $\boldsymbol{\Lambda_{10}} = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{10} \\ \mathbf{1} & \mathbf{0} & \mathbf{10} \\ \mathbf{1} & \mathbf{1} & \mathbf{0} \end{bmatrix}$

The updated confusion matrix is described as:

$$Confusion\ Matrix(i,j) = \begin{bmatrix} 0.6266 & 0.188 & 0.0025 \\ 0.1443 & 0.6103 & 0.0015 \\ 0.229 & 0.2016 & 0.996 \end{bmatrix}$$

Figure 6 shows the plot for correct vs incorrect classification using the loss matrix Λ_{10}.

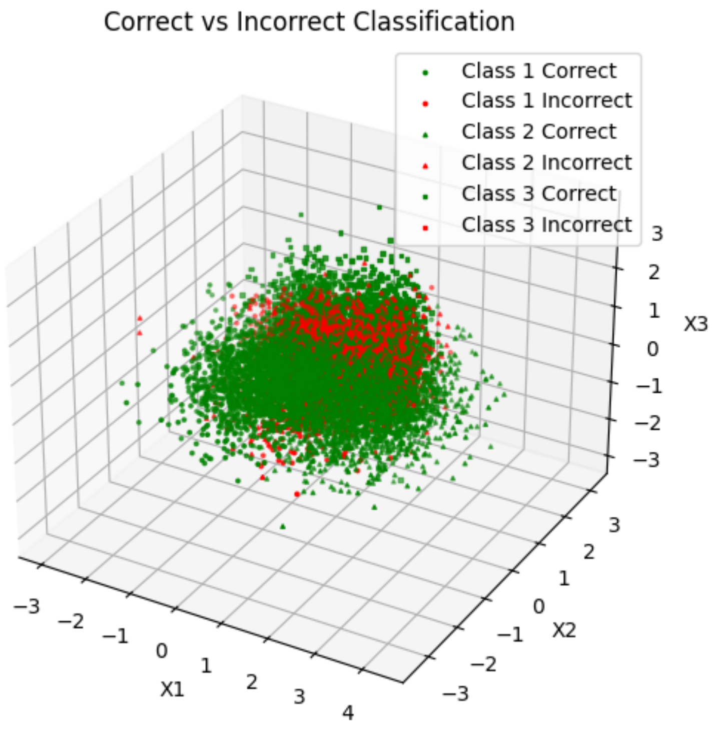


Figure 6: Correct vs Incorrect Classification for Λ_{10} loss matrix

2. **For** $\boldsymbol{\Lambda_{100}} = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{100} \\ \mathbf{1} & \mathbf{0} & \mathbf{100} \\ \mathbf{1} & \mathbf{1} & \mathbf{0} \end{bmatrix}$

The updated confusion matrix is described as:

$$Confusion\ Matrix(i,j) = \begin{bmatrix} 0.565 & 0.16766 & 0.0005 \\ 0.129 & 0.556 & 0 \\ 0.306 & 0.2763 & 0.9995 \end{bmatrix}$$

Figure 7 shows the plot for correct vs incorrect classification using the loss matrix Λ_{100}.

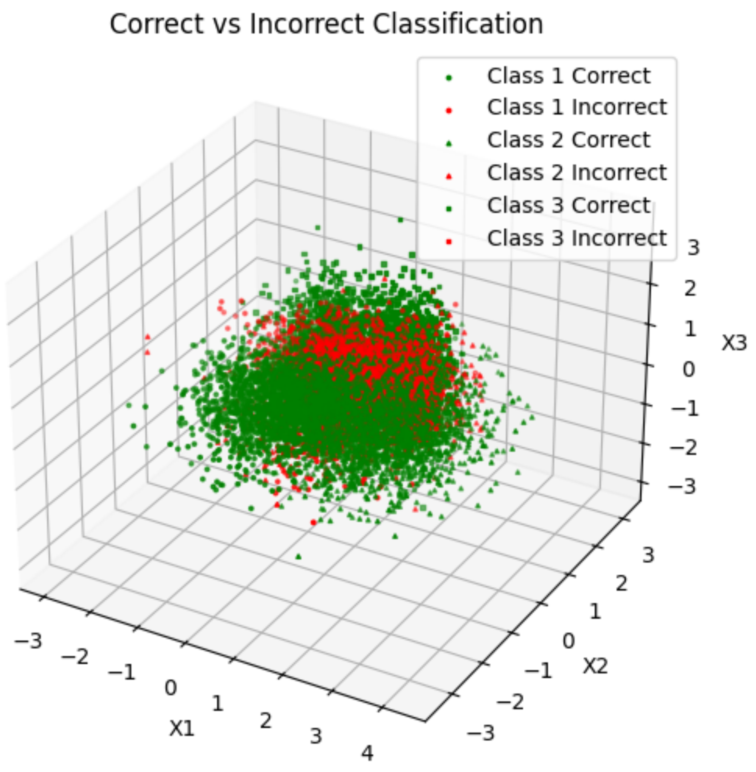


Figure 7: Correct vs Incorrect Classification for Λ_{100} loss matrix

Problem 1.3

Part A: Wine Quality Dataset

The class conditional PDFs for each class (0-10) are obtained on computing the mean and covariance of the samples of each class respectively. The estimated priors are obtained by using sample counts. The covariance matrix is regularized using the formula shown below.

$$\Sigma_{Regularized} = \Sigma_{SampleAverage} + \lambda I$$

where $\lambda > 0$: regularization parameter that ensures $\Sigma_{Regularized}$ has eigenvalues larger than this parameter. It is initialized as 0.01 in this implementation.

The Bayes decision rule is used for classification and 0-1 loss matrix is considered for minimum probability of error. Bayesian Decision Rule:

$$D(x) = \underset{i\in\{1,2,3\}}{\operatorname{argmin}} \sum_{j=1}^{c} \lambda_{ij} P(x|Y = j) P(Y = j)$$

where,

c: *Number of classes*
λ_{ij}: *Loss incurred*
$P(x|Y=j)$: *Likelihood Probability*
$P(Y=j)$: *Prior Probability*

The error probability estimate for this classifier is computed as 0.4624. Hence, the accuracy of the classifier is 53.76%. The confusion matrix, which is calculated by $P(D=i|Y=j)\ i,j \in \{0,1,2,3,4,5,6,7,8,9,10\}$ obtained on the implementation of the Bayes decision rule is shown below. Furthermore, the number of samples in each class and their priors are also described.

$$Confusion\ Matrix(i,j) = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.35 & 0.0122 & 0.0034 & 0.0045 & 0.0011 & 0.011 & 0 & 0 \\ 0 & 0 & 0 & 0.05 & 0.1411 & 0.0157 & 0.004 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.2 & 0.546 & 0.6135 & 0.247 & 0.0681 & 0.0457 & 0 & 0 \\ 0 & 0 & 0 & 0.4 & 0.2883 & 0.3486 & 0.6032 & 0.5011 & 0.4628 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0122 & 0.0178 & 0.1401 & 0.4227 & 0.4457 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.0006 & 0.0009 & 0.0056 & 0.0342 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.0011 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$Number\ of\ samples\ in\ each\ class = [0 \quad 0 \quad 0 \quad 20 \quad 163 \quad 1457 \quad 2198 \quad 880 \quad 175 \quad 5 \quad 0]$$

$$Estimated\ Prior\ Probabilities = [0 \quad 0 \quad 0 \quad 0.004 \quad 0.0332 \quad 0.2974 \quad 0.4487 \quad 0.1796 \quad 0.0357 \quad 0.001 \quad 0]$$

Classes 0, 1, 2, and 10 do not have any samples, hence no decisions should be made for these classes, which can be seen in the confusion matrix since all corresponding values are 0. The classifier makes most decisions for class 6, as it has the highest prior probability, which can be concluded from figure 8 as well. The confusion matrix also tells us that most decisions made are of classes 5, 6, or 7 as they have the largest priors. No decisions are made as class 9 as its prior probability, 0.001 is low. Moreover, the diagonal elements of each column in the confusion matrix, which represent probability of correct classification, do not have the highest values. Chances for misclassification are high as the means for each class may be overlapping. Decisions seem to be mostly made based on the prior probabilities. Therefore, we can conclude that Gaussian class conditional models seem to be not appropriate for this dataset as the classifier accuracy is low and the distributions do not follow a Gaussian model.

The PCA function in the Python sklearn module was used to reduce the dimensionality/features to 3 dimensions. The visualization of the datasets on implementing this is seen in figure 8.

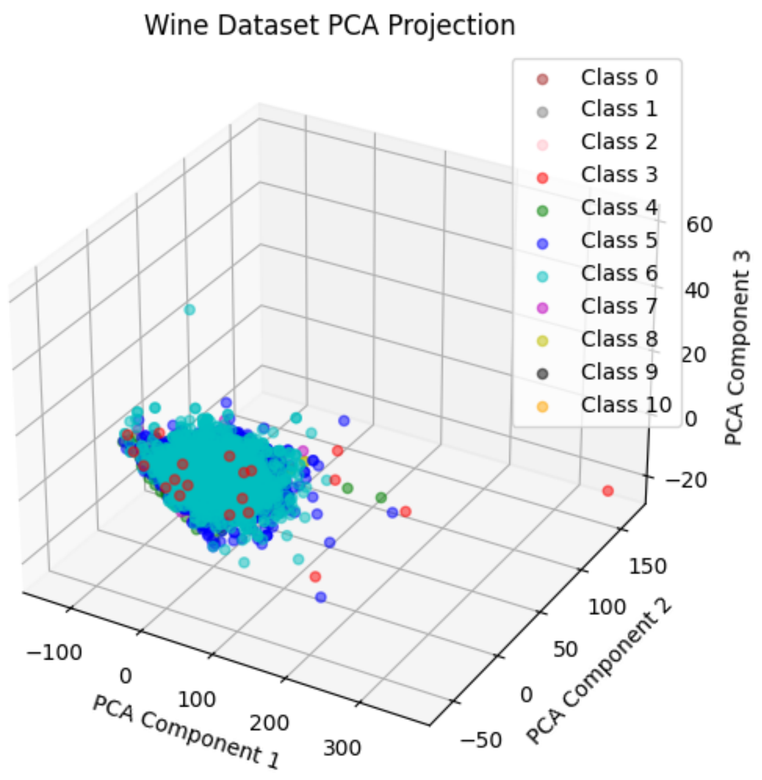


Figure 8: PCA Projection of the Wine Quality Dataset

Part B: Human Activity Recognition Dataset

The same classifier used in Part A is used here. The error probability estimate is 0.0163 and the accuracy of the classifier is 98.37%. The computed confusion matrix calculated using $P(D = i|Y = j)\ i, j\ \in \{1,2,3,4,5,6\}$, number of samples in each class, and their priors are described below.

$$Confusion\ Matrix(i,j) = \begin{bmatrix} 0.9982 & 0 & 0 & 0 & 0 & 0 \\ 0.001 & 1 & 0.0206 & 0 & 0 & 0 \\ 0.0005 & 0 & 0.9793 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.929 & 0.0052 & 0 \\ 0 & 0 & 0 & 0.0709 & 0.9947 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$Number\ of\ samples\ in\ each\ class = [1722 \quad 1544 \quad 1406 \quad 1777 \quad 1906 \quad 1944]$$

$$Estimated\ Prior\ Probabilities = [0.1672 \quad 0.1499 \quad 0.1365 \quad 0.1725 \quad 0.185 \quad 0.1887]$$

Classes 2 and 6 are always correctly classified as the corresponding entry has a probability of 1. Additionally, no other class is incorrectly classified as 6. In general, we can conclude from the confusion matrix that most samples are correctly classified or classified one class previous or next. The number of samples in each class are around the same range, making the prior probabilities almost the same as well. Therefore, as the classifier accuracy is high at 98.37%, we can conclude that the Gaussian distribution can be used for this dataset.

The visualization of the datasets on implementing Principal Component Analysis (PCA) on the 561 features to 3 dimensions is shown in figure 9.

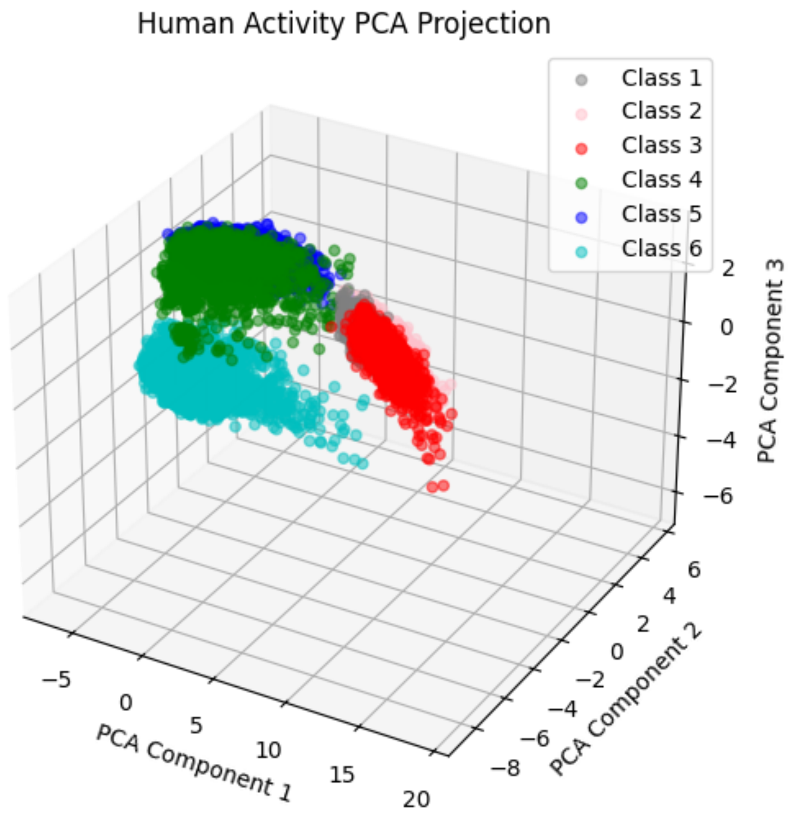


Figure 9: PCA Projection of the Human Activity Dataset

Appendix

Problem 1.1 Code:

```
#!/usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt
import math
from scipy.stats import multivariate_normal

################################## Parameters ######################################
# Priori Probabilites
PY = [0.7, 0.3]

# Gaussian PDF Parameters
mu_0 = np.array([-1, 1, -1, 1])
mu_1 = np.array([1, 1, 1, 1])

# For Part A
# 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]])

# For Part B
sigma_0 = np.array([[2, 0, 0, 0],
                    [0, 1, 0, 0],
                    [0, 0, 1, 0],
                    [0, 0, 0, 2]])
sigma_1 = np.array([[1, 0, 0, 0],
                    [0, 2, 0, 0],
                    [0, 0, 1, 0],
                    [0, 0, 0, 3]])

dimensions = 4
num_samples = 10000
num_samples_0 = int(PY[0] * num_samples)
num_samples_1 = int(PY[1] * num_samples)

################################# Data Generation ##################################
true_labels = np.hstack((np.zeros(num_samples_0), np.ones(num_samples_1)))
```

```
class0_samples = np.random.multivariate_normal(mean=mu_0, cov=sigma_0,
size=num_samples_0)
class1_samples = np.random.multivariate_normal(mean=mu_1, cov=sigma_1,
size=num_samples_1)
samples = np.vstack((class0_samples, class1_samples))

# print(true_labels.shape)
# print(samples.shape)
# for i, label in enumerate(true_labels):
#     print("i=",i, "label=",label, "samples:",samples[i])

########################### Classification and ROC Curve ############################
# Calculate Likelihoods using Gaussian PDF
fX_given_Y0 = multivariate_normal.pdf(samples, mean=mu_0, cov=sigma_0)
fX_given_Y1 = multivariate_normal.pdf(samples, mean=mu_1, cov=sigma_1)

# Classification and Calculation of TP, FP, TN, FN, Perror
thresholds = np.log(np.logspace(-18,18,1000))
TP_list = []
FP_list = []
Perror_list = []

for threshold in thresholds:
    # Decision Rule
    decisions = (np.log(fX_given_Y1) - np.log(fX_given_Y0)) > threshold

    # Get true & false positives and negatives
    TP = np.sum((decisions == 1) & (true_labels == 1)) / num_samples_1
    FP = np.sum((decisions == 1) & (true_labels == 0)) / num_samples_0
    TN = np.sum((decisions == 0) & (true_labels == 0)) / num_samples_0
    FN = np.sum((decisions == 0) & (true_labels == 1)) / num_samples_1
    TP_list.append(TP)
    FP_list.append(FP)

    # Error Probability Calculation
    Perror = FP * PY[0] + FN * PY[1]
    Perror_list.append(Perror)

# Empirical Minimum Error Probability Calculation
min_error_index = np.argmin(Perror_list)
empirical_thresh = math.exp(thresholds[min_error_index])
empirical_min_Perror = Perror_list[min_error_index]
empirical_str = '('
+str(round(FP_list[min_error_index],4))+','+str(round(TP_list[min_error_index],4))+'),
γ='+str(round(empirical_thresh,4))

# Theoretical Minimum Error Probability Calculation
theoretical_thresh = PY[0] / PY[1]
```

```
decisions = (fX_given_Y1 / fX_given_Y0) > theoretical_thresh
theoretical_TP = np.sum((decisions == 1) & (true_labels == 1)) / num_samples_1
theoretical_FP = np.sum((decisions == 1) & (true_labels == 0)) / num_samples_0
theoretical_TN = np.sum((decisions == 0) & (true_labels == 0)) / num_samples_0
theoretical_FN = np.sum((decisions == 0) & (true_labels == 1)) / num_samples_1
theoretical_min_Perror = theoretical_FP * PY[0] + theoretical_FN * PY[1]
theoretical_str =
'('+str(round(theoretical_FP,4))+','+str(round(theoretical_TP,4))+'),
γ='+str(round(theoretical_thresh,4))

# ROC Plot
fig = plt.figure(figsize=(8, 6))
plt.plot(FP_list, TP_list, linewidth=2, label='ROC Curve')
plt.scatter(FP_list[min_error_index], TP_list[min_error_index], color='red',
label='Empirical Min P_error='+str(round(empirical_min_Perror,4))+','+empirical_str)
plt.scatter(theoretical_FP, theoretical_TP, color='green', label='Theoretical Min
P_error='+str(round(theoretical_min_Perror,4))+','+theoretical_str)
plt.xlabel('False Positive Probability')
plt.ylabel('True Positive Probability')
plt.title('ROC Curve')
plt.xlim(-0.05,1)
plt.ylim(-0.05,1)
plt.grid()
plt.legend(loc='lower right')
plt.show()
```

Problem 1.2 Code:

```
#!/usr/bin/env python3
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import multivariate_normal
import random

################################## Parameters #####################################
# Priori Probabilites
PY = [0.3, 0.3, 0.4]

num_classes = 3
dimensions = 3

# Gaussian PDF Parameters
num_gaussians = 4

sigma = np.zeros((dimensions, dimensions, num_gaussians))
mean = np.zeros((dimensions, num_gaussians))
```

```
# Setting covariance matrix as diagonal random matrix
for i in range(num_gaussians):
    sigma[:, :, i] = [[random.random(), 0, 0],
                      [0, random.random(), 0],
                      [0, 0, random.random()]]

# Set distance between means as 2x standard deviation
avg_std_dev = np.mean([np.sqrt(sigma[d, d, i]) for i in range(num_gaussians) for d in
range(dimensions)])
distance = 2 * avg_std_dev
mean[:, 0] = [0, 0, 0]
mean[:, 1] = [distance, 0, 0]
mean[:, 2] = [0, distance, 0]
mean[:, 3] = [distance, distance, 0]

# Printing the mean and covariance matrix values
for i in range(num_gaussians):
    print('Covariance Matrix ', i, ': ', sigma[:,:,i])
    print('Mean ', i, ': ', mean[:,i])

# Calculating number of samples in each class
num_samples = 10000
num_samples_pc = np.zeros(num_classes, dtype=int)
for i in range(num_classes):
    num_samples_pc[i] = int(PY[i] * num_samples)

################################### Data Generation ###################################
true_labels = np.hstack((np.ones(num_samples_pc[0],dtype=int),
                         2*np.ones(num_samples_pc[1],dtype=int),
                         3*np.ones(num_samples_pc[2],dtype=int)))

class1_samples = np.random.multivariate_normal(mean=mean[:,0], cov=sigma[:,:,0],
size=num_samples_pc[0])
class2_samples = np.random.multivariate_normal(mean=mean[:,1], cov=sigma[:,:,1],
size=num_samples_pc[1])
class3_samples_1 = np.random.multivariate_normal(mean=mean[:,2], cov=sigma[:,:,2],
size=num_samples_pc[2]//2)
class3_samples_2 = np.random.multivariate_normal(mean=mean[:,3], cov=sigma[:,:,3],
size=num_samples_pc[2]//2)
class3_samples = np.vstack((class3_samples_1, class3_samples_2))
samples = np.vstack((class1_samples, class2_samples, class3_samples))

# Visualize True Label Distribution
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(samples[true_labels == 1, 0], samples[true_labels == 1, 1],
samples[true_labels == 1, 2], marker='o', s=3, color='r', label='Class 1')
```

```
ax.scatter(samples[true_labels == 2, 0], samples[true_labels == 2, 1],
samples[true_labels == 2, 2], marker='^', s=3, color='b', label='Class 2')
ax.scatter(samples[true_labels == 3, 0], samples[true_labels == 3, 1],
samples[true_labels == 3, 2], marker='s', s=3, color='g', label='Class 3')
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Feature 3')
ax.set_title('True Label Distributions')
ax.legend()

################################### Classification #############################
# Define PDFs
pdf1 = multivariate_normal(mean=mean[:,0], cov=sigma[:,:,0])
pdf2 = multivariate_normal(mean=mean[:,1], cov=sigma[:,:,1])
pdf3_part1 = multivariate_normal(mean=mean[:,2], cov=sigma[:,:,2])
pdf3_part2 = multivariate_normal(mean=mean[:,3], cov=sigma[:,:,3])

# Function to implement Classifier based on decision rule
def bayes_classifier(x, loss_matrix):
    pxgiven1 = pdf1.pdf(x)
    pxgiven2 = pdf2.pdf(x)
    pxgiven3 = 0.5 * pdf3_part1.pdf(x) + 0.5 * pdf3_part2.pdf(x)

    losses = np.zeros(num_classes)
    for i in range(num_classes):
        losses[i] = PY[0] * loss_matrix[i, 0] * pxgiven1 \
                  + PY[1] * loss_matrix[i, 1] * pxgiven2 \
                  + PY[2] * loss_matrix[i, 2] * pxgiven3

    return np.argmin(losses) + 1

# Classify samples
loss_matrix_01 = np.array([[0, 1, 1], [1, 0, 1], [1, 1, 0]])
decision_labels_01 = np.array([bayes_classifier(sample, loss_matrix_01) for sample in
samples])
loss_matrix_10 = np.array([[0, 1, 10], [1, 0, 10], [1, 1, 0]])
decision_labels_10 = np.array([bayes_classifier(sample, loss_matrix_10) for sample in
samples])
loss_matrix_100 = np.array([[0, 1, 100], [1, 0, 100], [1, 1, 0]])
decision_labels_100 = np.array([bayes_classifier(sample, loss_matrix_100) for sample
in samples])

# Visualization
def visualize_distribution(decision_labels):
    # Compute confusion matrix: P(D=i|Y=j) ∀ i,j∈{1,2,3}
    confusion_matrix = np.zeros((num_classes, num_classes))
    for i in range(1, num_classes+1):
        for j in range(1, num_classes+1):
            confusion_matrix[i-1, j-1] = np.sum((decision_labels == i) & (true_labels
== j)) / num_samples_pc[j-1]
    print("Confusion Matrix:\n", confusion_matrix)
    column_sums = np.sum(confusion_matrix, axis=0)
    print("Column sums:", column_sums)

    fig = plt.figure(figsize=(8, 6))
    ax = fig.add_subplot(111, projection='3d')
    markers = ['o', '^', 's']

    for i, label in enumerate([1, 2, 3]):
        idx = (true_labels == label)
        correct = (true_labels == decision_labels)

        ax.scatter(samples[idx & correct, 0], samples[idx & correct, 1], samples[idx &
correct, 2],
                marker=markers[label-1], color='green', label=f'Class {label}
Correct', s=3)
        ax.scatter(samples[idx & ~correct, 0], samples[idx & ~correct, 1], samples[idx
& ~correct, 2],
                marker=markers[label-1], color='red', label=f'Class {label}
Incorrect', s=3)

    ax.set_xlabel('X1')
    ax.set_ylabel('X2')
    ax.set_zlabel('X3')
    ax.set_title('Correct vs Incorrect Classification')
    ax.legend()

def visualize_class_distributions(decision_labels):
    markers = ['o', '^', 's']
    for i, label in enumerate([1, 2, 3]):
        fig = plt.figure(figsize=(8, 6))
        ax = fig.add_subplot(111, projection='3d')

        idx = (true_labels == label)
        correct = (true_labels == decision_labels)

        ax.scatter(samples[idx & correct, 0], samples[idx & correct, 1], samples[idx &
correct, 2],
                marker=markers[label-1], color='green', label='Correct', s=3)
        ax.scatter(samples[idx & ~correct, 0], samples[idx & ~correct, 1], samples[idx
& ~correct, 2],
                marker=markers[label-1], color='red', label='Incorrect', s=3)

        ax.set_xlabel('X1')
        ax.set_ylabel('X2')
        ax.set_zlabel('X3')
        ax.set_title('Class'+str(label) + ': Correct vs Incorrect Classification')
        ax.legend()

visualize_distribution(decision_labels_01)
visualize_class_distributions(decision_labels_01)
visualize_distribution(decision_labels_10)
visualize_distribution(decision_labels_100)

plt.show()
```

Problem 1.3 Code:

Wine Quality Dataset:

```
#!/usr/bin/env python3
import pandas as pd
import numpy as np
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

######################### Dataset and Parameter Estimation ##########################
# Load the dataset
wine_data = pd.read_csv('winequality-white.csv', delimiter=';')

# Extract Features and Labels
X = wine_data.drop(columns=['quality'])
true_labels = wine_data['quality'].values

n_samples = X.shape[0]
n_dimensions = X.shape[1]
n_classes = 11
print('Num samples: ', n_samples, ', Num dimensions: ', n_dimensions, ', Num classes: 
', n_classes)

# Calculate mean, covariance and priors for each class
class_means = []
class_covariances = []
class_priors = []
n_class_samples = []
for c in range(n_classes):
    X_c = X[true_labels == c]

    num_samples = X_c.shape[0]
    if (num_samples > 0):
        mean = np.mean(X_c, axis=0)
        cov = np.cov(X_c, rowvar=False)
        prior = num_samples / n_samples
    else:
        mean = np.zeros(n_dimensions)
        cov = np.zeros((n_dimensions, n_dimensions))
        prior = 0.0

    class_means.append(mean)
    class_covariances.append(cov)
    class_priors.append(prior)
    n_class_samples.append(num_samples)

# Regularize covariance matrix
def regularize_covariance(cov, lambda_val):
    return cov + lambda_val * np.identity(cov.shape[0])
regularized_covariances = [regularize_covariance(cov, 0.01) for cov in
class_covariances]

class_means = np.array(class_means)
class_covariances = np.array(regularized_covariances)
class_priors = np.array(class_priors)
n_class_samples = np.array(n_class_samples)

print('Priors: ', class_priors)
print('N Class Samples: ', n_class_samples)
############################ Classifier Implementation ##############################
# Function to implement Classifier based on decision rule
def bayes_classifier(X, means, covariances, priors):
    likelihoods = np.zeros((n_samples, n_classes))
    for i in range(n_classes):
        mean = means[i]
        cov = covariances[i]
        prior = priors[i]
        mvn = multivariate_normal(mean=mean, cov=cov)
        likelihoods[:, i] = mvn.pdf(X) * prior
    return np.argmax(likelihoods, axis=1)

# Classify samples
decision_labels = bayes_classifier(X.values, class_means, class_covariances,
class_priors)

# Compute confusion matrix: P(D=i|Y=j) ∀ i,j∈{0,1,....10}
confusion_matrix = np.zeros((n_classes, n_classes))
for i in range(n_classes):
    for j in range(n_classes):
        if (n_class_samples[j] > 0):
            confusion_matrix[i, j] = np.sum((decision_labels == i) & (true_labels ==
j)) / n_class_samples[j]
```

```
        else:
            confusion_matrix[i, j] = 0.0
print("Confusion Matrix:\n", confusion_matrix)
column_sums = np.sum(confusion_matrix, axis=0)
print("Column sums:", column_sums)

# Compute error probability
sum_errors = np.sum(decision_labels != true_labels)
error_probability = sum_errors / n_samples
print(f'Error Probability: ', error_probability)

# Visualization
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X)
colors = ['brown', 'grey', 'pink', 'r', 'g', 'b', 'c', 'm', 'y', 'k', 'orange']

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
for c in range(n_classes):
    ax.scatter(X_pca[true_labels == c, 0], X_pca[true_labels == c, 1],
X_pca[true_labels == c, 2],
               label=f'Class {c}', alpha=0.5, color=colors[c])

ax.set_xlabel('PCA Component 1')
ax.set_ylabel('PCA Component 2')
ax.set_zlabel('PCA Component 3')
ax.legend()
ax.set_title('Wine Dataset PCA Projection')
plt.show()
```

Human Activity Recognition Dataset:

```
#!/usr/bin/env python3
import pandas as pd
import numpy as np
from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

########################## Dataset and Parameter Estimation ##########################
# Load the dataset
X_test = pd.DataFrame([line.strip().split() for line in open('X_test.txt')])
X_train = pd.DataFrame([line.strip().split() for line in open('X_train.txt')])
y_test = pd.read_csv('y_test.txt', sep=' ', header=None)
y_train = pd.read_csv('y_train.txt', sep=' ', header=None)
X = pd.concat([X_test, X_train], axis=0).to_numpy().astype(float)
y = pd.concat([y_test, y_train], axis=0).to_numpy().ravel()
```

```
n_samples = X.shape[0]
n_dimensions = X.shape[1]
n_classes = 6
print('Num samples: ', n_samples, ', Num dimensions: ', n_dimensions, ', Num classes:
', n_classes)

# Calculate mean, covariance and priors for each class
class_means = []
class_covariances = []
class_priors = []
n_class_samples = []
for c in range(1, n_classes+1):
    X_c = X[y == c]

    num_samples = X_c.shape[0]
    mean = np.mean(X_c, axis=0)
    cov = np.cov(X_c, rowvar=False)
    prior = num_samples / n_samples

    class_means.append(mean)
    class_covariances.append(cov)
    class_priors.append(prior)
    n_class_samples.append(num_samples)

# Regularize covariance matrix
def regularize_covariance(cov, lambda_val):
    return cov + lambda_val * np.identity(cov.shape[0])
regularized_covariances = [regularize_covariance(cov, 0.01) for cov in
class_covariances]

class_means = np.array(class_means)
class_covariances = np.array(regularized_covariances)
class_priors = np.array(class_priors)
n_class_samples = np.array(n_class_samples)

print('Priors: ', class_priors)
print('N Class Samples: ', n_class_samples)
############################ Classifier Implementation ##############################
# Function to implement Classifier based on decision rule
def bayes_classifier(X, means, covariances, priors):
    likelihoods = np.zeros((n_samples, n_classes))
    for i in range(n_classes):
        mean = means[i]
        cov = covariances[i]
        prior = priors[i]
        mvn = multivariate_normal(mean=mean, cov=cov)
        likelihoods[:, i] = mvn.pdf(X) * prior
    return np.argmax(likelihoods, axis=1) + 1
```

```python
# Classify samples
decision_labels = bayes_classifier(X, class_means, class_covariances, class_priors)

# Compute confusion matrix: P(D=i|Y=j) ∀ i,j∈{1,....6}
confusion_matrix = np.zeros((n_classes, n_classes))
for i in range(1, n_classes+1):
    for j in range(1, n_classes+1):
        confusion_matrix[i-1, j-1] = np.sum((decision_labels == i) & (y == j)) / 
n_class_samples[j-1]
print("Confusion Matrix:\n", confusion_matrix)
column_sums = np.sum(confusion_matrix, axis=0)
print("Column sums:", column_sums)

# Compute error probability
error_probability = np.mean(decision_labels != y)
print(f'Error Probability: ', error_probability)

# Visualization
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X)
colors = ['brown', 'grey', 'pink', 'r', 'g', 'b', 'c', 'm', 'y', 'k', 'orange']

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
for c in range(1, n_classes+1):
    ax.scatter(X_pca[y == c, 0], X_pca[y == c, 1], X_pca[y == c, 2],
               label=f'Class {c}', alpha=0.5, color=colors[c])

ax.set_xlabel('PCA Component 1')
ax.set_ylabel('PCA Component 2')
ax.set_zlabel('PCA Component 3')
ax.legend()
ax.set_title('Human Activity PCA Projection')
plt.show()
```