Northeastern University
College of Engineering

October 7, 2024

EECE 5644: Introduction to Machine Learning & Pattern Recognition
Assignment 1

Name: Pradnyesh Choudhari
NUID: 002339243

Assignment 1 - Question 1

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import multivariate_normal
from sklearn.metrics import roc_curve
from scipy import linalg
```

In [ ]:
```
np.random.seed(1)
```

Question 1 (30%)

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 $P(L=0) = 0.35$ and $P(L=1) = 0.65$. 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.

In [ ]:
```
mean0 = np.array([-1, -1, -1, -1])
covariance0 = np.array(
    [[2, -0.5, 0.3, 0], [-0.5, 1, -0.5, 0], [0.3, -0.5, 1, 0], [0, 0, 0, 2]]
)
p_class_0 = 0.35

mean1 = np.array([1, 1, 1, 1])
covariance1 = np.array(
    [[1, 0.3, -0.2, 0], [0.3, 2, 0.3, 0], [-0.2, 0.3, 1, 0], [0, 0, 0, 3]]
)
p_class_1 = 0.65

number_of_samples = 10000
num_samples_class0 = int(p_class_0 * number_of_samples)
num_samples_class1 = int(p_class_1 * number_of_samples)


samples_class0 = np.random.multivariate_normal(mean0, covariance0, num_samples_class0)
samples_class1 = np.random.multivariate_normal(mean1, covariance1, num_samples_class1)

labels_class0 = np.zeros(num_samples_class0)
labels_class1 = np.ones(num_samples_class1)

samples = np.vstack((samples_class0, samples_class1))
labels = np.hstack((labels_class0, labels_class1))
```

In [ ]:
```
df = pd.DataFrame(samples, columns=["x1", "x2", "x3", "x4"])
df["Label"] = labels

df.to_csv("data_created_for_hw1_q1_10000_samples.csv", index=False)
```

In [ ]:
```
df
```

Out[ ]:

	x1	x2	x3	x4	Label
0	-2.885759	-0.199077	-2.716423	-1.865154	0.0
1	-3.172325	-1.774531	-0.538224	-4.254867	0.0
2	-2.410798	-2.655004	-1.048926	-1.352663	0.0
3	-1.317274	-2.400534	-0.514371	-1.543135	0.0
4	-0.753927	-0.834060	-0.614293	-2.241479	0.0
...	...	...	...	...	...
9995	1.981959	1.710207	2.784061	-0.120244	1.0
9996	2.029219	0.746992	0.546795	1.572723	1.0
9997	1.648817	2.504954	1.105592	0.246232	1.0
9998	0.148452	1.171763	2.323189	2.404018	1.0
9999	1.152027	-0.724877	0.124420	3.530430	1.0

10000 rows × 5 columns

Part A: ERM classification using the knowledge of true data pdf:

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.

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.

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?

Part A - 1

Reference from 2.2.1 Pattern Classification by Duda, Hart and Stork.

Considering λ_ij loss

$$R(L_0|X) = \lambda_{00}P(L_0|X) + \lambda_{01}P(L_1|X)$$
$$R(L_1|X) = \lambda_{10}P(L_0|X) + \lambda_{11}P(L_1|X)$$

The fundamental rule is to decide L_0 if $R(\alpha_0|X) < R(\alpha_1|X)$, in terms of the posterior probabilities, we decide L_0 if

$$(\lambda_{10} - \lambda_{00})P(L * 0|X) > (\lambda_{01} - \lambda_{11})P(L_1|X)$$

Using Bayes Theorem we get, in general

$$\frac{P(X|L_0)}{P(X|L_1)} \overset{?}{} \frac{(\lambda_{01} - \lambda_{11}).\,P(L_0)}{(\lambda_{10} - \lambda_{00}).\,P(L_1)}$$

Given $P(L_0) = 0.35$ & $P(L_1) = 0.65$

$$\frac{P(X|L_1)}{P(X|L_0)} \overset{?}{} \frac{(\lambda_{10} - \lambda_{00}).0.35}{(\lambda_{01} - \lambda_{11}).0.65}$$

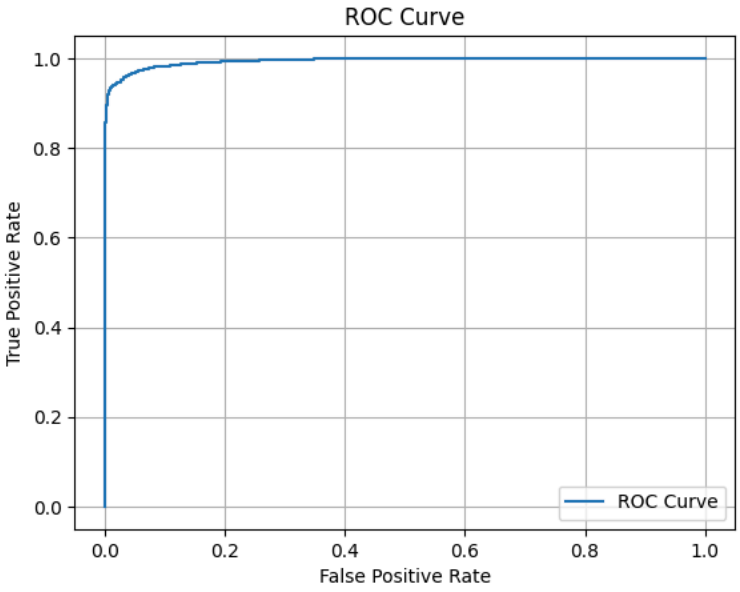
$$\frac{P(X|L_1)}{P(X|L_0)} . 1.8571 ? \frac{(\lambda_{10} - \lambda_{00})}{(\lambda_{01} - \lambda_{11})}$$

Part A - 2

In [ ]:
```
px_given_l0 = multivariate_normal.pdf(samples, mean=mean0, cov=covariance0)
px_given_l1 = multivariate_normal.pdf(samples, mean=mean1, cov=covariance1)
likelihood_ratio = px_given_l1 / px_given_l0
```

In [ ]:
```
fpr_list_likelihood, tpr_list_likelihood, gammas = roc_curve(labels, likelihood_ratio)
```

In [ ]:
```
plt.figure()
plt.plot(fpr_list_likelihood, tpr_list_likelihood, label="ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.grid(True)
plt.show()
```



Part A - 3

In [ ]:
```
perror_likelihood = (
    fpr_list_likelihood * p_class_0 + (1 - tpr_list_likelihood) * p_class_1
)
```

In [ ]:
```
gamma_theoretical = p_class_0 / p_class_1

decisions_theoretical = (likelihood_ratio >= gamma_theoretical).astype(int)

tp = np.sum((decisions_theoretical == 1) & (labels == 1))
fp = np.sum((decisions_theoretical == 1) & (labels == 0))
fn = np.sum((decisions_theoretical == 0) & (labels == 1))
tn = np.sum((decisions_theoretical == 0) & (labels == 0))

tpr_theoretical = tp / (tp + fn) if (tp + fn) > 0 else 0
fpr_theoretical = fp / (fp + tn) if (fp + tn) > 0 else 0

p_error_theoretical = fpr_theoretical * p_class_0 + (1 - tpr_theoretical) * p_class_1
```

In [ ]:
```
df_taus_tpr_fpr_likelihood = pd.DataFrame(gammas, columns=["Gamma"])
df_taus_tpr_fpr_likelihood["fpr"] = fpr_list_likelihood
df_taus_tpr_fpr_likelihood["tpr"] = tpr_list_likelihood
df_taus_tpr_fpr_likelihood["perror"] = perror_likelihood
df_taus_tpr_fpr_likelihood["perror - therotical"] = p_error_theoretical
```

In [ ]:
```
df_likelihood_result = df_taus_tpr_fpr_likelihood[
    df_taus_tpr_fpr_likelihood["perror"] == df_taus_tpr_fpr_likelihood["perror"].min()
].reset_index()
df_likelihood_result
```

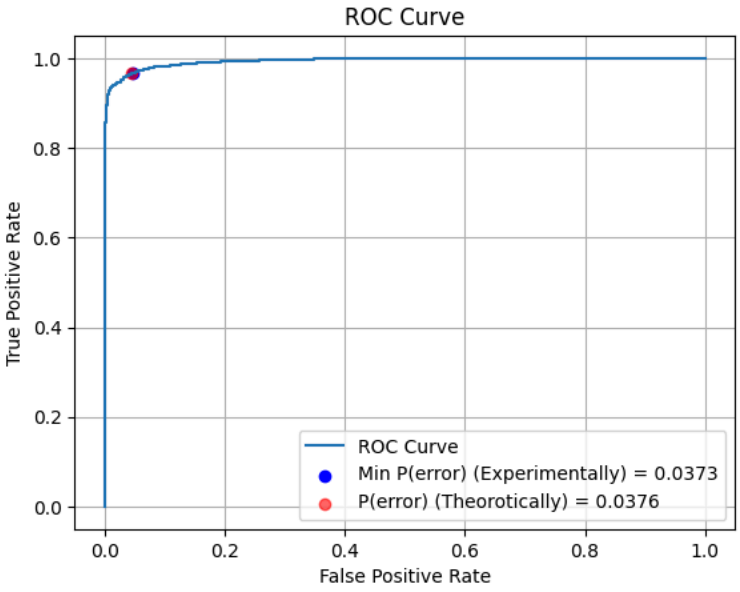
Out[ ]:

	index	Gamma	fpr	tpr	perror	perror - therotical
0	234	0.484447	0.047429	0.968154	0.0373	0.0376

In [ ]:
```
gamma_min_likelihood = df_likelihood_result.loc[0]["Gamma"]
fpr_min_likelihood = df_likelihood_result.loc[0]["fpr"]
```

```
tpr_min_likelihood = df_likelihood_result.loc[0]["tpr"]
perror_min_likelihood = df_likelihood_result.loc[0]["perror"]
```

In [ ]:
```
plt.figure()
plt.plot(fpr_list_likelihood, tpr_list_likelihood, label="ROC Curve")
plt.scatter(
    fpr_min_likelihood,
    tpr_min_likelihood,
    color="Blue",
    label=f"Min P(error) (Experimentally) = {perror_min_likelihood:.4f}",
)
plt.scatter(
    fpr_theoretical,
    tpr_theoretical,
    color="red",
    label=f"P(error) (Theorotically) = {p_error_theoretical:.4f}",
    alpha=0.6
)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:
```
print(f"Therotically Derived Gamma: {gamma_theoretical:.4f}")
print(f"Experimentally Derived Gamma: {gamma_min_likelihood:.4f}")
print(f"Delta: {abs(gamma_min_likelihood - gamma_theoretical):.4f}")

print(f"Therotically Derived PError: {p_error_theoretical:.4f}")
print(f"Experimentally Derived PError: {perror_min_likelihood:.4f}")
print(f"Delta: {abs(perror_min_likelihood - p_error_theoretical):.4f}")
```

```
Therotically Derived Gamma: 0.5385
Experimentally Derived Gamma: 0.4844
Delta: 0.0540
Therotically Derived PError: 0.0376
Experimentally Derived PError: 0.0373
Delta: 0.0003
```

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 matrices that are diagonal (with diagonal entries equal to true variances, off-diagonal entries equal to zeros, consistent with the independent feature assumption of Naive Bayes). 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?

In [ ]:
```
mean0 = np.array([-1, -1, -1, -1])
covariance0_naive = np.array([[2, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 2]])

mean1 = np.array([1, 1, 1, 1])
covariance1_naive = np.array([[1, 0, 0, 0], [0, 2, 0, 0], [0, 0, 1, 0], [0, 0, 0, 3]])
```

In [ ]:
```
px_given_l0_naive = multivariate_normal.pdf(samples, mean=mean0, cov=covariance0_naive)
px_given_l1_naive = multivariate_normal.pdf(samples, mean=mean1, cov=covariance1_naive)
likelihood_ratio_naive = px_given_l1_naive / px_given_l0_naive
```

In [ ]:
```
fpr_list_naive, tpr_list_naive, gammas = roc_curve(labels, likelihood_ratio_naive)
```

In [ ]:
```
perror_list_naive = fpr_list_naive * p_class_0 + (1 - tpr_list_naive) * p_class_1
```

In [ ]:
```
df_likelihood_naive = pd.DataFrame(gammas, columns=["Gamma"])
df_likelihood_naive["fpr"] = fpr_list_naive
df_likelihood_naive["tpr"] = tpr_list_naive
df_likelihood_naive["perror"] = perror_list_naive
df_likelihood_naive["perror - therotical"] = p_error_theoretical
```

In [ ]:
```
df_likelihood_ratio_naive_result = df_likelihood_naive[
    df_likelihood_naive["perror"] == df_likelihood_naive["perror"].min()
].reset_index()
df_likelihood_ratio_naive_result
```

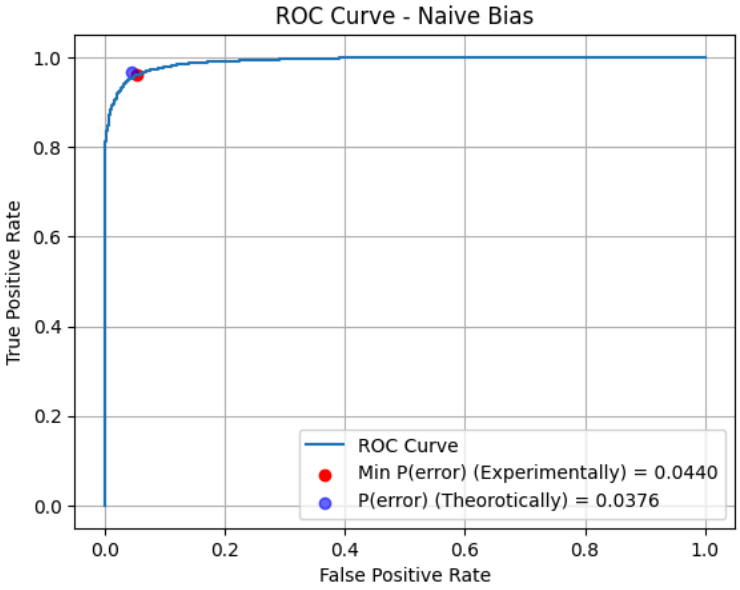
Out[ ]:

	index	Gamma	fpr	tpr	perror	perror - therotical
0	286	0.504565	0.052857	0.960769	0.044	0.0376

In [ ]:
```
gamma_min_likelihood_naive = df_likelihood_ratio_naive_result.loc[0]["Gamma"]
fpr_min_likelihood_naive = df_likelihood_ratio_naive_result.loc[0]["fpr"]
tpr_min_likelihood_naive = df_likelihood_ratio_naive_result.loc[0]["tpr"]
perror_min_likelihood_naive = df_likelihood_ratio_naive_result.loc[0]["perror"]
```

In [ ]:
```
plt.figure()
plt.plot(fpr_list_naive, tpr_list_naive, label="ROC Curve")
plt.scatter(
    [fpr_min_likelihood_naive],
    [tpr_min_likelihood_naive],
    color="red",
    label=f"Min P(error) (Experimentally) = {perror_min_likelihood_naive:.4f}",
)
plt.scatter(
    fpr_theoretical,
    tpr_theoretical,
    color="blue",
    label=f"P(error) (Theorotically) = {p_error_theoretical:.4f}",
    alpha=0.6
)
plt.plot([0, 0], [1, 1])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
```

```
plt.title("ROC Curve - Naive Bias")
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:
```
gamma_therotically_cal = p_class_0 / p_class_1
print(f"Therotically Derived Gamma: {gamma_therotically_cal:.4f}")
print(f"Experimentally Derived Gamma: {gamma_min_likelihood_naive:.4f}")
print(f"Delta: {abs(gamma_min_likelihood_naive - gamma_therotically_cal):.4f}")

print(f"Therotically Derived PError: {p_error_theoretical:.4f}")
print(f"Experimentally Derived PError: {perror_min_likelihood_naive:.4f}")
print(f"Delta: {abs(perror_min_likelihood - perror_min_likelihood_naive):.4f}")
```

```
Therotically Derived Gamma: 0.5385
Experimentally Derived Gamma: 0.5046
Delta: 0.0339
Therotically Derived PError: 0.0376
Experimentally Derived PError: 0.0440
Delta: 0.0067
```

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.

In [ ]:
```
def perform_lda(X, y):
    classes = np.unique(y)
    n_classes = len(classes)
    n_components = n_classes - 1

    mean_overall = np.mean(X, axis=0)

    means = []
    for cls in classes:
        means.append(np.mean(X[y == cls], axis=0))
    means = np.array(means)

    Sw = np.zeros((X.shape[1], X.shape[1]))
    for cls, mean in zip(classes, means):
        class_scatter = np.cov(X[y == cls].T)
        Sw += class_scatter * (np.sum(y == cls) - 1)

    Sb = np.zeros((X.shape[1], X.shape[1]))
    for cls, mean in zip(classes, means):
```

```
        n = np.sum(y == cls)
        mean_diff = (mean - mean_overall).reshape(-1, 1)
        Sb += n * (mean_diff @ mean_diff.T)

    evals, evecs = linalg.eigh(Sb, Sw)

    indices = np.argsort(evals)[::-1]
    evecs = evecs[:, indices]
    evals = evals[indices]

    w = evecs[:, :n_components]

    y_new = X @ w

    return w, y_new
```

In [ ]:
```
w, y = perform_lda(samples, labels)
```

In [ ]:
```
fpr, tpr, thresholds = roc_curve(labels, y)
```

In [ ]:
```
perror = fpr * p_class_0 + (1 - tpr) * p_class_1
```

In [ ]:
```
df_lda = pd.DataFrame(thresholds, columns=["Taus"])
df_lda["fpr"] = fpr
df_lda["tpr"] = tpr
df_lda["perror"] = perror
df_lda["perror - therotical"] = p_error_theoretical
```

In [ ]:
```
df_lda_result = df_lda[df_lda["perror"] == df_lda["perror"].min()].reset_index()
```

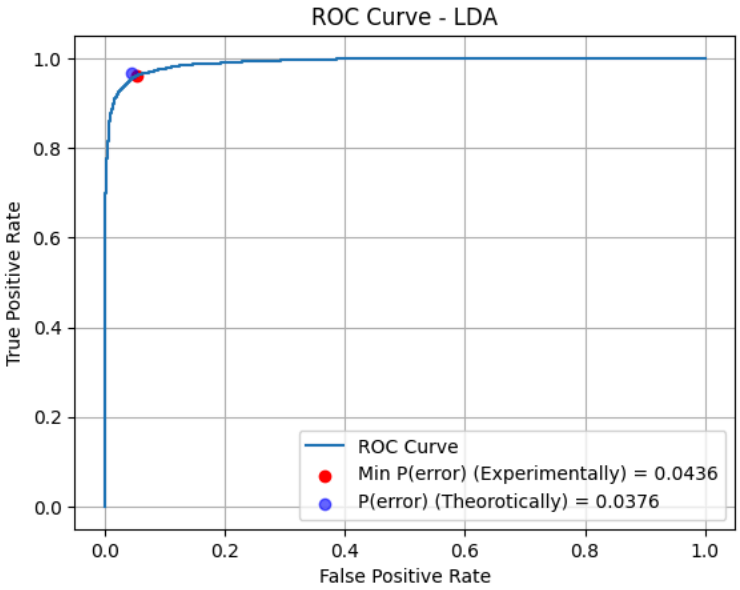
In [ ]:
```
df_lda_result
```

Out[ ]:

	index	Taus	fpr	tpr	perror	perror - therotical
0	304	-0.001791	0.053429	0.961692	0.0436	0.0376

In [ ]:
```
tau_min_lda = df_lda_result.loc[0]["Taus"]
fpr_min_lda = df_lda_result.loc[0]["fpr"]
tpr_min_lda = df_lda_result.loc[0]["tpr"]
perror_min_lda = df_lda_result.loc[0]["perror"]
```

In [ ]:
```
plt.figure()
plt.plot(fpr, tpr, label="ROC Curve")
plt.scatter(
    [fpr_min_lda],
    [tpr_min_lda],
    color="red",
    label=f"Min P(error) (Experimentally) = {perror_min_lda:.4f}",
)
plt.scatter(
    fpr_theoretical,
    tpr_theoretical,
    color="blue",
    label=f"P(error) (Theorotically) = {p_error_theoretical:.4f}",
    alpha=0.6
)
plt.plot([0, 0], [1, 1])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve - LDA")
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:
```
print(f"Therotically Derived PError: {p_error_theoretical:.4f}")
print(f"Experimentally Derived PError: {perror_min_lda:.4f}")
print(f"Delta: {abs(perror_min_likelihood - perror_min_lda):.4f}")
```

```
Therotically Derived PError: 0.0376
Experimentally Derived PError: 0.0436
Delta: 0.0063
```

In [ ]:
```
df_likelihood_result
```

Out[ ]:

	index	Gamma	fpr	tpr	perror	perror - therotical
0	234	0.484447	0.047429	0.968154	0.0373	0.0376

In [ ]:
```
df_likelihood_ratio_naive_result
```

Out[ ]:

	index	Gamma	fpr	tpr	perror	perror - therotical
0	286	0.504565	0.052857	0.960769	0.044	0.0376

In [ ]:
```
df_lda_result
```

Out[ ]:

	index	Taus	fpr	tpr	perror	perror - therotical
0	304	-0.001791	0.053429	0.961692	0.0436	0.0376

It can be observed that there is difference in the theoretical and experimental values of the thresholds and also the probabilities, the true nature of the model can be further understood when validated against a validation set.

perror(likelihood) < perrror(lda) < perror(naive_bais) which is expected as we have all the information while creating the likelihood model.

Assignment 1 - Question 2

In [ ]:
```
import numpy as np
import pandas as pd
from scipy.stats import multivariate_normal
import warnings
import random
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

plt.style.use('ggplot')
np.random.seed(37)
warnings.filterwarnings('ignore')
```

In [ ]:
```
np.random.seed(2)
```

Question 2 (40%)

A 3-dimensional random vector $\mathbf{X}$ takes values from a mixture of four Gaussians. One of these Gaussians represent the class-conditional pdf for class 1, and another Gaussian represents the class-conditional pdf for class 2. Class 3 data originates from a mixture of the remaining 2 Gaussian components with *equal weights*. For this setting where labels $L \in \{1,2,3\}$, pick your own class-conditional pdfs $p(\mathbf{x}|L=j),\ j \in \{1,2,3\}$ as described. Try to approximately set the distances between means of pairs of Gaussians to approximately 2 to 3 times the average standard deviation of the Gaussian compoenents, so that there is some significant overlap between class-conditional pdfs. Set class priors to $0.3, 0.3, 0.4$.

In [ ]:
```
def generate_gaussian_params(start_mean, start_cov, num_components=4):
    means = [start_mean]
    covariances = [start_cov]
    distance_factor = random.uniform(2.0, 3.0)

    avg_std = np.mean(np.sqrt(np.diag(start_cov)))
    target_distance = distance_factor * avg_std

    for _ in range(num_components - 1):
        while True:
            direction = np.random.randn(3)
            direction /= np.linalg.norm(direction)

            new_mean = start_mean + direction * target_distance

            if all(np.linalg.norm(new_mean - m) >= target_distance for m in means):
                means.append(new_mean)

                A = np.random.randn(3, 3)
                new_cov = np.dot(A, A.T)
                new_cov = new_cov * (np.sum(np.diag(start_cov)) / np.sum(np.diag(new_cov)))
                covariances.append(new_cov)

                break

    return means, covariances
```

In [ ]:
```
mean1 = [-1, -1, -1]
covariance1 = [[2, -0.5, 0.3], [-0.5, 1, -0.5], [0.3, -0.5, 1]]

means, covariances = generate_gaussian_params(mean1, covariance1)

priors = np.array([0.3, 0.3, 0.4])
```

In [ ]:
```
for i, mean in enumerate(means):
    print(f"Mean {i+1}: ", mean)
```

```
Mean 1:  [-1, -1, -1]
Mean 2:  [ 0.97351623 -3.15779871 -2.01276087]
Mean 3:  [-2.4000455  -1.04492683  1.75942029]
Mean 4:  [-2.97755192 -3.36653905 -1.25559244]
```

In [ ]:
```
for i, cov in enumerate(covariances):
    print(f"Cov {i+1}: \n", cov)
    print()
```

```
Cov 1:
 [[2, -0.5, 0.3], [-0.5, 1, -0.5], [0.3, -0.5, 1]]

Cov 2:
 [[ 1.07760713 -1.31577081  0.31069627]
 [-1.31577081  2.35386679  0.21434559]
 [ 0.31069627  0.21434559  0.56852608]]

Cov 3:
 [[1.7884787  1.32753134 1.09044667]
 [1.32753134 1.43563752 0.83734056]
 [1.09044667 0.83734056 0.77588378]]

Cov 4:
 [[ 3.14991402 -0.72127957 -0.5791761 ]
 [-0.72127957  0.5733632  -0.07348668]
 [-0.5791761  -0.07348668  0.27672278]]
```

Part A: Minimum probability of error classification (0-1 loss, also referred to as Bayes Decision rule or MAP classifier).

1. Generate 10000 samples from this data distribution and keep track of the true labels of each sample.
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 $i, j \in \{1,2,3\}$.
3. Provide a visualization of the data (scatter-plot in 3-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) classified with a different marker color as indicated in parentheses.

Part A - 1

In [ ]:
```python
mean2 = means[1]
mean3 = means[2]
mean4 = means[3]

covariance2 = covariances[1]
covariance3 = covariances[2]
covariance4 = covariances[3]

p_class_1 = 0.3
p_class_2 = 0.3
p_class_3 = 0.4

number_of_samples = 10000

num_samples_class1 = int(p_class_1 * number_of_samples)
num_samples_class2 = int(p_class_2 * number_of_samples)
num_samples_class3 = int(0.5 * p_class_3 * number_of_samples)
num_samples_class4 = int(0.5 * p_class_3 * number_of_samples)

samples_class1 = np.random.multivariate_normal(mean1, covariance1, num_samples_class1)
samples_class2 = np.random.multivariate_normal(mean2, covariance2, num_samples_class2)
samples_class3 = np.random.multivariate_normal(mean3, covariance3, num_samples_class3)
samples_class4 = np.random.multivariate_normal(mean4, covariance4, num_samples_class4)

samples_class3_combined = np.vstack((samples_class3, samples_class4))

labels_class1 = np.ones(num_samples_class1)
labels_class2 = 2 * np.ones(num_samples_class2)
labels_class3 = 3 * np.ones(num_samples_class3 + num_samples_class4)

samples = np.vstack((samples_class1, samples_class2, samples_class3_combined))
labels = np.hstack((labels_class1, labels_class2, labels_class3))
```

In [ ]:
```python
df = pd.DataFrame(samples, columns=['x1', 'x2', 'x3'])
df['Label'] = labels

df.to_csv('data_created_for_hw1_q2_10000_samples.csv', index=False)
```

In [ ]:
```python
df
```

Out[ ]:

	x1	x2	x3	Label
0	-1.460081	-0.071713	-1.157930	1.0
1	0.548614	-2.411437	1.891539	1.0
2	-0.909978	-1.529053	-0.361921	1.0
3	0.102712	-1.074835	-0.049047	1.0
4	-0.299916	-1.085325	-1.106438	1.0
...	...	...	...	...
9995	-1.737476	-4.571486	-1.091317	3.0
9996	-2.491255	-3.042180	-1.335511	3.0
9997	-0.753233	-3.819275	-1.649710	3.0
9998	-2.867664	-3.313555	-1.206537	3.0
9999	-3.238531	-3.985722	-0.943668	3.0

10000 rows × 4 columns

Part A - 2

Classification Rule for this problem is

$$D_{ERM}(X) = arg \min_{d \in \{1,2,3\}} R(D = d|X)$$

$$D_{MAP}(X) = \arg \min_{d \in \{1,2,3\}} (\lambda(d,1)p(1|X) + \lambda(d,2)p(2|X) + \lambda(d,3)p(3|X))$$

By using 0-1 loss matrix, we minimize the probability of error

$$D_{MAP}(X) = \arg \min_{d \in \{1,2,3\}} 1 - p(d|X)$$

$$D_{MAP}(X) = \arg \max_{d \in \{1,2,3\}} p(d|X)$$

Using Bayes Theorem we get,

$$D_{MAP}(X) = \arg \max_{d \in \{1,2,3\}} p(x|d).P(d)$$

In [ ]:
```python
def MAP(samples, number_of_samples, priors, means, covariances, lossMatrix):
    N = number_of_samples
    C = len(priors)
    pxgivenl = np.zeros((C, N))

    pxgivenl[0, :] = multivariate_normal.pdf(samples, mean=means[0], cov=covariances[0])
    pxgivenl[1, :] = multivariate_normal.pdf(samples, mean=means[1], cov=covariances[1])
    pxgivenl[2, :] = 0.5 * (multivariate_normal.pdf(samples, mean=means[2], cov=covariances[2]) +
                            multivariate_normal.pdf(samples, mean=means[3], cov=covariances[3]))

    px = priors @ pxgivenl
    classPosteriors = (priors[:, np.newaxis] * pxgivenl) / px
    expectedRisks = lossMatrix @ classPosteriors
    decisions = np.argmin(expectedRisks, axis=0)

    return decisions
```

In [ ]:
```python
def plot_result(decisions, labels):
    markers = "o*^"
    marker_size = 40
    alpha = 0.7

    fig, axs = plt.subplots(1, 3, figsize=(18, 6), subplot_kw={'projection': '3d'})

    titles = ['Class 1 Samples', 'Class 2 Samples', 'Class 3 Samples']

    for j, ax in enumerate(axs):
        wrong = 0
        for i in range(len(samples)):
            if labels[i] == j+1:
                x, y, z = samples[i]
                true_label = int(labels[i])
                predicted_label = int(decisions[i]) + 1

                if true_label == predicted_label:
                    color = 'green'
                else:
                    color = 'red'
                    wrong+=1

                ax.scatter(x, y, z, marker=markers[j], color=color, s=marker_size, alpha=alpha)
        print(f"Wrongly Labeled {j+1}: ", wrong)
```

```
        ax.set_xlabel('X axis', fontsize=10)
        ax.set_ylabel('Y axis', fontsize=10)
        ax.set_zlabel('Z axis', fontsize=10)
        ax.scatter([], [], [], color='green', marker=markers[j], label='Prediction == Label')
        ax.scatter([], [], [], color='red', marker=markers[j], label='Prediction != Label')
        ax.legend(loc='upper right')
        ax.set_title(titles[j], fontsize=12)
        ax.view_init(elev=30, azim=120)

    plt.tight_layout()
    plt.show()
```

In [ ]:
```
def getConfusionMatrix(labels, decisions):
    confusion_matrix = np.zeros((3, 3), dtype=int)
    for true, pred in zip(labels, decisions):
        true = int(true)
        pred += 1
        confusion_matrix[pred - 1, true - 1] += 1

    return confusion_matrix
```

In [ ]:
```
lossMatrix_0_1 = np.array([[0, 1, 1],
                           [1, 0, 1],
                           [1, 1, 0]])

decisions_0_1 = MAP(samples, number_of_samples, priors, means, covariances, lossMatrix_0_1)
```

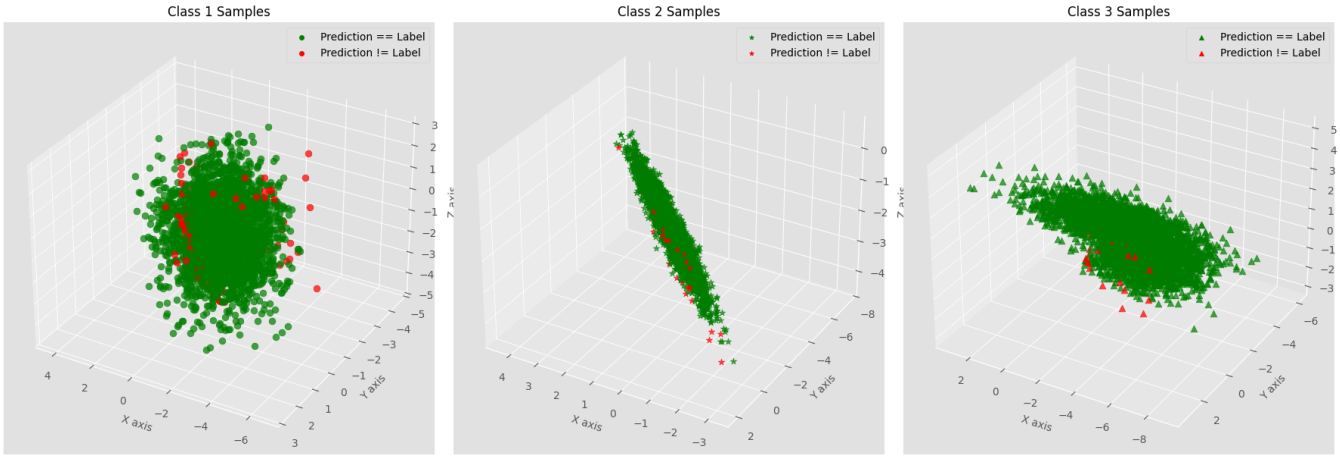
In [ ]:
```
getConfusionMatrix(labels, decisions_0_1)
```

Out[ ]:
```
array([[2825,   29,   17],
       [ 137, 2971,   12],
       [  38,    0, 3971]])
```

Part A - 3

In [ ]:
```
plot_result(decisions_0_1, labels)
```

```
Wrongly Labeled 1:  175
Wrongly Labeled 2:  29
Wrongly Labeled 3:  29
```



Part B: Repeat the exercise for the ERM classification rule with the following loss matrices which respectively care 10 times or 100 times more about not making mistakes when $L = 3$:

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

Note that, the $(i, j)^{th}$ entry of the loss matrix indicates the loss incurred by deciding on class i when the true label is j. 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.

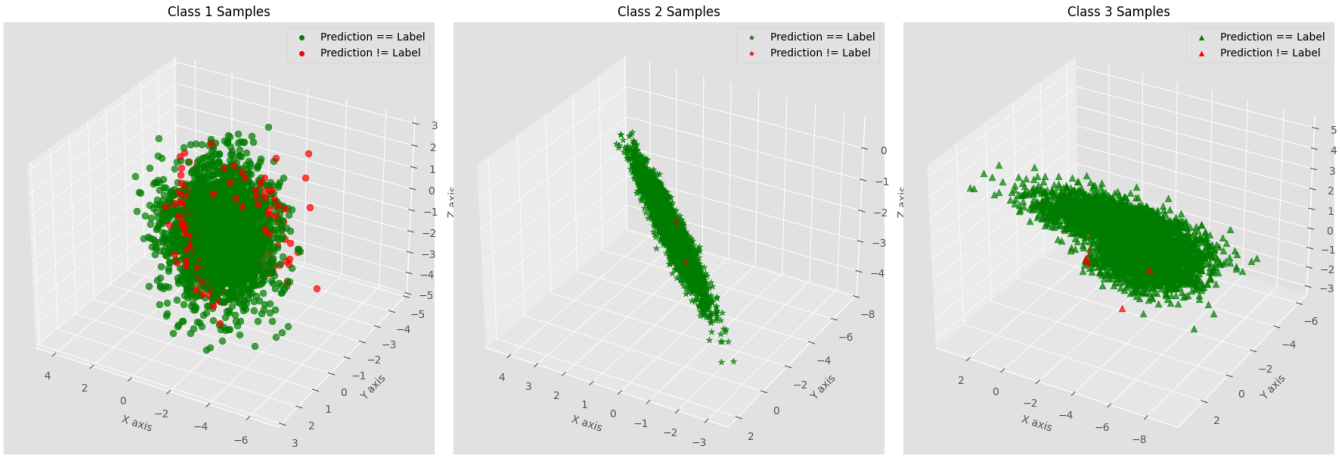
In [ ]:
```
lossMatrix_0_10 = np.array([[0, 10, 10],
                            [1, 0, 10],
                            [1, 1, 0]])

decisions_0_10 = MAP(samples, number_of_samples, priors, means, covariances, lossMatrix_0_10)
```

In [ ]:
```
getConfusionMatrix(labels, decisions_0_10)
```

Out[ ]:
```
array([[2714,    1,    2],
       [ 207, 2990,    8],
       [  79,    9, 3990]])
```

In [ ]:
```
plot_result(decisions_0_10, labels)
```

```
Wrongly Labeled 1:  286
Wrongly Labeled 2:  10
Wrongly Labeled 3:  10
```



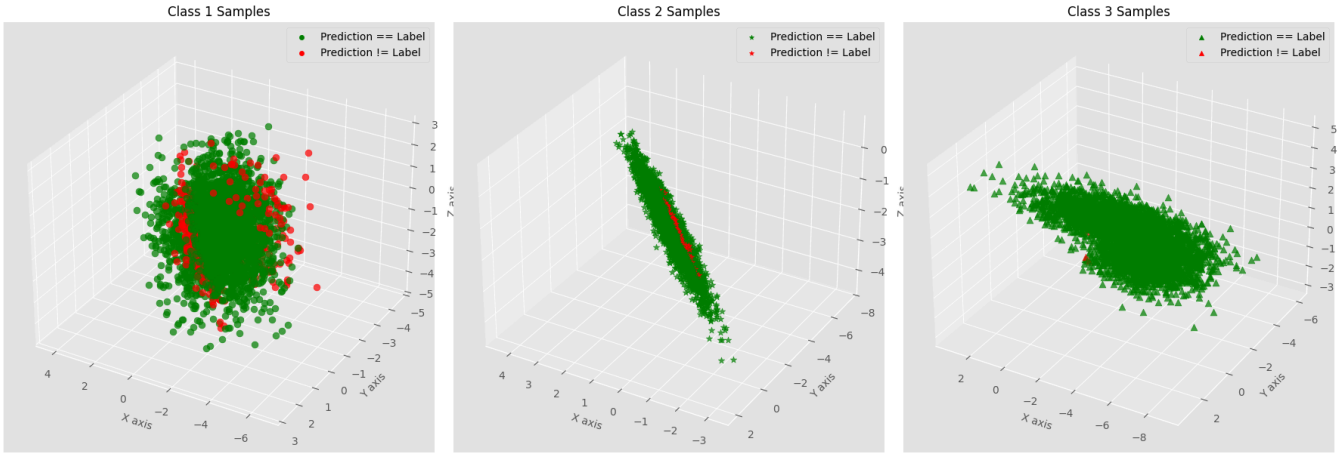
In [ ]:
```
lossMatrix_0_100 = np.array([[0, 100, 100],
                             [1, 0, 100],
                             [1, 1, 0]])

decisions_0_100 = MAP(samples, number_of_samples, priors, means, covariances, lossMatrix_0_100)
```

In [ ]:
```
getConfusionMatrix(labels, decisions_0_100)
```

Out[ ]:
```
array([[2609,    0,    0],
       [ 253, 2823,    4],
       [ 138,  177, 3996]])
```

In [ ]:
```
plot_result(decisions_0_100, labels)
```

```
Wrongly Labeled 1:  391
Wrongly Labeled 2:  177
Wrongly Labeled 3:  4
```



Results from the confusion matrix and from the plots both suggests that the loss matrix (0-1) performed the best in all the three loss matrices. Introducing a greater loss does make the model better in a way that we can more accurately predict that class in this case class 3 but negatively impacts the other two classes.

Assignment 1 - Question 3

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.stats import multivariate_normal
import warnings
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler


plt.style.use("ggplot")
np.random.seed(37)
warnings.filterwarnings("ignore")
```

Question 3 (30%)

Download the following datasets...

- Wine Quality dataset located at `https://archive.ics.uci.edu/ml/datasets/Wine+Quality` consists of 11 features, and class labels from 0 to 10 indicating wine quality scores. There are 4898 samples.
- Human Activity Recognition dataset located at `https://archive.ics.uci.edu/ml/datasets/Human+Activity+Recognition+Using+Smartphones` consists of 561 features, and 6 activity labels. There are 10299 samples.

In [ ]:
```
winequality_red_df = pd.read_csv(
    "/Users/pradnyeshchoudhari/EECE 5644 - LOCAL/Assignments/Assignment 01/wine+quality/winequality-red.csv",
    delimiter=";",
)
winequality_white_df = pd.read_csv(
    "/Users/pradnyeshchoudhari/EECE 5644 - LOCAL/Assignments/Assignment 01/wine+quality/winequality-white.csv",
    delimiter=";",
)
```

In [ ]:
```
X_train = pd.read_csv(
    "/Users/pradnyeshchoudhari/EECE 5644 - LOCAL/Assignments/Assignment 01/human+activity+recognition+using+smartphones/UCI HAR
    delim_whitespace=True,
    header=None,
)
```

In [ ]:
```
X_test = pd.read_csv(
    "/Users/pradnyeshchoudhari/EECE 5644 - LOCAL/Assignments/Assignment 01/human+activity+recognition+using+smartphones/UCI HAR
    delim_whitespace=True,
    header=None,
)
```

In [ ]:
```
Y_train = pd.read_csv(
    "/Users/pradnyeshchoudhari/EECE 5644 - LOCAL/Assignments/Assignment 01/human+activity+recognition+using+smartphones/UCI HAR
    delim_whitespace=True,
    header=None,
)
```

In [ ]:
```
Y_test = pd.read_csv(
    "/Users/pradnyeshchoudhari/EECE 5644 - LOCAL/Assignments/Assignment 01/human+activity+recognition+using+smartphones/UCI HAR
    delim_whitespace=True,
    header=None,
)
```

In [ ]:
```
winequality_df = pd.concat(
    [winequality_red_df, winequality_white_df], ignore_index=True
)
```

In [ ]:
```
print("Red Wine df Shape: ", winequality_red_df.shape)
print("White Wine df Shape: ", winequality_white_df.shape)
print("Combined Wine df Shape: ", winequality_df.shape)
print()
print("Smartphone df X_train Shape: ", X_train.shape)
print("Smartphone df Y_train Shape: ", Y_train.shape)
print("Smartphone df X_test Shape: ", X_test.shape)
print("Smartphone df Y_test Shape: ", Y_test.shape)
```

```
Red Wine df Shape:  (1599, 12)
White Wine df Shape:  (4898, 12)
Combined Wine df Shape:  (6497, 12)

Smartphone df X_train Shape:  (7352, 561)
Smartphone df Y_train Shape:  (7352, 1)
Smartphone df X_test Shape:  (2947, 561)
Smartphone df Y_test Shape:  (2947, 1)
```

Implement minimum-probability-of-error classifiers for these problems, assuming that the class conditional pdf of features for each class you encounter in these examples is a Gaussian. Using all available samples from a class, with sample averages, estimate mean vectors and covariance matrices. Using sample counts, also estimate class priors. In case your sample estimates of covariance matrices are ill-conditioned, consider adding a regularization term to your covariance estimate as in: $\mathbf{C}_{Regularized} = \mathbf{C}_{SampleAverage} + \lambda\mathbf{I}$ where $\lambda > 0$ is a small regularization parameter that ensures the regularized covariance matrix $\mathbf{C}_{Regularized}$ has all eigenvalues larger than this parameter.

With these estimated (trained) Gaussian class conditional pdfs and class priors, apply the minimum-P(error) classification rule on all (training) samples, count the errors, and report the error probability estimate you obtain for each problem. Also report the confusion matrices for both datasets, for this classification rule.

Functions to apply Minimum Probability of Error classifier, creating the confusion matrix and getting the perror based on the confusion matrix.

In [ ]:
```python
def getConfusionMatrix(labels, decisions, n):
    confusion_matrix = np.zeros((n, n), dtype=int)
    for true, pred in zip(labels, decisions):
        true = int(true)
        confusion_matrix[pred, true] += 1

    return confusion_matrix
```

In [ ]:
```python
def getPerror(confusion_matrix):
    total_predictions = confusion_matrix.sum()

    correct_classifications = confusion_matrix.diagonal().sum()
    misclassifications = total_predictions - correct_classifications

    perror = misclassifications / total_predictions

    return perror
```

In [ ]:
```python
def applyMapOnDataset(case, df=None, target=None, X=None, y=None):
    if case == 1:
        feature_list = df.columns.to_list()[:-1]
        features = df[feature_list].values
        labels = df[target].values
        potential_labels = [i for i in range(0, 11)]

    if case == 2:
        features = X.values
        labels = y.T.values[0]
        potential_labels = [i for i in range(0, 7)]

    pl = len(potential_labels)

    rows, cols = features.shape[0], features.shape[1]

    priors = np.zeros(pl)
    unique_labels = np.unique(labels)

    means = {}
    cov = {}

    for cls in unique_labels:
        count_cls = (labels == cls).sum()
        priors[cls] = round((count_cls / rows), 2)
        means[cls] = np.mean(features[labels == cls], axis=0)
        epl = 1e-2
        cov[cls] = np.cov(features[labels == cls], rowvar=False) + epl * np.eye(cols)

    N = rows
    C = pl
    pxgivenl = np.zeros((C, N))

    for cls in unique_labels:
        pxgivenl[cls, :] = multivariate_normal.pdf(
            features, mean=means[cls], cov=cov[cls]
        )

    px = priors @ pxgivenl
    classPosteriors = (priors[:, np.newaxis] * pxgivenl) / px
```

```
    lossMatrix_0_1 = np.ones((C, C)) - np.eye(C)

    expectedRisks = lossMatrix_0_1 @ classPosteriors

    decisions = np.argmin(expectedRisks, axis=0)

    mat = getConfusionMatrix(labels, decisions, pl)

    perror = getPerror(mat)

    model_params = {
        "priors": priors,
        "means": means,
        "cov": cov,
        "unique_labels": unique_labels,
    }

    return decisions, mat, perror, model_params
```

In [ ]:
```
def predictMapModel(model_params, test_features):
    priors = model_params["priors"]
    means = model_params["means"]
    cov = model_params["cov"]
    unique_labels = model_params["unique_labels"]

    N = test_features.shape[0]
    C = len(priors)
    pxgivenl = np.zeros((C, N))

    for cls in unique_labels:
        pxgivenl[cls, :] = multivariate_normal.pdf(
            test_features, mean=means[cls], cov=cov[cls]
        )

    px = priors @ pxgivenl
    classPosteriors = (priors[:, np.newaxis] * pxgivenl) / px

    lossMatrix_0_1 = np.ones((C, C)) - np.eye(C)
    expectedRisks = lossMatrix_0_1 @ classPosteriors

    decisions = np.argmin(expectedRisks, axis=0)

    return decisions
```

Wine Quality Dataset

Case 1: Only Red Wine Samples

In [ ]:
```
decisions_out_red, confusion_matrix_red, perror_red, model_params_red = (
    applyMapOnDataset(1, df=winequality_red_df, target="quality")
)
```

In [ ]:
```
print("Confusion Matrix Red Wines")
confusion_matrix_red
```

```
Confusion Matrix Red Wines
```

Out[ ]:
```
array([[  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,   7,   5,   9,   1,   0,   0,   0,   0],
       [  0,   0,   0,   0,   3,  10,   6,   0,   0,   0,   0],
       [  0,   0,   0,   2,  28, 483, 189,  11,   1,   0,   0],
       [  0,   0,   0,   1,  15, 167, 398, 126,   8,   0,   0],
       [  0,   0,   0,   0,   2,  12,  40,  59,   4,   0,   0],
       [  0,   0,   0,   0,   0,   0,   4,   3,   5,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0]])
```

In [ ]:
```
print(f"PError Red Wines Data: {perror_red:.4f}")
```

```
PError Red Wines Data: 0.4028
```

Wine Quality Dataset

Case 2: Only White Wine Samples

In [ ]:
```
decisions_out_white, confusion_matrix_white, perror_white, model_params_white = (
    applyMapOnDataset(1, df=winequality_white_df, target="quality")
)
```

In [ ]:
```
print("Confusion Matrix White Wines")
confusion_matrix_white
```

```
Confusion Matrix White Wines
```

Out[ ]:
```
array([[   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,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    5,   20,   18,    9,    0,    0,    0,    0],
       [   0,    0,    0,    5,   94,  901,  551,   60,    8,    0,    0],
       [   0,    0,    0,    9,   47,  510, 1326,  442,   81,    2,    0],
       [   0,    0,    0,    0,    2,   26,  308,  372,   78,    3,    0],
       [   0,    0,    0,    1,    0,    2,    4,    6,    8,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0]])
```

In [ ]:
```
print(f"PError White Wines Data: {perror_white:.4f}")
```

```
PError White Wines Data: 0.4637
```

Wine Quality Dataset

Case 3: Combined Wine Samples

In [ ]:
```
decisions_out_wines, confusion_matrix_wines, perror_wines, model_params_wine = (
    applyMapOnDataset(1, df=winequality_df, target="quality")
)
```

In [ ]:
```
print("Confusion Matrix Combined Wines")
confusion_matrix_wines
```

```
Confusion Matrix Combined Wines
```

Out[ ]:
```
array([[   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,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    4,    9,   14,    7,    0,    0,    0,    0],
       [   0,    0,    0,   11,  122, 1353,  742,   75,    7,    0,    0],
       [   0,    0,    0,    9,   78,  704, 1633,  523,   87,    2,    0],
       [   0,    0,    0,    2,    5,   64,  451,  472,   93,    3,    0],
       [   0,    0,    0,    4,    2,    3,    3,    9,    6,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0],
       [   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0]])
```

In [ ]:
```
print(f"PError Combined Wines Data: {perror_wines:.4f}")
```

```
PError Combined Wines Data: 0.4654
```

Human Activity Dataset

In [ ]:
```
decisions_out_train, confusion_matrix_train, perror_train, model_params_train = (
    applyMapOnDataset(2, X=X_train, y=Y_train)
)
```

In [ ]:
```
np.unique(decisions_out_train)
```

Out[ ]:
```
array([1, 2, 3, 4, 5, 6])
```

In [ ]:
```
print("Confusion Matrix Train Data Human Activity Dataset")
confusion_matrix_train
```

```
Confusion Matrix Train Data Human Activity Dataset
```

Out[ ]:
```
array([[   0,    0,    0,    0,    0,    0,    0],
       [   0, 1226,    0,    0,    0,    0,    0],
       [   0,    0, 1073,    1,    0,    0,    0],
       [   0,    0,    0,  985,    0,    0,    0],
       [   0,    0,    0,    0, 1196,    1,    0],
       [   0,    0,    0,    0,   90, 1373,    0],
       [   0,    0,    0,    0,    0,    0, 1407]])
```

In [ ]:
```
print(f"PError Train Data Human Activity Dataset: {perror_train:.4f}")
```

```
PError Train Data Human Activity Dataset: 0.0125
```

In [ ]:
```
decisions_out_test = predictMapModel(model_params_train, X_test.values)
```

In [ ]:
```
labels_test = Y_test.T.values[0]
```

In [ ]:
```
mat_test = getConfusionMatrix(labels_test, decisions_out_test, 7)
```

In [ ]:
```
print("Confusion Matrix Test Data Human Activity Dataset")
mat_test
```

```
Confusion Matrix Test Data Human Activity Dataset
```

Out[ ]:
```
array([[  0,   0,   0,   0,   0,   0,   0],
       [  0, 482,   0,   4,   0,   0,   0],
       [  0,   1, 470,  42,   0,   0,   0],
       [  0,  13,   1, 374,   0,   0,   0],
       [  0,   0,   0,   0, 387,   7,   0],
       [  0,   0,   0,   0, 103, 525,   0],
       [  0,   0,   0,   0,   1,   0, 537]])
```

In [ ]:
```
print(
    f"PError Test Data Human Activity Dataset: {getPerror(confusion_matrix = mat_test):.4f}"
)
```

```
PError Test Data Human Activity Dataset: 0.0584
```

Visualize the datasets in various 2 or 3 dimensional projections (either subsets of features, or using the first few principal components). Discuss if Gaussian class conditional models are appropriate for these datasets and how your model choice might have influenced the confusion matrix and probability of error values you obtained in the experiments conducted above. Make sure you explain in rigorous detail what your modeling assumptions are, how you estimated/selected necessary parameters for your model and classification rule, and describe your analyses in mathematical terms supplemented by numerical and visual results in a way that conveys your understanding of what you have accomplished and demonstrated.

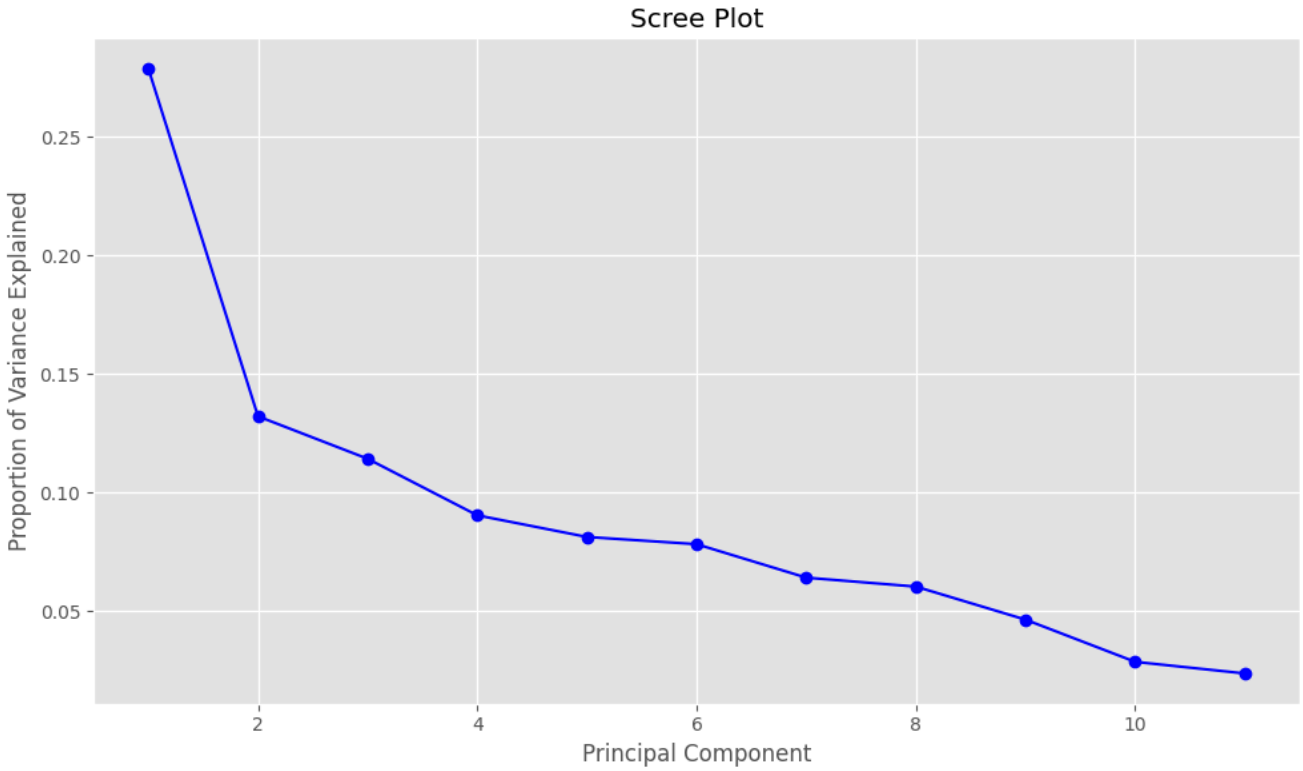
White Wine Dataset

In [ ]:
```
scaler = StandardScaler()
```

In [ ]:
```
features_white = winequality_white_df.values
labels_white = winequality_white_df["quality"].values
```

In [ ]:
```
features_white_scaled = scaler.fit_transform(features_white)
pca_white = PCA(n_components=11)
components_white = pca_white.fit_transform(features_white_scaled)
```

In [ ]:
```
explained_variance_ratio = pca_white.explained_variance_ratio_
plt.figure(figsize=(10, 6))
plt.plot(
    range(1, len(explained_variance_ratio) + 1, 1), explained_variance_ratio, "bo-"
)
plt.xlabel("Principal Component")
plt.ylabel("Proportion of Variance Explained")
plt.title("Scree Plot")

plt.grid(True)

plt.tight_layout()
plt.show()
```

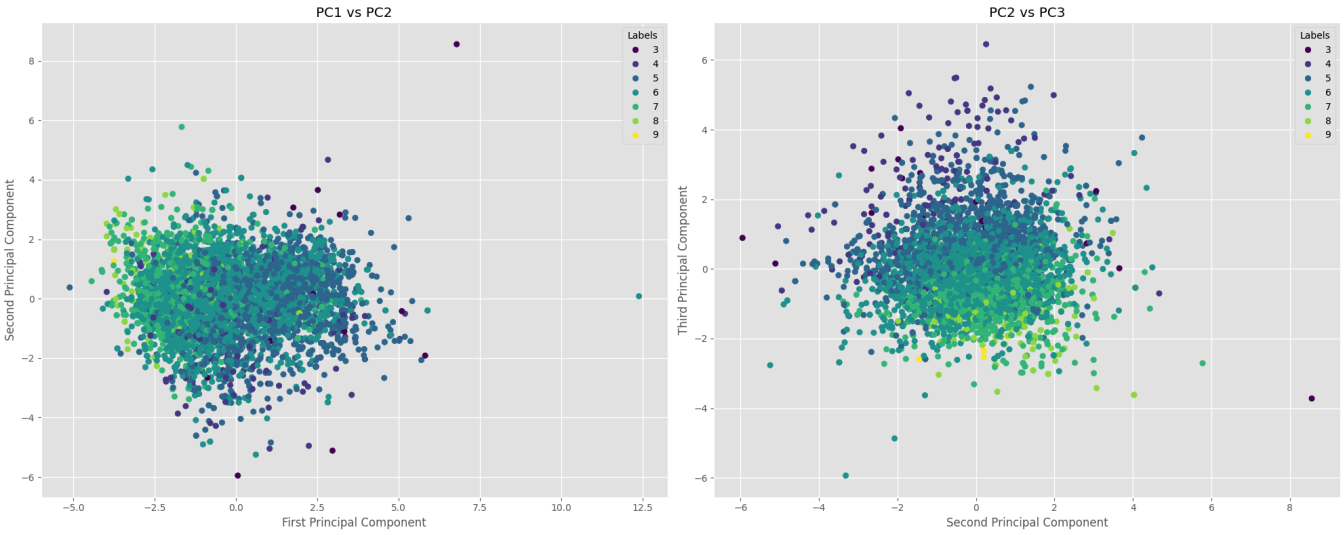


In [ ]:
```
print("Explained Variance Ratio:")
print(f"PC1: {pca_white.explained_variance_ratio_[0]:.4f}")
print(f"PC2: {pca_white.explained_variance_ratio_[1]:.4f}")
print(f"PC3: {pca_white.explained_variance_ratio_[2]:.4f}")
print(f"\nTotal Variance Explained:")
print(
    f"PC1 + PC2: {pca_white.explained_variance_ratio_[0] + pca_white.explained_variance_ratio_[1]:.4f}"
)
```

```
print(
    f"PC2 + PC3: {pca_white.explained_variance_ratio_[1] + pca_white.explained_variance_ratio_[2]:.4f}"
)
```

```
Explained Variance Ratio:
PC1: 0.2789
PC2: 0.1322
PC3: 0.1143

Total Variance Explained:
PC1 + PC2: 0.4111
PC2 + PC3: 0.2464
```

In [ ]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8))

scatter1 = ax1.scatter(components_white[:, 0], components_white[:, 1], c=labels_white)
ax1.set_xlabel("First Principal Component")
ax1.set_ylabel("Second Principal Component")
ax1.set_title("PC1 vs PC2")
ax1.grid(True)

legend1 = ax1.legend(*scatter1.legend_elements(), title="Labels", loc="upper right")
ax1.add_artist(legend1)

scatter2 = ax2.scatter(components_white[:, 1], components_white[:, 2], c=labels_white)
ax2.set_xlabel("Second Principal Component")
ax2.set_ylabel("Third Principal Component")
ax2.set_title("PC2 vs PC3")
ax2.grid(True)

legend2 = ax2.legend(*scatter2.legend_elements(), title="Labels", loc="upper right")
ax2.add_artist(legend2)

plt.tight_layout()
plt.show()
```

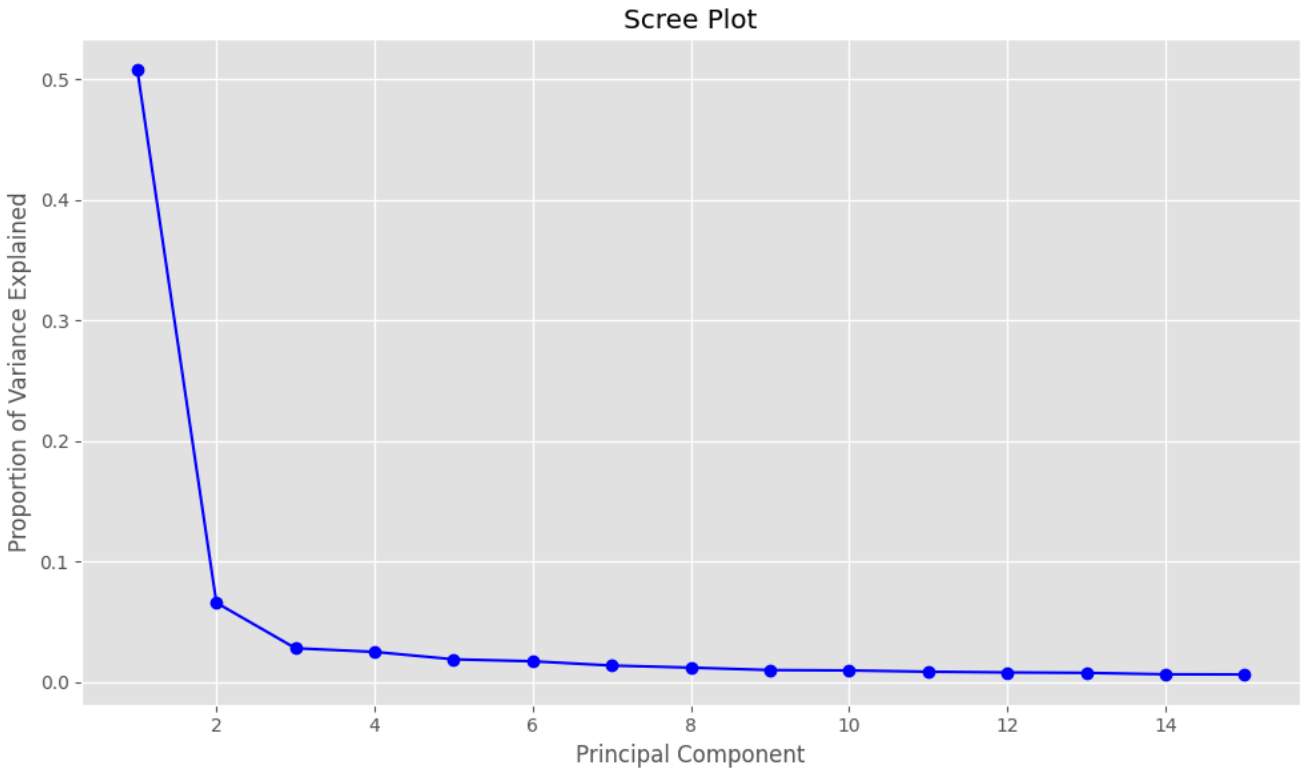


Human Activity Dataset

In [ ]:
```
features_harr = X_train.values
labels_harr = Y_train.values
```

In [ ]:
```
scaler = StandardScaler()
features_harr_scaled = scaler.fit_transform(features_harr)
pca_harr = PCA(n_components=15)
pca_out_harr = pca_harr.fit_transform(features_harr_scaled)
```

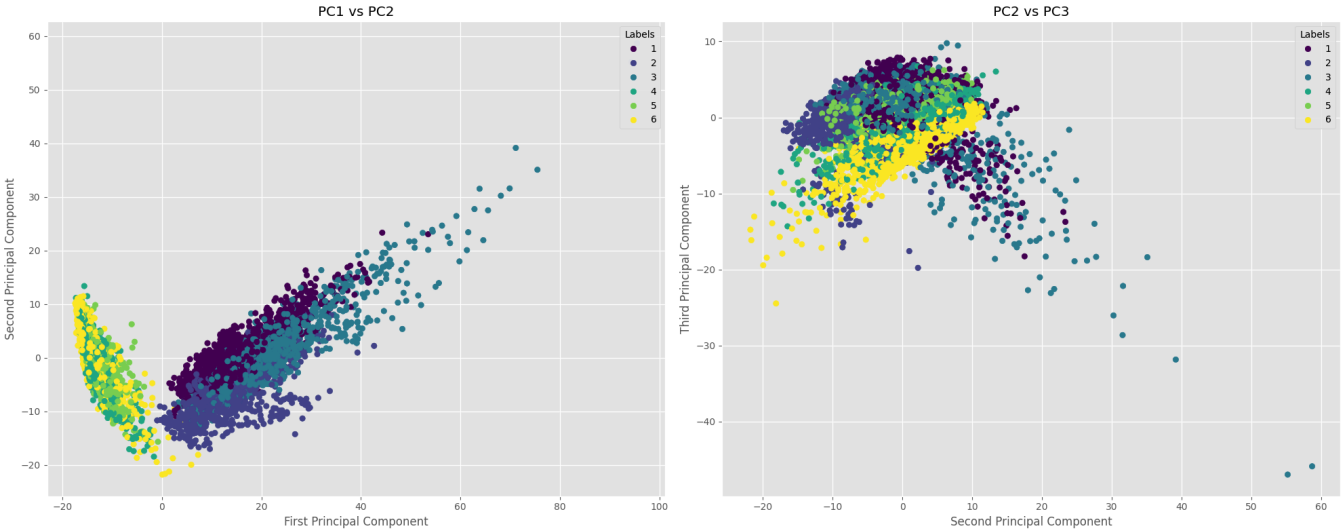
In [ ]:
```
explained_variance_ratio = pca_harr.explained_variance_ratio_
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio, "bo-")
plt.xlabel("Principal Component")
plt.ylabel("Proportion of Variance Explained")
plt.title("Scree Plot")
plt.grid(True)

plt.tight_layout()
plt.show()
```



In [ ]:
```
print("Explained Variance Ratio:")
print(f"PC1: {pca_harr.explained_variance_ratio_[0]:.4f}")
print(f"PC2: {pca_harr.explained_variance_ratio_[1]:.4f}")
print(f"PC3: {pca_harr.explained_variance_ratio_[2]:.4f}")
print(f"\nTotal Variance Explained:")
print(
    f"PC1 + PC2: {pca_harr.explained_variance_ratio_[0] + pca_harr.explained_variance_ratio_[1]:.4f}"
)
print(
    f"PC2 + PC3: {pca_harr.explained_variance_ratio_[1] + pca_harr.explained_variance_ratio_[2]:.4f}"
)
```

```
Explained Variance Ratio:
PC1: 0.5078
PC2: 0.0658
PC3: 0.0281

Total Variance Explained:
PC1 + PC2: 0.5736
PC2 + PC3: 0.0939
```

In [ ]:
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 8))

scatter1 = ax1.scatter(pca_out_harr[:, 0], pca_out_harr[:, 1], c=labels_harr)
ax1.set_xlabel("First Principal Component")
ax1.set_ylabel("Second Principal Component")
ax1.set_title("PC1 vs PC2")
ax1.grid(True)

legend1 = ax1.legend(*scatter1.legend_elements(), title="Labels", loc="upper right")
ax1.add_artist(legend1)

scatter2 = ax2.scatter(pca_out_harr[:, 1], pca_out_harr[:, 2], c=labels_harr)
ax2.set_xlabel("Second Principal Component")
ax2.set_ylabel("Third Principal Component")
ax2.set_title("PC2 vs PC3")
ax2.grid(True)

legend2 = ax2.legend(*scatter2.legend_elements(), title="Labels", loc="upper right")
ax2.add_artist(legend2)

plt.tight_layout()
plt.show()
```



From the results, it can be inferred that Gaussian models may not always be the advisable choice for solving classification models where we do not possess domain knowledge of the problem. The Perror is significantly higher in case of the Wine Quality data showing the model not being appropriate for the set. There is a possibility that the class posteriors were highly influenced by the priors selected based on the sample count, resulting in misleading results on new samples. For the second dataset, a Gaussian classifier tends to yield better results as compared to the Wine Quality Classification problem. PCA helps as an effective tool to identify meaningful information from the abundant feature set. But the Principal Components for the Human Activity Dataset have more data than that of the Wine Quality Dataset, this is visible in the Spree Plot of the explained variance ratios.

References

1. Lecture Notes

2. Code shared as part of the Google Drive

3. Pattern Classification by Duda, Hart and Stork

4. StatsQuest - https://statquest.org/