Data 558: Statistical Machine Learning

Spring 2023

Homework - 2

Arjun Sharma

Conceptual Questions:

Problem 1 (18 points): Suppose we are in a binary classification setting (i.e. $y \in \{0, 1\}$) with a single feature X. We perform discriminate analysis where the distribution of $X|y = k$, with density $p(X = x|y = k)$ is modeled to be Gaussian where both the mean and variance can change with k; this corresponds to Quadratic Discriminate Analysis (QDA). Derive the discriminate functions in this case. What is the maximum number of transition points in this case? Draw a figure of what the decision boundary can look like. How is it different from LDA?

HW- Problem 1

Given :→ Binary Classification setting. ($y \in \{0,1\}$)

→ A single feature X.

→ Distribution of $X/y = k$, with the density $P(X=x/y=k)$ is modeled to be Gaussian where both mean and variance can change with K.

To find : $P(Y=k/X=x)$

By Bayes' Theorem:

$$P(Y=k/X=x) = \frac{P(X=x/Y=k).P(Y=k)}{P(X=x)}$$

$$f'(x) = \underset{k}{\text{argmax}} \frac{P(X=x/Y=k)\,P(Y=k)}{P(X=x)}$$

$$\Rightarrow f'(x) = \underset{k}{\text{argmax}}\ P(X=x/Y=k).P(Y=k)$$

$$\Rightarrow f'(x) = \underset{k}{\text{argmax}} \frac{1}{\sqrt{2\pi}\sigma} e^{-1/2\left(\frac{x-\mu}{\sigma}\right)^2}.\pi_k$$ [Density of X is Gaussian]

$$\log(f'(x)) = \underset{k}{\text{argmax}} \log\left(\frac{1}{\sqrt{2\pi}\sigma} e^{-1/2\left(\frac{x-\mu}{\sigma}\right)^2}\right) + \log(\pi_k)$$

$$\Rightarrow \log(f'(x)) = \underset{k}{\text{argmax}} \log\frac{1}{\sqrt{2\pi}} - \frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2 + \log(\pi_k)$$

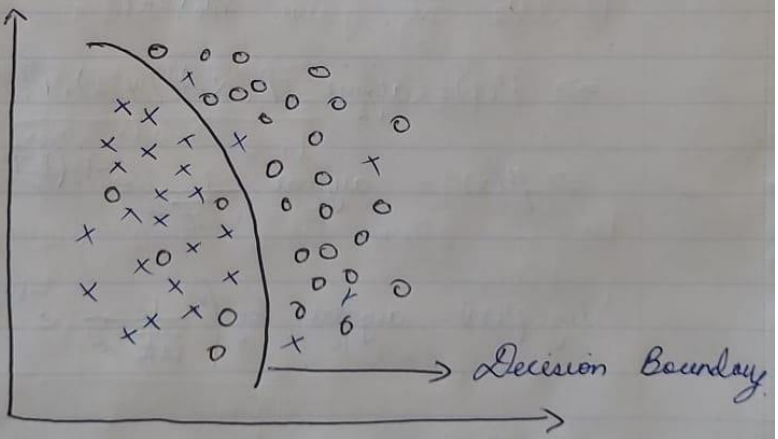
$$= \underset{k}{\arg\max} \log \pi_k - \frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2$$

∴ The discriminate function : $\underset{k}{\arg\max} \boxed{\log \pi_k - \frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$

→ The function corresponds to that of a quadratic case since the x term is of degree 2.

And, We can have a maximum of 2 transition points, because Y has only 2 classes and because the discriminant function is quadratic.

The decision boundary can be visualized like this :



where O and X are 2 classes

Problem 2 (12 points): Suppose we collect data for a group of students in a statistics class with variables X_1 = hours studied, X_2 = undergrad GPA, and Y = receive an A (binary). We fit a logistic regression and produce estimated coefficients, $\widehat{\beta}_0 = -2$, $\widehat{\beta}_1 = 0.1$ and $\widehat{\beta}_2 = 0.6$.

(a) Estimate the probability that a student who studies for 50 hours ($X_1 = 50$) and has an undergrad GPA of 3.7 ($X_2 = 3.7$) gets an A in the class ($Y = 1$).

(b) How many hours would a student with GPA 3.0 need to study to have a 50% chance of getting an A in the class?

HW - Problem 2

Given: Logistic Regression model with coefficients:

$\hat{\beta}_0 = -2$, $\hat{\beta}_1 = 0.1$, $\hat{\beta}_2 = 0.6$.

To find:

(1) Probability that student gets 'A' grade for given values:
$X_1 = 50$
$X_2 = 3.7$

(2) Number of hours needed for a student with 3.0 GPA to have 50% Chance of getting A grade.

Solution:

(1) We are given the model:

$$\hat{P}(X) = \frac{e^{-0.2 + 0.1X_1 + 0.6X_2}}{1 + e^{-0.2 + 0.1X_1 + 0.6X_2}}$$

For the given set of values, we have the following prediction:

$$\hat{P}(X) = \frac{e^{-0.2 + 5 + 2.22}}{1 + e^{-0.2 + 5 + 2.22}} = \frac{184.934}{185.934} =$$

$$\Rightarrow \boxed{\hat{P}(X) = 0.9946}$$

2) $$\hat{P}(X) = \frac{e^{-2 + 0.1X_1 + 0.6(3.0)}}{1 + e^{-2 + 0.1X_1 + 0.6(3.0)}} = 0.5.$$

$$\left(1 + e^{-0.2 + 0.1X_1}\right) \times 0.5 = e^{-0.2 + 0.1X_1}$$

$$1 + e^{-0.2 + 0.1X_1} = 2e^{-0.2 + 0.1X_1}$$

$$e^{-0.2 + 0.1X_1} = 1$$

$$\Rightarrow \quad 0.1X_1 = 0.2$$

$$\boxed{X_1 = 2.}$$

Problem 3 (12 points): As we talked about in class, imbalanced data is a challenge with classification. Often, we would like to design a classifier that does well in a minority class, but we do not see many labeled data points from that minority class. Suppose we are in a binary classification setting with a large number of negative examples and only a small number of positive examples. We will examine two approaches to deal with data imbalance.

(a) A common data processing approach is to oversample or undersample a class. Random oversampling duplicates examples from the minority class in the training dataset; Random undersampling deletes examples from the majority class. What effects does oversampling have on the false positive and false negative rates? What about undersampling?

(b) An equivalent approach in logistic regression is to weigh the samples from the minority class differently than the samples from the majority class in the maximum likelihood estimator. As an example, let $i = 1, 2, \ldots, n_1$ be samples from the minority class and $i = n_1 + 1, \ldots, n$ be samples from the majority class. Then, we modify the logistic objective function to:

$$\left[\omega \sum_{i=1}^{n_1} x^{(i)}\beta y^{(i)} - \log(1 + \exp(\beta x^{(i)}))\right] + \sum_{i=n_1+1}^{n} x^{(i)}\beta y^{(i)} - \log(1 + \exp(\beta x^{(i)})).$$

Setting $\omega = 1$ recovers the standard logistic regression. Does setting $\omega > 1$ lead to a larger or smaller false negative rate than the standard logistic regression? What about $\omega < 1$?

(a)

Oversampling the minority class leads to duplication of the positive class labelled datapoints. Now, because the frequency of positive class datapoints increases, the model becomes likely to predict the positive class more often. This can lead to a higher false positive rate. Conversely, the false negative rate decreases since the model is relatively more likely to predict the positive class compared to a model which has been trained on data without any sampling.

Under-sampling the majority class leads to some information loss about the relationship between the predictors and outcome. In addition, due to the relatively balanced composition of the classes in the data, the model would have a higher false negative rate. The increase in false negatives would also imply a decreased false positive rate.

(b)

For ω > 1, the added weights to the minority class increases the model's likelihood to predict in favor of the minority class. Consequently, the false negative rate would decrease since the model's frequency of prediction to be negative decreases.

For ω < 1, the reduced weights to the minority class increases the model's likelihood to predict in favor of the majority class. Thus, the false negative rate would increase since the model is more likely to predict an outcome as negative.

Applied Questions:

Problem 1 (18 points): In this question you will implement your own logistic regression. Let X_1, X_2 be two standard normal variables, and let $Y \in \{0, 1\}$ be a binary variable with

$$\mathbb{P}(Y = 1|X_1 = x_1, X_2 = x_2) = \frac{\exp\left(\beta_1^\star x_1 + \beta_2^\star x_2\right)}{1 + \exp\left(\beta_1^\star x_1 + \beta_2^\star x_2\right)}.$$

We let $\beta_1^\star = \beta_2^\star = 0.2$. Generate $n = 400$ samples of (X_1, X_2) and of Y according to the conditional distribution. We will denote positive examples as ones where $Y = 1$ and negative examples as ones where $Y = 0$.

(a) Write your own gradient descent algorithm to estimate $\beta_1^\star, \beta_2^\star$ from the data. Setting $\beta_1 = \beta_2 = 0$ as your initialization, perform gradient descent until convergence. Set your learning rate at 0.1 (feel free to adjust the learning rate to see a faster convergence). Track your function value at every step of your algorithm and make sure that it is increasing. Compare your estimate to the one produced by either `R` or `Python`.

(b) Once you have obtained estimates for $\widehat{\beta}_1$ and $\widehat{\beta}_2$, use the prediction rule $\widehat{y}^{(i)} = 1$ if

$$\frac{\exp\left(\widehat{\beta} x_1^{(i)} + \widehat{\beta}_2 x_2^{(i)}\right)}{1 + \exp\left(\widehat{\beta}_1 x_1^{(i)} + \widehat{\beta}_2 x_2^{(i)}\right)} \geq \delta,$$

and $\widehat{y}^{(i)} = 0$, otherwise. Produce an ROC curve by varying δ.

Answer:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
```

```
random.seed(1)
```

```
# Generating the samples
X1_1 = np.random.normal(1,1,200) # mean = 1, var = 1, n = 200
X2_1 = np.random.normal(0,1,200) # mean = 0, var = 1, n = 200

X1_2 = np.random.normal(-1,1,200) # mean = -1, var = 1, n = 200
X2_2 = np.random.normal(0,1,200) # mean = 0, var = 1, n = 200

X1 = np.concatenate((X1_1, X1_2))
X2 = np.concatenate((X2_1, X2_2))
X = np.vstack((X1, X2)).T
```

```
# Generate values for Y based on above random values
def generate_y(X1, X2):
    beta_1 = 5
    beta_2 = -1

    probs = np.exp(beta_1*X1 + beta_2*X2)/(1+np.exp(beta_1*X1 + beta_2*X2))

    y = np.random.binomial(n = 1, p = probs)
    return(y)
```

```
Y = generate_y(X1, X2)
```

```
# Initialize the coefficients and learning rate
beta1 = 0
beta2 = 0
alpha = 0.1 # Learning Rate
tolerance = 1e-6
log_likelihood = np.array([])
```

```
def gradient_descent(X1, X2, Y, beta1, beta2):
    coefs_change = 1
    N = 0 # number of iterations
    max_iters = 1000000
    while coefs_change > tolerance:
        exp_term = np.exp(beta1*X1 + beta2*X2)
        y_prob = exp_term / (1 + exp_term)
        residual = Y - y_prob

        beta1_new = beta1 + alpha*np.mean(residual*X1)
        beta2_new = beta2 + alpha*np.mean(residual*X2)

        coefs_change = np.sqrt((beta1_new - beta1)**2 + (beta2_new - beta2)**2)

        beta1 = beta1_new
        beta2 = beta2_new

        log_likelihood = np.sum(Y*np.log(y_prob) + (1-Y)*np.log(1-y_prob))

        if N > max_iters:
            break
        N += 1

        if N%100 == 0:
            print(f'For iteration no.: {N}, the Log-Likelihood is: = {log_likelihood}')
            print(f'Beta 1: {beta1}, and Beta 2: {beta2}')
    return beta1, beta2
```

```
beta1, beta2 =  gradient_descent(X1, X2, Y, beta1, beta2)
```

```
Beta 1: 5.971431610260454S, and Beta 2: -0.8808750658786433

For iteration no.: 17200, the Log-Likelihood is: = -48.96027731744708
Beta 1: 5.971568345560693, and Beta 2: -0.8808993914705516
For iteration no.: 17300, the Log-Likelihood is: = -48.96027659845217
Beta 1: 5.971700301236775, and Beta 2: -0.8809228667102846
For iteration no.: 17400, the Log-Likelihood is: = -48.960275928841114
Beta 1: 5.97182764465487, and Beta 2: -0.8809455213772805
For iteration no.: 17500, the Log-Likelihood is: = -48.96027530521928
Beta 1: 5.971950537300436, and Beta 2: -0.8809673842044166
For iteration no.: 17600, the Log-Likelihood is: = -48.960274724425574
Beta 1: 5.9720691349862305, and Beta 2: -0.8809884829150415
For iteration no.: 17700, the Log-Likelihood is: = -48.960274183516475
Beta 1: 5.972183588052848, and Beta 2: -0.8810088442586823
For iteration no.: 17800, the Log-Likelihood is: = -48.96027367975101
Beta 1: 5.972294041562133, and Beta 2: -0.8810284940454678
For iteration no.: 17900, the Log-Likelihood is: = -48.96027321057677
Beta 1: 5.972400635483614, and Beta 2: -0.8810474571793242
For iteration no.: 18000, the Log-Likelihood is: = -48.96027277361688
Beta 1: 5.972503504874331, and Beta 2: -0.8810657576899836
```

```
# Scikit-Learn Implementation:
from sklearn.linear_model import LogisticRegression

X = np.vstack((X1, X2)).T

clf = LogisticRegression(random_state=1, solver='lbfgs').fit(X, Y)

# estimates
print("Beta1:", clf.coef_[0][0])
print("Beta2:", clf.coef_[0][1])
```

```
Beta1: 4.235778539384465
Beta2: -0.5552441738379492
```

The coefficients of my model are:

Beta1 = 5.972503

Beta2 = -0.881065

While the coefficients of the model built on scikit-learn are:

Beta1: 4.235778

Beta2: -0.5552441

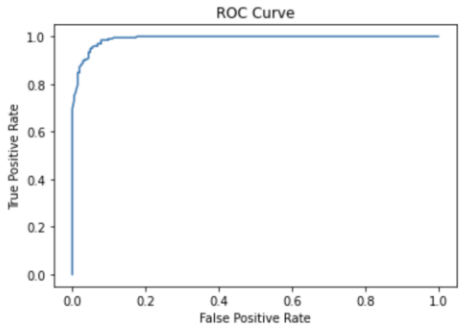
The coefficients of the models are similar, although they are not the same.

```
delta = np.linspace(0, 1, num = 100) # Varying delta from 0 to 1
```

```
def predicted_Y(X1, X2, beta1, beta2, delta):
    y_hat = np.where((np.exp(beta1*X1 + beta2*X2)/(1+np.exp(beta1*X1 + beta2*X2))) >= delta, 1, 0)
    return y_hat
```

```
def performance_metrics(y, y_hat):
    TP = 0 # True Positive
    FP = 0 # False positives
    TN = 0 # True Negatives
    FN = 0 # False Negatives
    for i in range(len(y_hat)):
        if y[i]==y_hat[i] and y[i]==1:
            TP += 1
        elif y_hat[i]==1 and y[i]!=y_hat[i]:
            FP += 1
        elif y[i]==y_hat[i] and y[i]==0:
            TN += 1
        else:
            FN += 1

    TPR = TP/(TP+FN)
    FPR = FP/(FP+TN)
    TNR = TN/(TN+FP)
    FNR = FN/(TP+FN)
    return TPR, FPR, TNR, FNR

TPR_values = []
FPR_values = []
```

```
for i in delta:
    y_hat = predicted_Y(X1, X2, beta1, beta2, i)
    TPR, FPR, TNR, FNR = performance_metrics(Y, y_hat)
    TPR_values.append(TPR)
    FPR_values.append(FPR)


# Specific value of delta
delta_half = 0.5
y_preds = predicted_Y(X1, X2, beta1, beta2, delta_half)
TPR_delta_half, FPR_delta_half, TNR_delta_half, FNR_delta_half = performance_metrics(Y, y_preds)

plt.plot(FPR_values, TPR_values)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()

print("False positive rate:",FPR_delta_half)
print("False negative rate:",FNR_delta_half)
```



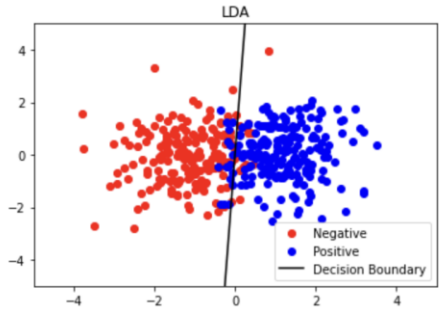

```
False positive rate: 0.059113300492610835
False negative rate: 0.04060913705583756
```

Problem 2 (20 points): In this question, you will implement your own linear discriminant analysis algorithm. Take the data that you generated in the previous question. Then, train a linear discriminant analysis model where the covariances across the two categories are identical but the means vary. Plot the data points in a scatter plot and the decision boundary you obtain using LDA. What are the false positive and false negative error rates? Compare these numbers with the logistic regression approach from the previous question with $\delta = 0.5$. [Hint: Use the sample means of the data in each class, and the sample covariance matrix of the entire data, to specify the multivariate Gaussian distributions in each category].

Note: Y is the data that was generated in Problem 1.

```
mu_0 = np.array([np.mean(X1[Y == 0]), np.mean(X2[Y == 0])])
mu_1 = np.array([np.mean(X1[Y == 1]), np.mean(X2[Y == 1])])

cov_0 = np.cov(X1[Y == 0], X2[Y == 0])
cov_1 = np.cov(X1[Y == 1], X2[Y == 1])

pooled_cov = (cov_0*(sum(Y == 0)-1) + cov_1*(sum(Y == 1)-1)) / (n-2)

w = np.linalg.solve(pooled_cov, mu_1 - mu_0)
w0 = -0.5*np.dot(w, (mu_0 + mu_1))

plt.scatter(X1[Y==0], X2[Y==0], c='red', marker='o', label='Negative')
plt.scatter(X1[Y==1], X2[Y==1], c='blue', marker='o', label='Positive')

x = np.linspace(-5, 5, 100)
y = -(w[0]/w[1])*x - w0/w[1]

plt.plot(x, y, c='black', label='Decision Boundary')
plt.legend(loc='lower right')
plt.title('LDA')
plt.xlim(-5,5)
plt.ylim(-5,5)
plt.show()
```

```
z = np.dot(w, np.vstack((X1, X2))) + w0
y_pred = (z > 0).astype(int)
tp = np.sum((Y == 1) & (y_pred == 1))
tn = np.sum((Y == 0) & (y_pred == 0))
fp = np.sum((Y == 0) & (y_pred == 1))
fn = np.sum((Y == 1) & (y_pred == 0))
FPR = fp/(fp+tn)
FNR = fn/(tp+fn)

print("False positive rate:", FPR)
print("False negative rate:", FNR)
```




```
False positive rate: 0.06842105263157895
False negative rate: 0.0761904761904762
```

Above are the False Positive and False Negative Rates obtained using LDA.

Below are the False Positive and False Negative Rates obtained using Logistic Regression:

```
False positive rate: 0.059113300492610835
False negative rate: 0.04060913705583756
```

The FPR and FNR are quite similar for both Logistic Regression and LDA, however it is observed that the Logistic Regression performs marginally better.

Problem 3 (20 points): Using the Pima Indians Diabetes Dataset in file `diabetes.csv`, fit classification models in order to predict the outcome. Explore logistic regression, LDA, naïve Bayes, and KNN models using various subsets of the predictors. Describe your findings.

Pima Indians Diabetes Dataset Description: This dataset contains 768 samples of Pima Indian women, each with features such as glucose level, BMI, and blood pressure, along with an outcome variable indicating whether the person has diabetes (1) or not (0).

Models were developed with the following sets of predictors:

1. 'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', and 'Age'

2. 'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', and 'Insulin'

3. 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', and 'Age'

Overall, the Logistic Regression Model performs best for this problem.

```
import pandas as pd
import numpy as np
import random
```

```
random.seed(1)
```

```
import warnings
warnings.filterwarnings("ignore")
```

```
df= pd.read_csv('C://Users//arjun//Desktop//Quarter_3//ML//Homeworks//HW2//diabetes.csv')
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

```
df.describe()
```

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

```
df.shape
```

```
(768, 9)
```

```
df['Outcome'].unique()
```

```
array([1, 0], dtype=int64)
```

```
df.columns
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

```
X=df.loc[:,df.columns!='Outcome'].values
y=df.loc[:,'Outcome'].values
```

```
X1 = df[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin']].values
```

```
X2 = df[['SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age']].values
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
#Naive-Bayes
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
from sklearn.metrics import accuracy_score
print( 'Accuracy Score',accuracy_score(y_test, y_pred)*100)
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
Accuracy Score 79.22077922077922
              precision    recall  f1-score   support

           0       0.87      0.84      0.85       111
           1       0.62      0.67      0.64        43

    accuracy                           0.79       154
   macro avg       0.74      0.76      0.75       154
weighted avg       0.80      0.79      0.79       154
```

```
#Naive-Bayes
X1_train, X1_test, y_train, y_test = train_test_split(X1, y, test_size=0.2, random_state=0)
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X1_train, y_train)
y_pred = classifier.predict(X1_test)
from sklearn.metrics import accuracy_score
print( 'Accuracy Score',accuracy_score(y_test, y_pred)*100)
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
Accuracy Score 76.62337662337663
              precision    recall  f1-score   support

           0       0.86      0.81      0.84       113
           1       0.55      0.63      0.59        41

    accuracy                           0.77       154
   macro avg       0.71      0.72      0.71       154
weighted avg       0.78      0.77      0.77       154
```

```
#Naive-Bayes
X2_train, X2_test, y_train, y_test = train_test_split(X2, y, test_size=0.2, random_state=0)
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X2_train, y_train)
y_pred = classifier.predict(X2_test)
from sklearn.metrics import accuracy_score
print( 'Accuracy Score',accuracy_score(y_test, y_pred)*100)
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
Accuracy Score 72.07792207792207
              precision    recall  f1-score   support

           0       0.87      0.76      0.81       122
           1       0.38      0.56      0.46        32

    accuracy                           0.72       154
   macro avg       0.63      0.66      0.63       154
weighted avg       0.77      0.72      0.74       154
```

```
# Linear Discriminant Analysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda_classifier = LDA()
lda_classifier.fit(X_train, y_train)
y_pred = lda_classifier.predict(X_test)
from sklearn.metrics import accuracy_score
print( 'Accuracy Score',accuracy_score(y_test, y_pred)*100)
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
Accuracy Score 82.46753246753246
              precision    recall  f1-score   support

           0       0.92      0.84      0.88       116
           1       0.62      0.76      0.68        38

    accuracy                           0.82       154
   macro avg       0.77      0.80      0.78       154
weighted avg       0.84      0.82      0.83       154
```

```
# Linear Discriminant Analysis
X1_train, X1_test, y_train, y_test = train_test_split(X1, y, test_size=0.2, random_state=0)
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda_classifier = LDA()
lda_classifier.fit(X1_train, y_train)
y_pred = lda_classifier.predict(X1_test)
from sklearn.metrics import accuracy_score
print( 'Accuracy Score',accuracy_score(y_test, y_pred)*100)
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
Accuracy Score 76.62337662337663
              precision    recall  f1-score   support

           0       0.86      0.81      0.84       113
           1       0.55      0.63      0.59        41

    accuracy                           0.77       154
   macro avg       0.71      0.72      0.71       154
weighted avg       0.78      0.77      0.77       154
```

```
# Linear Discriminant Analysis
X2_train, X2_test, y_train, y_test = train_test_split(X2, y, test_size=0.2, random_state=0)
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda_classifier = LDA()
lda_classifier.fit(X2_train, y_train)
y_pred = lda_classifier.predict(X2_test)
from sklearn.metrics import accuracy_score
print( 'Accuracy Score',accuracy_score(y_test, y_pred)*100)
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
Accuracy Score 72.07792207792207
              precision    recall  f1-score   support

           0       0.92      0.74      0.82       132
           1       0.28      0.59      0.38        22

    accuracy                           0.72       154
   macro avg       0.60      0.67      0.60       154
weighted avg       0.82      0.72      0.76       154
```

```
#Logistic Regression
from sklearn.linear_model import LogisticRegression
clf= LogisticRegression(max_iter=150)
clf.fit(X_train, y_train)
y_pred= clf.predict(X_test)
from sklearn.metrics import accuracy_score
print( 'Accuracy Score',accuracy_score(y_test, y_pred)*100)
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
Accuracy Score 82.46753246753246
              precision    recall  f1-score   support

           0       0.92      0.84      0.88       116
           1       0.62      0.76      0.68        38

    accuracy                           0.82       154
   macro avg       0.77      0.80      0.78       154
weighted avg       0.84      0.82      0.83       154
```

```
#Logistic Regression
X1_train, X1_test, y_train, y_test = train_test_split(X1, y, test_size=0.2, random_state=0)
from sklearn.linear_model import LogisticRegression
clf= LogisticRegression(max_iter=150)
clf.fit(X1_train, y_train)
y_pred= clf.predict(X1_test)
from sklearn.metrics import accuracy_score
print( 'Accuracy Score',accuracy_score(y_test, y_pred)*100)
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
Accuracy Score 76.62337662337663
              precision    recall  f1-score   support

           0       0.86      0.81      0.84       113
           1       0.55      0.63      0.59        41

    accuracy                           0.77       154
   macro avg       0.71      0.72      0.71       154
weighted avg       0.78      0.77      0.77       154
```

```
#Logistic Regression
X2_train, X2_test, y_train, y_test = train_test_split(X2, y, test_size=0.2, random_state=0)
from sklearn.linear_model import LogisticRegression
clf= LogisticRegression(max_iter=150)
clf.fit(X2_train, y_train)
y_pred= clf.predict(X2_test)
from sklearn.metrics import accuracy_score
print( 'Accuracy Score',accuracy_score(y_test, y_pred)*100)
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
Accuracy Score 72.72727272727273
              precision    recall  f1-score   support

           0       0.92      0.75      0.82       131
           1       0.30      0.61      0.40        23

    accuracy                           0.73       154
   macro avg       0.61      0.68      0.61       154
weighted avg       0.82      0.73      0.76       154
```

```
#KNN Classifier
from sklearn.neighbors import KNeighborsClassifier
for i in range (1,20):
    knn2 = KNeighborsClassifier(n_neighbors=i)
    knn2.fit(X_train, y_train)
    print("For k = %d accuracy is"%i,knn2.score(X_test,y_test))
```

```
For k = 1 accuracy is 0.6168831168831169
For k = 2 accuracy is 0.7142857142857143
For k = 3 accuracy is 0.7207792207792207
For k = 4 accuracy is 0.7727272727272727
For k = 5 accuracy is 0.7532467532467533
For k = 6 accuracy is 0.7792207792207793
For k = 7 accuracy is 0.7597402597402597
For k = 8 accuracy is 0.7792207792207793
For k = 9 accuracy is 0.7727272727272727
For k = 10 accuracy is 0.7922077922077922
For k = 11 accuracy is 0.7662337662337663
For k = 12 accuracy is 0.7857142857142857
For k = 13 accuracy is 0.7922077922077922
For k = 14 accuracy is 0.7922077922077922
For k = 15 accuracy is 0.7922077922077922
For k = 16 accuracy is 0.7922077922077922
For k = 17 accuracy is 0.7857142857142857
For k = 18 accuracy is 0.7857142857142857
For k = 19 accuracy is 0.7987012987012987
```

```
from sklearn.neighbors import KNeighborsClassifier
knn1 = KNeighborsClassifier(n_neighbors=19)
knn1.fit(X_train, y_train)
y_pred = knn1.predict(X_test)
print("Score is",knn1.score(X_test,y_test)*100)
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
Score is 79.87012987012987
              precision    recall  f1-score   support

           0       0.91      0.82      0.86       118
           1       0.55      0.72      0.63        36

    accuracy                           0.80       154
   macro avg       0.73      0.77      0.74       154
weighted avg       0.82      0.80      0.81       154
```

```
X1_train, X1_test, y_train, y_test = train_test_split(X1, y, test_size=0.2, random_state=0)
from sklearn.neighbors import KNeighborsClassifier
knn1 = KNeighborsClassifier(n_neighbors=19)
knn1.fit(X1_train, y_train)
y_pred = knn1.predict(X1_test)
print("Score is",knn1.score(X1_test,y_test)*100)
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
Score is 75.97402597402598
              precision    recall  f1-score   support

           0       0.87      0.80      0.83       116
           1       0.51      0.63      0.56        38

    accuracy                           0.76       154
   macro avg       0.69      0.72      0.70       154
weighted avg       0.78      0.76      0.77       154
```

```
X2_train, X2_test, y_train, y_test = train_test_split(X2, y, test_size=0.2, random_state=0)
from sklearn.neighbors import KNeighborsClassifier
knn1 = KNeighborsClassifier(n_neighbors=19)
knn1.fit(X2_train, y_train)
y_pred = knn1.predict(X2_test)
print("Score is",knn1.score(X2_test,y_test)*100)
from sklearn.metrics import classification_report
print(classification_report(y_pred, y_test))
```

```
Score is 71.42857142857143
              precision    recall  f1-score   support

           0       0.82      0.78      0.80       113
           1       0.47      0.54      0.50        41

    accuracy                           0.71       154
   macro avg       0.65      0.66      0.65       154
weighted avg       0.73      0.71      0.72       154
```