PRML LAB 04 REPORT

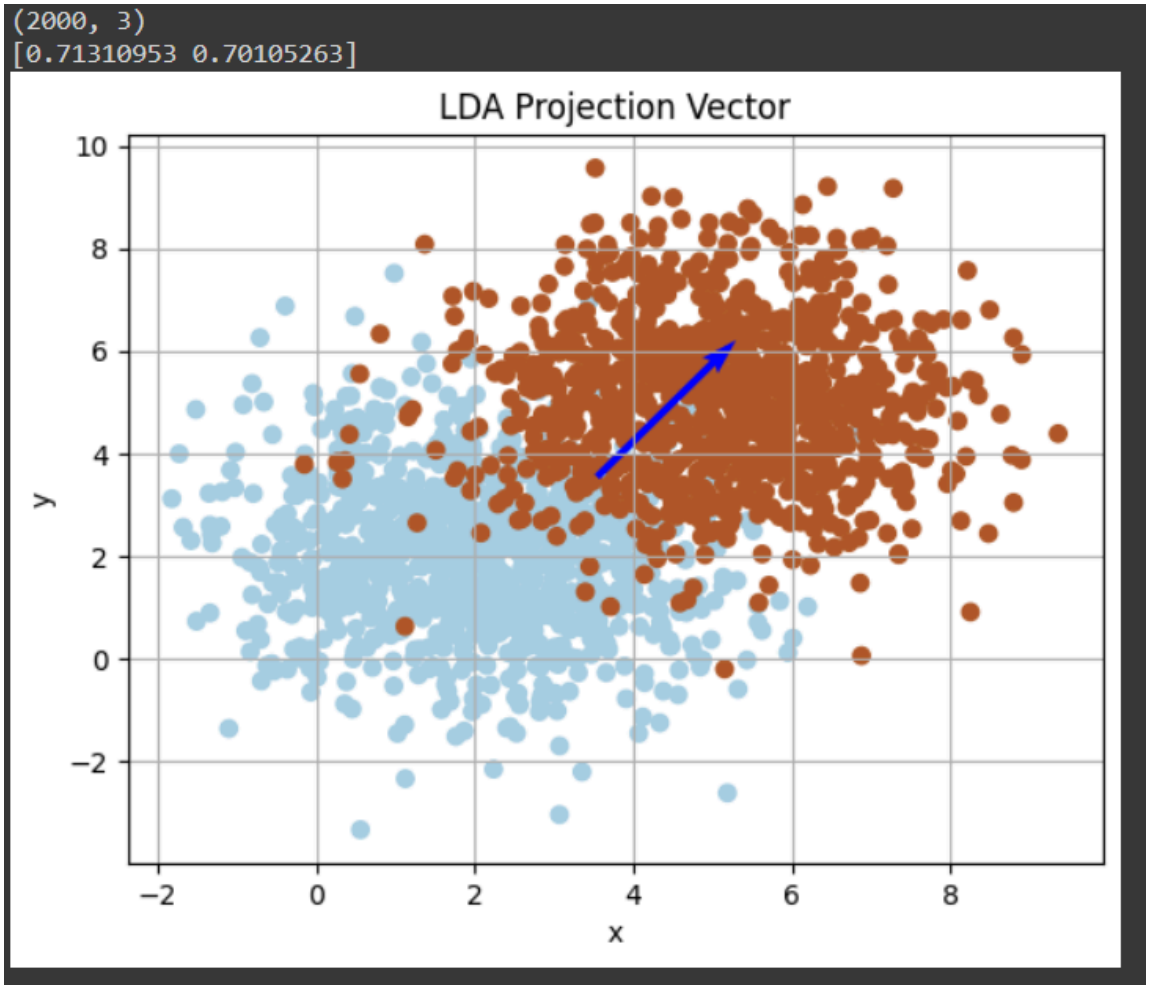
Anushka Dadhich
B22CS097
Link - **B22CS097_all.ipynb**

QUESTION 1, LDA

TASK 1:
Submitted B22CS097_myLDA.py

TASK 2:
Using the code as in the submitted file for GetLDAProjectionVector(X), plotting the LDA vector using the quiver plot of the matplotlib, which is used for plotting the vector points with the directions, and the scatter plot to plot the data points on the same plot.

Output:



TASK 3:
Using train_test_split() to split the original data into 80:20 split and then apply the KNeighborsClassifier(n_neighbors=1), similarly getting the projected data by using
w = GetLDAProjectionVector(X)
X_proj = np.dot(X_features, w.reshape(-1, 1))
And then again apply the KNeighborsClassifier(n_neighbors=1) on the projected data.
Computing the accuracies by using accuracy_score() function.

Output:

With random_state = 50

```
Accuracy of 1-NN classifier on original data: 0.9075
Accuracy of 1-NN classifier on projected data: 0.9
```

With random_state = 59

```
Accuracy of 1-NN classifier on original data: 0.88
Accuracy of 1-NN classifier on projected data: 0.905
```

There isn't any general trend as to which accuracy must be higher always, because it depends on various factors like the random_state through which testing data might be changing, due to which for some random data the accuracy on original data is higher while for some other state the accuracy on projected data is higher as per my trials on the dataset as shown above.
These might be due to:
LDA aims to find a projection that maximizes the separation between classes. If the classes in the dataset are well-separated in the feature space, LDA is likely to find an effective projection that further enhances this separation, leading to better classification performance.
LDA assumes that the data is normally distributed within each class and that the classes have equal covariance matrices.
Overall, in scenarios where the assumptions of LDA are met and the classes in the dataset are well-separated, the classifier trained on the projected data using LDA is expected to have a higher accuracy compared to the classifier trained on the original data.

QUESTION 2, (Naive Bayes)

TASK 0:
Reading the file using pd.read_csv and then using train_test_split() to split the data.

Output:

```
X_train shape: (12, 4)
X_test shape: (2, 4)
y_train shape: (12,)
y_test shape: (2,)
```

TASK 1:
Iterating over the rows and then keeping a count of the ‘yes’ and ‘no’ to find prior probabilities on training data.

Output:

```
P(Play=yes) on training data: 0.5833333333333334
P(Play=no) on training data: 0.4166666666666667
```

TASK 2:
Created dict for likelihood probabilities of ‘yes’ and ‘no’, iterating over the data features and then keeping a count of each unique occurrences by
count = data[(data[feature] == value) & (data['Play'] == 'yes')].shape[0]
And then divide by total ‘yes’ to get the corresponding probabilities.

Output:

```
Likelihood probabilities for 'Play = yes' on the training dataset:
P(Outlook = Sunny | Play = yes): 0.43
P(Outlook = Rainy | Play = yes): 0.29
P(Outlook = Overcast | Play = yes): 0.29
P(Temp = Cool | Play = yes): 0.43
P(Temp = Hot | Play = yes): 0.14
P(Temp = Mild | Play = yes): 0.43
P(Humidity = Normal | Play = yes): 0.71
P(Humidity = High | Play = yes): 0.29
P(Windy = t | Play = yes): 0.29
P(Windy = f | Play = yes): 0.71

Likelihood probabilities for 'Play = no' on the training dataset:
P(Outlook = Sunny | Play = no): 0.40
P(Outlook = Rainy | Play = no): 0.60
P(Outlook = Overcast | Play = no): 0.00
P(Temp = Cool | Play = no): 0.20
P(Temp = Hot | Play = no): 0.40
P(Temp = Mild | Play = no): 0.40
P(Humidity = Normal | Play = no): 0.20
P(Humidity = High | Play = no): 0.80
P(Windy = t | Play = no): 0.60
P(Windy = f | Play = no): 0.40
```

TASK 3:
Iterating over the test samples and multiplying the likelihood probabilities with the prior probability and dividing by the evidence for both ‘yes’ and ‘no’ labels.
Then printing the probabilities.

Output:

```
Zero likelihood probability found for feature 'Outlook' and value 'Overcast'
Zero likelihood probability found for feature 'Outlook' and value 'Overcast'
Posterior Probabilities for Testing Split:
           Play = yes  Play = no
Sample ID
11               1.0        0.0
12               1.0        0.0
```

TASK 4:
Iterating over the posterior probabilities obtained above and if that of ‘yes’ > ‘no’ then predicting the label as ‘yes’, similar for ‘no’.

Output:

```
Predictions:
           Prediction
Sample ID
11                yes
12                yes
Accuracy: 1.0
```

TASK 5:
Making changes as per -

$$P(w'|positive) = \frac{\text{number of reviews with w' and y = positive } + \ \alpha}{\text{N } + \ \alpha * K}$$

Here,
alpha represents the smoothing parameter,
K represents the number of dimensions (features) in the data, and
N represents the number of reviews with y=positive

In the previous above codes without Laplace smoothing, changes are made by adding an extra alpha (=1) i.e.
count = data[(data[feature] == value) & (data['Play'] == 'yes')].shape[0] + alpha
And (alpha)*(K) into the total counts of labels.
Rest code is the same.

Output:

```
Laplace smoothed likelihood probabilities for 'Play = yes':
P(Outlook = Sunny | Play = yes): 0.36
P(Outlook = Rainy | Play = yes): 0.27
P(Outlook = Overcast | Play = yes): 0.27
P(Temp = Cool | Play = yes): 0.36
P(Temp = Hot | Play = yes): 0.18
P(Temp = Mild | Play = yes): 0.36
P(Humidity = Normal | Play = yes): 0.55
P(Humidity = High | Play = yes): 0.27
P(Windy = t | Play = yes): 0.27
P(Windy = f | Play = yes): 0.55

Laplace smoothed likelihood probabilities for 'Play = no':
P(Outlook = Sunny | Play = no): 0.33
P(Outlook = Rainy | Play = no): 0.44
P(Outlook = Overcast | Play = no): 0.11
P(Temp = Cool | Play = no): 0.22
P(Temp = Hot | Play = no): 0.33
P(Temp = Mild | Play = no): 0.33
P(Humidity = Normal | Play = no): 0.22
P(Humidity = High | Play = no): 0.56
P(Windy = t | Play = no): 0.44
P(Windy = f | Play = no): 0.33

Posterior Probabilities with Laplace Smoothing for Testing Split:
   Play = yes  Play = no
0    0.530357   0.469643
1    0.882746   0.117254

Predictions with Laplace Smoothing:
0    yes
1    yes
dtype: object
Accuracy with Laplace Smoothing: 1.0
```

Generally Laplace smoothed probabilities are more preferred and accurate because:

- When calculating Laplace smoothed posterior probabilities, the likelihood probabilities are adjusted using the Laplace smoothed counts, leading to more robust estimates, especially for feature-value pairs that are not present or are rare in the training data.
- Laplace smoothing helps address the issue of overfitting that can occur when estimating likelihood probabilities from limited data, by providing a more balanced estimation that is less sensitive to rare events.

Predictions by both of them are correct on this random test dataset.