PA4: LDA and Naive Bayes Classifier

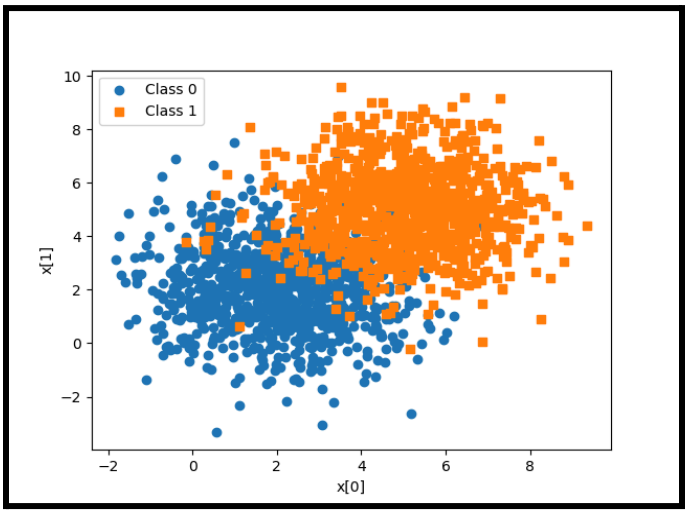
Colab Link: PA4_B22CS090.ipynb

Question 1:

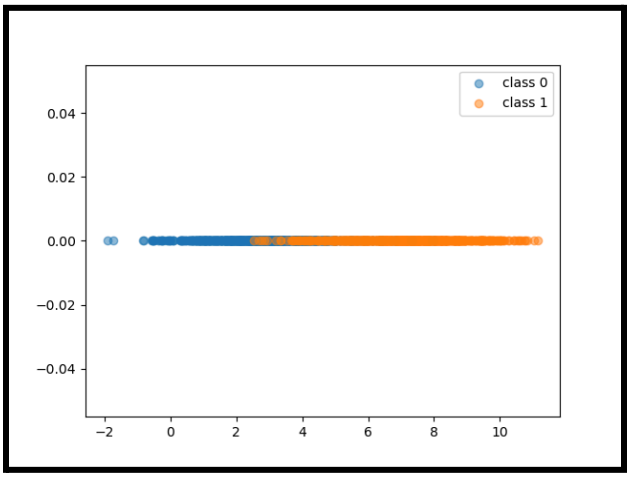
Note for Q1: B22CS090_myLDA.py follows the exact same template as given in google classroom. Just the switch cases have been put inside the main method. This is done because I am importing the myLDA.py file inside my jupyter notebook so that I can use the necessary functions without having to go through the switch cases, as importing a file does not run the main method.

Also, **80-20 splitting** is done randomly, keeping the label distribution stratified.

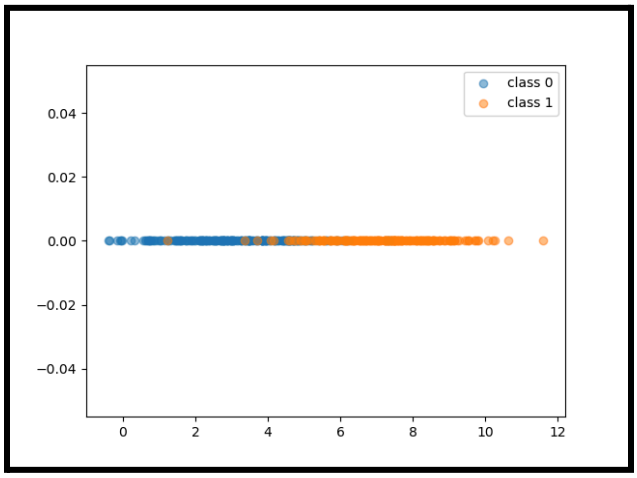
Some plots before visualising the projection vector:



Scatter plot of whole dataset



Training scatter plot after LDA transformation

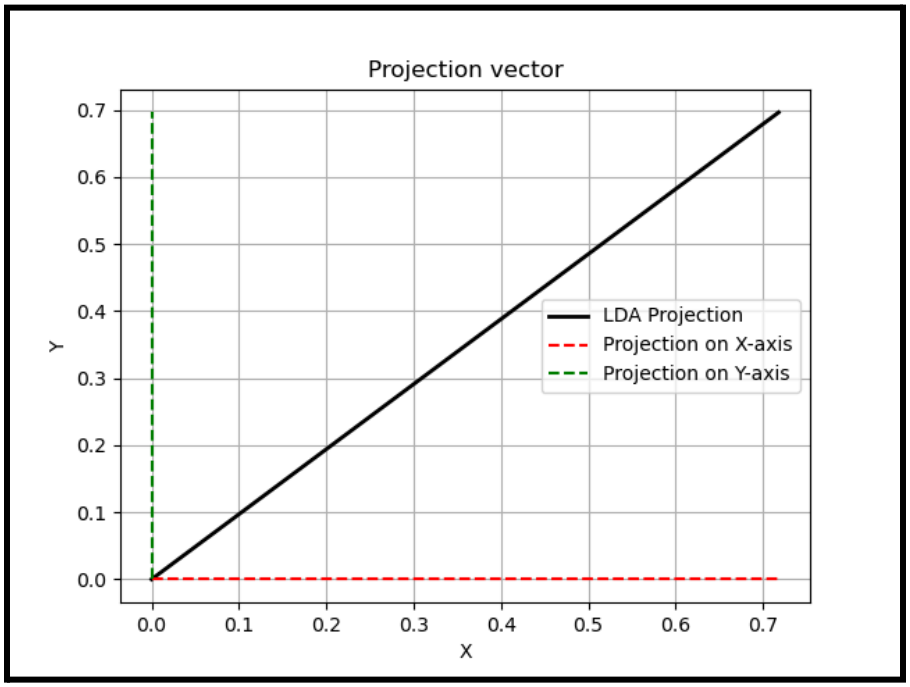


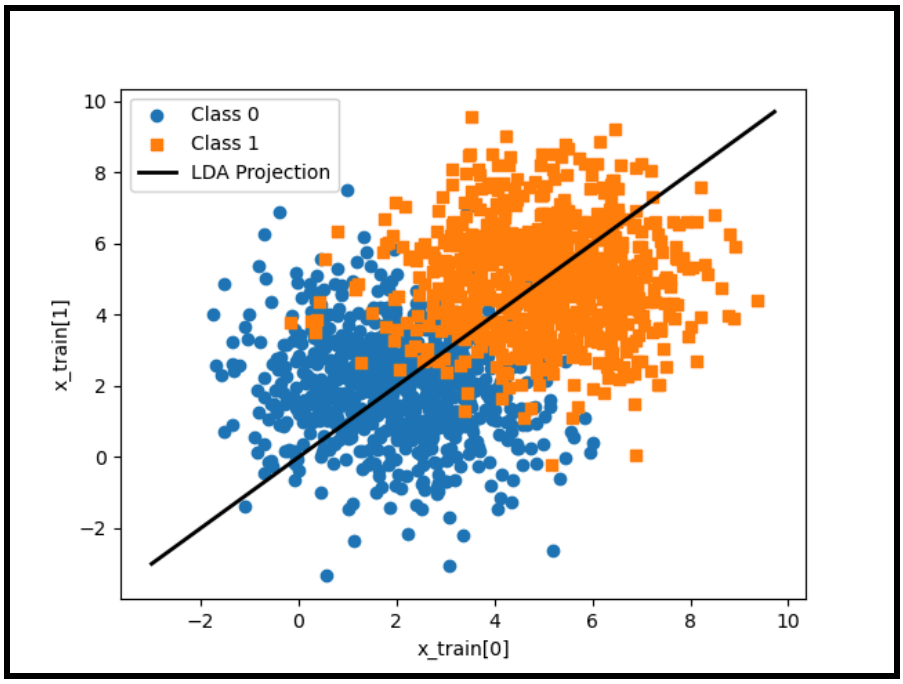
Testing scatter plot after LDA transformation

Observations using training data:

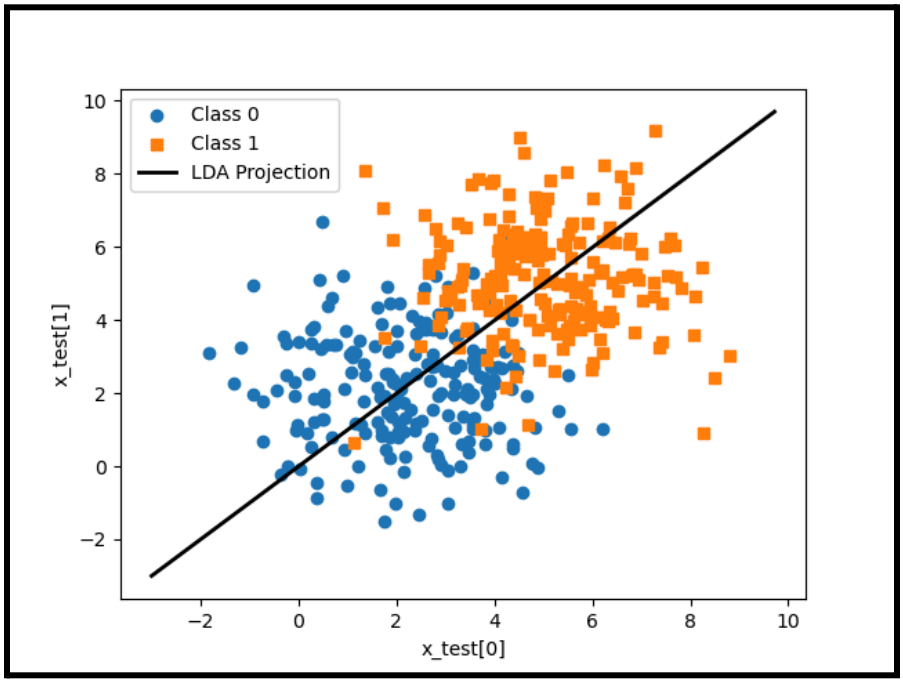
```
Mean difference:
[[-3.02497273 -3.02879727]]

Within class Scatter matrix:
[[3514.44729897 -195.81656663]
 [-195.81656663 3633.79324678]]

Between class Scatter matrix:
[[9.15046003 9.16202916]
 [9.16202916 9.17361291]]

Projection vector:
[[0.71781551]
 [0.69623336]]
```

Task 2:





Projection vector with training dataset



Projection vector with testing dataset

The projection vector is the line (in this problem, the vector is 2D, so it is a line) on which the data points are projected to get the LDA transformation. It is quite obvious that projecting the data points on this particular line will result in substantial

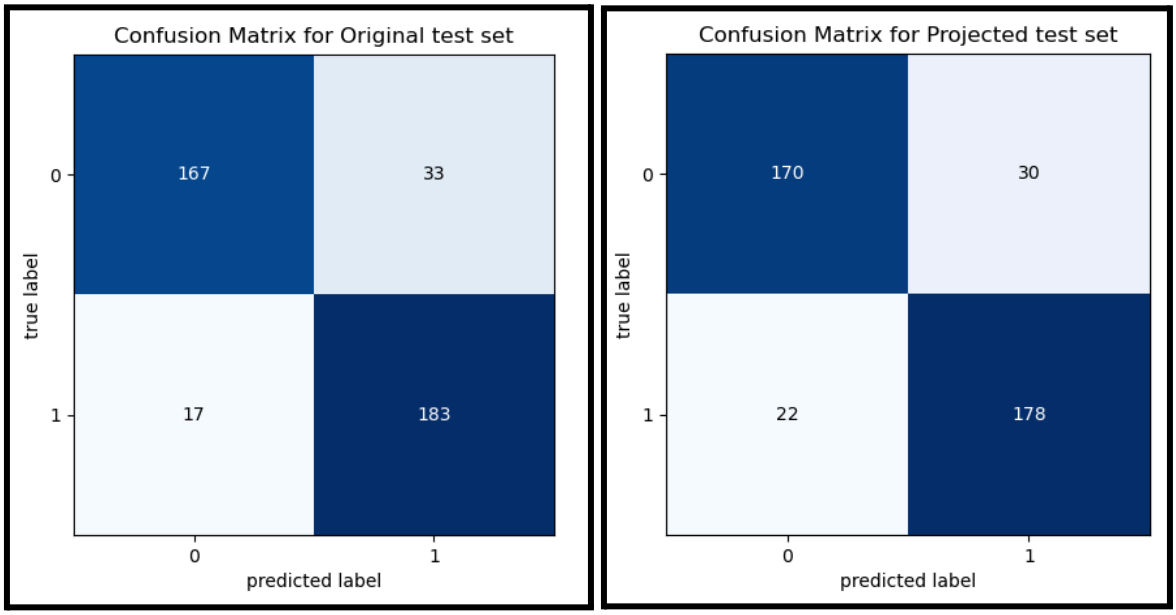
separability, as opposed to some other line. At the meeting of the clusters, we will encounter some overlap.

Task 3:

For 1-NN classifier, following are the observations:

	Training accuracy	**Testing accuracy**
Original	100%	87.5%
Projected	100%	87%

Accuracy on testing dataset slightly reduces due to the loss of one axis in this case. Slight information loss occurs, due to which few misclassifications might occur.

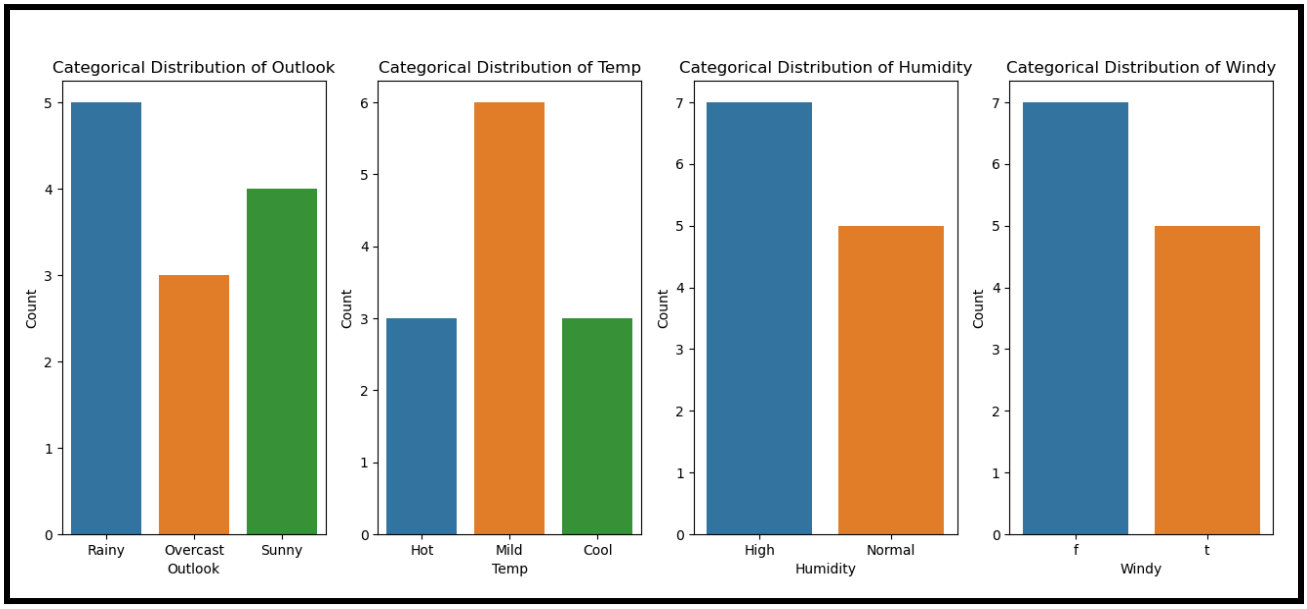


Confusion matrices for original and projected test sets respectively

Question 2:

Task 0:

- The dataset is split into training and test set, where test set contains 2 samples. Splitting is done randomly and we have one class 0 sample and class 1 sample in the test set.



Categorical distribution in training dataset

Note:

- To perform functions related to Naive Bayes Classification such as calculating prior probabilities or calculating likelihood, I have created a class **NaiveBayesClassifier**, which implements all these functions and predicts the class label.
- The class also implements Laplace smoothing, where values of α = **0, K = 0** by default, which means no Laplace smoothing. Entering values of α and K ensures Laplace smoothing.
- For Naive Bayes Classifier, **evidence is ignored** as that is just a scaling factor and won't hamper classification.
- **Logarithm is not used** for this problem as the posterior probabilities won't be low enough for numerical instability to occur.

	Outlook	Temp	Humidity	Windy	Play
0	Rainy	Hot	High	f	0
2	Overcast	Hot	High	f	1
13	Sunny	Mild	High	t	0
1	Rainy	Hot	High	t	0
9	Sunny	Mild	Normal	f	1
6	Overcast	Cool	Normal	t	1
10	Rainy	Mild	Normal	t	1
7	Rainy	Mild	High	f	0
3	Sunny	Mild	High	f	1
4	Sunny	Cool	Normal	f	1
8	Rainy	Cool	Normal	f	1
11	Overcast	Mild	High	t	1

Training set

	Outlook	Temp	Humidity	Windy	Play
12	Overcast	Hot	Normal	f	1
5	Sunny	Cool	Normal	t	0

Test set

Task 1:

- The prior probabilities are calculated in the method **calc_prior()**, where training labels are passed on as an argument to the method. In my code, I have **mapped 'yes' to 1 and 'no' to 0** for my ease. From now onwards, I shall use these notations to depict probabilities.
  Eg: Prior probability **P (Play = 1) = N_1 / N**
  N_y = number of samples where class label = 1
  N = number of samples

Observations:
P (Play = 0) = 0.333
P (Play = 1) = 0.667

Task 2:

- Likelihood probabilities for every possible case are calculated in the **calc_likelihood()** method, where the training feature matrix and their corresponding labels are passed on as arguments.
  Eg: Likelihood **P (Outlook = Sunny | Play = 1) = N_{S1} / N_1**
  N_{S1} = number of samples where outlook = sunny AND play = 1
  N_1 = number of samples where class label = 1

Observations:

```
{'Outlook': {'Rainy': [0.75, 0.25],
  'Overcast': [0.0, 0.375],
  'Sunny': [0.25, 0.375]},
 'Temp': {'Hot': [0.5, 0.125], 'Mild': [0.5, 0.5], 'Cool': [0.0, 0.375]},
 'Humidity': {'High': [1.0, 0.375], 'Normal': [0.0, 0.625]},
 'Windy': {'f': [0.5, 0.625], 't': [0.5, 0.375]}}
```

where the first cell of an array represents likelihood for a particular value of a feature, given play = 0, and the second cell represents the same but play = 1.

Eg: { ‘Outlook’ : {‘Rainy’ : [0.75] } } represents P (Outlook = Rainy | Play = 0)

Task 3:

- Posterior probabilities are calculated in the **predict()** method. It takes a feature matrix as an argument and returns the predicted class labels as well as the posterior probabilities.
  Eg: **Posterior probability P (play = 0 | features = [rainy, hot, high, f])**
  = [P (rainy | play = 0) x P (hot | play = 0) x P (high | play = 0) x P (f | play = 0)]
  x P (play = 0)
  = **Likelihoods x Prior**

  Here, **overall likelihood** = **product of individual likelihoods** because **Naive Bayes assumes that features are independent of each other.**
  Also, **evidence is ignored** as mentioned above.

Observations:

Training:

```
[[0.0625 , 0.00488],
 [0.     , 0.00732],
 [0.02083, 0.01758],
 [0.0625 , 0.00293],
 [0.     , 0.04883],
 [0.     , 0.02197],
 [0.     , 0.01953],
 [0.0625 , 0.01953],
 [0.02083, 0.0293 ],
 [0.     , 0.03662],
 [0.     , 0.02441],
 [0.     , 0.01758]]
```

where **each row represents a sample**, **first column represents posterior probability for play = 0** and **second column represents posterior probability for play = 1**

Testing:

```
[[0.     , 0.01221],
 [0.     , 0.02197]]
```

Task 4:

- Predictions are based on the prior probabilities. For a sample, the **label corresponding to the higher posterior probability is chosen as the prediction**.
  Eg: For **first sample** of training set, **prediction = 0 as 0.0625 > 0.00488**

Observations:

Training: `[0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1]`
Testing: `[1, 1]`

We encountered one misclassification in the test set.

Task 5:

- Laplace smoothing is done to prevent zero probabilities.
- It is used when there is insufficient data or when some events have never occurred in the dataset.
- **Scenario where this is useful:** Let us assume that we are attempting to estimate the probability of an event based on previous data. If we only count occurrences and find an event that has never occurred in our data, the probability estimated from the dataset is 0. However, even if the event hasn't occurred in the dataset yet, there is a slight possibility that it will later. Laplace smoothing solves this problem by adding a little amount to each count before calculating probability.

Formula:
P (Rainy | Play = 0) = (N_{R0} + α) / (N_0 + α * K)
α = smoothing parameter
K = number of features (dimensions) in data

Observations:
Using α = **1** and **K** = **4:**

Prior probabilities:

```
[0.3333333333333333, 0.6666666666666666]
```

Likelihoods:

```
{'Outlook': {'Rainy': [0.5, 0.25],
  'Overcast': [0.125, 0.3333333333333333],
  'Sunny': [0.25, 0.3333333333333333]},
 'Temp': {'Hot': [0.375, 0.16666666666666666],
  'Mild': [0.375, 0.4166666666666667],
  'Cool': [0.125, 0.3333333333333333]},
 'Humidity': {'High': [0.625, 0.3333333333333333], 'Normal': [0.125,
0.5]},
 'Windy': {'f': [0.375, 0.5], 't': [0.375, 0.3333333333333333]}}
```

Posterior probabilities:

Training:

```
[[0.01465, 0.00463],
 [0.00366, 0.00617],
 [0.00732, 0.01029],
 [0.01465, 0.00309],
 [0.00146, 0.02315],
 [0.00024, 0.01235],
 [0.00293, 0.01157],
 [0.01465, 0.01157],
 [0.00732, 0.01543],
 [0.00049, 0.01852],
 [0.00098, 0.01389],
 [0.00366, 0.01029]]
```

Testing:

```
[[0.00073, 0.00926],
 [0.00049, 0.01235]]
```

Predictions:

Training: `[0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1]`

Testing: `[1, 1]`

One misclassification in training set and one in test set.

Unfortunately, we don't get the test classifications right, however, the prior probabilities which were zero previously, **are non-zero now**, each having a **low probability value**. Further **fine tuning of** α, or a **better split**, or **more data in training set** might provide better result.