PRML ASSIGNMENT - 4

Name : Aditya Sahani
Roll No : B22CS003
Google Colab :
https://colab.research.google.com/drive/1TnkSg8eXKqTVh8ofZ9wf2L-2nvBcKHKI#scrollTo=TEv-J8UlRT7j

Linear Discriminant Analysis

Task - 1: Function Definitions and Computations

ComputeMeanDiff:
Computes the difference of class-wise means (m1 - m2).
Extracts features and labels from the dataset and calculates means for each class.

ComputeSW:
Computes the total within-class scatter matrix (S_W).
Iterates through each class to calculate the scatter matrix for each class and sums them up.

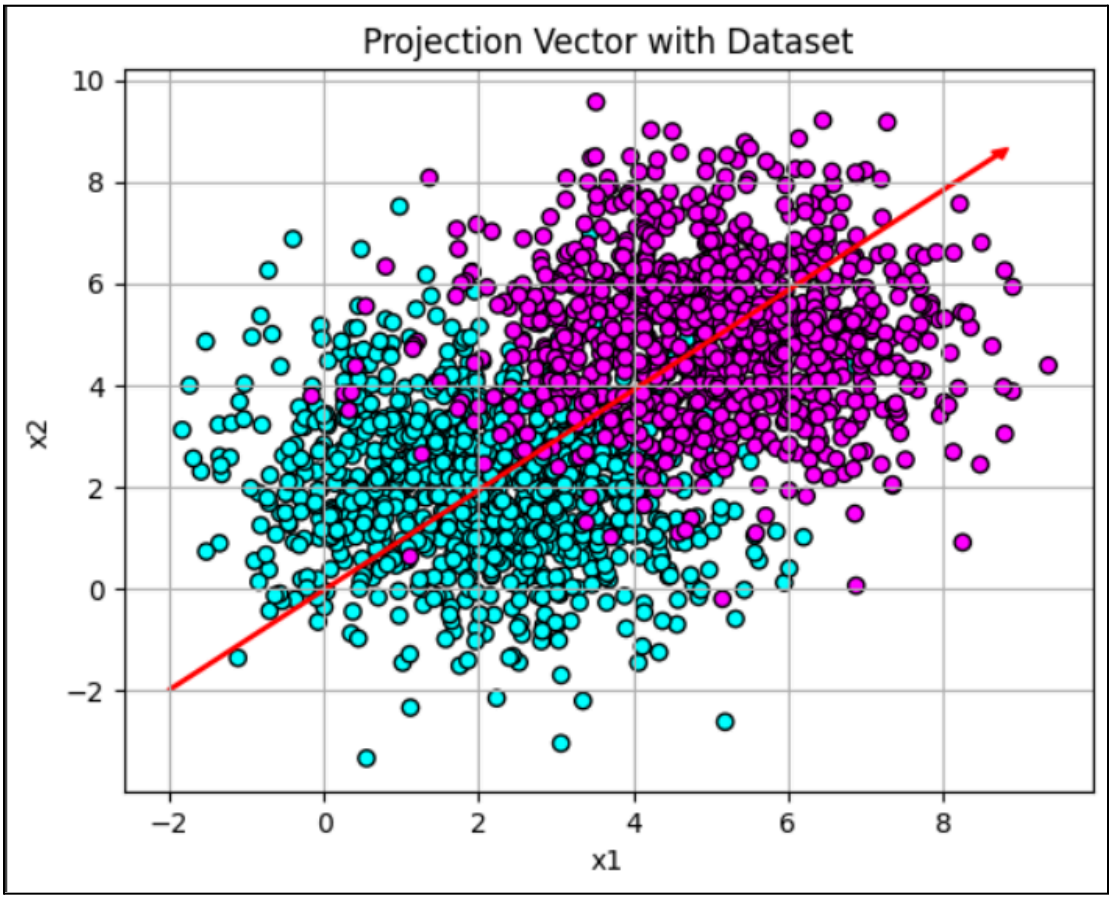
ComputeSB:
Computes the between-class scatter matrix (S_B).
Calculates the overall mean and iterates through each class to compute the scatter matrix.

GetLDAProjectionVector:
Computes the projection vector using Linear Discriminant Analysis (LDA).
Calculates the matrix S_W^-1 * S_B and extracts eigenvectors corresponding to the highest eigenvalue.

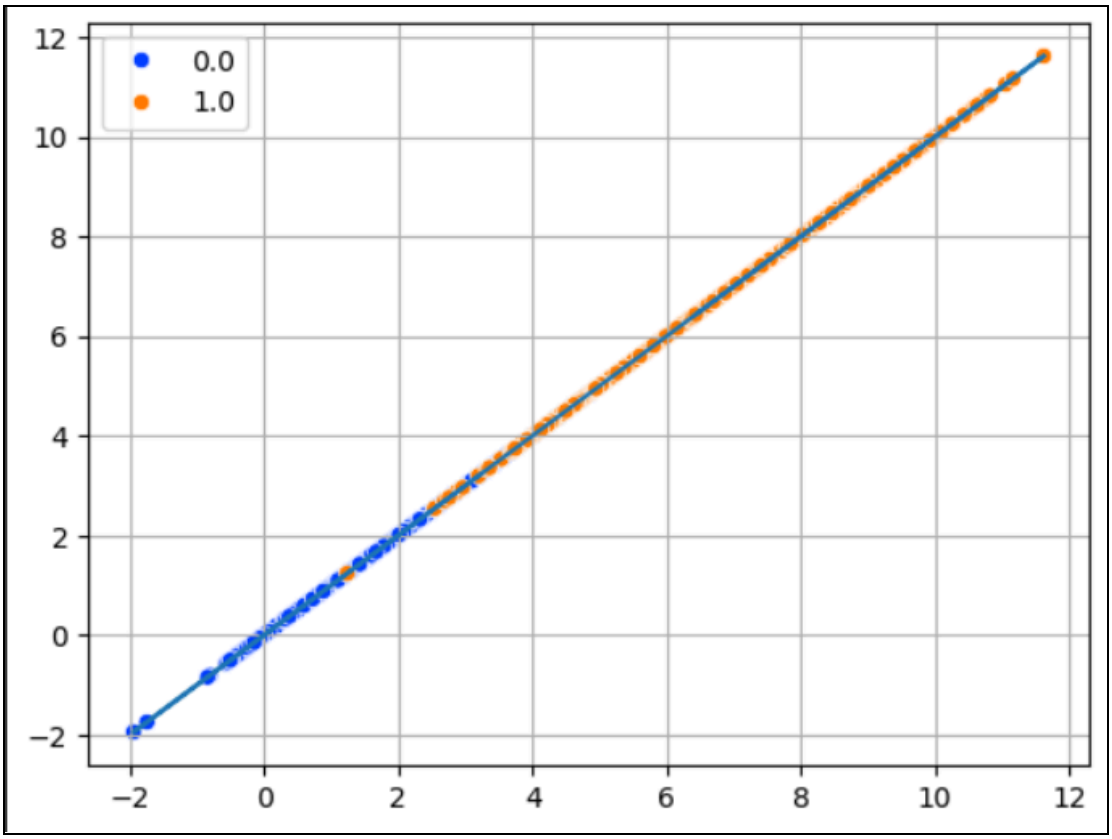
project:
Projects a 2-D point onto the LDA projection vector.

Option Selection:
Allows users to select among different options to compute and print specific terms.

Task - 2: Visualization of Results
Difference of class-wise means is calculated and printed.
Total within-class scatter matrix (S_W), between-class scatter matrix (S_B), and LDA projection vector are computed and printed.
The projection vector is visualized along with the dataset using a scatter plot.



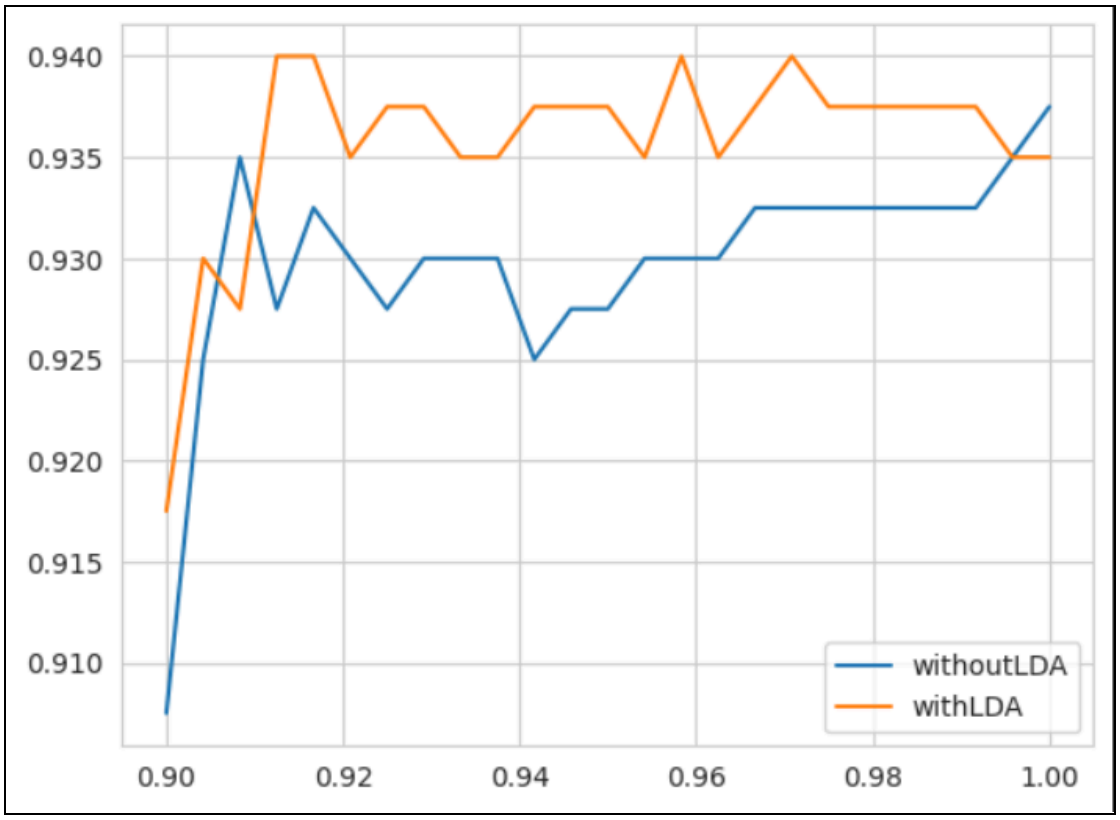
The projected data is plotted to show the transformation after applying LDA.



Task - 3: Model Training and Comparison

The K Nearest Neighbors (KNN) model is trained using both the original dataset and the projected data after LDA.
Nearest_neighbors is taken as 1.
Model accuracy is computed and compared for both scenarios.
Additionally, the LDA transformation is compared with the LDA implementation from the sklearn library by training a KNN model on the sklearn LDA-transformed data.

Accuracy without LDA : **90.7 %**
Accuracy with LDA : **91.75 %**
Accuracy with LDA from sklearn : **91.75 %**



Observation : Clearly, accuracy after LDA has improved. Accuracy is not improving much using LDA and the original model is performing well. The reason is data is 2-dimensional and already linearly separable to a good extent, so even on converting to 1-dimensional, accuracy is almost the same.

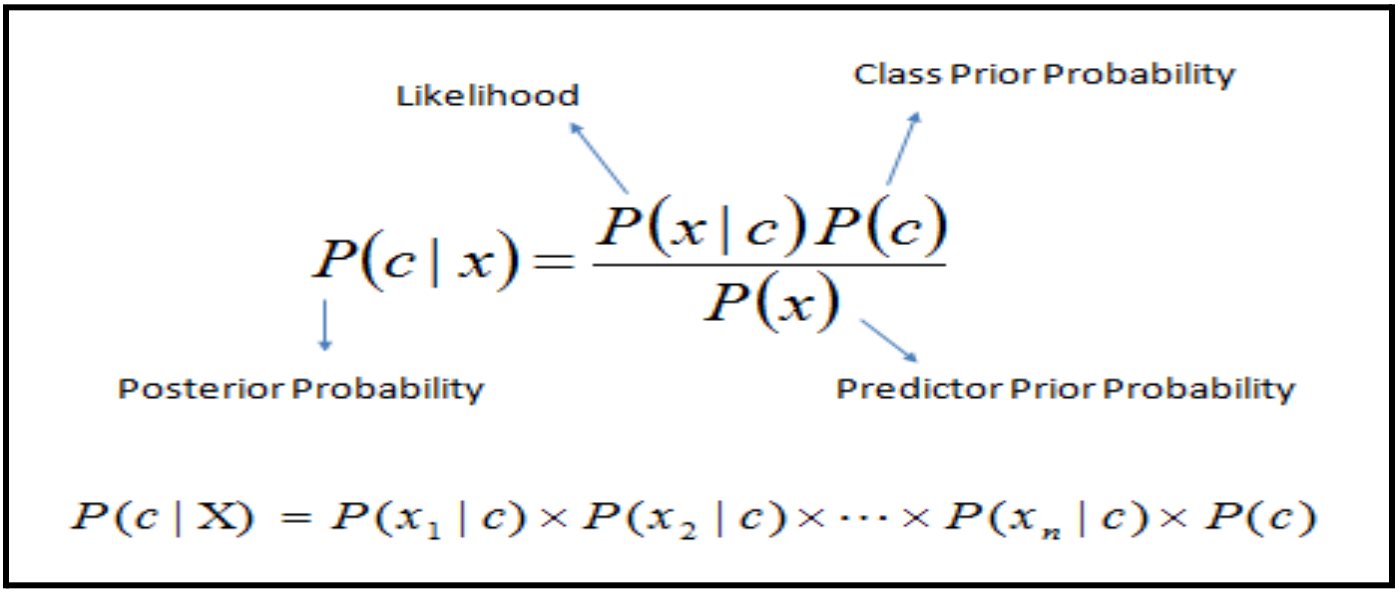
Reason :

- We are using LDA for dimensionality reduction to 1 feature and KNN performs better with low-dimensional data.
- LDA aims to find an optimal decision boundary that maximizes class separability. This decision boundary can be more effective for classification tasks compared to the nearest neighbor approach used by KNN, especially when the dataset has distinct class clusters.
- LDA is robust to outliers because we are projecting 2-D data to 1-D, so the outliers which may be present in 2D data, would be projected on the same line along with other normal points. Hence, it no longer remains an outlier.

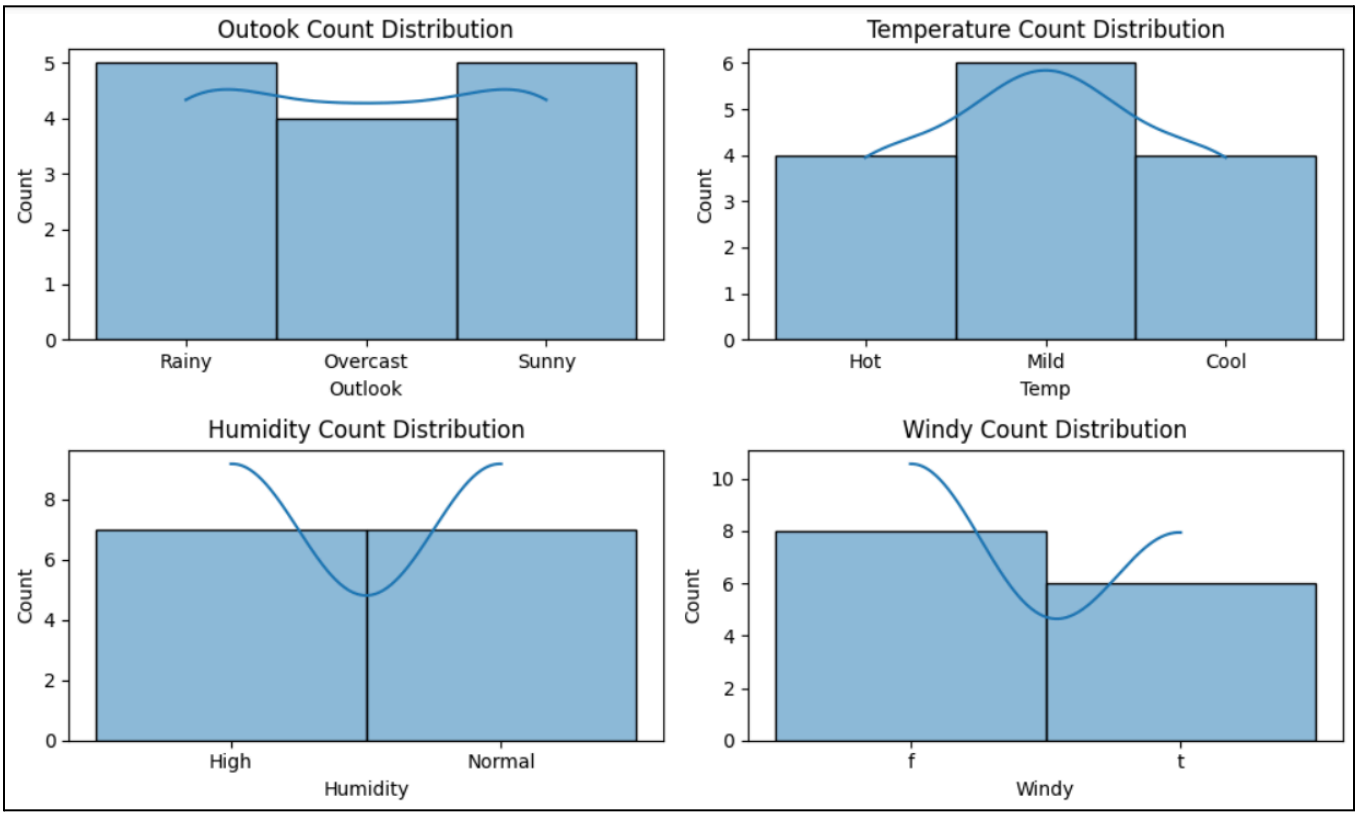
NAIVE BAYES CLASSIFIER

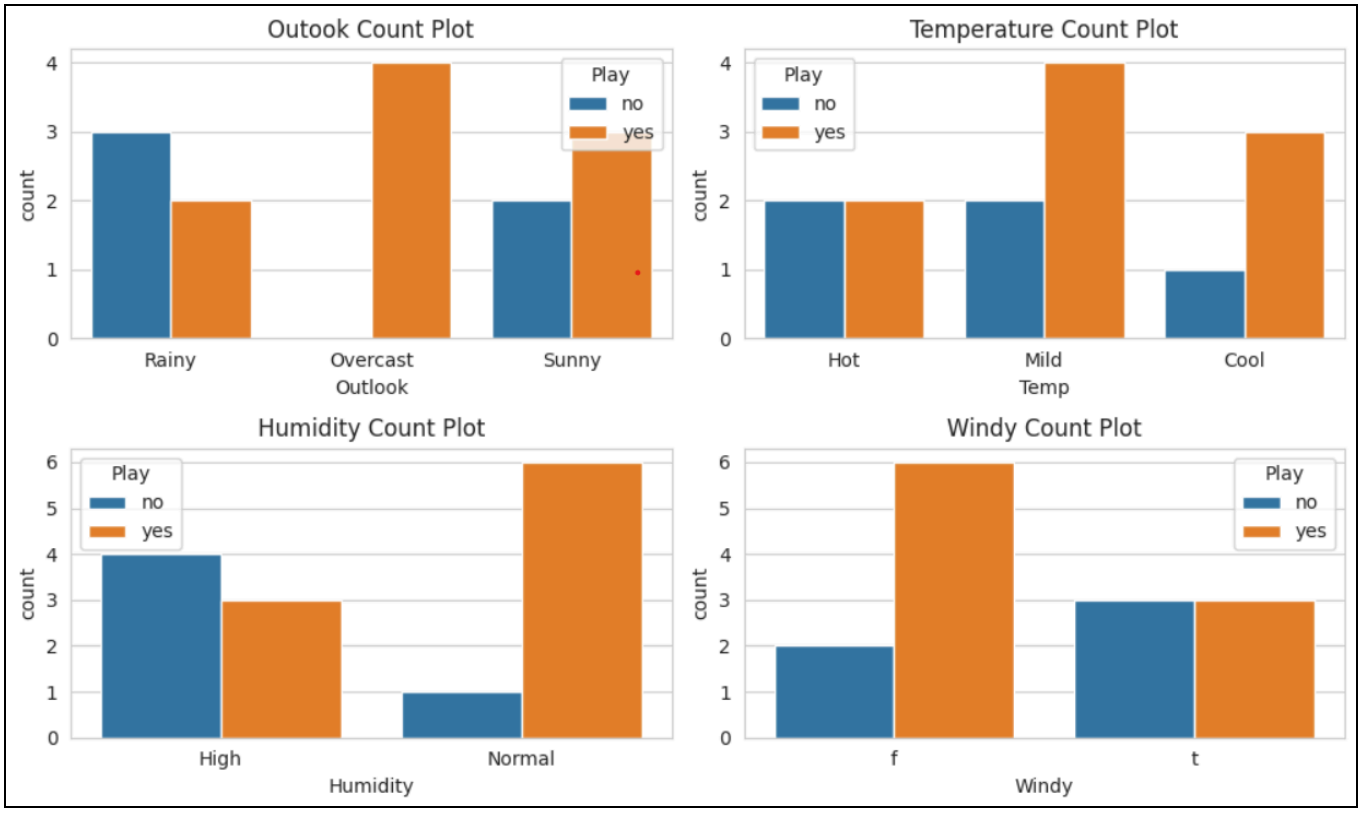
Task - 0: Data Preprocessing

The dataset is loaded from a CSV file using pandas.
The 'Play' column is considered as the target variable, and the rest are features.
The data is split into training and testing sets using the train_test_split function from sklearn.



Count Distribution Plot of Various Features :





Task - 1: Prior Probabilities

The prior probabilities for the target classes 'yes' and 'no' are calculated.
For random_state = 25
Prior probability for 'yes': **0.666**
Prior probability for 'no': **0.333**

Depending on the “random_state” value, the dataset will change and hence, the prior probability will also change.

Task - 2: Likelihood Probabilities

Likelihood probabilities for each feature given the target class are calculated.
Features include 'Outlook', 'Temp', 'Humidity', and 'Windy' :
P(Outlook = Sunny / Play = Yes), P(Temp = Cool / Play = No) etc.

This dictionary show the likelihood probabilities :
Without Laplace :

```
{'Outlook': {'Sunny': {'yes': 0.25, 'no': 0.5},
  'Overcast': {'yes': 0.5, 'no': 0.0},
  'Rainy': {'yes': 0.25, 'no': 0.5}},
 'Temp': {'Hot': {'yes': 0.25, 'no': 0.25},
  'Mild': {'yes': 0.375, 'no': 0.5},
  'Cool': {'yes': 0.375, 'no': 0.25}},
 'Humidity': {'High': {'yes': 0.375, 'no': 0.75},
  'Normal': {'yes': 0.625, 'no': 0.25}},
 'Windy': {'f': {'yes': 0.625, 'no': 0.25}, 't': {'yes': 0.375, 'no': 0.75}}}
```

With Laplace :

```
{'Outlook': {'Sunny': {'yes': 0.36363636363636365, 'no': 0.42857142857142855},
  'Overcast': {'yes': 0.36363636363636365, 'no': 0.14285714285714285},
  'Rainy': {'yes': 0.2727272727272727, 'no': 0.42857142857142855}},
 'Temp': {'Hot': {'yes': 0.18181818181818182, 'no': 0.2857142857142857},
  'Mild': {'yes': 0.45454545454545453, 'no': 0.42857142857142855},
  'Cool': {'yes': 0.36363636363636365, 'no': 0.2857142857142857}},
 'Humidity': {'High': {'yes': 0.4, 'no': 0.6666666666666666},
  'Normal': {'yes': 0.6, 'no': 0.3333333333333333}},
 'Windy': {'f': {'yes': 0.6, 'no': 0.3333333333333333},
  't': {'yes': 0.4, 'no': 0.6666666666666666}}}
```

Task - 3: Posterior Probabilities

Posterior probabilities for each target class given a test instance are calculated. Uses the likelihood probabilities and prior probabilities. Above mentioned formula was used to calculate the posterior probability.

Task - 4: Predictions and Accuracy

Predictions are made for the training set using posterior probabilities. Accuracy is calculated using sklearn's accuracy_score function.

Accuracy On Test Data : 50 %
Accuracy On Training Data : 91.66 %

Task - 5: Class Implementation for Naive Bayes

Laplace smoothing, also known as additive smoothing or pseudocount smoothing, is a technique used to address the issue of zero probabilities in statistical models, particularly in the context of probability estimation. It is commonly applied in situations where there might be unseen or unobserved data in the training set, leading to zero probabilities for certain events.

$$P(w_i|class) = \frac{freq(w_i, class)}{N_{class}} \qquad class \in \{Positive, Negative\}$$

$$P(w_i|class) = \frac{freq(w_i, class) + 1}{N_{class} + V}$$

N_{class} = frequency of all words in class

V = number of unique words in vocabulary

A class NaiveBayes is defined to train and predict using Naive Bayes without Laplace smoothing.
It includes methods for calculating likelihood and posterior probabilities, fitting the model, and making predictions.

Another class NaiveBayesLaplace is defined, extending the previous class to incorporate Laplace smoothing.

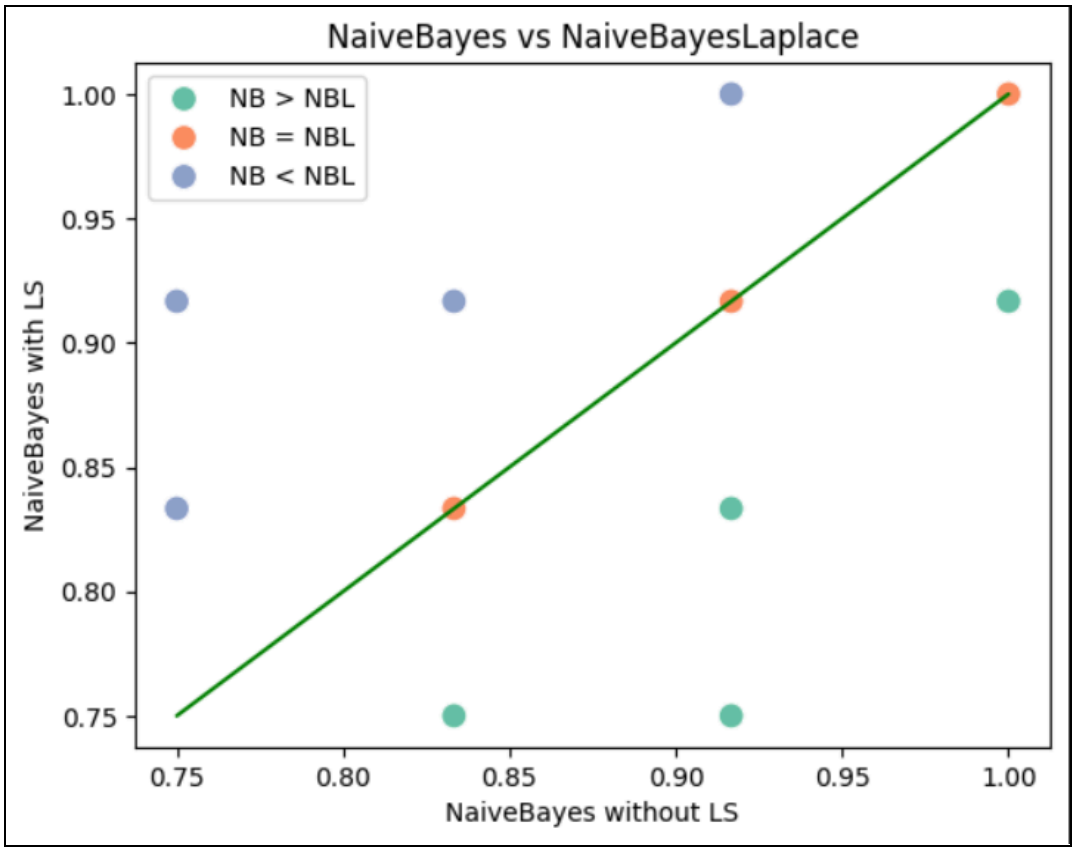
It overrides methods for likelihood probability calculation and model fitting.

Comparison of Naive Bayes and Naive Bayes with Laplace Smoothing:

Accuracy :

	NB	NBwithLS
0	0.833333	0.750000
1	0.916667	0.916667
2	0.833333	0.916667
3	0.916667	0.916667
4	0.916667	0.916667

Model performance is compared across multiple random splits of the data. Scatter plot visualizes the accuracy of Naive Bayes without Laplace smoothing against Naive Bayes with Laplace smoothing against various random_state.



Green line represents equality line (where accuracy of both models is equal).

Counts are provided for instances where Naive Bayes performs better, Naive Bayes with Laplace smoothing performs better, and both models have equal accuracy.

NB > NBL : 16 (Naive Bayes better than Naive Bayes with Laplace)
NB < NBL : 16 (Naive Bayes Laplace better than Naive Bayes)
NB = NBL : 68 (Both coming equal on accuracy)

Observation :

- I have calculated accuracy for both the models (with or without Laplace Smoothing) while varying the random_state, both the models

are performing equally well, somewhere NB with Laplace is dominating while somewhere NB without NB is good.

- It is not guaranteed that using NB with Laplace Smoothing, our accuracy will improve.

For this dataset : There would be significant difference in probabilities and likelihood probabilities would be different in both the cases.

	Outlook	Temp	Humidity	Windy	Play
0	Sunny	Mild	High	f	yes
1	Sunny	Cool	Normal	t	no
2	Overcast	Mild	High	t	yes
3	Rainy	Hot	High	t	no
4	Overcast	Hot	Normal	f	yes
5	Sunny	Mild	High	t	no
6	Rainy	Cool	Normal	f	yes
7	Overcast	Hot	High	f	yes
8	Rainy	Mild	High	f	no
9	Overcast	Cool	Normal	t	yes
10	Rainy	Mild	Normal	t	yes
11	Sunny	Cool	Normal	f	yes

	NB Predict
0	no
1	yes
2	yes
3	no
4	yes
5	no
6	yes
7	yes
8	no
9	yes
10	no
11	yes

	NB Laplace
0	yes
1	yes
2	yes
3	no
4	yes
5	no
6	yes
7	yes
8	yes
9	yes
10	yes
11	yes

Without Laplace :

```
{'Outlook': {'Sunny': {'yes': 0.25, 'no': 0.5},
  'Overcast': {'yes': 0.5, 'no': 0.0},
  'Rainy': {'yes': 0.25, 'no': 0.5}},
 'Temp': {'Hot': {'yes': 0.25, 'no': 0.25},
  'Mild': {'yes': 0.375, 'no': 0.5},
  'Cool': {'yes': 0.375, 'no': 0.25}},
 'Humidity': {'High': {'yes': 0.375, 'no': 0.75},
  'Normal': {'yes': 0.625, 'no': 0.25}},
 'Windy': {'f': {'yes': 0.625, 'no': 0.25}, 't': {'yes': 0.375, 'no': 0.75}}}
```

With Laplace :

```
{'Outlook': {'Sunny': {'yes': 0.36363636363636365, 'no': 0.42857142857142855},
  'Overcast': {'yes': 0.36363636363636365, 'no': 0.14285714285714285},
  'Rainy': {'yes': 0.2727272727272727, 'no': 0.42857142857142855}},
 'Temp': {'Hot': {'yes': 0.18181818181818182, 'no': 0.2857142857142857},
  'Mild': {'yes': 0.45454545454545453, 'no': 0.42857142857142855},
  'Cool': {'yes': 0.36363636363636365, 'no': 0.2857142857142857}},
 'Humidity': {'High': {'yes': 0.4, 'no': 0.6666666666666666},
  'Normal': {'yes': 0.6, 'no': 0.3333333333333333}},
 'Windy': {'f': {'yes': 0.6, 'no': 0.3333333333333333},
  't': {'yes': 0.4, 'no': 0.6666666666666666}}}
```

Laplace Smoothing is distributing the probabilities uniformly around the dataset so that probability is not biased to one feature and hence is reducing the overfitting in the model. One can see from the above predictions comparison.

Reason :

- Zero frequencies in training data are addressed via Laplace smoothing. Traditional Naive Bayes assign a probability of zero to feature-value

pairs that are not in the training set for a class, resulting in a zero conditional probability. Laplace smoothing adds a modest count to each feature-value combination to prevent zero probability estimates. This keeps the model from overfitting and improves probability estimations.

- Laplace smoothing balances probability estimates across classes and feature-value pairs. Adding a constant pseudo-count to each count makes probabilities more uniformly distributed, preventing unusual events and outliers from dominating probability estimations.

———————————— END ————————————————————————————