



Programming Assignment 4

Link to google colab-: [co B22ES018_all](#)

Name-: Anuj chincholikar

Roll no -: B22ES018

Q1. LDA -:

- **TASK(1) -:**

In Task 1, the code provided aims to compute various terms related to Linear Discriminant Analysis (LDA) based on the given dataset. Here's a breakdown of what each function does:

1. **ComputeMeanDiff(X)**: This function calculates the difference between the class-wise means. It separates the samples into two classes based on their labels and then computes the mean for each class. Finally, it calculates the difference between these means.

2. **ComputeSW(X)**: This function computes the Total Within-class Scatter Matrix (SW). It separates the samples into two classes based on their labels and then computes the covariance matrix for each class. The SW matrix is obtained by summing up these covariance matrices.

3. **ComputeSB(X)**: This function computes the Between-class Scatter Matrix (SB). It calculates the outer product of the difference between the class means and returns this matrix.

4. **GetLDAProjectionVector(X)**: This function computes the LDA projection vector. It first calculates the SW matrix similar to ComputeSW(). Then, it calculates the difference between class means and multiplies it by the inverse of SW to obtain the projection vector.

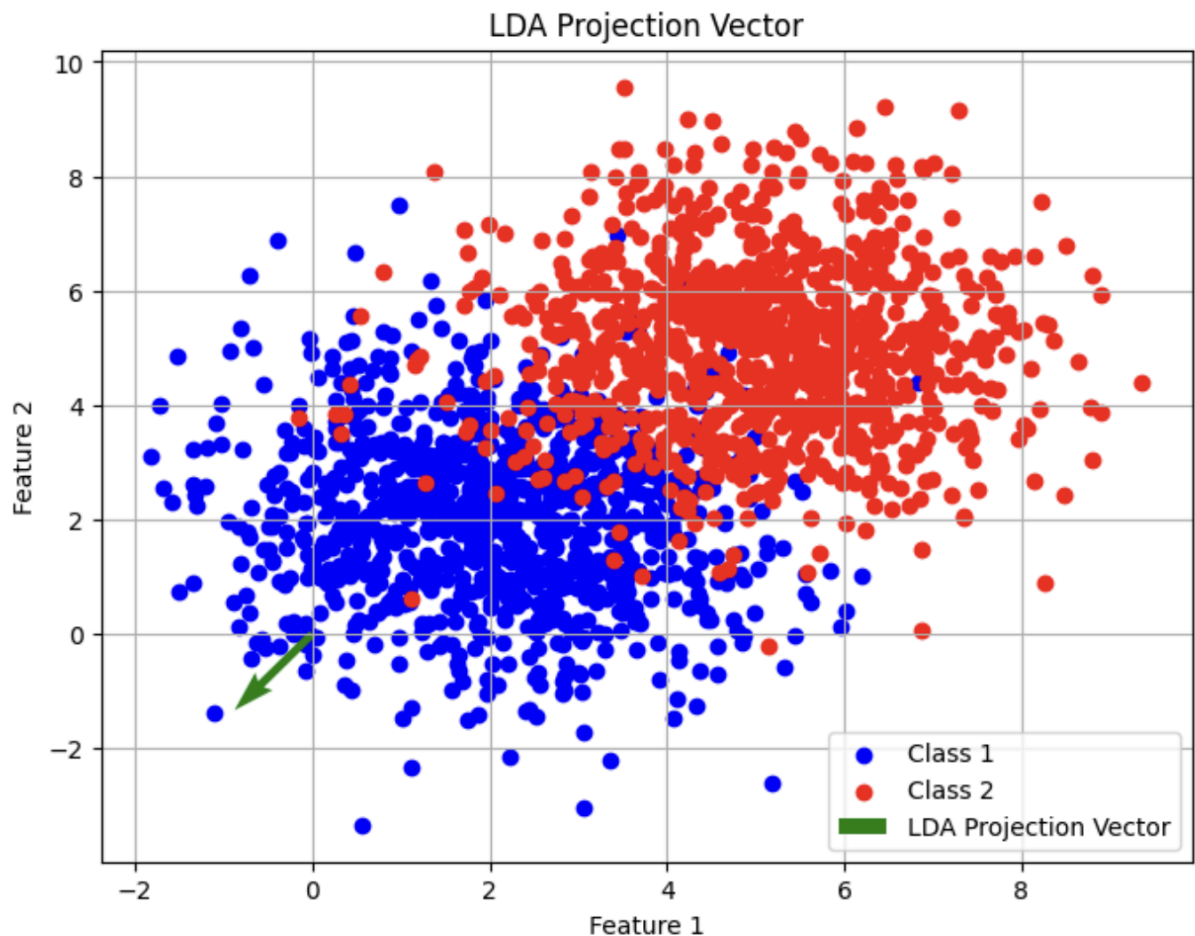
5. **project(x, y, w)**: This function projects a given 2-dimensional point onto the LDA projection vector. It takes the x and y coordinates of the point, along with the projection vector w. It first calculates the projection vector using the same steps as GetLDAProjectionVector(), then computes the dot product of the input point with the projection vector to obtain the projection.

- **TASK(2) -:**

In Task 2, the code visualizes the LDA (Linear Discriminant Analysis) projection vector on a plot using Python's matplotlib library. Initially, it loads a dataset stored in a CSV file, dividing it into two classes based on labels. Then, it computes the LDA projection vector, which optimally projects the data points onto a lower-dimensional space to maximize class separation. The plotted graph shows each data point, with samples from one class depicted in blue and samples from the other in red. Additionally, the LDA projection vector is displayed as a green arrow, indicating

its direction and magnitude. This visualization helps to intuitively grasp how the LDA projection vector separates the classes in the dataset, offering insights into its effectiveness in achieving discrimination between classes.


❖ **BELOW IS THE OBTAINED PLOT WITH CLASS 1 , CLASS 2 AND LDA PROJECTION VECTORS MARKED WITH BLUE , RED AND GREEN LABELS -:**



● **TASK(3) -:**

In Task 3, the code compares the performance of a 1-NN (1-Nearest Neighbor) classifier on the original dataset versus the dataset projected onto the Linear Discriminant Analysis (LDA) subspace. Initially, the data is loaded from a CSV file, with features and labels separated accordingly. Then, the original dataset is split into training and testing sets using a standard train-test split. A 1-NN classifier is trained on the original data, and its accuracy is evaluated on the test set. Subsequently, the original data is projected onto the LDA subspace using the previously computed LDA projection vector. The projected dataset is then split into training and testing sets again. Another 1-NN classifier is trained on the projected data, where the features are reshaped to have a single dimension. Finally, the accuracy of the 1-NN classifier on the projected data is evaluated and compared with the accuracy on the original data. This comparison helps in assessing how the LDA projection affects the performance of the classifier, providing insights into the effectiveness of dimensionality reduction techniques like LDA in improving classification accuracy.

❖ **BELOW ARE THE OBTAINED ACCURACIES OF ORIGINAL DATA AND PROJECTED DATA:-**

 Accuracy of 1-NN classifier on original data: 0.8875
Accuracy of 1-NN classifier on projected data: 0.88

Q2. NAIVE BAYES -:

- **TASK(0) -:**

In Task 0, the code reads a dataset from a CSV file containing information about weather conditions and whether people played a certain outdoor sport. The dataset is then split into training and testing sets using the `train_test_split` function from `scikit-learn`. The split is such that 12 samples are placed in the training set (`x_train` and `y_train`), and 2 samples are placed in the testing set (`x_test` and `y_test`). This splitting allows for the creation of separate datasets for training the model and evaluating its performance. The shapes of the training and testing sets are printed to verify that the split has been performed correctly.

- **TASK(1) -:**

In Task 1, the code calculates the Prior Probabilities for the classes "yes" and "no" based on the target variable "Play" in the dataset. It iterates through each entry in the "Play"

column of the dataframe and counts the occurrences of "yes" and "no". The total count of "yes" occurrences is divided by the total number of samples in the dataset to calculate the prior probability of "yes", while the total count of "no" occurrences is divided by the total number of samples to calculate the prior probability of "no". These prior probabilities represent the likelihood of observing each class in the dataset before considering any features, and they serve as the basis for subsequent probability calculations in Naive Bayes classification. Finally, the calculated prior probabilities are printed to the console for analysis and verification.

❖ **OUTPUT:-**

```
⇒ Prior probability of yes: 0.6428571428571429  
   Prior probability of no : 0.35714285714285715
```

● **TASK(2) :-**

In Task 2, the code calculates the Likelihood Probabilities for each feature given the class, where the class can be either "yes" or "no". The Likelihood Probability represents the probability of observing a specific feature value given a certain class.

The code iterates over the DataFrame and counts occurrences for each possible combination of feature value and class. For example, it counts how many times the Outlook is "Sunny" and the class is "yes", "no", "Rainy" and the class is "yes", "no", and so on for all possible combinations of feature values and classes.

After counting occurrences, the code calculates the Likelihood Probabilities by dividing the count of each specific combination by the total count of the corresponding class. This gives the probability of observing a particular feature value given a specific class.

Finally, the calculated Likelihood Probabilities are printed for each feature value and class combination. This information will be used later in the Naive Bayes algorithm to compute the posterior probabilities and make predictions.

Additionally, the code provides a generalized function `calculate_likelihood_probabilities()` to calculate Likelihood Probabilities for any dataset and set of feature columns. This function can be used with different datasets and feature sets by passing appropriate arguments, providing a modular and reusable solution.

❖ **OUTPUT-:**


```

probability of Outlook = Sunny and play = Yes : 0.3333333333333333
probability of Outlook = Sunny and play = No : 0.4
probability of Outlook = Rainy and play = Yes : 0.2222222222222222
probability of Outlook = Rainy and play = No : 0.6
probability of Outlook = Overcast and play = Yes : 0.4444444444444444
probability of Outlook = Overcast and play = No : 0.0
probability of Temp = Mild and play = Yes : 0.4444444444444444
probability of Temp = Mild and play = No : 0.4
probability of Temp= Cool and play = Yes : 0.3333333333333333
probability of Temp = Cool and play = No : 0.2
probability of Temp = Hot and play = Yes : 0.2222222222222222
probability of Temp = Hot and play = No : 0.4
probability of Humidity= High and play = Yes : 0.3333333333333333
probability of Humidity= High and play = No : 0.8
probability of Humidity= Normal and play = Yes : 0.6666666666666666
probability of Humidity= Normal and play = No : 0.2
probability of Windy= f and play = Yes : 0.6666666666666666
probability of Windy= f and play = No : 0.4
probability of Windy= t and play = Yes : 0.3333333333333333
probability of Windy= t and play = No : 0.6

```

- **TASK(3)-:**

In Task 3, the code calculates the posterior probabilities for the samples in the test dataset `x_test`. It does so by applying the Naive Bayes formula, which involves computing the likelihood probabilities, prior probabilities, and individual probabilities for each feature given the class.

For each sample in the test dataset, the code calculates the likelihood probabilities based on the observed feature values. Then, it computes the prior probabilities, which are the probabilities of each class (e.g., "yes" or "no") occurring

in the dataset. Additionally, it calculates individual probabilities for each feature value.

Using these probabilities, the code calculates the posterior probabilities for both classes ("yes" and "no") for each sample in the test dataset. This is done by applying the Naive Bayes formula, which involves multiplying the likelihood probabilities, prior probabilities, and individual probabilities for each feature value given the class. Finally, it prints the posterior probabilities for each sample.

Additionally, the code provides a generalized function `calculate_posterior_probabilities()` that automates this process. This function takes the dataset, feature columns, class column, and input combination (feature values for a specific sample) as inputs and calculates the posterior probabilities accordingly. This function can be used with different datasets and input combinations, providing a modular and reusable solution for computing posterior probabilities in Naive Bayes classification.

❖ OUTPUT OBTAINED AFTER CALCULATING POSTERIOR PROBABILITY ON TESTING DATASET:-

```
Posterior probability of Yes given sample 1: 0.9679012345679012
Posterior probability of No given sample 1: 0.10453333333333337
*****
*****
*****
Posterior probability of Yes given sample 2: 0.5377229080932785
Posterior probability of No given sample 2: 0.0
```

This is the output that we get after manual calculation , same is the output from the generalized function as well .

❖ **BELOW IS THE OUTPUT FROM GENERALIZED FUNCTION:-**

```
Posterior probability of no given Outlook=Overcast, Temp=Mild, Humidity=High, Windy=t: 0.0
Posterior probability of yes given Outlook=Overcast, Temp=Mild, Humidity=High, Windy=t: 0.5377229080932785
Posterior probability of no given Outlook=Sunny, Temp=Mild, Humidity=Normal, Windy=f: 0.10453333333333337
Posterior probability of yes given Outlook=Sunny, Temp=Mild, Humidity=Normal, Windy=f: 0.9679012345679012
```

- **TASK(4) :-**

In Task 4, the code implements a function predict() that takes the posterior probabilities calculated in Task 3 as input and predicts whether it's suitable to play or not for each sample in the test dataset x_test.

For each sample in the test dataset, the function compares the posterior probabilities of both classes ("yes" and "no") calculated in Task 3. If the posterior probability of the "yes" class is greater than that of the "no" class, the function predicts "Yes, we can play!!". Otherwise, it predicts "We cannot play!!".


The code then extracts the posterior probabilities for both samples in the test dataset (p_{yes_x1} , p_{no_x1} for sample 1, and p_{yes_x2} , p_{no_x2} for sample 2) obtained from the results of Task 3. Finally, it calls the `predict()` function twice, passing the extracted posterior probabilities for each sample as arguments, which results in predicting whether it's suitable to play or not for each sample in the test dataset.

❖ **OUTPUT OF PREDICTION FROM TEST DATASET:-**

```
Yes we can play !!  
Yes we can play !!
```

● **TASK(5) :-**

In Task 5, Laplace Smoothing is applied to the likelihood probabilities calculated in Task 2. Laplace Smoothing is used to handle the situation where some probabilities might be zero due to the absence of certain feature combinations in the training dataset.



The Laplace Smoothing constant (alpha) is defined as 1. Then, Laplace Smoothing is applied to calculate the likelihood probabilities for each feature value and class combination. For each feature, Laplace Smoothing is used to compute the likelihood probabilities of each class given that feature. This involves adding the Laplace Smoothing constant to the count of occurrences of each feature value for each class and then dividing by the sum of the counts for all feature values plus alpha times the total number of unique feature values.

Next, the posterior probabilities are calculated using the Laplace smoothed likelihood probabilities, prior probabilities, and the test data. This involves multiplying the likelihood probabilities for each class given the test data by the prior probability of that class and then normalizing the probabilities.

Finally, the code prints out the renewed likelihood probabilities and the renewed posterior probabilities for the two samples in the test dataset, showing the impact of Laplace Smoothing on the probability calculations. Additionally, a generalized function is provided to calculate likelihood probabilities and posterior probabilities with Laplace Smoothing for any given dataset and input combination, allowing for easier application of Laplace Smoothing in other scenarios.

❖ OUTPUT OBTAINED -:

Renewed Likelihood Probabilities:

```

Probability of outlook = Sunny and play = Yes: 0.3333333333333333
Probability of outlook = Sunny and play = No: 0.375
Probability of outlook = Rainy and play = Yes: 0.25
Probability of outlook = Rainy and play = No: 0.5
Probability of outlook = Overcast and play = Yes: 0.4166666666666667
Probability of outlook = Overcast and play = No: 0.125
Probability of Temperature = Mild and play = Yes: 0.4166666666666667
Probability of Temperature = Mild and play = No: 0.375
Probability of Temperature = Cool and play = Yes: 0.3333333333333333
Probability of Temperature = Cool and play = No: 0.25
Probability of Temperature = Hot and play = Yes: 0.25
Probability of Temperature = Hot and play = No: 0.375
Probability of Humidity = High and play = Yes: 0.3636363636363636
Probability of Humidity = High and play = No: 0.7142857142857143
Probability of Humidity = Normal and play = Yes: 0.6363636363636364
Probability of Humidity = Normal and play = No: 0.2857142857142857
Probability of Windy = f and play = Yes: 0.6363636363636364
Probability of Windy = f and play = No: 0.4285714285714285
Probability of Windy = t and play = Yes: 0.3636363636363636
Probability of Windy = t and play = No: 0.5714285714285714

```


Renewed Posterior Probabilities:

```

Posterior probability of Yes given sample 1: 0.854638487208009
Posterior probability of No given sample 1: 0.1453615127919911
Posterior probability of Yes given sample 2: 0.6835222319093288
Posterior probability of No given sample 2: 0.31647776809067124

```

Also, Before Laplace Smoothing, likelihood probabilities directly calculated from the training data might result in zero probabilities for certain feature combinations that were not observed in the dataset. This occurrence could lead to zero posterior probabilities, especially when such feature combinations are encountered during inference. Consequently, the model might fail to provide meaningful



predictions for these unseen combinations. However, after applying Laplace Smoothing, likelihood probabilities are adjusted to prevent zero probabilities, ensuring that all feature combinations have non-zero probabilities. This adjustment, in turn, impacts the posterior probabilities, leading to more realistic and reliable estimates of class probabilities given the input features. By avoiding zero probabilities and mitigating the effects of data sparsity, Laplace Smoothing contributes to the stability, generalization, and robustness of the model, allowing it to provide more meaningful predictions even for previously unseen feature combinations. Thus, Laplace Smoothing is a valuable technique in probabilistic modeling, particularly when dealing with sparse datasets or when aiming for more reliable inference results.