PATTERN RECOGNITION AND MACHINE LEARNING
LAB REPORT
ASSIGNMENT-4

Lavangi Parihar
B22EE044

Question 1 LDA

Task 1– Generating myLDA

- Data Loading and Preparation:

  The code loads data from a CSV file named 'data.csv', where each row represents a sample with two features (x, y) and a class label (0 or 1).

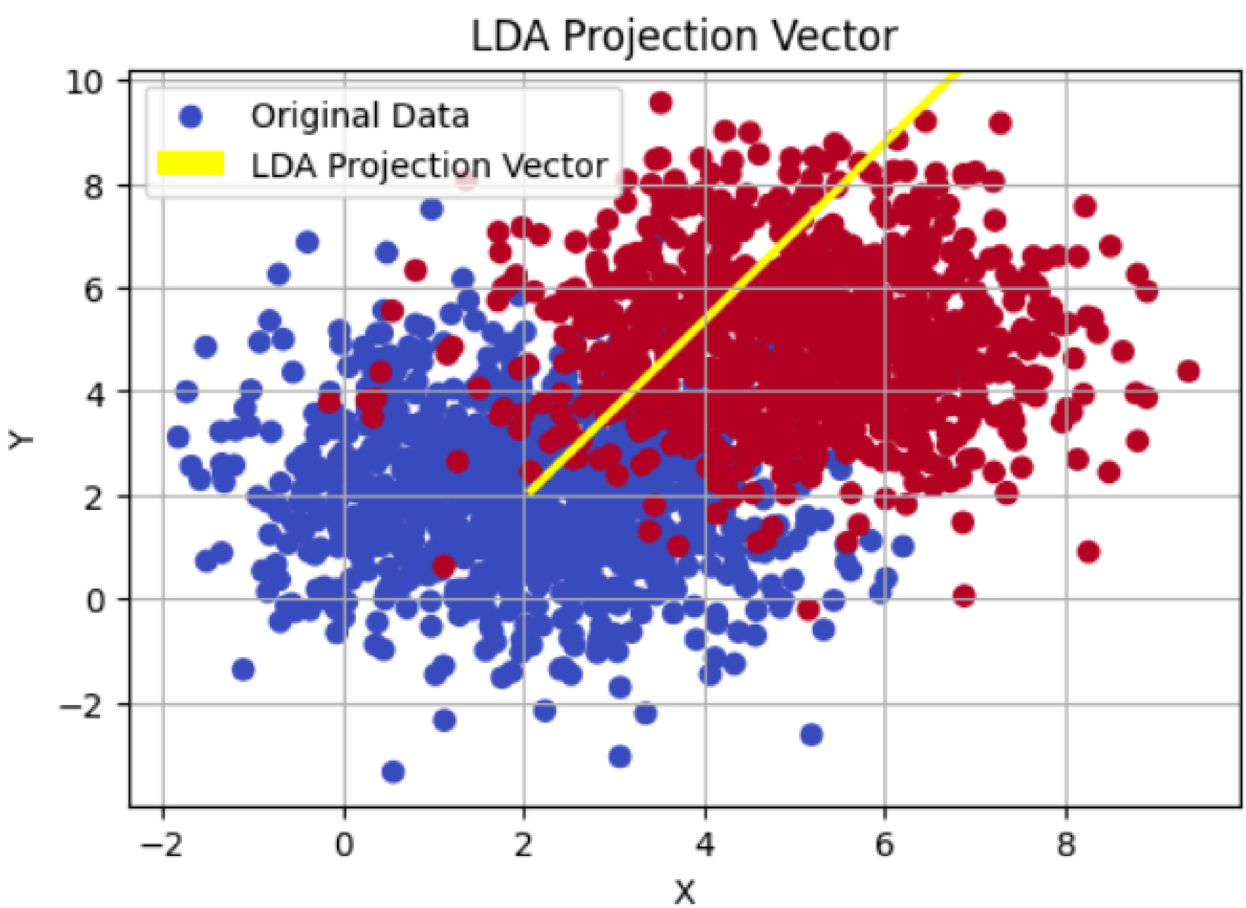
- LDA Functions:
  - `ComputeMeanDiff(X)`: Computes the mean difference vector between the class means.
  - `ComputeSW(X)`: Computes the within-class scatter matrix.
  - `ComputeSB(X)`: Computes the between-class scatter matrix.
  - `GetLDAProjectionVector(X)`: Computes the LDA projection vector.
  - `project(x, y, w)`: Projects a 2-dimensional point onto the LDA projection vector.

- Results:
  - `ComputeMeanDiff(X)`:
    ```
    [2.986 3.024]
    ```
    .
  - `ComputeSW(X)`:
    ```
    [[ 2.213 -0.119]
     [-0.119  2.28 ]]
    ```
  - `ComputeSB(X)`:
    ```
    [[8.914 9.029]
     [9.029 9.146]]
    ```
  - `GetLDAProjectionVector(X)`:
    ```
    [0.713 0.701]
    ```
  - `project(x, y, w)`:
    ```
    Input x dimension of a 2-dimensional point: 5
    Input y dimension of a 2-dimensional point: 7
    8.472916093359904
    ```

Task 2– Plotting LDA projection Vector

- LDA Computation:
  - The part initializes an LDA classifier using `LDA()` from scikit-learn.
  - It fits the LDA model to the data using `lda.fit(X, y)`.
  - The LDA model learns the optimal projection vector during the fitting process.
  - The learned projection vector is stored in `lda_projection_vector`.

- Plotting:
  - The scatter plot visualizes the original data points in 2D space, where each class is represented by a different color.
  - The yellow arrow represents the LDA projection vector, which maximizes the separation between the classes.
  - LDA projects the data onto this vector, effectively reducing the dimensionality while preserving class discriminability.
  - The plot provides an intuitive understanding of how LDA transforms the data for classification.



Task 3– Computing Performance

- 1. Data Splitting:
  - The part splits the original dataset (`X`, `y`) into training and testing sets (`X_train`, `X_test`, `y_train`, `y_test`) using `train_test_split` from scikit-learn.
  - The testing set size is set to 30% of the original data, and a random state of 42 is used for reproducibility.

- Original Data Classification (1-NN):
  - It initializes a 1-nearest neighbor classifier (`knn_original`) using `KNeighborsClassifier` from scikit-learn.
  - The classifier is trained on the original training data (`X_train`, `y_train`) using `knn_original.fit()`.
  - Predictions are made on the testing set (`X_test`) using `knn_original.predict()`.
  - The accuracy of the classifier on the original data is computed using `accuracy_score` from scikit-learn.

- LDA Projection:
  - An LDA object is initialized (`lda`) using `LDA()` from scikit-learn.
  - The training data is projected onto a lower-dimensional space using LDA with `lda.fit_transform(X_train, y_train)`. Similarly, the testing data is transformed using `lda.transform(X_test)`.

- Projected Data Classification (1-NN):
  - A new 1-nearest neighbor classifier (`knn_projected`) is initialized.
  - This classifier is trained on the projected training data (`X_train_projected`, `y_train`) using `knn_projected.fit()`.
  - Predictions are made on the projected testing set (`X_test_projected`) using `knn_projected.predict()`.
  - The accuracy of the classifier on the projected data is computed using `accuracy_score`.

- Accuracy Scores:

```
Accuracy on Original Data: 0.89
Accuracy on Projected Data: 0.8716666666666667
```

Question 1   LDA

Task 1– Calculating Prior Probabilities:

- Probability Calculation:
  - `p_play_yes`: The prior probability of the event 'play' (label 'yes'), calculated by dividing the count of 'yes' samples by the total number of samples.
  - `p_play_no`: The prior probability of the event 'not play' (label 'no'), calculated by dividing the count of 'no' samples by the total number of samples.

- Prior probabilities represent the probabilities of classes before observing any evidence or features of the data.

- In this context, `p_play_yes` represents the probability of playing given no other information, and `p_play_no` represents the probability of not playing given no other information.

- These probabilities serve as the baseline probabilities for the classes in the classification problem.
- They are essential for Bayesian inference and are used in many machine learning algorithms, including Naive Bayes classifiers.

```
Prior Probability of playing : 0.5833333333333334
Prior Probability of not playing: 0.4166666666666667
```

Task 2 and Task 5– Calculating Likelihood Probabilities Using laplace smoothing:

- Pseudocount Definition:
  - `pseudocount`: This variable holds the value of the pseudocount used in Laplace smoothing. It's set to 1 in this code.

- Likelihood Probabilities with Laplace Smoothing:
  - A dictionary `likelihood_probabilities_smoothed` is initialized to store the likelihood probabilities with Laplace smoothing.
  - The part iterates over each feature in the training data and for each unique feature value calculates the likelihood probability for each class using Laplace smoothing.
  - Laplace smoothing is applied by adding the pseudocount to the numerator and adjusting the denominator to account for the added counts.
  - The calculated likelihood probabilities are stored in the dictionary.

- Prior Probabilities with Laplace Smoothing:
  - Prior probabilities for both classes ('yes' and 'no') are calculated using Laplace smoothing.
  - Laplace smoothing is applied by adding the pseudocount to the numerator and adjusting the denominator to account for the added counts.

- Laplace smoothing is applied to avoid zero probabilities when a feature value has not been observed with a particular class label in the training data.
- Pseudocount is added to both the numerator and the denominator to ensure that every feature value has a nonzero probability estimate.
- Likelihood probabilities represent the probability of observing a particular feature value given the class label.
- Prior probabilities represent the probability of each class occurring before observing any feature information.

```
Likelihood Probabilities with Laplace smoothing:
P(Outlook_Overcast=0|Play=no) = 0.8571428571428571
P(Outlook_Overcast=0|Play=yes) = 0.5555555555555556
P(Outlook_Overcast=1|Play=no) = 0.14285714285714285
P(Outlook_Overcast=1|Play=yes) = 0.4444444444444444
P(Outlook_Rainy=1|Play=no) = 0.5714285714285714
P(Outlook_Rainy=1|Play=yes) = 0.3333333333333333
……….
```

Task 3– Calculating Posterior Probabilities:

- Calculation of Posterior Probabilities:
  - For each sample in the testing set (`X_test`), the code calculates the posterior probability for both classes ('yes' and 'no') using the Naive Bayes formula.
  - The prior probability of each class is multiplied by the likelihood probabilities of the features given the class, which were previously calculated with Laplace smoothing.
  - The resulting posterior probabilities for each class are stored in the respective dictionaries.

- Normalization:
  - The probabilities in `posterior_probabilities_yes` and `posterior_probabilities_no` are normalized to ensure that they sum up to 1.
  - This is done by dividing each probability by the total probability of all samples classified as 'yes' and 'no', respectively.

- Posterior probabilities represent the probability of each class given the observed features of a sample.
- By comparing the posterior probabilities of the classes, the code can predict the class label for each sample in the testing set.

```
Posterior probabilities for Play = yes:
Sample 9: 0.9090909090909091
Sample 11: 0.09090909090909088

Posterior probabilities for Play = no:
Sample 9: 0.28825622775800713
Sample 11: 0.7117437722419929
```

Task 4– Making Predictions for test dataset:

- For each sample in the testing set (`X_test`), the code compares the posterior probabilities calculated for both classes ('yes' and 'no').

- If the posterior probability for class 'yes' is greater than the posterior probability for class 'no', the prediction for the sample is 'yes'. Otherwise, it is 'no'.

- The predicted class label for each sample is appended to the `predictions` list.

```
Predictions for test split examples:
Sample 0: yes
Sample 1: no
```