

## Experiment No 10

**Title: To implement LDA (Linear Discriminant Analysis)**

**Lab Outcome: LO6: To demonstrate the dimensionality reduction techniques.**

### Theory:

Both PCA and LDA are linear transformation techniques. However, PCA is an unsupervised while LDA is a supervised dimensionality reduction technique.

PCA has no concern with the class labels. In simple words, PCA summarizes the feature set without relying on the output. PCA tries to find the directions of the maximum variance in the dataset. In a large feature set, there are many features that are merely duplicate of the other features or have a high correlation with the other features. Such features are basically redundant and can be ignored. The role of PCA is to find such highly correlated or duplicate features and to come up with a new feature set where there is minimum correlation between the features or in other words feature set with maximum variance between the features. Since the variance between the features doesn't depend upon the output, therefore PCA doesn't take the output labels into account.

Unlike PCA, LDA tries to reduce dimensions of the feature set while retaining the information that discriminates output classes. LDA tries to find a decision boundary around each cluster of a class. It then projects the data points to new dimensions in a way that the clusters are as separate from each other as possible and the individual elements within a cluster are as close to the centroid of the cluster as possible. The new dimensions are ranked on the basis of their ability to maximize the distance between the clusters and minimize the distance between the data points within a cluster and their centroids. These new dimensions form the linear discriminants of the feature set.

Let us now see how we can implement LDA using Python's Scikit-Learn.

### Implementing LDA with Scikit-Learn

Like PCA, the Scikit-Learn library contains built-in classes for performing LDA on the dataset. In this section we will apply LDA on the Iris dataset since we used the same dataset for the PCA article and we want to compare results of LDA with PCA.

### Program Code:

```
Data set url; https://archive.ics.uci.edu/ml/datasets/iris
Importing Libraries
import numpy as np
import pandas as pd
Importing the Dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
dataset = pd.read_csv(url, names=names)
```

### Data Preprocessing

Once dataset is loaded into a pandas data frame object, the first step is to divide dataset into features and corresponding labels and then divide the resultant dataset into training and test sets. The following code divides data into labels and feature set:

```
X = dataset.iloc[:, 0:4].values
y = dataset.iloc[:, 4].values
```

The above script assigns the first four columns of the dataset i.e. the feature set to X variable while the values in the fifth column (labels) are assigned to the y variable.

The following code divides data into training and test sets:

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

### Feature Scaling

As was the case with PCA, we need to perform feature scaling for LDA too. Execute the following script to do so:

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.transform(X_test)
```

### Performing LDA

It requires only four lines of code to perform LDA with Scikit-Learn. The LinearDiscriminantAnalysis class of the sklearn.discriminant\_analysis library can be used to Perform LDA in Python. Take a look at the following script:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
```

```
lda = LDA(n_components=1)
```

```
X_train = lda.fit_transform(X_train, y_train)
```

```
X_test = lda.transform(X_test)
```

In the script above the LinearDiscriminantAnalysis class is imported as LDA. Like PCA, we have to pass the value for the n\_components parameter of the LDA, which refers to the number of linear discriminates that we want to retrieve. In this case we set the n\_components to 1, since we first want to check the performance of our classifier with a single linear discriminant. Finally, we execute the fit and transform methods to actually retrieve the linear discriminants.

Notice, in case of LDA, the transform method takes two parameters: the X\_train and the y\_train. However in the case of PCA, the transform method only requires one parameter i.e. X\_train. This reflects the fact that LDA takes the output class labels into account while selecting the linear discriminants, while PCA doesn't depend upon the output labels.

### Training and Making Predictions

Since we want to compare the performance of LDA with one linear discriminant to the performance of PCA with one principal component, we will use the same Random Forest classifier that we used to evaluate performance of PCA-reduced algorithms.

Execute the following code:

```
from sklearn.ensemble import RandomForestClassifier
```

```
classifier = RandomForestClassifier(max_depth=2, random_state=0)
```

```
classifier.fit(X_train, y_train)
```

```
y_pred = classifier.predict(X_test)
```

### Evaluating the Performance

As always, the last step is to evaluate performance of the algorithm with the help of a confusion matrix and find the accuracy of the prediction. Execute the following script:

```
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import accuracy_score
```

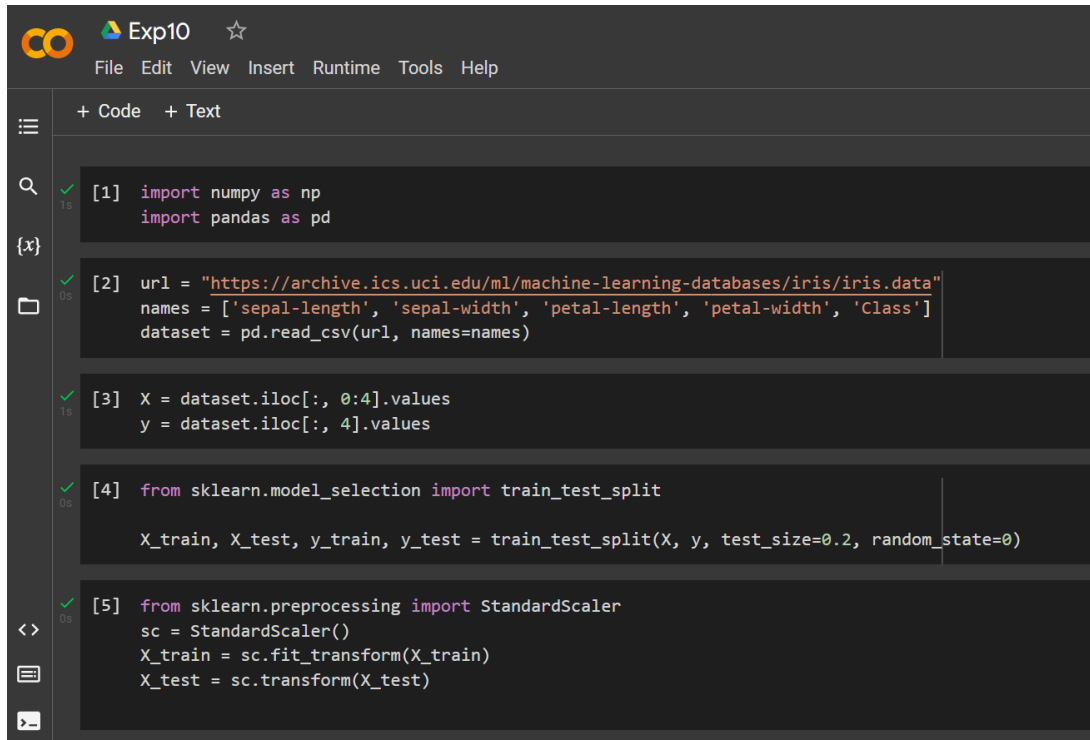
```
cm = confusion_matrix(y_test, y_pred)
```

```
print(cm)
```

```
print('Accuracy' + str(accuracy_score(y_test, y_pred)))
```

You can see that with one linear discriminant, the algorithm achieved an accuracy of 100%, which is greater than the accuracy achieved with one principal component, which was 93.33%.

## Implementation:



```
Exp10
File Edit View Insert Runtime Tools Help

+ Code + Text

[1] import numpy as np
    import pandas as pd

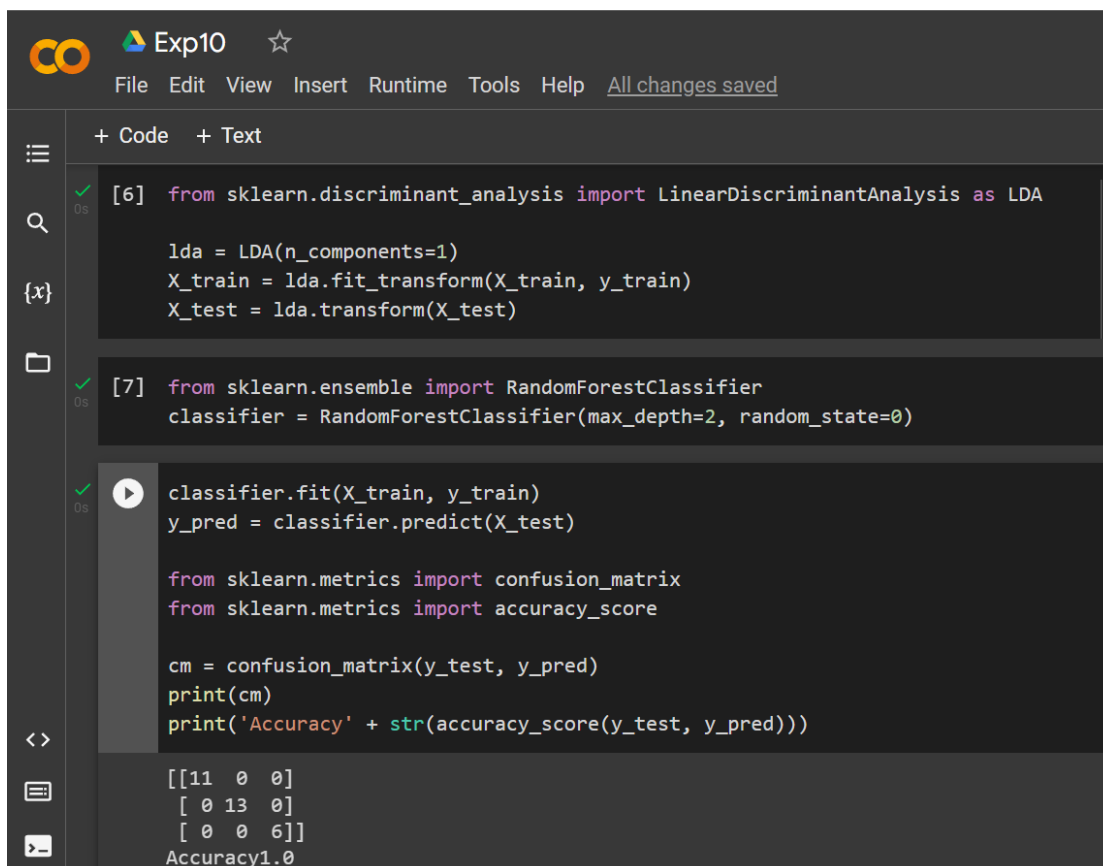
[2] url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
    names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']
    dataset = pd.read_csv(url, names=names)

[3] X = dataset.iloc[:, 0:4].values
    y = dataset.iloc[:, 4].values

[4] from sklearn.model_selection import train_test_split

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

[5] from sklearn.preprocessing import StandardScaler
    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)
```



```
Exp10
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

[6] from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

    lda = LDA(n_components=1)
    X_train = lda.fit_transform(X_train, y_train)
    X_test = lda.transform(X_test)

[7] from sklearn.ensemble import RandomForestClassifier
    classifier = RandomForestClassifier(max_depth=2, random_state=0)

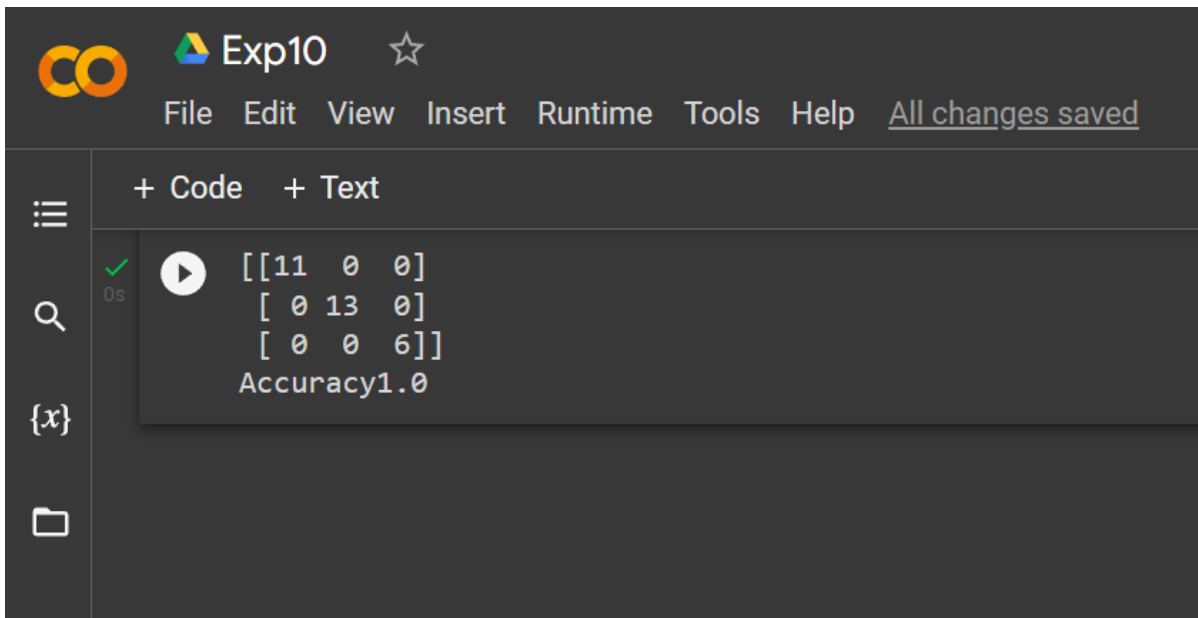
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)

    from sklearn.metrics import confusion_matrix
    from sklearn.metrics import accuracy_score

    cm = confusion_matrix(y_test, y_pred)
    print(cm)
    print('Accuracy' + str(accuracy_score(y_test, y_pred)))

[[11  0  0]
 [ 0 13  0]
 [ 0  0  6]]
Accuracy1.0
```

## Output:



**Conclusion:** Thus, in this practical we studied and performed LDA (Linear Discriminant Analysis) using python programming language.