Dimensionality Reduction with PCA and LDA Using Sklearn

Dimensionality reduction refers to reducing the number of features in a dataset in such a way that the overall performance of the algorithms trained on the dataset is minimally affected. With dimensionality reduction, the training time of statistical algorithms can be significantly reduced, and data can be visualized more easily since it is not easy to visualize datasets in higher dimensions.

There are two main approaches used for dimensionality reduction: Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). In this chapter, you will study both of them.

10.1. Principal Component Analysis

Principal component analysis is an unsupervised dimensionality reduction technique that doesn't depend on the labels of a dataset. Principal component analysis prioritizes features on the basis of their ability to cause maximum variance in the output. The idea behind PCA is to capture those features that contain maximum features about the dataset. The feature that causes the maximum variance in the output is called the first principal component, the feature that causes the second-highest variance is called the second principal component, and so on.

§ Why Use PCA?

The following are the advantages of PCA:

1. Correlated features can be detected and removed using PCA
2. Reduces overfitting because of reduction in the number of features
3. Model training can be expedited.

§ Disadvantages of PCA

There are two major disadvantages of PCA:

1. You need to standardize the data before you apply PCA
2. The independent variable becomes less integrable
3. Some amount of information is lost when you reduce features.

§ Implementing PCA with Python's Sklearn Library

In this section, you will see how to use PCA to select two of the most important features in the Iris dataset using the Sklearn library. The following script imports the required libraries:

Script 1:

```
1. import pandas as pd
2. import numpy as np
3. import seaborn as sns
```

The following script imports the Iris dataset using the Seaborn library and prints the first five rows of the dataset.

Script 2:

```
1. #importing the dataset
2. iris_df = sns.load_dataset("iris" )
3.
4. #print dataset header
5. iris_df.head()
```

Output:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

The above output shows that the dataset contains four features: sepal_length, sepal_width, petal_length, petal_width, and one output label, i.e., species. For PCA, we will only use the feature set.

The following script divides the data into the features and labels sets.

Script 3:

```
#creating feature set
X = iris_df.drop(['species' ], axis=1)


#creating label set
y = iris_df["species" ]

#converting labels to numbers
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
y = le.fit_transform(y)
```

Before we apply PCA on a dataset, we will divide it into the training and test sets, as shown in the following script.

Script 4:

```
#dividing data into 80-20% traning 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=0.20, random_state=0)
```

Finally, both the training and test sets should be scaled before PCA could be applied to them.

Script 5:

```
#applying scaling on training and test data
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform (X_test)
```

To apply PCA via Sklearn, all you have to do is import the PCA class from the Sklearn.decomposition module. Next, to apply PCA to the training set, pass the training set to the fit_tansform() method of the PCA class object. To apply PCA on the test set, pass the test set to the transform() method of the PCA class object. This is shown in the following script.

Script 6:

```
#importing PCA class
from sklearn.decomposition import PCA

#creating object of the PCA class
pca = PCA()

#training PCA model on training data
X_train = pca.fit_transform(X_train)

#making predictions on test data
X_test = pca.transform(X_test)
```

Once you have applied PCA on a dataset, you can use the explained_variance_ratio_ feature to print variance caused by all the features in the dataset. This is shown in the following script:

Script 7:

```
#printing variance ratios
variance_ratios = pca.explained_variance_ratio_
print (variance_ratios)
```

Output:

```
[0.72229951 0.2397406 0.03335483 0.00460506]
```

The output above shows that 72.22 percent of the variance in the dataset is caused by the first principal component, while 23.97 percent of the variance is caused by the second principal component.

Let's now select the two principal components that caused a collective variance of 96.19 percent (72.22% + 23.97% = 96.19%).

To select two principal components, all you have to do is pass 2 as a value to the n_components attribute of the PCA class. The following script selects two principal components from the Iris training and test sets.

Script 8:

```
#use one principal component
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

Let's train a classification model using logistic regression, which predicts the label of the iris plant using the two principal components or features, instead of the original four features.

Script 9:

```
#making predictions using logistic regression
from sklearn.linear_model import LogisticRegression

#training the logistic regression model
lg = LogisticRegression()
lg.fit(X_train, y_train)


# Predicting the Test set results
y_pred = lg.predict(X_test)

#evaluating results

from sklearn.metrics import accuracy_score

print (accuracy_score(y_test, y_pred))
```

Output:

```
0.8666666666666667
```

The output shows that even with two features, the accuracy for correctly predicting the label for the iris plant is 86.66.

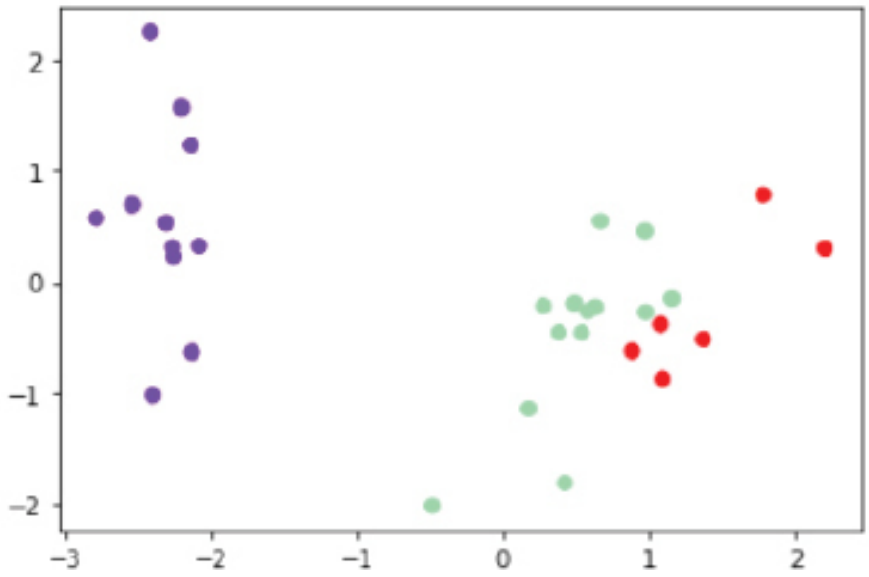
Finally, with two features, you can easily visualize the dataset using the following script.

Script 10:

```
1. from matplotlib import pyplot as plt
2. %matplotlib inline
3.
4. #print actual datapoints
5.
6. plt.scatter(X_test[:,0], X_test[:,1], c= y_test, cmap='rainbow' )
```

Output:

```
<matplotlib.collections.PathCollection at 0x12ea1737610>
```



10.2. Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a supervised dimensionality reduction technique, where a decision boundary is formed around data points belonging to each cluster of a class. The data points are projected to new dimensions in a way that the distance between the data points within a cluster is minimized, while the distance between the clusters is maximized. The new dimensions are ranked w.r.t. their ability to (i) minimize the distance between data points within a cluster, and (ii) maximize the distance between individual clusters.

§ Why Use LDA?

The following are the advantages of LDA:

1. Reduces overfitting because of reduction in the number of features
2. Model training can be expedited.

§ Disadvantages of LDA

There are three major disadvantages of LDA:

1. Not able to detect correlated features
2. Cannot be used with unsupervised or unlabeled data
3. Some amount of information is lost when you reduce features.

§ Implementing LDA with Sklearn Library

Let's see how you can implement LDA using the Sklearn library.

As always, the first step is to import the required libraries.

Script 11:

```
1. import pandas as pd
2. import numpy as np
3. import seaborn as sns
```

You will be using the "banknote.csv" dataset from the *Datasets* folder in the GitHub repository. The following script imports the dataset and displays its first five rows.

Script 12:

```
1. #importing dataset
2. banknote_df = pd.read_csv(r"E:\Hands on Python for Data Science and Machine
      Learning\Datasets\banknote.csv" )
3.
4. #displaying dataset header
5. banknote_df.head()
```

Output:

	variance	skewness	curtosis	entropy	class
0	3.62160	8.6661	-2.8073	-0.44699	0
1	4.54590	8.1674	-2.4586	-1.46210	0
2	3.86600	-2.6383	1.9242	0.10645	0
3	3.45660	9.5228	-4.0112	-3.59440	0
4	0.32924	-4.4552	4.5718	-0.98880	0

Let's divide the dataset into features and labels.

Script 13:

```
1. # dividing data into features and labels
2. X = banknote_df.drop(["class" ], axis = 1)
3. y = banknote_df.filter(["class" ], axis = 1)
```

Finally, the following script divides the data into training and test sets.

Script 14:

```
1. #dividing data into 80-20% training and test sets
2. from sklearn.model_selection import train_test_split
3.
4. X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=0)
```

Like PCA, you need to scale the data before you can apply LDA on it. The data scaling is performed in the following step.

Script 15:

```
1. #applying scaling on training and test data
2. from sklearn.preprocessing import StandardScaler
3. sc = StandardScaler()
4. X_train = sc.fit_transform(X_train)
5. X_test = sc.transform (X_test)
```

To apply LDA via Sklearn, all you have to do is import the LinearDiscriminantAnalysis class from the Sklearn.decomposition module. Next, to apply LDA to the training set, pass the training set to the fit_tansform() method of

the LDA class object. To apply LDA on the test set, pass the test set to the transform() method of the LDA class object. This is shown in the following script.

Script 16:

```
#importing LDA class
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA


#creating object of the LDA class
lda = LDA()

#training PCA model on training data
X_train = lda.fit_transform(X_train, y_train)

#making predictions on test data
X_test = lda.transform(X_test)
```

Like PCA, you can find variance ratios for LDA using the explained_variance_ratio attribute.

Script 17:

```
#printing variance ratios
variance_ratios = lda.explained_variance_ratio_
print (variance_ratios)
```

Output:

```
[1.]
```

The above output shows that even with one component, the maximum variance can be achieved.

Next, we select only a single component from our dataset using LDA. To do so, you have to pass 1 as the attribute value for the n_components attribute of the LDA class, as shown below.

Script 18:

```
#creating object of the LDA class
lda = LDA(n_components = 1)

#training PCA model on training data
X_train = lda.fit_transform(X_train, y_train)

#making predictions on test data
X_test = lda.transform(X_test)
```

Next, we will try to class whether or not a banknote is fake using a single feature. We will use the LogisticRegression algorithm for that. This is shown in the following script.

Script 19:

```
#making predictions using logistic regression
from sklearn.linear_model import LogisticRegression

#training the logistic regression model
lg = LogisticRegression()
lg.fit(X_train, y_train)


# Predicting the Test set results
y_pred = lg.predict(X_test)

#evaluating results

from sklearn.metrics import accuracy_score

print (accuracy_score(y_test, y_pred))
```

Output:

```
0.9890909090909091
```

The output shows that even with a single feature, we are able to correctly predict whether or not a banknote is fake with 98.90 percent accuracy.