2019103555 PRANAVA RAMAN B M S 15/11/2021

LAB - 10 - PCA AND LDA

DATASET:- https://archive-beta.ics.uci.edu/ml/datasets/iris

1. Principal Component Analysis

PCA is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.It reduces the number of variables of a data set, while preserving as much information as possible.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from sklearn.neural_network import MLPClassifier
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

[421] ✓ 0.3s Python

```
from sklearn.datasets import load_iris

iris = load_iris()
X=iris.data
y = iris.target
df=pd.DataFrame(iris['data'],columns=iris['feature_names'])
df.head()
```

[422] ✓ 0.3s Python

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
def model_fit_and_predict(train_x, train_y, test_x, test_y):
    mlp = MLPClassifier(hidden_layer_sizes=(10,5),max_iter=1000)
    start = time.time()
    mlp.fit(train_x,train_y)
    stop = time.time()
    print(f"Training time: {stop - start}s")
    predict = mlp.predict(test_x)
    print("Accuracy: ", accuracy_score(predict, test_y))
    print("Confusion Matrix")
    conf_mat = confusion_matrix(predict,test_y)
    print(conf_mat)
    print("Performance Evaluation")
    print(classification_report(predict,test_y))
```

[423] ✓ 0.3s Python

```
def kmeans_cluster(X, y, plot):
    Kmean = KMeans(n_clusters=3)
    Kmean.fit(X)
    centers = Kmean.cluster_centers_
    plot.figure(figsize=(8,6))
    fig, ax = plot.subplots()
    scatter = ax.scatter(X[:,0],X[:,1],s=50,c=y)
    ax.scatter(centers[:,0],centers[:,1], s=200,marker='s',c='r')
    legend1 = ax.legend(*scatter.legend_elements(),
                        loc="lower left", title="Classes")
    ax.add_artist(legend1)
    plot.xlabel('First principle component')
    plot.ylabel('Second principle component')
```

[424] ✓ 0.3s Python

Before using PCA:-

```python
xtrain,xtest,ytrain,ytest = train_test_split(X,y,test_size=0.35)
```

[425] ✓ 0.3s Python

```python
model_fit_and_predict(xtrain,ytrain,xtest,ytest)
```

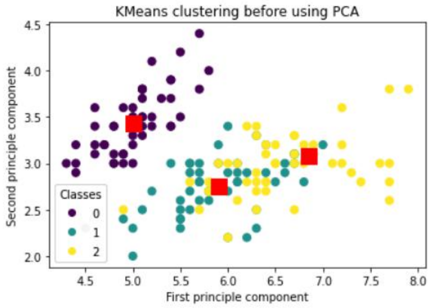
[426] ✓ 0.4s Python

```
Training time: 0.40201640129089355s
Accuracy:  0.9433962264150944
Confusion Matrix
[[20  0  0]
 [ 0 16  2]
 [ 0  1 14]]
Performance Evaluation
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       0.94      0.89      0.91        18
           2       0.88      0.93      0.90        15

    accuracy                           0.94        53
   macro avg       0.94      0.94      0.94        53
weighted avg       0.94      0.94      0.94        53
```

```python
kmeans_cluster(X, y, plt)
plt.title("KMeans clustering before using PCA")
plt.show()
```

[427] ✓ 0.3s Python

```
<Figure size 576x432 with 0 Axes>
```



PCA definition:-

```python
def PCA(X , num_components):

    #Step-1
    X_meaned = X - np.mean(X , axis = 0)

    #Step-2
    cov_mat = np.cov(X_meaned , rowvar = False)

    #Step-3
    eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)

    #Step-4
    sorted_index = np.argsort(eigen_values)[::-1]
    sorted_eigenvalue = eigen_values[sorted_index]
    sorted_eigenvectors = eigen_vectors[:,sorted_index]

    #Step-5
    eigenvector_subset = sorted_eigenvectors[:,0:num_components]

    #Step-6
    X_reduced = np.dot(eigenvector_subset.transpose() , X_meaned.transpose() ).transpose()

    return X_reduced
```

[428] ✓ 0.3s Python

After applying PCA:-

```
X_modified = PCA(X, num_components=2)
```

[429] ✓ 0.3s Python

```
xtrain,xtest,ytrain,ytest = train_test_split(X_modified,y,test_size=0.35)
```

[430] ✓ 0.3s Python

```
model_fit_and_predict(xtrain,ytrain,xtest,ytest)
```

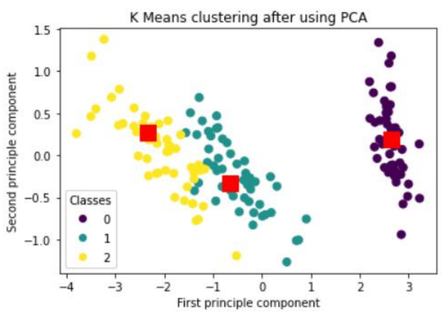
[431] ✓ 0.4s Python

```
Training time: 0.40332937240600586s
Accuracy:  0.9622641509433962
Confusion Matrix
[[20  0  0]
 [ 0 17  1]
 [ 0  1 14]]
Performance Evaluation
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       0.94      0.94      0.94        18
           2       0.93      0.93      0.93        15

    accuracy                           0.96        53
   macro avg       0.96      0.96      0.96        53
weighted avg       0.96      0.96      0.96        53
```

```
kmeans_cluster(X_modified,y, plt)
plt.title("K Means clustering after using PCA")
plt.show()
```

[432] ✓ 0.3s Python

```
<Figure size 576x432 with 0 Axes>
```



After applying PCA, the accuracy has increased marginally while maintaining the same training time, but the Clustering is cleanly done. Before PCA, the clusters were intermingled, but after PCA the segregation is more pronounced.

2. Linear Discriminant Analysis

It is a dimensionality reduction technique. It is used as a pre-processing step in Machine Learning and applications of pattern classification. The goal of LDA is to project the features in higher dimensional space onto a lowerdimensional space in order to avoid the curse of dimensionality and also reduce resources and dimensional costs. LDA is a supervised classification technique that is considered a part of crafting competitive machine learning models. This category of dimensionality reduction is used in areas like image recognition and predictive analysis in marketing

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from sklearn.neural_network import MLPClassifier
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

[352] ✓ 0.3s Python

```python
from sklearn.datasets import load_iris

iris = load_iris()
X=iris.data
y = iris.target
df=pd.DataFrame(iris['data'],columns=iris['feature_names'])
df.head()
```

[353] ✓ 0.3s Python

...

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```python
def model_fit_and_predict(train_x, train_y, test_x, test_y):
    mlp = MLPClassifier(hidden_layer_sizes=(10,8,5),max_iter=1000)
    start = time.time()
    mlp.fit(train_x,train_y)
    stop = time.time()
    print(f"Training time: {stop - start}s")
    predict = mlp.predict(test_x)
    print("Accuracy: ", accuracy_score(predict, test_y))
    print("Confusion Matrix")
    conf_mat = confusion_matrix(predict,test_y)
    print(conf_mat)
    print("Performance Evaluation")
    print(classification_report(predict,test_y))
```

54] ✓ 0.9s Python

```python
def kmeans_cluster(X, y, plot):
    Kmean = KMeans(n_clusters=3)
    Kmean.fit(X)
    centers = Kmean.cluster_centers_
    plot.figure(figsize=(8,6))
    fig, ax = plot.subplots()
    scatter = ax.scatter(X[:,0],X[:,1],s=50,c=y)
    ax.scatter(centers[:,0],centers[:,1], s=200,marker='s',c='r')
    legend1 = ax.legend(*scatter.legend_elements(),
                        loc="lower left", title="Classes")
    ax.add_artist(legend1)
    plot.xlabel('First principle component')
    plot.ylabel('Second principle component')
```

[355] ✓ 0.4s Python

Performance of MLP and K means clustering before LDA

```
xtrain,xtest,ytrain,ytest = train_test_split(X,y,test_size=0.35)
```

[356] ✓ 0.3s Python

```
model_fit_and_predict(xtrain,ytrain,xtest,ytest)
```

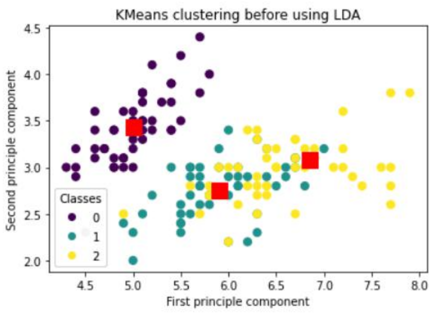
[357] ✓ 0.5s Python

```
Training time: 0.4759352207183838s
Accuracy:  0.9433962264150944
Confusion Matrix
[[20  0  0]
 [ 0 19  0]
 [ 0  3 11]]
Performance Evaluation
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        20
           1       0.86      1.00      0.93        19
           2       1.00      0.79      0.88        14

    accuracy                           0.94        53
   macro avg       0.95      0.93      0.94        53
weighted avg       0.95      0.94      0.94        53
```

```
kmeans_cluster(X,y,plt)
plt.title("KMeans clustering before using LDA")
plt.show()
```

[358] ✓ 0.3s Python

```
<Figure size 576x432 with 0 Axes>
```



Here we can see that the classes of type 2 and 3 are intermingled before using LDA.

```
class LDA:

    def __init__(self, n_components):
        self.n_components = n_components
        self.linear_discriminants = None

    def fit(self, X, y):
        n_features = X.shape[1]
        class_labels = np.unique(y)

        mean_overall = np.mean(X, axis=0)
        SW = np.zeros((n_features, n_features))
        SB = np.zeros((n_features, n_features))
        for c in class_labels:
            X_c = X[y == c]
            mean_c = np.mean(X_c, axis=0)
            SW += (X_c - mean_c).T.dot((X_c - mean_c))

            n_c = X_c.shape[0]
            mean_diff = (mean_c - mean_overall).reshape(n_features, 1)
            SB += n_c * (mean_diff).dot(mean_diff.T)
```

```python
        # Determine SW^-1 * SB
        A = np.linalg.inv(SW).dot(SB)
        # Get eigenvalues and eigenvectors of SW^-1 * SB
        eigenvalues, eigenvectors = np.linalg.eig(A)
        # -> eigenvector v = [:,i] column vector, transpose for easier calculations
        # sort eigenvalues high to low
        eigenvectors = eigenvectors.T
        idxs = np.argsort(abs(eigenvalues))[::-1]
        eigenvalues = eigenvalues[idxs]
        eigenvectors = eigenvectors[idxs]
        # store first n eigenvectors
        self.linear_discriminants = eigenvectors[0:self.n_components]

    def transform(self, X):
        # project data
        return np.dot(X, self.linear_discriminants.T)
```
[359] ✓ 0.3s Python

```python
LDA_object = LDA(n_components=2)
LDA_object.fit(X, y)
X_modified = LDA_object.transform(X)
```
[360] ✓ 0.2s Python

```python
xtrain,xtest,ytrain,ytest = train_test_split(X_modified,y,test_size=0.35)
```
[361] ✓ 0.3s Python

+ Code + Markdown

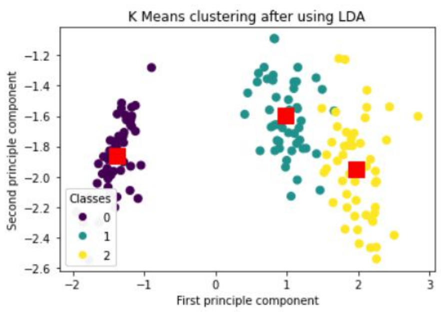
Now after using LDA:-

```python
model_fit_and_predict(xtrain,ytrain,xtest,ytest)
```
[362] ✓ 0.4s Python

```
Training time: 0.4393279552459717s
Accuracy:  0.9811320754716981
Confusion Matrix
[[17  0  0]
 [ 0 18  0]
 [ 0  1 17]]
Performance Evaluation
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        17
           1       0.95      1.00      0.97        18
           2       1.00      0.94      0.97        18

    accuracy                           0.98        53
   macro avg       0.98      0.98      0.98        53
weighted avg       0.98      0.98      0.98        53
```

```python
kmeans_cluster(X_modified,y,plt)
plt.title("K Means clustering after using LDA")
plt.show()
```
[363] ✓ 0.3s Python

```
<Figure size 576x432 with 0 Axes>
```



Similarly for LDA, the accuracy has also increased slightly while segregating the clusters better.