26/4/23

EXPERIMENT - 8A

AIM

To implement the Principal Component Analysis Algorithm in Python

ALGORITHM

* The mean obtained 'm' is subtracted from the target dataset 'x'. Thus, the adjusted dataset is X − m.
* The covariance 'c' is obtained, eigen values and eigen vectors of the covariance matrix are calculated.
* Eigenvector of highest eigenvalue is PC of the dataset and the feature vector is formed.
* PCA transform is $y = A \times (x - m)$ where, $A \rightarrow$ transpose of the feature vector.

THEORY

PCA is used to transform a given set of measurements to a new set of features so that the features exhibit high information packing properties. This leads to a reduced and compact set of features. PCA transform is, $y = A(x - m)$ and the original vector can be reconstructed as, $x = A^T y + m$

OUTPUT

```
Input Data :

[[2, 6], [1, 7]]

PCA Transformed Matrix :

[[ 1.11022302e-16 -1.11022302e-16]
 [-7.07106781e-01  7.07106781e-01]]
```

CODE

```
import numpy as np

def computeMeanVector ( data , features ) :
   feature_values = [ [] for i in range ( features ) ]
   for row in range ( len ( data ) ) :
      for feat in range ( features ) :
         feature_values [ feat ].append ( data [ row ] [ feat ] )
   meanVec = np.mean ( feature_values , axis = 1 )
   return meanVec

def computeAdjustedData ( data , meanVec , samples ) :
   adjustedData = [ [] for i in range ( samples ) ]
   for row in range ( len ( data ) ) :
      for val in range ( len ( data [ row ] ) ) :
         adjustedData [ row ].append ( data [ row ] [ val ] - meanVec [ val ] )
   return adjustedData

def computeCovarianceVectors ( adjustedData ) :
   covVectors = []
   adjustedData = np.array ( adjustedData )
   for vec in adjustedData :
      if len ( vec.shape ) == 1 :
         mul = vec.reshape ( 1 , len ( vec ) )
      else :
         mul = vec.reshape ( len ( vec [ 0 ] ) , len ( vec ) )
      covVectors.append ( np.dot ( mul.T , mul ) )
   return covVectors

def computeCovarianceMatrix ( covVectors , samples , features ) :
   covMatrix = np.zeros ( ( samples , features ) )
   for vec in range ( len ( covVectors ) ) :
      for i in range ( samples ) :
         for j in range ( features ) :
            covMatrix [ i ] [ j ] += covVectors [ vec ] [ i ] [ j ]
   return covMatrix

def computeEigenVectorMatrix ( covMatrix ) :
   eigenValues , eigenVectors = np.linalg.eig ( covMatrix )
   idx = eigenValues.argsort () [ : : -1 ]
   eigenvalues = eigenValues [ idx ]
   eigenVectorMatrix = eigenVectors [ : , idx ]
   return eigenVectorMatrix

def computePCAMatrix ( adjustedData , eigenVectorMatrix ) :
   pcaMatrix = np.dot ( eigenVectorMatrix , adjustedData )
   return pcaMatrix

def PCATransform ( numFeatures , numSamples , inputData ) :
   inputData = np.array ( inputData )
   inputData = inputData.reshape ( numFeatures , numSamples )
   meanVec = computeMeanVector ( inputData , numFeatures )
```

```
    adjustedData = computeAdjustedData ( inputData , meanVec , numSamples )
    covVectors = computeCovarianceVectors ( adjustedData )
    covMatrix = computeCovarianceMatrix ( covVectors , numSamples , numFeatures )
    eigenVectorMatrix = computeEigenVectorMatrix ( covMatrix )
    pcaMatrix = computePCAMatrix ( adjustedData , eigenVectorMatrix )
    return pcaMatrix

numFeatures = 2
numSamples = 2
inputData = [ [ 2 , 6 ] , [ 1 , 7 ] ]
pcaMatrix = PCATransform ( numFeatures , numSamples , inputData )
print ( "\n Input Data :\n\n" , inputData , "\n\n PCA Transformed Matrix :\n\n" , pcaMatrix )
```

RESULT

Hence, the PCA algorithm has been implemented successfully

26/4/23

EXPERIMENT - 8B

AIM

To implement the Linear Discriminant Analysis algorithm in Python.

ALGORITHM

* Calculate the mean and standard deviation of each feature.
* Within-class and between-class scatter matrices are calculated.
* These matrices are then used to calculate the eigenvectors & eigenvalues.
* LDA chooses the k eigenvectors with the largest eigenvalues to form a transformation matrix, mapping into a new space.
* LDA can then be used for dimensionality reduction.

THEORY

LDA is a feature reduction technique that projects higher dimension data to a line. It can also be used to predict the class label of new data points.

$$S_B = \sum_{i=1}^{c} N_i (\mu_i - \mu_t)(\mu_i - \mu_t)^T$$

$$S_W = S_1 + S_2 + \dots + S_c$$

OUTPUT

```
Data read from dataset.csv :

   Feature 1  Feature 2 Target
0          4          1     C1
1          2          4     C1
2          9         10     C2
3          2          3     C1
4          3          6     C1
5          6          8     C2
6          9          5     C2
7          4          4     C1
8          8          7     C2
9         10          8     C2

Linear Discriminants :  [[0.91955932 0.39295122]]

Shape of X :  (10, 2)

Shape of transformed X :  (10, 1)
```

CODE

```
import numpy as np
import pandas as pd

def LDA_fit ( 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 clas in class_labels :
        X_cls = X [ y == clas ]
        mean_cls = np.mean ( X_cls , axis = 0 )
        SW += ( ( X_cls - mean_cls ).T).dot ( ( X_cls - mean_cls ) ).astype ( 'float64' )
        n_cls = X_cls.shape [ 0 ]
        mean_diff = ( mean_cls - mean_overall ).reshape ( n_features , 1 )
        SB += n_cls * ( ( mean_diff ).dot ( mean_diff.T ) ).astype ( 'float64' )
    A = np.linalg.inv ( SW ).dot ( SB )
    eigenvalues , eigenvectors = np.linalg.eig ( A )
    eigenvectors = eigenvectors.T
    idxs = np.argsort ( abs ( eigenvalues ) ) [ : : -1 ]
    eigenvalues = eigenvalues [ idxs ]
    eigenvectors = eigenvectors [ idxs ]
    linear_discriminants = eigenvectors [ 0 : n_components ]
    return linear_discriminants

data = pd.read_csv ( "lda_dataset.csv" )
print ( "\n Data read from dataset.csv :\n\n" , data )

arr = np.array ( data )
X = arr [ : , : -1 ]
y = arr [ : , -1 ]

n_components , linear_discriminants = 1 , None
linear_discriminants = LDA_fit ( X , y )
X_projected = np.dot ( X , linear_discriminants.T )
print ( "\n Linear Discriminants : " , linear_discriminants )
print ( "\n Shape of X : " , X.shape )
print ( "\n Shape of transformed X : " , X_projected.shape )
```

RESULT

Hence, the LDA algorithm has been implemented successfully.