**LAB EXERCISE – 6**

**Principal Component Analysis**

**Aim of the Experiment**

To write python program for finding principal component analysis (PCA) for the given problem given in Chapter 2 and to a randomly generated dataset.

**Reference to Textbook and Explanation**

Chapter 2 and Appendix 2 for details about principal component analysis (PCA).

Consider the dataset



Apply PCA and Inverse Transform and Prove that they are similar.

In listing 2, the methods of computing mean matrix, covariance matrix, eigen values and eigen vectors computed are illustrated.

In listing 3, Iris dataset is taken and PCA is applied. It can be verified that after applying PCA, the cross score remains unchanged. That means, all the features of Iris are not important.

**Listing 1**

import numpy as np

from sklearn import decomposition

X = np.array([[2,6],[1,7]])

print("Orginal Matrx X and its Shape")

print(X)

print("Original Shape:",X.shape)

print("Original matrix\n\n")

# Apply Transform for X

pca = decomposition.PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

print("Transformed Matrix and its Shape")

print(X\_pca)

print("Transformed Shape:",X\_pca.shape)

print("Transformed Matrix\n\n")

# Apply Inverse Transform

print("After Inverse Transform")

X\_new=pca.inverse\_transform(X\_pca)

print(X\_new)

print("After Inverse Transform\n\n\n")

# Explain variance

print('Explained variance\n')

print(pca.explained\_variance\_ratio\_)

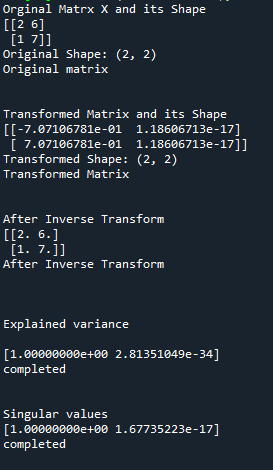
print('completed\n\n')

print('Singular values')

print(pca.singular\_values\_)

print('completed\n\n')

**Output**

****

**Listing 2**

**This explains how the eigen values and eigen vectors are calculated.**

import numpy as np

from numpy.linalg import eig

# define a matrix

X = np.array([[3, 6], [4,7]])

print("Orginal Matrx X and its Shape")

print(X)

print("Original matrix Shape")

print("Original Shape:",X.shape)

# calculate the mean of each column

M = np.mean(X.T, axis=1)

print("\nMean matrix")

print(M)

# center columns by subtracting column means

C = X - M

print("\nCentre the matrix")

print(C)

# calculate covariance matrix of centered matrix

V = np.cov(C.T)

print("\nCovariance of the matrix\n")

print(V)

# eigendecomposition of covariance matrix

values, vectors = eig(V)

print('\n Eigen vectors')

print(vectors)

print('\n Eigen values')

print(values)

**Output**

**Text

Description automatically generated**

**Listing 3**

import pandas as pd

import numpy as np

from sklearn.model\_selection import KFold

from sklearn.model\_selection import cross\_val\_score

from sklearn.neighbors import KNeighborsClassifier

from sklearn import decomposition

import seaborn as sns

df = pd.read\_csv("iris.csv")

print(df.head(10))

array = df.values

X = array[:,0:4]

y = array[:,4]

kfold = KFold(n\_splits=10)

model = KNeighborsClassifier(n\_neighbors=3)

score = cross\_val\_score(model,X,y,cv=10)

print('\n\n')

print("Cross score before applying PCA\n")

print(score.mean())

print("Apply PCA now...")

pca = decomposition.PCA(n\_components=1)

X\_pca = pca.fit\_transform(X)

core = cross\_val\_score(model,X\_pca,y,cv=10)

print('\n\n')

print("Cross score After applying PCA\n")

print(score.mean())

**Output**

**A screenshot of a computer

Description automatically generated**

*If more attributes are removed, it leads to more information loss. So optimal replacement of features is required. In the above case, retaining only one component does not result in reduction of scores.*