Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Continuous Assessment for Laboratory / Assignment sessions

Academic Year 2022-23

Name: Ayush Jain SAP ID: 60004200182

Course: Machine Learning Laboratory Course Code: **DJ19CEEL6021**

Year: **T.Y. B.Tech.** Sem: **VII** Batch: B 3

Department: Computer Engineering

Performance Indicators (Any no. of Indicators) (Maximum 5 marks per indicator)	1	2	3	4	5	6	7	8	9	10	11	Σ	Avg	A1	A2	Σ	Avg
Course Outcome	2, 4	2, 4	2, 4	2, 4	2, 4	3	2, 4	2, 4	5								
1. Knowledge (Factual/Conceptual/Procedural/ Metacognitive)	4	5	5	5													
2. Describe (Factual/Conceptual/Procedural/ Metacognitive)	4	4	4	5													
3. Demonstration (Factual/Conceptual/Procedural/ Metacognitive)	5	5	5	5													
4. Strategy (Analyse & / or Evaluate) (Factual/Conceptual/ Procedural/Metacognitive)	4	4	4	4													
5. Interpret/ Develop (Factual/Conceptual/ Procedural/Metacognitive)	–	–	–	–													
6. Attitude towards learning (receiving, attending, responding, valuing, organizing, characterization by value)	4	4	4	4													
7. Non-verbal communication skills/ Behavior or Behavioural skills (motor skills, hand-eye coordination, gross body movements, finely coordinated body movements speech behaviours)	–	–	–	–													
Total	21	22	22	23													
Signature of the faculty member																	

Outstanding (5), Excellent (4), Good (3), Fair (2), Needs Improvement (1)

Laboratory marks Σ Avg. =	Assignment marks Σ Avg. =	Total Term-work (25) =
Laboratory Scaled to (15) =	Assignment Scaled to (10) =	Sign of the Student:

Signature of the Faculty member:
Name of the Faculty member:

Signature of Head of the Department
Date:

Machine Learning

Experiment 4

Ayush Jain 60004200132 B3

Aim : To implement PCA in python without using any inbuilt function and without inbuilt function.

Theory:

What is Principal Component Analysis (PCA)?

In short, PCA is a dimensionality reduction technique that transforms a set of features in a dataset into a smaller number of features called principal components while at the same time trying to retain as much information in the original dataset as possible:

The key aim of PCA is to reduce the number of variables of a data set, while preserving as much information as possible.

Instead of explaining the theory of how PCA works in this article, I shall leave the explanation to the following video, which provides an excellent walkthrough of how PCA works.

So what are the advantages of using PCA on your dataset? Here are several reasons why you want to use PCA:

- Removes correlated features. PCA will help you remove all the features that are correlated, a phenomenon known as *multi-collinearity*. Finding features that are correlated is time consuming, especially if the number of features is large.
- Improves machine learning algorithm performance. With the number of features reduced with PCA, the time taken to train your model is now significantly reduced.
- Reduce overfitting. By removing the unnecessary features in your dataset, PCA helps to overcome overfitting.

On the other hand, PCA has its disadvantages:

- Independent variables are now less interpretable. PCA reduces your features into smaller number of components. Each component is now a

linear combination of your original features, which makes it less readable and interpretable.
- Information loss. Data loss may occur if you do not exercise care in choosing the right number of components.
- Feature scaling. Because PCA is a *variance maximizing* exercise, PCA requires features to be scaled prior to processing.

PCA is useful in cases where you have a large number of features in your dataset.

Steps for PCA algorithm

1. **Getting the dataset**
   Firstly, we need to take the input dataset and divide it into two subparts X and Y, where X is the training set, and Y is the validation set.
2. **Representing data into a structure**
   Now we will represent our dataset into a structure. Such as we will represent the two-dimensional matrix of independent variable X. Here each row corresponds to the data items, and the column corresponds to the Features. The number of columns is the dimensions of the dataset.
3. **Standardizing the data**
   In this step, we will standardize our dataset. Such as in a particular column, the features with high variance are more important compared to the features with lower variance.
   If the importance of features is independent of the variance of the feature, then we will divide each data item in a column with the standard deviation of the column. Here we will name the matrix as Z.
4. **Calculating the Covariance of Z**
   To calculate the covariance of Z, we will take the matrix Z, and will transpose it. After transpose, we will multiply it by Z. The output matrix will be the Covariance matrix of Z.
5. **Calculating the Eigen Values and Eigen Vectors**
   Now we need to calculate the eigenvalues and eigenvectors for the resultant covariance matrix Z. Eigenvectors or the covariance matrix are the directions of the axes with high information. And the coefficients of these eigenvectors are defined as the eigenvalues.
6. **Sorting the Eigen Vectors**
   In this step, we will take all the eigenvalues and will sort them in decreasing order, which means from largest to smallest. And

simultaneously sort the eigenvectors accordingly in matrix P of eigenvalues. The resultant matrix will be named as P*.

7. **Calculating the new features Or Principal Components**
   Here we will calculate the new features. To do this, we will multiply the P* matrix to the Z. In the resultant matrix Z*, each observation is the linear combination of original features. Each column of the Z* matrix is independent of each other.
8. **Remove less or unimportant features from the new dataset.**
   The new feature set has occurred, so we will decide here what to keep and what to remove. It means, we will only keep the relevant or important features in the new dataset, and unimportant features will be removed out.

Code without inbuilt function:

```
import numpy as np

def PCA(X, num_components):

    X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)

    cov_matrix = np.cov(X.T)

    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)


    indices = np.argsort(eigenvalues)[::-1][:num_components]
    top_eigenvectors = eigenvectors[:, indices]

    transformed_data = np.dot(X, top_eigenvectors)

    return transformed_data

X = np.array([
    [7, 2, 3],
    [4, 5, 6],
    [7, 8, 9],
    [10, 11, 12]
])

X_pca = PCA(X, 2)
print(X_pca)
```

Output:

```
[[-1.63655433  0.96004684]
 [-1.26109477 -0.89979996]
 [ 0.54551811 -0.32001561]
 [ 2.35213099  0.25976874]]
```

Code with inbuilt function:

```
import numpy as np
from sklearn.decomposition import PCA

data = np.array([[ 0.6, -0.2,  0.4,  0.4,  0.3],
                 [-0.2,  0.7, -0.5, -0.1, -0.3],
                 [ 0.3, -0.5,  0.9,  0.3,  0.2],
                 [ 0.2, -0.1,  0.3,  0.8,  0.7],
                 [ 0.5, -0.3,  0.2,  0.7,  0.6],
                 [-0.1,  0.8, -0.7, -0.3, -0.5],
                 [ 0.2, -0.5,  0.3,  0.6,  0.8],
                 [ 0.4, -0.1,  0.2,  0.7,  0.6],
                 [-0.2,  0.6, -0.4, -0.2, -0.4],
                 [ 0.1, -0.4,  0.6,  0.2,  0.2]])
n_train = 8
X_train = data[:n_train, :] # replace n_train with the number of training samples
X_val = data[n_train:, :] # replace n_train with the number of validation sample
s

X_train_std = (X_train - X_train.mean(axis=0)) / X_train.std(axis=0)

cov_mat = np.cov(X_train_std.T)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:, i]) for i in range(len(eig_vals))]
eig_pairs.sort(reverse=True, key=lambda x: x[0])
sorted_eig_vecs = np.stack([eig_pairs[i][1] for i in range(len(eig_vals))], axis=1
)
X_train_pca = X_train_std.dot(sorted_eig_vecs)

n_components =2
X_train_pca = X_train_pca[:, :n_components]
X_val_std = (X_val - X_train.mean(axis=0)) / X_train.std(axis=0)
X_val_pca = X_val_std.dot(sorted_eig_vecs[:, :n_components])
X_val_pca
```

Output:

```
array([[-3.24368083, -0.19499751],
       [ 0.24929579, -0.91519733]])
```

Conclusion: We successfully implemented PCA in python without using any inbuilt function and without inbuilt function.