# PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

**APRIL 2022: IN SEMESTER ASSESSMENT (ISA) B.TECH. IV SEMESTER**

**UE20MA251- LINEAR ALGEBRA**

# Project / Seminar

Session: Jan-May 2022

**Branch : Computer Science and Engineering**

**Semester & Section : Semester IV Section B**

| Sl No. | Name of the Student | SRN | Marks Allotted (Out of 10) |
|---|---|---|---|
| 1. | CHIRANTH R | PES1UG20CS116 | |
| 2. | CHITTOOR SAI SUSHAEN | PES1UG20CS117 | |
| 3. | B.N.SHIVADATTA KUMAR | PES1UG20CS100 | |
| 4. | ARUN KUMAR RATH | PES1UG20CS076 | |

Name of the Course Instructor :**Prof. Chaitra GP**

Signature of the Course Instructor

(with Date) : _____

TITLE : PRINCIPAL COMPONENT ANALYSIS

## Literature review:

We have referred many research papers cited on PCA,and tutorials from youtube,learnings from class.The main paper referred was https://tinyurl.com/y2k5k7kt .

## Theoretical background:

Working with a image of larger dimsensions is cumbersome and takes too much computations. With the help of a technique called PCA we can reduce dimensionality,keeping much of the original data . PCA is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables to a set of uncorrelated variables. We will consider only those variables which have the maximum variablity . PCA can also be used for denoising and data compression.PCA considers eigenvectors as its orthogonal counter-parts which are ordered based on its eigen-values in increasing order of significance.

We can summarize the goals of PCA as follows:

- Extract the most important information from the data table.
- Compress the size of the data set by keeping only this important information
- Compress the data, by reducing the number of dimensions, without much loss of information
- This technique used in image compression

## Application of LA techniques

The data we have taken is:

| Variable X | Variable Y | Deviation from Mean for X | Deviation from Mean for Y |
|---|---|---|---|
| 2.5 | 2.4 | 0.69 | 0.49 |
| 0.5 | 0.7 | -1.31 | -1.21 |
| 2.2 | 2.9 | 0.39 | 0.99 |
| 1.9 | 2.2 | 0.09 | 0.29 |
| 3.1 | 3.0 | 1.29 | 1.09 |
| 2.3 | 2.7 | 0.49 | 0.79 |
| 2 | 1.6 | 0.19 | -0.31 |
| 1 | 1.1 | -0.81 | -0.81 |
| 1.5 | 1.6 | -0.31 | -0.31 |
| 1.1 | 0.9 | -.71 | -1.01 |

- The project is based on computing a co-variance matrix which is found by the formula

  co-var matrix = $1/n-1 * (BB^T)$

  where n is no of observations, B is the original data subtracted with its mean represented as a vector

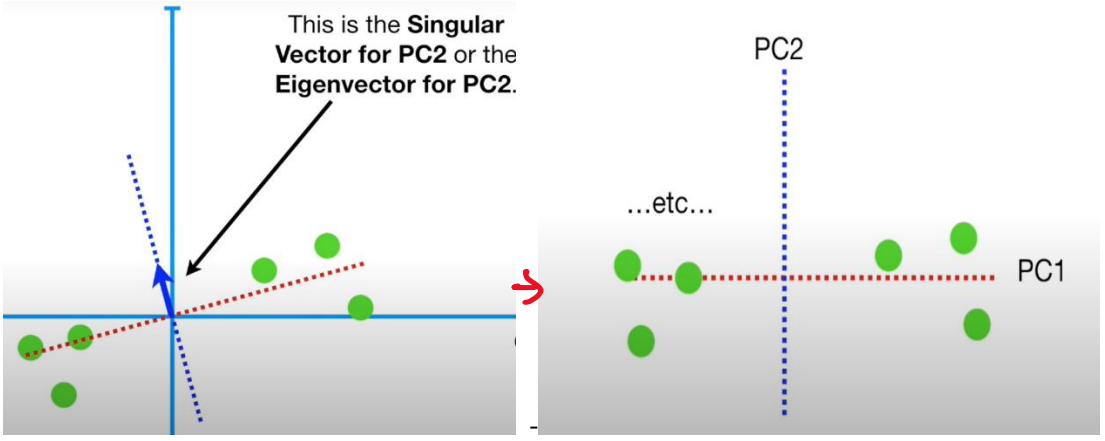
$$\text{cov} = \begin{bmatrix} 0.616555556 & 0.615444444 \\ 0.615444444 & 0.716555558 \end{bmatrix}$$

- Then we do eigen-decomposition of the covariance matrix to obtain the eigen-values and its respective eigen-vectors

$$\text{Eigenvalue} = \begin{bmatrix} 0.0490833989 \\ 1.28402771 \end{bmatrix}$$

$$\text{Eigenvector} = \begin{bmatrix} -0.735178656 & -0.677873399 \\ 0.677873399 & -0.735178656 \end{bmatrix}$$

- As we know that, the eigen-vectors should be orthogoanal to each other ,so we can represent the data represented in the n-dimesnional axis , in the newly found eigen dimensions.




Standard x-y axes

In newly found eigen axes

- We must order our eigen-vectors in order of significance. In fact the eigenvector wiht the highest eigenvalue is principle component of dataset.
  The eigenvector with the largest eigenvalue was the one that pointed down the middle of the data.

- In general, once eigenvectors are found from the covariance matrix, the next step is to order them by eigenvalue, highest to lowest. This gives you the components in order of significance. After that components with lesser significance were ignored.

- If some the components were leaves out, the final data set will have fewer dimensions than the original.

  Given with the example set of data, and the fact that here will be 2 eigenvectors, that means there will be two choices. A feature vector can be form with either of the eigenvectors-

$$\begin{bmatrix} 0.677873399 & -0.735178656 \\ -0.735178656 & -0.677873399 \end{bmatrix}$$

  or, the smaller one can be chosen to leave out, less significant component and only have a single column vector-

$$\begin{bmatrix} 0.677873399 \\ -0.735178656 \end{bmatrix}$$

- After we have found out our feature vector (which is eigen-vectors in order) ,we can derive the final data-set

  Final Data = Row Feature Vector × Row Data Adjust

  where row data adjust was the data after subtracting with mean

| Transformed Data (Single eigenvector) for Variable X |
|---|
| -0.8279 70186 |
| 1.77758033 |
| -0.992197494 |
| -0.274210416 |
| -1.67580142 |
| -0.912949103 |
| 0.0991094375 |
| 1.14457216 |
| 0.438046137 |
| 1.22382056 |

- So from above we can see that we have tried to reduce a 2-d dataset to a single dimension which covers most of the data.

Code:

1) Reading the MNIST dats set which is a 28x28 pixel images of digits

| | label | pixel0 | pixel1 | pixel2 | pixel3 | pixel4 | pixel5 | pixel6 | pixel7 | pixel8 | pixel9 | pixel10 | pixel11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

2) Standardizing the data

```
from sklearn.preprocessing import StandardScaler
X = train.values
X_std = StandardScaler().fit_transform(X)
```

3) Finding covariance matrix, and its eigenvalues and eigenvectors

```
# Calculating Eigenvectors and eigenvalues of Cov matirx
mean_vec = np.mean(X_std, axis=0)
cov_mat = np.cov(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
# Create a list of (eigenvalue, eigenvector) tuples
eig_pairs = [ (np.abs(eig_vals[i]),eig_vecs[:,i]) for i in range(len(eig_vals))]
```

4) Sorting the eigenvectors in order of significance

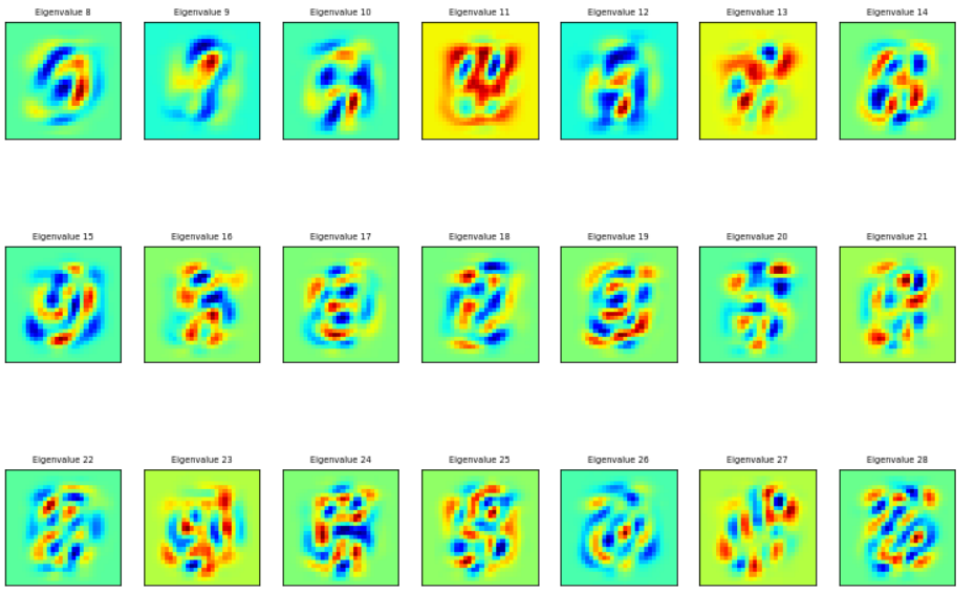
```
# Sort the eigenvalue, eigenvector pair from high to low
eig_pairs.sort(key = lambda x: x[0], reverse= True)
```

5) We can use sklearn's PCA method which seeks to obtain the optimal directions (or eigenvectors) that captures the most variance ( spreads out the data points the most )

```
# Invoke SKlearn's PCA method
n_components = 30
pca = PCA(n_components=n_components).fit(train.values)

eigenvalues = pca.components_.reshape(n_components, 28, 28)

# Extracting the PCA components ( eignevalues )
#eigenvalues = pca.components_.reshape(n_components, 28, 28)
eigenvalues = pca.components_
```

## 6) Visualizing the heatplots of eigenvalues

```
n_row = 4
n_col = 7

# Plot the first 8 eignenvalues
plt.figure(figsize=(13,12))
for i in list(range(n_row * n_col)):
    offset =0
    plt.subplot(n_row, n_col, i + 1)
    plt.imshow(eigenvalues[i].reshape(28,28), cmap='jet')
    title_text = 'Eigenvalue ' + str(i + 1)
    plt.title(title_text, size=6.5)
    plt.xticks(())
    plt.yticks(())
plt.show()
```




When one compares the first component "Eigenvalue 1" to the 28th component "Eigenvalue 28", it is obvious that more complicated directions or components are being generated in the search to maximise variance in the new feature subspace.

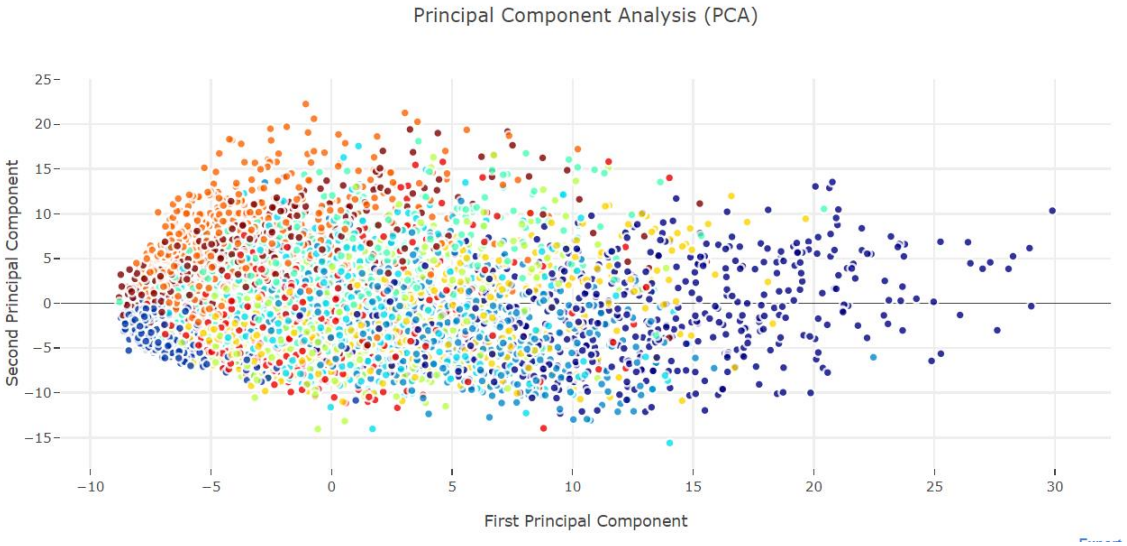
7) Plotting some of the numbers

```
# plot some of the numbers
plt.figure(figsize=(14,12))
for digit_num in range(0,70):
    plt.subplot(7,10,digit_num+1)
    grid_data = train.iloc[digit_num].as_matrix().reshape(28,28)  # reshape from 1d to 2d pixel array
    plt.imshow(grid_data, interpolation = "none", cmap = "afmhot")
    plt.xticks([])
    plt.yticks([])
plt.tight_layout()
```

8) Visualization of the new transformed data with respect to newly found eigenvectors(PCA1 AND PCA2) using plotly

```
trace0 = go.Scatter(
    x = X_5d[:,0],
    y = X_5d[:,1],
#     name = Target,
#     hoveron = Target,
    mode = 'markers',
    text = Target,
    showlegend = False,
    marker = dict(
        size = 8,
        color = Target,
        colorscale ='Jet',
        showscale = False,
        line = dict(
            width = 2,
            color = 'rgb(255, 255, 255)'
        ),
        opacity = 0.8
    )
)
data = [trace0]
```

```
layout = go.Layout(
    title= 'Principal Component Analysis (PCA)',
    hovermode= 'closest',
    xaxis= dict(
         title= 'First Principal Component',
        ticklen= 5,
        zeroline= False,
        gridwidth= 2,
    ),
    yaxis=dict(
        title= 'Second Principal Component',
        ticklen= 5,
        gridwidth= 2,
    ),
    showlegend= True
)


fig = dict(data=data, layout=layout)
py.iplot(fig, filename='styled-scatter')
```



9) We know that PCA is a unsupervised learning,so we cant separate our data points in a new feature space.So we can apply a clustering algorithm on our new PCA projection data and hopefully arrive at distinct clusters which would tell us something about the underlying class separation in the data

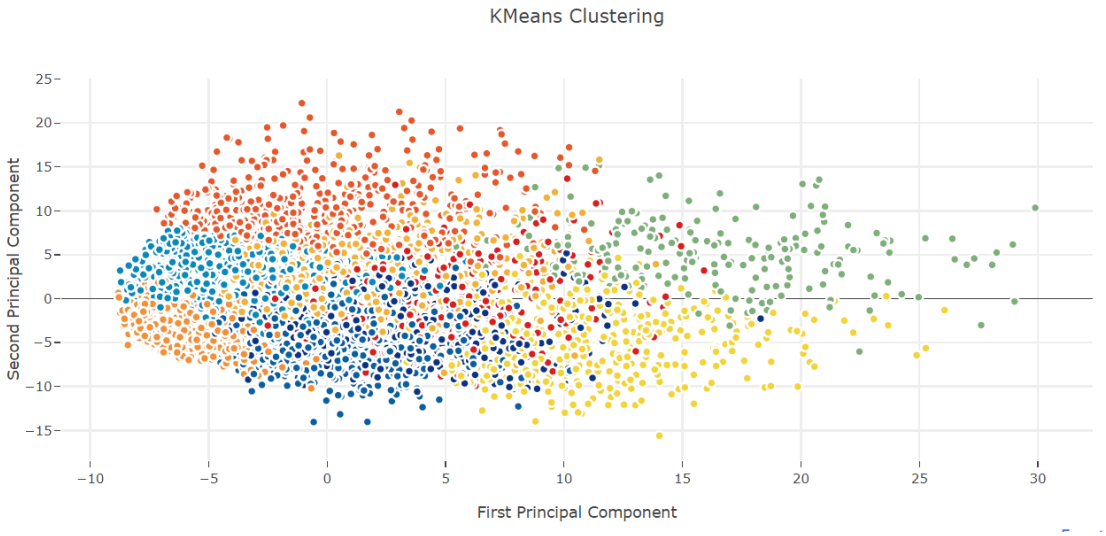
## Using sklearns kmeans clustering

```
from sklearn.cluster import KMeans # KMeans clustering
# Set a KMeans clustering with 9 components ( 9 chosen sneakily ;) as hopefully we get back our 9 class labels)
kmeans = KMeans(n_clusters=9)
# Compute cluster centers and predict cluster indices
X_clustered = kmeans.fit_predict(X_5d)
```

## Plotting the new clusters in the PCA spaces

```
trace_Kmeans = go.Scatter(x=X_5d[:, 0], y= X_5d[:, 1], mode="markers",
                    showlegend=False,
                    marker=dict(
                            size=8,
                            color = X_clustered,
                            colorscale = 'Portland',
                            showscale=False,
                            line = dict(
            width = 2,
            color = 'rgb(255, 255, 255)'
        )
                    ))
```

```
layout = go.Layout(
    title= 'KMeans Clustering',
    hovermode= 'closest',
    xaxis= dict(
         title= 'First Principal Component',
        ticklen= 5,
        zeroline= False,
        gridwidth= 2,
    ),
    yaxis=dict(
        title= 'Second Principal Component',
        ticklen= 5,
        gridwidth= 2,
    ),
    showlegend= True
)

data = [trace_Kmeans]
fig1 = dict(data=data, layout= layout)
# fig1.append_trace(contour_list)
py.iplot(fig1, filename="svm")
```




## Results and conclusions:

Basically we have transformed our data so that is expressed in terms of the patterns between them, where the patterns are the lines that most closely describe the relationships between the data.

And a 28x28 pixels image data was compressed to be shown in a graph of only two-dimensions in the PCA's newly found axes which captured most of the data. So while working in future projects matrix computations,image recognition techniques could be done easily without having to worry much about loss of data.

## Future work:

To explore PCA in detail and their insight contributions in the image recogniton field .

## References:

1) https://www.researchgate.net/publication/316652806_Principal_Component_Analysis
2) https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202
3) https://www.sciencedirect.com/science/article/pii/S1110016820304543