PRINCIPAL COMPONENT ANALYSIS

PCA is one of the old and most important dimensionality reduction technique which is based on idea called variance maximisation based Reduction

As a part of our Journey,I did my second task as Dimensionality Reduction of 784d data into 2d by using PCA on an interesting data set called SIGNED ALPHABETS MNIST and LINK for data set is
[https://www.kaggle.com/datamunge/sign-language-mnist (https://www.kaggle.com/datamunge/sign-language-mnist)](https://www.kaggle.com/datamunge/sign-language-mnist)

In [0]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [4]:
```
data = pd.read_csv("/content/drive/My Drive/Data_Scientist/pca-tsne/sign_mnist_train.csv")
data.head()
```

Out[4]:

	label	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	pi›
0	3	107	118	127	134	139	143	146	150	153	156	158	160	163	165	
1	6	155	157	156	156	156	157	156	158	158	157	158	156	154	154	
2	2	187	188	188	187	187	186	187	188	187	186	185	185	185	184	
3	2	211	211	212	212	211	210	211	210	210	211	209	207	208	207	
4	13	164	167	170	172	176	179	180	184	185	186	188	189	189	190	

5 rows × 785 columns

In [5]:
```
data.shape
```

Out[5]: (27455, 785)

In [6]:
```
labels = data['label']
data = data.drop('label',axis= 1)
data.head()
```

Out[6]:

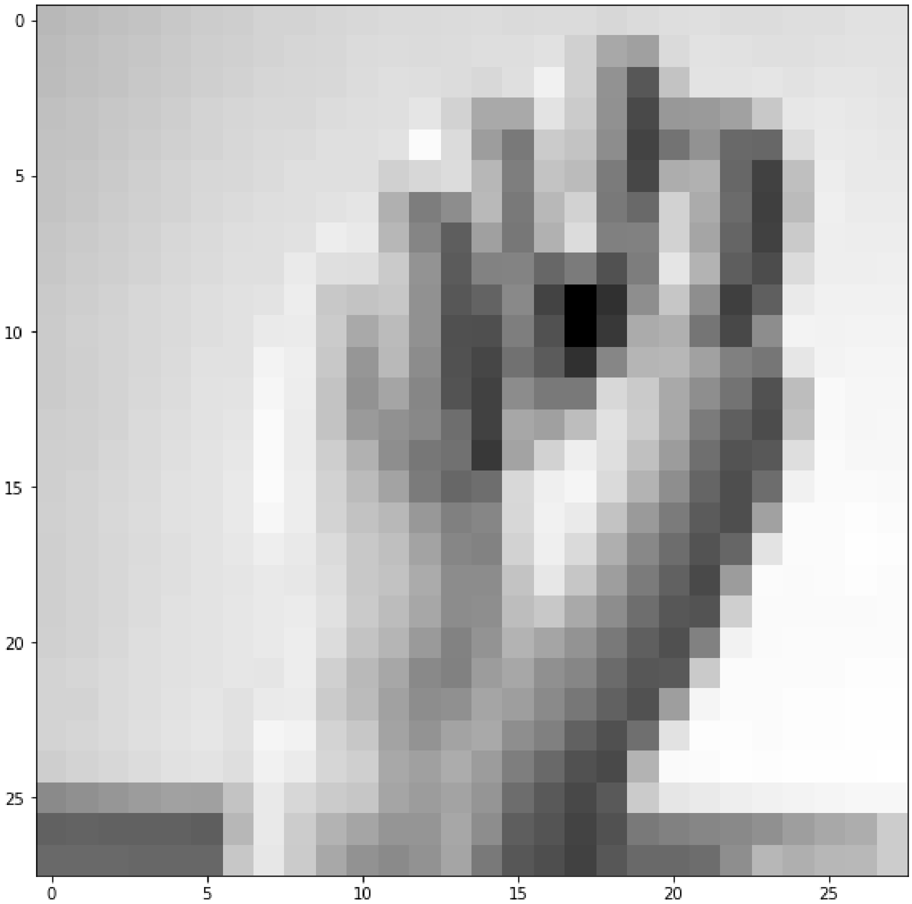
	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	pixel11	pixel12	pixel13	pixel14	pixel15
0	107	118	127	134	139	143	146	150	153	156	158	160	163	165	159
1	155	157	156	156	156	157	156	158	158	157	158	156	154	154	153
2	187	188	188	187	187	186	187	188	187	186	185	185	185	184	184
3	211	211	212	212	211	210	211	210	210	211	209	207	208	207	206
4	164	167	170	172	176	179	180	184	185	186	188	189	189	190	191

5 rows × 784 columns

In [7]:
```
print(labels.shape)
```

(27455,)

In [18]:
```
plt.figure(figsize=(10,10))
ids = 4

data_matrix = data.iloc[ids].as_matrix().reshape(28,28)  # reshape from 1d to 2d pixel arra
plt.imshow(data_matrix, interpolation = "none", cmap = "gray")
plt.show()

print(labels[ids])
```



```
N
```

2D Representation and Visualisation using PCA

In [9]:
```
data = data.head(15000)
labels = labels.head(15000)
print("The shape of the data becomes",data.shape," and labels become",labels.shape)
```

```
The shape of the data becomes (15000, 784)  and labels become (15000,)
```

In [10]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
standardised_data = scaler.fit_transform(data)
print("The standardised data shape is",standardised_data.shape)
```

```
The standardised data shape is (15000, 784)
```

Covariance Matrix

Covariance matrix is used to understand how the variables of input data set are varying from mean wrt to each other or is there any relation ship among one another.

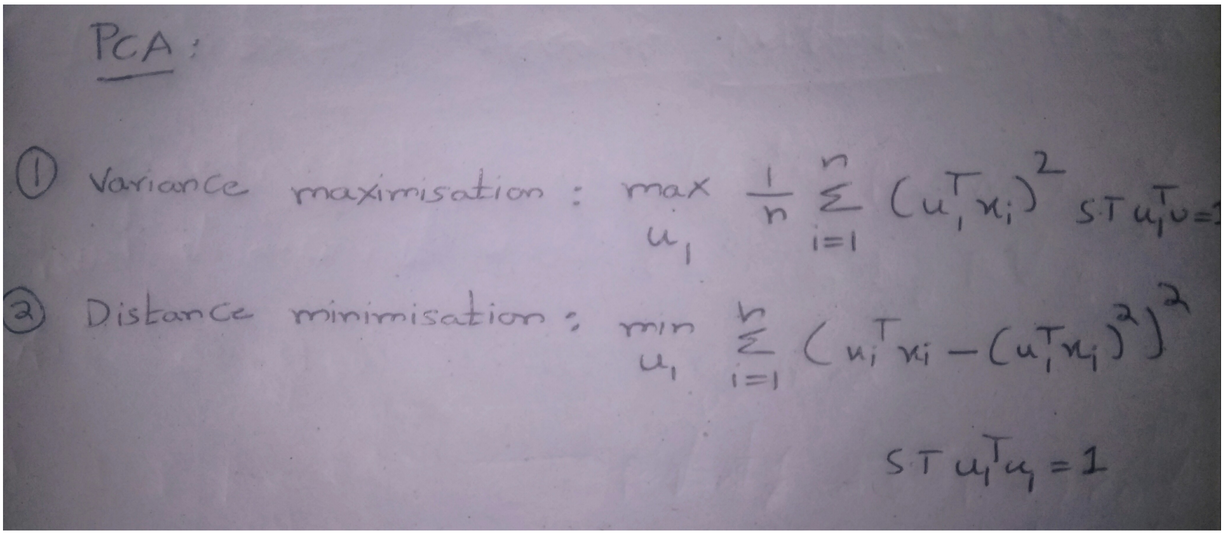
- It is a Symmetric matrix obtained by X^T * X
- cov(a,a) ---> var(a)
- cov(xi,yi) ---> sum[(xi-Ux) * (yi-Uy)] where sum means summation over i=1 to n && U means mean
- Covarince matrix is not more than a table that summarises correlation between all possible pairs of variables

In [11]:
```
resultant_data = standardised_data

# matrix multiplication using numpy
covariance_matrix = np.matmul(resultant_data.T , resultant_data)

print ( "The shape of co_variance matrix = ", covariance_matrix.shape)
```

```
The shape of co_variance matrix =  (784, 784)
```

What is the use of Eigen Values and Eigen Vectors ?

Eigen vectors are the unit vectors which gives the direction of gives the direction of maximum spread of the points in that dimension



In [12]:
```
# finding the top two eigen-values and corresponding eigen-vectors

from scipy.linalg import eigh

# eigh gives eigen values and vectors in ascending order and eigvals of parameters 782,783
values, vectors = eigh(covariance_matrix, eigvals=(782,783))

print("Shape of eigen vectors = ",vectors.shape)

# transposing of vector to give(2,d) dimension
vectors = vectors.T

print("Updated shape of eigen vectors = ",vectors.shape)

# here the vectors[1] represent the eigen vector corresponding 1st principal eigen vector
# here the vectors[0] represent the eigen vector corresponding 2nd principal eigen vector
```

```
Shape of eigen vectors =  (784, 2)
Updated shape of eigen vectors =  (2, 784)
```

In [13]:
```
# projecting the original data sample on the plane

import matplotlib.pyplot as plt

reshaping_of_original_coordinates = np.matmul(vectors,resultant_data.T)

print (" resultanat new data point's shape ", vectors.shape, "X", data.T.shape," = ", resha
```

```
 resultanat new data point's shape  (2, 784) X (784, 15000)  =  (2, 15000)
```

In [0]:
```
words_dict ={
    0:'A',
    1:'B',
    2:'C',
    3:'D',
    4:'E',
    5:'F',
    6:'G',
    7:'H',
    8:'I',
    9:'J',
    10:'K',
    11:'L',
    12:'M',
    13:'N',
    14:'O',
    15:'P',
    16:'Q',
    17:'R',
    18:'S',
    19:'T',
    20:'U',
    21:'V',
    22:'W',
    23:'X',
    24:'Y',
    25:'Z'
}
```

In [15]:
```
words_dict[4]
```

Out[15]:
```
'E'
```

In [16]:
```
for i in range(len(labels)) :
  if(labels[i] in words_dict.keys()) :
    labels[i] = words_dict[labels[i]]
labels = pd.Series(labels)
print(labels.head())
```

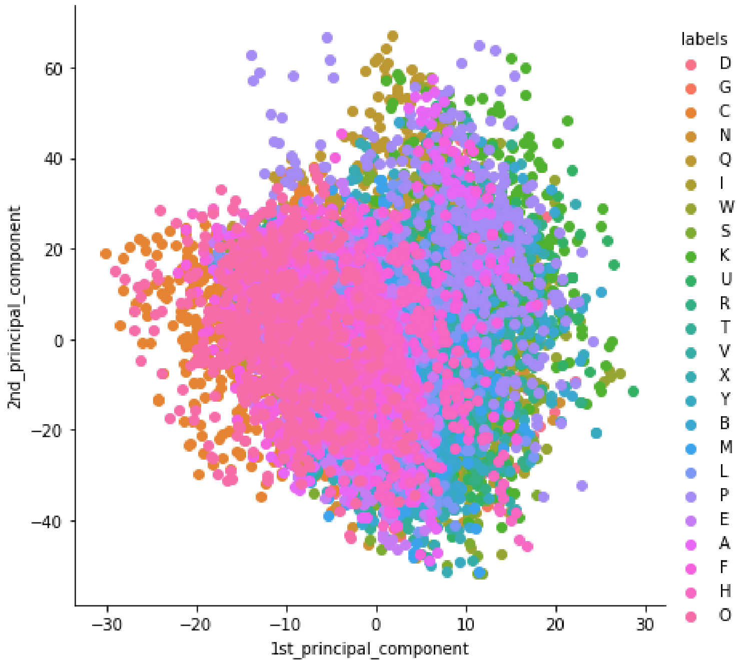
```
0    D
1    G
2    C
3    C
4    N
Name: label, dtype: object
```

In [17]:
```
import pandas as pd

# appending label to the data
final_coordinates = np.vstack((reshaping_of_original_coordinates, labels)).T

# Final2d Data set
two_d_data = pd.DataFrame(data=final_coordinates, columns=("1st_principal_component", "2nd_
print(two_d_data.head())
```

```
  1st_principal_component 2nd_principal_component labels
0                0.347344                 4.62431      D
1                -4.45913                 6.69406      G
2                -20.6547               -0.336218      C
3                -20.3266                -9.53171      C
4                -2.89753                -6.60637      N
```

Visualising our data using seaborn

In [19]:
```
import seaborn as sn
sn.FacetGrid(two_d_data, hue="labels", size=6).map(plt.scatter, '1st_principal_component',
plt.show()
```



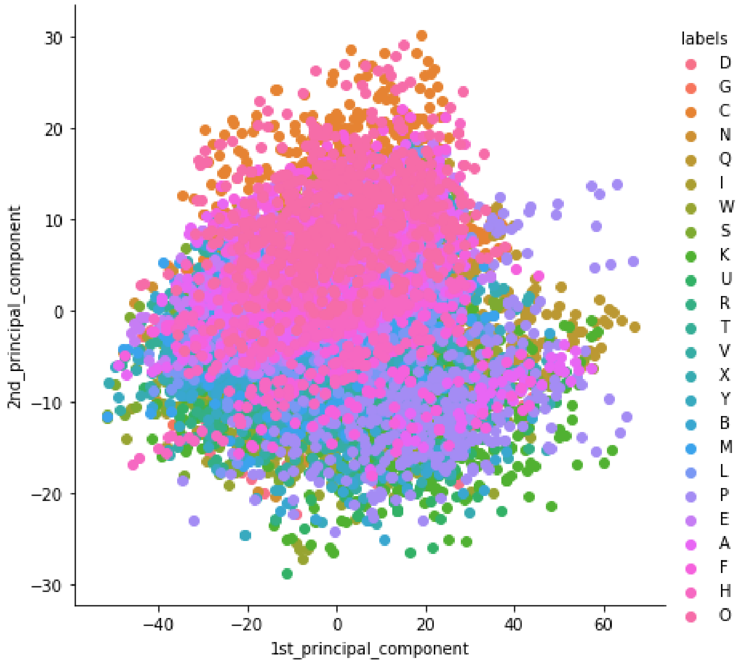
PCA using Scikit-Learn

Let's implement same using sklearn's implementation

In [20]:
```
from sklearn import decomposition
pca = decomposition.PCA(n_components = 2)
pca_data = pca.fit_transform(resultant_data)
print("shape of sklearn's pca implemented data's shape = ", pca_data.shape)
```

```
shape of sklearn's pca implemented data's shape =  (15000, 2)
```

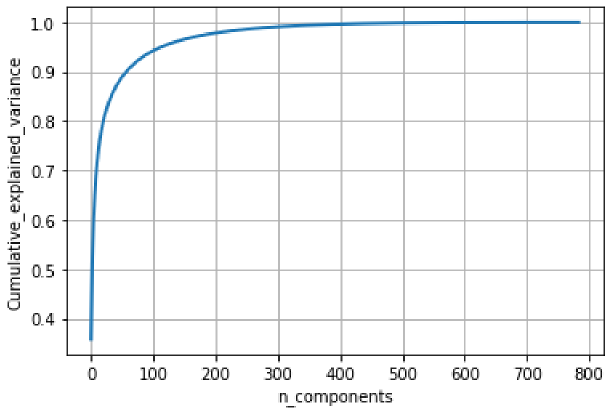
In [21]:
```
pca_data = np.vstack((pca_data.T, labels)).T

# creating a new data fram which help us in ploting the result data
pca_resultant_data = pd.DataFrame(data=pca_data, columns=("1st_principal_component", "2nd_p
sns.FacetGrid(pca_resultant_data, hue="labels", size=6).map(plt.scatter, '1st_principal_com
plt.show()
```



PCA estimation using cumulative sum of percentage Variance Share

- lambda_i gives eigen values
- lambda_i/(sum(lambda_i)) gives percentage of variance explained and we plot the cummulative sums of percentage of variances explained

In [22]:
```
pca.n_components = 784
pca_data = pca.fit_transform(resultant_data)

percentage_var_explained = pca.explained_variance_ / np.sum(pca.explained_variance_);

cum_var_explained = np.cumsum(percentage_var_explained)

#plotting values of PCA
plt.figure(1, figsize=(6, 4))

plt.clf()
plt.plot(cum_var_explained, linewidth=2)
plt.axis('tight')
plt.grid()
plt.xlabel('n_components')
plt.ylabel('Cumulative_explained_variance')
plt.show()
```



CONCLUSIONS : ***By just considering only 100 components we preserves 95% of data***

LIMITATIONS OF PCA :

- Won't work for Sinusoidal data sets
- Won't work for equally distributed data along axis
- Data Loss to some extent