**Music Recommendation Using K-Means Clustering**

About Dataset

The data set which I used in this project is having about 21.5k samples and 21 features originally

-No NaN entries to handle

- Pretty clean data set

-I had to drop a few columns which I considered to be insignificant for the analysis

['type','id','uri','track\_href','analysis\_url','Unnamed: 0']

-The data was left with columns like ['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo','duration\_ms', 'time\_signature', 'genre', 'song\_name'] , named this data frame ‘df’

- Then I removed the features [‘genre’,’song\_name’] from this data frame and saved it as df\_

-**SCALED** df\_ using StandardScaler , I scaled it because I was going to apply PCA and KMeans

**REASON FOR SCALING THE DATA SET**

PCA-I wanted to scale the features because a feature which is having higher scale will have higher variance as compared to a feature which is on a considerably lower scale , and we know that variance of each feature is used in the covariance matrix which is used to determine the eigen value and vectors , i.e Principal Components , thus a feature with higher scale data will have more say in determining the Principal Components , thus it is important to scale the data before applying PCA

KMeans- I was sure about using KMeansClustering algo in my project , and the background mathematics of this algo involves calculating Euclidian distance between samples and centroids , and we know that a feature with higher scale influences Euclidian distance more and ultimately that feature will single handedly dominate the process of determining clusters ,Which is not good !!

-Then after scaling I performed **PCA** and the reason for doing it was to perform dimensionality reduction to minimize curse of dimensionality , (in higher dimensions distance metric becomes meaningless , also because of significantly increased sparsity it becomes quite difficult for the ML models to identify the intricate patterns of the data set ), reduced 2 features , i.e I generated 11 principal components, and the variance explained by these 11 principal components was about 94 percent .

**KMeansClustering**

Determined the appropriate number of clusters =15 , found the optimal number of clusters using elbow plot

The elbow plot is the standard way of figuring out the optimal number of clusters required for your data set

The elbow plot has the wcss (within cluster squared sum ) value on the y axis and the number of clusters on the x axis . to calculate wcss for a particular number of clusters

km=KMeans(n\_clusters=3)

wcss\_3=km.**inertia\_**

**-**The inertia attribute is used to calculate the wcss value for n clusters

**Y**

I got the cluster value of the samlples on the df\_pca (post pca data)

I made a new data frame by concatenating df\_pca and ‘Y’ also made a feature which had row number of samples and saved it as ‘index’, named this new data frame df\_pca\_dep

Overall by the end of all this ,I am left with **df, df\_, df\_pca, df\_pca\_dep**

**PREDICTIONS**

Now I have clusters , and for a query point I can very easily predict its cluster using the KMeans object and I can very easily recommend 10 random samples from that cluster

**BUT**

Say that cluster has 2k entries and randomly 10 out of 2k entries is not going to be a good prediction or recommendation .

To solve this issue I decided to choose 10 nearest neighbours of that query point from that cluster and that’s is how I recommend relevant songs to the person who is listening the current song .

To perform this I made a function which would take in the query point and give us 10 recommendations

* the query point , a list of len 13 , inside the function was firstly scaled using the standard scaler object used to scale the original data set , then transformed using PCA
* Then the subset of the orignal data set which belonged to the cluster of the query point was stored as a data frame named ‘df\_cluster’
* the distance of the query point was calculated from each sample of ‘df\_cluster’ and saved a new feature ‘distance ’,,[THIS WAS A BIT MORE COMPLICATED -Refer code for the same]
* And then post calculating the distance , the sample index was extracted of the 10 nearest neighbours , and they were printed as output

**FUNCTION CODE**

def recommend(query=[0.83, 0.81, 2, -7.36, 1.0, 0.42, 0.06, 0.01, 0.06, 0.39, 156.99, 124539.0, 4.0]):

query=np.array(query)

query=query.reshape(1,13)

query=pd.DataFrame(query,columns=df\_.columns)

query=ss.transform(query)

query\_=pca.transform(query)

cluster=km.predict(query\_)

df\_cluster=df\_pca\_dep[df\_pca\_dep['Y']==cluster[0]]

df\_cluster2=df\_cluster.drop(columns=['Y','index'])

df\_cluster['distances'] = np.linalg.norm(df\_cluster2 - query\_, axis=1)

dfuk=df\_cluster.sort\_values(by='distances')

song\_index=dfuk['index'].head(10)

song\_index=list(song\_index)

values\_at\_indexes = df.loc[song\_index, 'song\_name']

return list(values\_at\_indexes)

np.linalg.norm(df\_cluster2-query\_,axis=1) was used to calculate the distance of the query point from each sample of the cluster it belonged to

**MAJOR QUESTION**

**Why didn’t you directly figure out the 10 nearest neighbours of the query point? Why did you even do clustering ?**

* The reason of applying both the things one after the other is quite easy to understand
* Post clustering I got clusters of similar songs , then I chose 10 NN from that cluster .
* Say the clustering boundary was not there then there is a chance that 10NN would have been different form the above ones , may be not form the same cluster , which clearly implies that clustering has a significant effect on the determination of 10 NNs
* thus in this scenario firstly we clustered similar songs and then again out of the similar songs we found the 10 most similar songs ,ULTIMATELY IMPROVING THE QUALITY OF MUSIC RECOMMENDATION

**END**