MUHAMMAD AHSAN ASIF

218606833

In [77]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns12
from sklearn.cluster import KMeans
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.metrics.cluster import adjusted_mutual_info_score
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from nltk.cluster import KMeansClusterer
import nltk
from sklearn import cluster
from sklearn import metrics
from sklearn.preprocessing import scale
```

Part 1 - Clustering

1)

In [8]:

```
#QUESTION 1

#Creating a dataframe M from the csv
M = pd.read_csv('digitData3.csv')
print('Dimensions of M matrix:', M.shape)

# Assigning m rows and n-1 columns from M dataframe
X=M.values[:,:-1]
print('Dimensions of X matrix:', X.shape)

#Assigning nth column of M
trueLabels=M.values[:,-1]
print('Dimensions of trueLabels:', trueLabels.shape)
```

```
Dimensions of M matrix: (1692, 65)
Dimensions of X matrix: (1692, 64)
Dimensions of trueLabels: (1692,)
```

2)

In [3]:

```
#QUESTION 2

#Computing kmeans for 5 clustering
kmeans = KMeans(n_clusters=5)
kmeans.fit(X)

#Getting the centroids and labels
centroids = kmeans.cluster_centers_
labels    = kmeans.labels_
```

In [4]:

```
#computing adjusted rand index
print(adjusted_rand_score(trueLabels,labels))
```

```
0.35193463573681966
```

In [5]:

```
#computing adjusted mutual information
print(adjusted_mutual_info_score(trueLabels,labels,'arithmetic'))
```

```
0.534419320121989
```

In [6]:

```
averaged_rand_score = 0
averaged_mutual_score = 0
i=0
```

In [10]:

```
while i<50:
    #running the kmeans for 50 random initialization
    kmeanstest = KMeans(n_clusters=5,init='random',n_init=1)
    kmeanstest.fit(X)
    centroids = kmeans.cluster_centers_
    labels    = kmeans.labels_

    #Calculating the average of adjusted rand scord and adjusted mutual informat
ion
    ad_rand = adjusted_rand_score(trueLabels,labels)
    ad_mutual = adjusted_mutual_info_score(trueLabels,labels,'arithmetic')


    averaged_rand_score = averaged_rand_score+ad_rand
    averaged_mutual_score = averaged_mutual_score + ad_mutual
    i += 1
```

In [11]:

```
#adjusted mutual information score after 50 iterations
averaged_mutual_score = averaged_mutual_score/50
print('The average adjusted mutual information score after 50 iterations : ', av
eraged_mutual_score)
```

```
The average adjusted mutual information score after 50 iterations :
0.5344193201219883
```

In [12]:

```
#adjusted rand score after 50 iterations
averaged_rand_score = averaged_rand_score/50
print('The average adjusted rand score score after 50 iterations : ', averaged_r
and_score)
```

```
The average adjusted rand score score after 50 iterations :  0.35193
46357368198
```

3)

In [74]:

```
#Now for instance we had an initial value of 0.7 as ARI

ARI = 0.7
k=0
# Lets run the kmeans for 20 runs

while k<20:
    #running the kmeans for 50 random initialization
    kmeanstest = KMeans(n_clusters=5,init='k-means++',n_init=1)
    kmeanstest.fit(X)
    centroids3 = kmeans.cluster_centers_
    labels3    = kmeans.labels_

    #Calculating the average of adjusted rand scord
    ad_rand3 = adjusted_rand_score(trueLabels,labels3)
    ARI = ARI+ad_rand3
    k +=1
```

In [75]:

```
#After running for 20 runs we can see that the ARI is as followed
ARI = ARI/20
print(ARI)
```

```
0.38693463573681963
```

After seeing the value of ARI, it has been decreased from 0.7 to 0.39. It could be concluded that with every single run the accuracy between the similarity of predicted labels and trained labels were decreasing. When computing kmeans ++ the value should be nearer to 1 otherwise if close to 0 it would indicate that centroids were initialized at random which is not the case. The higher the value of ARI the more similarity there is between the predicted and trained labels, as we have seen. Also, the less value of ARI could also be due to outliers, which might be chosen as data points. Kmeans++ also always selects the centroids which are far away from the data points as that increases its efficient and accuracy in overall results.

4)

In [13]:

```
# Doing kmeans through cosine distance similarity measure from the NLTK library
# Initializing the clusters to use cosine distance

kclusterer = KMeansClusterer(5, distance=nltk.cluster.util.cosine_distance, repe
ats=25)
assigned_clusters = kclusterer.cluster(X, assign_clusters=True)

kmeans_cos = cluster.KMeans(n_clusters=5)
kmeans_cos.fit(X)

cosine_centroids = kmeans_cos.cluster_centers_
cosine_labels = kmeans_cos.labels_
```

In [14]:

```
#computing adjusted rand index
print(adjusted_rand_score(trueLabels,cosine_labels))
```

```
0.32654943868304015
```

In [15]:

```
#computing adjusted mutual information
print(adjusted_mutual_info_score(trueLabels,cosine_labels,'arithmetic'))
```

```
0.5140142810253878
```

In [16]:

```
averaged_rand_score2 = 0
averaged_mutual_score2 = 0
j=0
```

In [17]:

```
while j<50:
    #running the kmeans using cosine distance for 50 random initialization

    kclusterer = KMeansClusterer(5, distance=nltk.cluster.util.cosine_distance,
repeats=25)
    assigned_clusters = kclusterer.cluster(X, assign_clusters=True)
    kmeans_cos = cluster.KMeans(n_clusters=5)
    kmeans_cos.fit(X)

    cosine_centroids = kmeans_cos.cluster_centers_
    cosine_labels = kmeans_cos.labels_

    #Calculating the average of adjusted rand scord and adjusted mutual informat
ion
    ad_rand2 = adjusted_rand_score(trueLabels,cosine_labels)
    ad_mutual2 = adjusted_mutual_info_score(trueLabels,cosine_labels,'arithmeti
c')


    averaged_rand_score2 = averaged_rand_score2+ad_rand2
    averaged_mutual_score2 = averaged_mutual_score2 + ad_mutual2
    j += 1
```

In [21]:

```
#adjusted mutual information score after 50 iterations
averaged_mutual_score2 = averaged_mutual_score2/50
print('The average adjusted mutual information score after 50 iterations : ', av
eraged_mutual_score2)
```

```
The average adjusted mutual information score after 50 iterations :
0.5411483639670291
```

In [20]:

```
#adjusted rand score after 50 iterations
averaged_rand_score2 = averaged_rand_score2/50
print('The average adjusted rand score score after 50 iterations : ', averaged_r
and_score2)
```

```
The average adjusted rand score score after 50 iterations :  0.35523
449001910706
```

Before the comparison, just want to report that this method took me 25 minutes to run this loop for cosine distance. This method could be very costly for companies as it sacrifices the run time. Now if we compare the average ARI values from the kmeans++ using euclidean distance with random kmeans using cosine, we can see that the values are better than euclidean distance. This means that cosine distance gave more similarity between the predicted and trained data, which is good but at the cost of run time. It took 25 times the time euclidean distance took to compute the results. Therefore, i would conclude that cosine distance as a similarity measure to give more accurate results but at the cost of run time, if compared to kmeans with euclidean distance as a similarity measure.

Question 2

1)

In [23]:

```
#Normalizing X
Xnorm = scale(X)
```

In [26]:

```
#Performing PCA for 40 dimensions
pca = PCA(n_components=40)
```

In [27]:

```
pca.fit(Xnorm)
```

Out[27]:

```
PCA(copy=True, iterated_power='auto', n_components=40, random_state=
None,
  svd_solver='auto', tol=0.0, whiten=False)
```

In [29]:

```
#Calculating Variance
var= pca.explained_variance_ratio_
```

In [31]:

```
#Calculating cumulative variance
var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
```

In [32]:

```
# As we can see that if i use 40 dimensions i can get a variance of atleast 95%
 by then
print(var1)
```

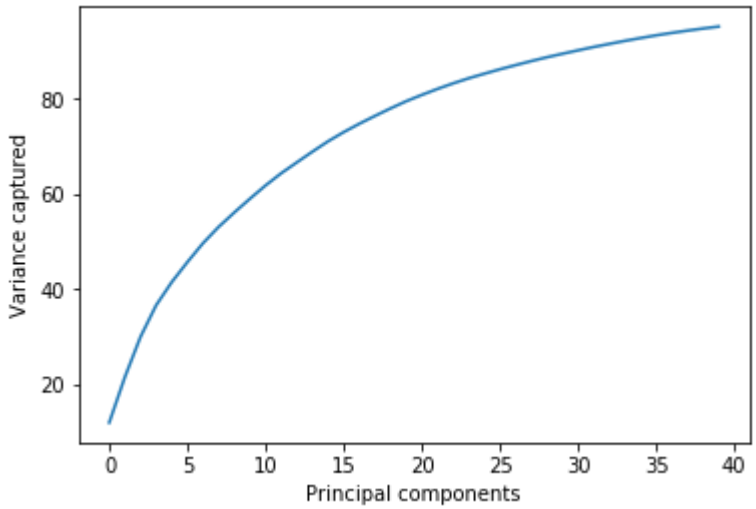
```
[12.   21.56 30.01 36.64 41.51 45.71 49.68 53.1  56.12 59.01 61.8  6
4.36
 66.64 68.9  71.07 73.   74.76 76.4  77.99 79.49 80.82 82.07 83.24 8
4.31
 85.27 86.2  87.07 87.92 88.72 89.46 90.18 90.88 91.53 92.17 92.76 9
3.33
 93.84 94.32 94.76 95.17]
```

In [34]:

```
#Plotting the captured variance with respect to increasing latent dimensionaliot
y
plt.plot(var1)
plt.xlabel("Principal components")
plt.ylabel("Variance captured")
```

Out[34]:

```
Text(0, 0.5, 'Variance captured')
```



2)

In [47]:

```
#Performing PCA for two principle components

#Scaling the data
scaler = StandardScaler()
scaler.fit(X)

#Transforming the data
scaled_data = scaler.transform(X)

#Performing PCA for 2 principle components
pca2 = PCA(n_components= 2)
pca2.fit(scaled_data)
x_pca = pca2.transform(scaled_data)
```

In [41]:

```
#Now if we check the dimensions of x_pca we can see our two principle components
as dimensions
print(x_pca.shape)
```

```
(1692, 2)
```

In [60]:

```
# Plotting the total rows of Xprojected onto the first two principal components
plt.figure(figsize=(8,6))
plot = plt.scatter(x_pca[:,0],x_pca[:,1],c = trueLabels, cmap='plasma')
plt.xlabel('First Principle Component, V1')
plt.ylabel('Second Principle Component, V2')
plt.title('Scatter plot of Rows and principle components')
plt.colorbar(plot)
```

Out[60]:

```
<matplotlib.colorbar.Colorbar at 0x1a202f4c50>
```