Problem Set 7

by Chu Zhuang

```
import numpy as np
import pandas as pd
from copy import deepcopy
import matplotlib.pyplot as plt
import seaborn
```

k-Means Clustering "By Hand"

Import the data:

```
x1=[5,8,7,8,3,4,2,3,4,5]  #5,8,7,8,3,4,2,3,4,5
x2=[8,6,5,4,3,2,2,8,9,8]  #8,6,5,4,3,2,2,8,9,8
x=np.array(list(zip(x1,x2)))
```

1. Initiate k-means randomly:

```
np.random.seed(123)  #ensure the same randomn process
clusters=np.array([1,1,1,2,2,2,3,3,3,3])   #initial assignemnt
np.random.shuffle(clusters)
```

```
#calculate centroid based on the first assignment
k_num=3                     #num of clusters
cent=np.zeros([k_num,2])      #to calculate centroids
for i in range(k_num):
    data_in_cluster=[x[j,:] for j in range(len(x)) if clusters[j]==(i+1)]
    cent[i]=np.mean(data_in_cluster,axis=0)
```

```
#centroids of the first assignment:
cent
```

```
array([[5.        , 5.33333333],
       [5.66666667, 4.66666667],
       [4.25      , 6.25      ]])
```

2. Compute centroid and updated by distance with iterations:

```
#define function to calculate euclidean distance
def dist(x1, x2, axis=1):
    return np.linalg.norm(x1-x2, axis=axis)
```

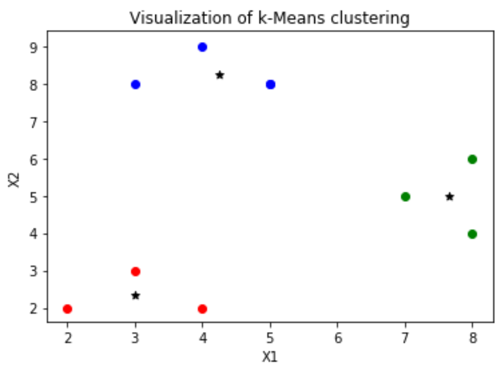
```
#iteration flag to signal when to stop
iteration_flag=np.array([1,1,1])#dist(cent,cent_old,1)
#time of iterations
iter_num=1

while iteration_flag.any()!=0 and iter_num<30:
    #re-assign points according to the new centroids
    for i in range(len(x)):
        #calculate distance of each data point to the centroid
        distance_x_c=dist(x[i],cent,1)
        #assign data point according to the cloest distance
        clusters[i]=np.argmin(distance_x_c)+1

    cent_old=deepcopy(cent)
    #calculate new centroids
    for i in range(k_num):
        data_in_cluster=[x[j,:] for j in range(len(x)) if clusters[j]==(i+1)]
        cent[i]=np.mean(data_in_cluster,axis=0)

    iter_num+=1
    iteration_flag=dist(cent,cent_old,1)
```

```
print("Number of iterations:" , iter_num)
print("Clusters assignment of each point:", clusters)
```

```
Number of iterations: 4
Clusters assignment of each point: [3 2 2 2 1 1 1 3 3 3]
```

3. Visualization

```python
#plot prediction results based on Test dataset
labels=['Cluster1','Cluster2','Cluster3']
colors=['r','g','b']
fig, ax = plt.subplots()

#plot points in each cluster
for i in range(k_num):
    x0=np.array([x[j,:] for j in range(len(x)) if clusters[j]==(i+1)])
    ax.scatter(x0[:,0],x0[:,1],c=colors[i],label=labels[i])
#plot centroids
ax.scatter(cent[:, 0], cent[:, 1], marker='*', c='black')

#plt.legend(loc = 'center right', title = 'Catgories')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Visualization of k-Means clustering')
plt.show()
```



4. With 2 clusters, k=2

```python
clusters2=np.array([1,1,1,1,1,2,2,2,2,2])   #initial assignemnt
np.random.shuffle(clusters2)
```

```python
#calculate centroid based on the first assignment
k_num2=2                        #num of clusters
cent2=np.zeros([k_num2,2])      #to calculate centroids
for i in range(k_num2):
    data_in_cluster=[x[j,:] for j in range(len(x)) if clusters2[j]==(i+1)]
    cent2[i]=np.mean(data_in_cluster,axis=0)
```

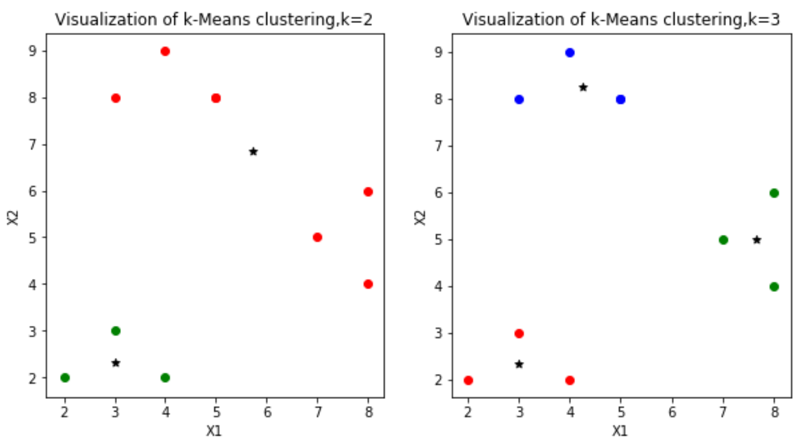
```python
#iteration flag to signal when to stop
iteration_flag=np.array([1,1,1])#dist(cent,cent_old,1)
#time of iterations
iter_num=1

while iteration_flag.any()!=0 and iter_num<30:
    #re-assign points according to the new centroids
    for i in range(len(x)):
        #calculate distance of each data point to the centroid
        distance_x_c=dist(x[i],cent2,1)
        #assign data point according to the cloest distance
        clusters2[i]=np.argmin(distance_x_c)+1

    cent_old=deepcopy(cent2)
    #calculate new centroids
    for i in range(k_num2):
        data_in_cluster=[x[j,:] for j in range(len(x)) if clusters2[j]==(i+1)]
        cent2[i]=np.mean(data_in_cluster,axis=0)

    iter_num+=1
    iteration_flag=dist(cent2,cent_old,1)
```

```python
print("Number of iterations:" , iter_num)
print("Clusters assignment of each point:", clusters)
```

```
Number of iterations: 3
Clusters assignment of each point: [3 2 2 2 1 1 1 3 3 3]
```

```python
#plot prediction results based on Test dataset
plt.figure(12,figsize=(10,5))
labels=['Cluster1','Cluster2']
colors=['r','g','b']
```

```
plt.subplot(121)

#plot points in each cluster
for i in range(k_num2):
    x0=np.array([x[j,:] for j in range(len(x)) if clusters2[j]==(i+1)])
    plt.scatter(x0[:,0],x0[:,1],c=colors[i],label=labels[i])
#plot centroids
plt.scatter(cent2[:, 0], cent2[:, 1], marker='*', c='black')

#plt.legend(loc = 'center right', title = 'Catgories')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Visualization of k-Means clustering,k=2')
#plt.show()

#--------------------------------------#
#plot the results of k=3
labels=['Cluster1','Cluster2','Cluster3']
colors=['r','g','b']
plt.subplot(122)

#plot points in each cluster
for i in range(k_num):
    x0=np.array([x[j,:] for j in range(len(x)) if clusters[j]==(i+1)])
    plt.scatter(x0[:,0],x0[:,1],c=colors[i],label=labels[i])
#plot centroids
plt.scatter(cent[:, 0], cent[:, 1], marker='*', c='black')

#plt.legend(loc = 'center right', title = 'Catgories')
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('Visualization of k-Means clustering,k=3')
plt.show()
```



As we could see above, the initial guess of 3 clusters works well in this dataset. The ten data-points are clearly divided into 3 clusters with clear separation in feature space; while with k=2, two visually separated clusters are merged together; also there is no more sub-clusters in the 3 clusters as well, so 3 clusters capture the pattern/simialrity in the dataset very well.

Calculate the silhouetter score for further verification

the silhouette score for 3 clusters is larger than 2 clusters which further validate the visual check results- the best number of clusters in this dataset k=3.

```
#import silhouette score
from sklearn.metrics import silhouette_score
score_k2=silhouette_score(x, clusters2)
score_k3=silhouette_score(x, clusters)
print("silhouette score for k=2:",score_k2)
print("silhouette score for k=3:",score_k3)
```

```
silhouette score for k=2: 0.5084966486673459
silhouette score for k=3: 0.6924487175226476
```

Application

```
#load the data
df_wiki=pd.read_csv('data/wiki.csv')
df_wiki.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	age	gender	phd	yearsexp	userwiki	pu1	pu2	pu3	peu1	peu2	...	exp5	domain_Sciences	doma
0	40	0	1	14	0	4	4	3	5	5	...	2	1	0
1	42	0	1	18	0	2	3	3	4	4	...	4	0	0
2	37	0	1	13	0	2	2	2	4	4	...	3	0	0
3	40	0	0	13	0	3	3	4	3	3	...	4	0	0
4	51	0	0	8	1	4	3	5	5	4	...	4	0	0

5 rows × 57 columns

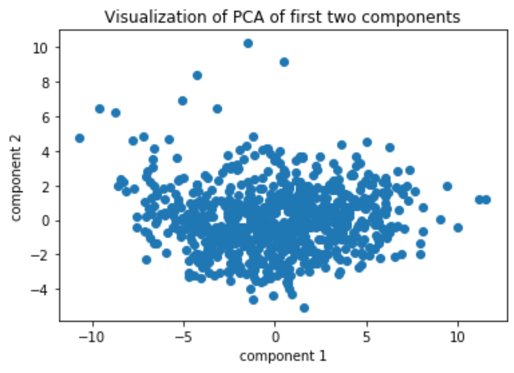
```python
#organized as array
np_wiki=df_wiki.values
```

```python
#preprocessing the data, standardize the dataset
from sklearn import preprocessing
np_wiki1=np_wiki
np_wiki1=preprocessing.StandardScaler().fit_transform(np_wiki1)
```

1. Perform PCA and plot the first and second components

```python
from sklearn.decomposition import PCA
#build the pca model according to sklearn package
pca=PCA()                                      #automatically find number of components, n_components="mle"
pca_wiki_model=pca.fit(np_wiki1)               #fit in the data
pca_wiki_score=pca.fit_transform(np_wiki1)  #derive the transformed score
```

```python
#visualization
plt.scatter(pca_wiki_score[:,0],pca_wiki_score[:,1])
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.title('Visualization of PCA of first two components')
plt.show()
```



```python
#save loadings of the first two components
pca_wiki_loadings=pca_wiki_model.components_
pca_wiki_loadings=np.transpose(pca_wiki_loadings[:2,:])
df_pca=pd.DataFrame(pca_wiki_loadings,index=df_wiki.columns)
df_pca.columns=['PC1','PC2']
```

```python
df_pca.sort_values(by='PC1',ascending=False)[0:5]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	PC1	PC2
bi2	0.230924	0.083431
bi1	0.226193	0.056374
use3	0.218809	0.155152
use4	0.214558	0.160865
pu3	0.210863	0.028776

```
df_pca.sort_values(by='PC2')[0:5]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	PC1	PC2
peu1	0.061228	-0.271741
inc1	0.104667	-0.245440
sa3	0.120376	-0.242325
sa1	0.121658	-0.229926
enj2	0.131110	-0.227602

```
df_pca.sort_values(by='PC2',ascending=False)[0:5]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	PC1	PC2
exp4	0.099873	0.228494
use2	0.147852	0.218629
use1	0.181477	0.197827
vis3	0.175351	0.197635
domain_Engineering_Architecture	0.051309	0.171484

```
df_pca.sort_values(by='PC2')[0:5]
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

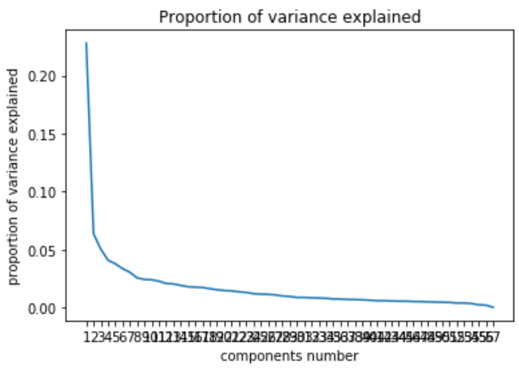
	PC1	PC2
peu1	0.061228	-0.271741
inc1	0.104667	-0.245440
sa3	0.120376	-0.242325
sa1	0.121658	-0.229926
enj2	0.131110	-0.227602

As we could see above from the loadings for principal components, `bi1/2`, `use3/4` and `pu3` contribute most to the first component in a positive direction; for the second component, `peu1`, `inc1`, `sa3/sa1` contribute most in a negative direction, while `exp4` also contributes a lot in a positve direction. The patterns of feature captured by these two components are quite different.

7. Plot PVE and cumulative PVE

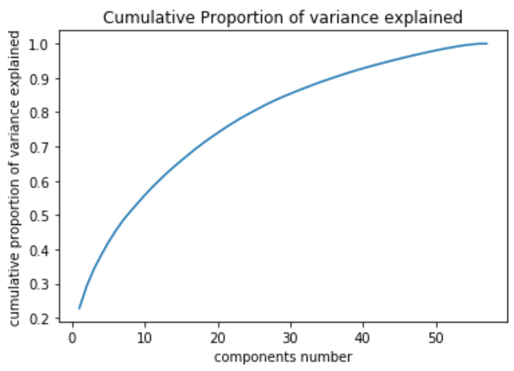
```
#retrieve the pve score
pve=pca_wiki_model.explained_variance_ratio_
```

```
#plot PVE by numbers of component
x=list(range(len(pve)))
x=[i+1 for i in x]
plt.plot(x,pve)
plt.xlabel('components number')
plt.ylabel('proportion of variance explained')
plt.title('Proportion of variance explained')
plt.xticks(np.arange(1,len(pve)+1))
plt.show()
```



```
#calculate cumulative PVE
cum_pve=np.zeros(len(pve))
for i in range(len(pve)):
    cum_pve[i]=np.sum(pve[:i+1])
```

```
#plot cumulative PVE by numbers of component
x=list(range(len(cum_pve)))
x=[i+1 for i in x]
plt.plot(x,cum_pve)
plt.xlabel('components number')
plt.ylabel('cumulative proportion of variance explained')
plt.title('Cumulative Proportion of variance explained')
plt.show()
```



Cumulative proportion of variance explained goes to 1 when all the components are considered.

```
#print the pve and cumulative pve for the first two components
print('PVE of the first component:')
print(round(pve[0],4))
print('PVE of the second component:')
print(round(pve[1],4))
print('PVE of the third component:')
print(round(pve[2],4))
print('Cumulative PVE of the first two components:')
print(round(cum_pve[1],4))
```

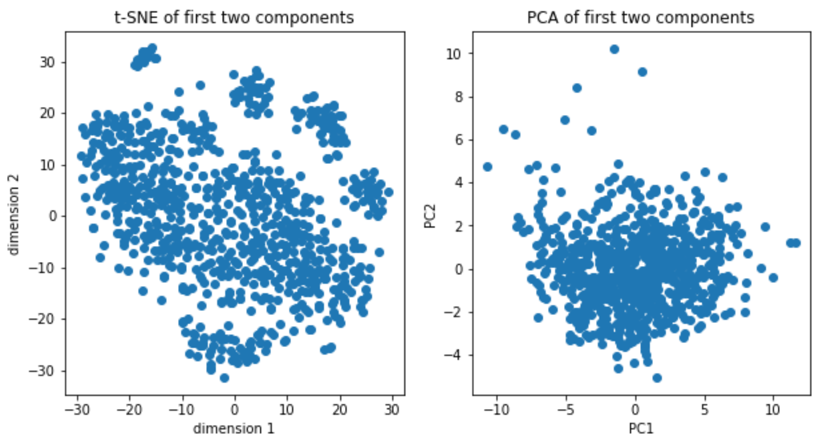
```
PVE of the first component:
0.2281
PVE of the second component:
0.0637
PVE of the third component:
0.0502
Cumulative PVE of the first two components:
0.2918
```

As we could see from the figure and numbers above, the first component explained around 22.8% variance, while the sencond componenet explained around 6.37% variance, then the third component explained about 5.02% variance. The first two components capture around **29.18% variance cumulatively**. The first component captures the most variance in the dataset.

8. t-SNE and visualization

```
from sklearn.manifold import TSNE #For T-SNE
tsne_wiki = TSNE().fit_transform(np_wiki1)
```

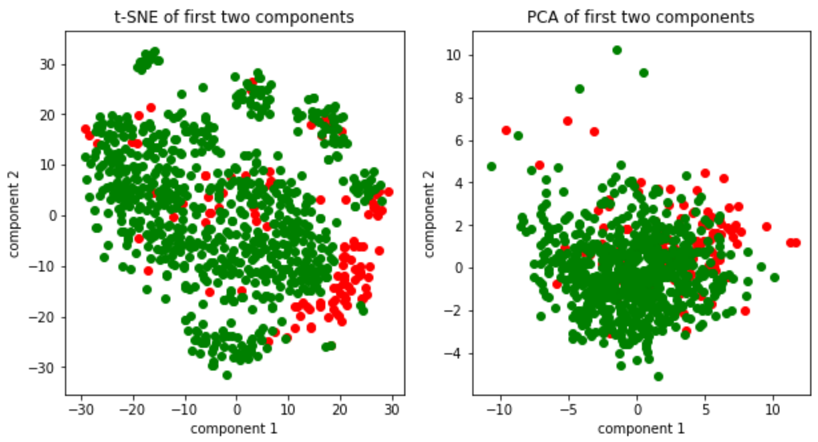
```
plt.figure(12,figsize=(10,5))

#visualization for t-SNE
plt.subplot(121)
plt.scatter(tsne_wiki[:,0],tsne_wiki[:,1])
plt.xlabel('dimension 1')
plt.ylabel('dimension 2')
plt.title('t-SNE of first two components')

#visualization for PCA
plt.subplot(122)
plt.scatter(pca_wiki_score[:,0],pca_wiki_score[:,1])
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA of first two components')
plt.show()
```



Then I further plot with label (whether 'wiki user' or not) to view the data points more clearly:

```
#plot with label (wiki_user), more clearly
wiki_user=np_wiki[:,4]                  #presumably, 1=wiki user, 0=not wiki user
index_wiki=np.where(wiki_user==1)[0]    #index for wiki user
index_notwiki=np.where(wiki_user==0)[0] #index for non-wiki user

#figure settings
plt.figure(12,figsize=(10,5))
colors=['r','g']

#visualization for t-SNE
plt.subplot(121)
plt.scatter(tsne_wiki[index_wiki,0],tsne_wiki[index_wiki,1],c=colors[0])
plt.scatter(tsne_wiki[index_notwiki,0],tsne_wiki[index_notwiki,1],c=colors[1])
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.title('t-SNE of first two components')
```

```
#visualization for PCA
plt.subplot(122)
plt.scatter(pca_wiki_score[index_wiki,0],pca_wiki_score[index_wiki,1],c=colors[0])
plt.scatter(pca_wiki_score[index_notwiki,0],pca_wiki_score[index_notwiki,1],c=colors[1])
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.title('PCA of first two components')
plt.show()
```



The reduced dimension derived by t-SNE is quite direrent from the one derived by PCA; (while both of them cannot separate 'wiki-users), the distribution and configuration of the reduced dimensions from these two dimension reduction methods differ a lot. This discrepancy probably comes from the differece of each algorithm: one-PCA is linear, while the t-SNE is non-linear and probablistic based, and hence yields different 'reduced feature space' after dimension reduction.

Also, from the figure above, we could also see that by t-SNE, the data points are more evenly spread out across the axis which might comes from the t-distribution transformation to lower dimension which reduce the effect of 'outliers', while by PCA, there are some 'obvious outliers' that the original difference and variance in the dataset are retained.

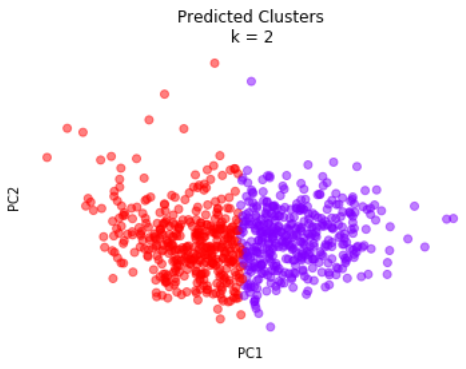
Clustering

9. k-means

```
from sklearn.cluster import KMeans

#build model for num of categories of 2
k_num=2
wikiKM=KMeans(n_clusters = k_num, init='k-means++')
wikiKM.fit(np_wiki1)
clusters=wikiKM.labels_
```

```
#set label for clusters
colors = list(plt.cm.rainbow(np.linspace(0,1, k_num)))
colors_k=[colors[c] for c in wikiKM.labels_]
```

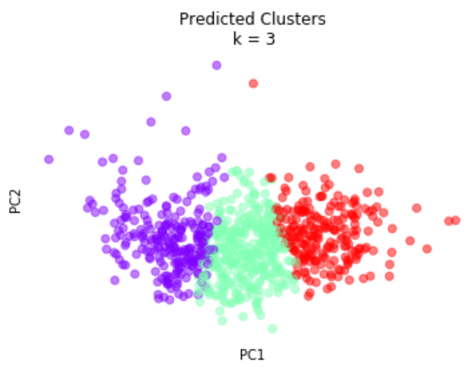
```
#visulization
fig = plt.figure(1)
ax = fig.add_subplot(111)
ax.set_frame_on(False)
plt.scatter(pca_wiki_score[:, 0], pca_wiki_score[:, 1], color = colors_k, alpha = 0.5)
plt.xticks(())
plt.yticks(())
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Predicted Clusters\n k = {}'.format(k_num))
plt.show()
```



```
#build model for num of categories of 3
k_num=3
wikiKM=KMeans(n_clusters = k_num, init='k-means++')
wikiKM.fit(np_wiki1)
clusters=wikiKM.labels_
cent=wikiKM.cluster_centers_
```

```
#set label for clusters
colors = list(plt.cm.rainbow(np.linspace(0,1, k_num)))
colors_k=[colors[c] for c in wikiKM.labels_]
```

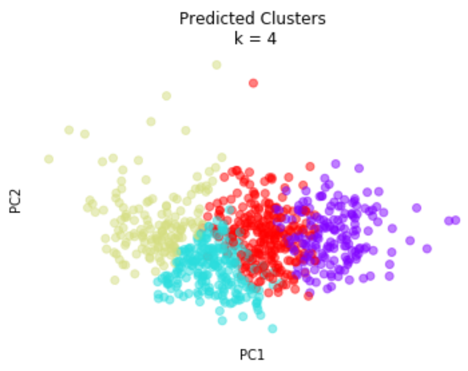
```
#visulization for k=3
fig = plt.figure(1)
ax = fig.add_subplot(111)
ax.set_frame_on(False)
plt.scatter(pca_wiki_score[:, 0], pca_wiki_score[:, 1], color = colors_k, alpha = 0.5)
plt.xticks(())
plt.yticks(())
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Predicted Clusters\n k = {}'.format(k_num))
plt.show()
```



```
#build model for num of categories of 4
k_num=4
wikiKM=KMeans(n_clusters = k_num, init='k-means++')
wikiKM.fit(np_wiki1)
clusters=wikiKM.labels_
```

```
#set label for clusters
colors = list(plt.cm.rainbow(np.linspace(0,1, k_num)))
colors_k=[colors[c] for c in wikiKM.labels_]
```

```
#visulization
fig = plt.figure(1)
ax = fig.add_subplot(111)
ax.set_frame_on(False)
plt.scatter(pca_wiki_score[:, 0], pca_wiki_score[:, 1], color = colors_k, alpha = 0.5)
plt.xticks(())
plt.yticks(())
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('Predicted Clusters\n k = {}'.format(k_num))
plt.show()
```



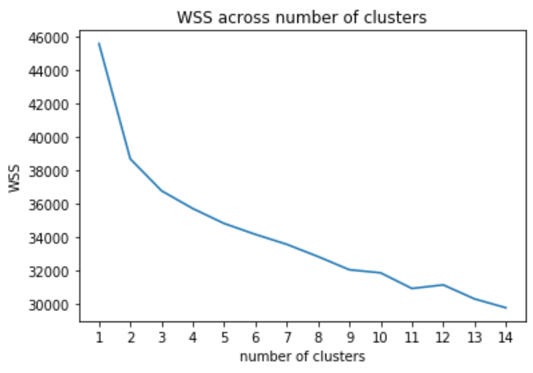
From the visulization by PCA, we could see that each cluster is clearly separated by K-means. With the number of clusters increasing, the clusters seem to be divided evenly across the feature space (there seems no clear subgroups, no within group division when k number increases). Moreover, from the visualization, we could see that the separation is mainly caused by 'PC1'. The division of dataset is mainly based on the difference of observations in the first component which captures the most varaince in the dataset.

10. Identify optimal number of clusters (elbow method and silhouette score)

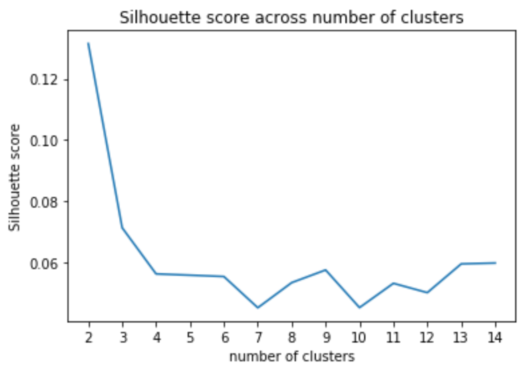
```
#pre-define a series of k
k_nums=np.arange(1,15)
wss_all=np.zeros(len(k_nums))
sil_all=np.zeros(len(k_nums)-1)

for k in k_nums:
    wikiKM=KMeans(n_clusters = k, init='k-means++')
    wikiKM.fit(np_wiki1)
    clusters=wikiKM.labels_
    cent=wikiKM.cluster_centers_

    wss=0
    for j in range(wiki_size):
        c1=cent[clusters[j]]
        x1=np_wiki1[j]
        wss+=sum((x1-c1)**2)
    wss_all[k-1]=wss
    if k!=1:
        sil_all[k-2]=silhouette_score(np_wiki1, clusters)
```

Visualizations of WSS and silouette score:

```
#plot wss with number of clusters
plt.plot(k_nums,wss_all)
plt.xticks(k_nums);
plt.xlabel("number of clusters")
plt.ylabel("WSS")
plt.title("WSS across number of clusters");
```



```
#plot silhouette with number of clusters
plt.plot(np.arange(2,15),sil_all)
plt.xticks(np.arange(2,15));
plt.xlabel("number of clusters")
plt.ylabel("Silhouette score")
plt.title("Silhouette score across number of clusters");
```

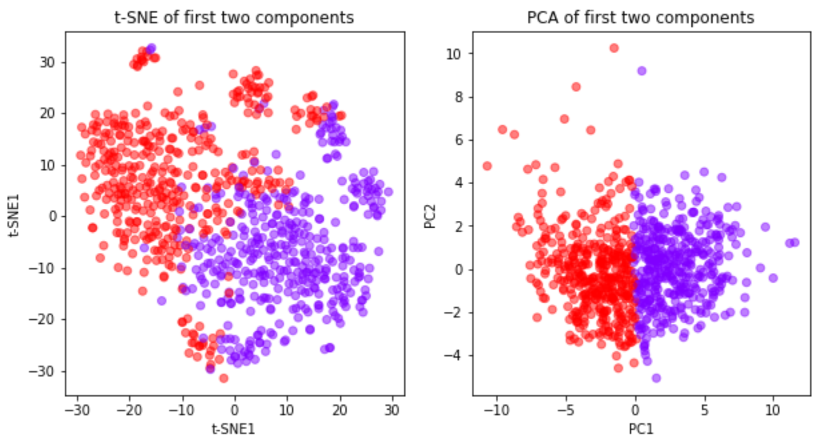


From the figure above and below, we could more clearly figure out that k=2 seems to be the optimal number of clusters: 1) elbow point in the WSS plot; 2) it has distinctively highest silouette score.

11. Visualization of the optimal k=2:

```
k_num=2
wikiKM=KMeans(n_clusters = k_num, init='k-means++')
```

```
wikiKM.fit(np_wiki1)
clusters=wikiKM.labels_

colors = list(plt.cm.rainbow(np.linspace(0,1, k_num)))
colors_k=[colors[c] for c in wikiKM.labels_]

#visualization
plt.figure(12,figsize=(10,5))

#visualization for t-SNE
plt.subplot(121)
plt.scatter(tsne_wiki[:,0],tsne_wiki[:,1],color = colors_k, alpha = 0.5)
plt.xlabel('t-SNE1')
plt.ylabel('t-SNE1')
plt.title('t-SNE of first two components')

#visualization for PCA
plt.subplot(122)
plt.scatter(pca_wiki_score[:,0],pca_wiki_score[:,1],color = colors_k, alpha = 0.5)
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title('PCA of first two components')
plt.show()
```



From the above visualizations, we could see that in both PCA and t-SNE dimension reduction, the first component captures the most obvious difference/variance in the dataset (division happens in the first component axis).

While in PCA, this division is more evenly and we could see a clear separation by the point of 0 in 'PC1' axis. That is because PCA is a linear dimension reduction and retains the spatial difference linearly which corresponds well with the 'eulicidean distance' used in k-Means. However, in t-SNE, this separation is not very clear. Also the 'point 0 separation' is not seen in t-SNE as well, since t-SNE is a non-linear dimension reduction algorithm and retain similarity/distance in dataset based on conditional probability which might distort the original feature space nonlinearly, hence, it does not show a clear boundary in the current clustering calculated on linear distance.