modelhw7

March 15, 2020

[14]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import pylab
import math
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.manifold import TSNE
from sklearn.metrics import silhouette_score


import warnings
warnings.filterwarnings("ignore")
np.random.seed(1234)
```

1 k-Means Clustering "By Hand"

1.0.1 1. Imitate the k-means random initialization part of the algorithm by assigning each observation to a cluster at random.

[26]:
```
input_1 = [5,8,7,8,3,4,2,3,4,5]
input_2= [8,6,5,4,3,2,2,8,9,8]
```

[27]:
```
df_k3 = pd.DataFrame({'input_1': input_1, 'input_2': input_2})
k3_labels = np.random.choice(3, 10, replace=True)
df_k3['k_label'] = k3_labels
df_k3
```

[27]:

	input_1	input_2	k_label
0	5	8	0
1	8	6	1

2	7	5	2
3	8	4	2
4	3	3	2
5	4	2	0
6	2	2	0
7	3	8	1
8	4	9	2
9	5	8	2

1.0.2 2. Compute the cluster centroid and update cluster assignments for each observation iteratively based on spatial similarity.

[28]:
```
def fit_kmeans(num_k, df):
    fit_df = df.copy()
    for i in range(50):
        centroid = {}
        for k in range(num_k):
            cent1 = fit_df[fit_df['k_label'] == k]['input_1'].mean()
            cent2 = fit_df[fit_df['k_label'] == k]['input_2'].mean()
            centroid[k] = (cent1, cent2)
        new_k_labels = []
        for idx, row in fit_df.iterrows():
            min_distance = 50
            for k, v in centroid.items():
                eu_distance = np.sqrt((row['input_1'] - v[0]) ** 2 +␣
↪(row['input_2'] -v[1]) ** 2)

                if eu_distance < min_distance:
                    min_distance = eu_distance
                    new_k = k
            new_k_labels.append(new_k)
        fit_df['k_label'] = new_k_labels
    return fit_df
```

[29]:
```
fit_k3 = fit_kmeans(3, df_k3)
fit_k3
```

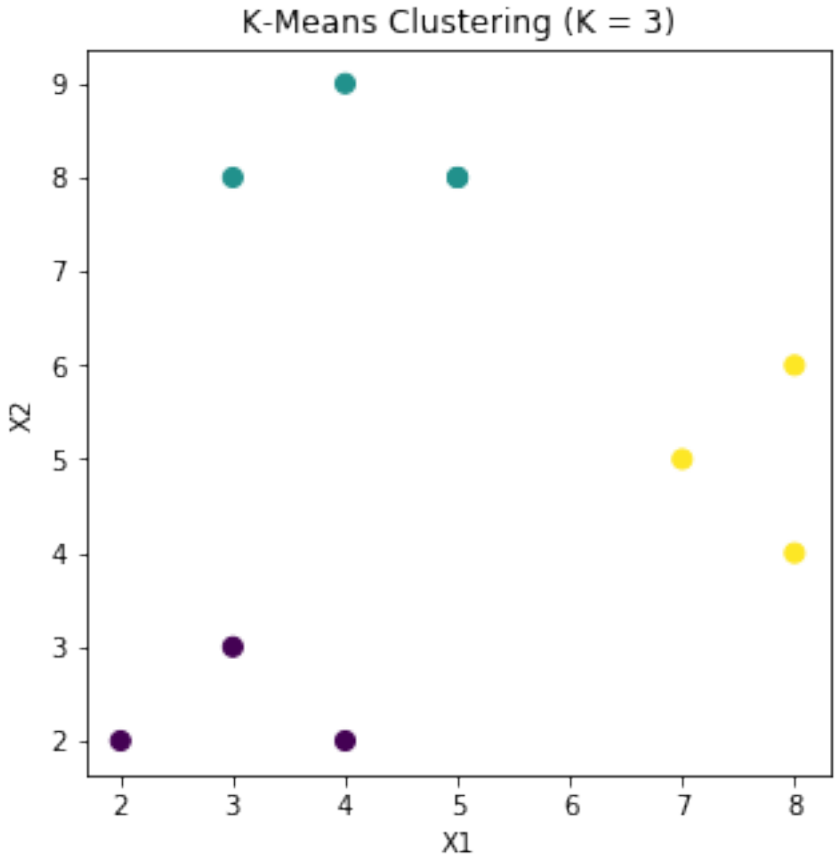
[29]:

	input_1	input_2	k_label
0	5	8	1
1	8	6	2
2	7	5	2
3	8	4	2
4	3	3	0
5	4	2	0
6	2	2	0
7	3	8	1
8	4	9	1

```
9         5         8         1
```

1.0.3 3. Present a visual description of the final, converged (stopped) cluster assignments.

[30]:
```
fig = plt.figure(figsize=(5, 5))
colors = list(fit_k3['k_label'])
plt.scatter(fit_k3['input_1'], fit_k3['input_2'], c=colors, s=50)
plt.xlabel('X1')
plt.ylabel('X2')
plt.title('K-Means Clustering (K = 3)')
plt.show()
```



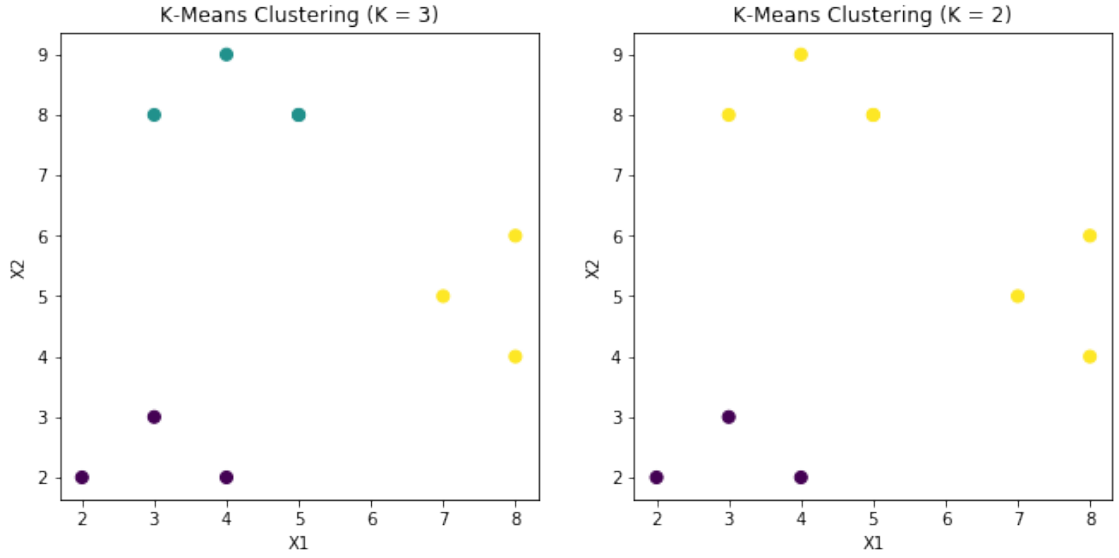
1.0.4 4. Now, repeat the process, but this time initialize at k = 2 and present a final cluster assignment visually next to the previous search at k = 3.

[31]:
```
fit_k2 = fit_kmeans(2, df_k3)
fit_k2
```

[31]:

	input_1	input_2	k_label
0	5	8	1
1	8	6	1
2	7	5	1
3	8	4	1
4	3	3	0
5	4	2	0
6	2	2	0
7	3	8	1
8	4	9	1
9	5	8	1

[32]:
```
fig, axs = plt.subplots(1, 2, figsize=(11,5))
axs[0].scatter(fit_k3['input_1'], fit_k3['input_2'], c=fit_k3['k_label'], s=50)
axs[0].set_xlabel('X1')
axs[0].set_ylabel('X2')
axs[0].set_title('K-Means Clustering (K = 3)')
axs[1].scatter(fit_k2['input_1'], fit_k2['input_2'], c=fit_k2['k_label'], s=50)
axs[1].set_xlabel('X1')
axs[1].set_ylabel('X2')
axs[1].set_title('K-Means Clustering (K = 2)');
```



1.0.5 5. Did your initial hunch of 3 clusters pan out, or would other values of k, like 2, fit these data better? Why or why not?

k=3 fits the data better, because if k=2, the within-cluster distance of yellow cluster would be as high as the cross-cluster distance between the purple cluster and the sub-part of yellow one on the right. Whereas when k=3, the two sub-parts of yellow cluster can be successfully teased apart and the within-cluster distance would be much shorter.

2 Application exercises

2.1 Dimension Reduction

[70]:
```
wiki_data = pd.read_csv('wiki.csv')
```

6. Perform PCA on the dataset and plot the observations on the first and second principal components. Describe your results, e.g., What variables appear strongly correlated on the first principal component?. What about the second principal component?

[79]:
```
wiki_std=StandardScaler().fit_transform(wiki_data)
```

[80]:
```
pca = PCA(n_components=2)
PComponents = pca.fit_transform(wiki_std)
PCDF = pd.DataFrame(pca.components_, columns = wiki_data.columns, index =␣
↪['PC1', 'PC2']).T
```

[87]:
```
PCDF.sort_values(by=['PC1'], ascending=False).head()
```

[87]:

	PC1	PC2
bi2	0.230924	0.083421
bi1	0.226193	0.056372
use3	0.218809	0.155147
use4	0.214558	0.160853
pu3	0.210863	0.028807

The five most correlated variables on the first component are: [‘pu3’, ‘use4’, ‘use3’, ‘bi1’, ‘bi2’]

[88]:
```
PCDF.sort_values(by=['PC2'], ascending=False).head()
```

[88]:

	PC1	PC2
exp4	0.099873	0.228485
use2	0.147852	0.218639
use1	0.181477	0.197807
vis3	0.175351	0.197626
domain_Engineering_Architecture	0.051309	0.171501

The five most correlated variables on the second component are: [‘exp4’, ‘use2’, ‘use1’, ‘vis3’ ,‘domain_Engineering_Architecture’]

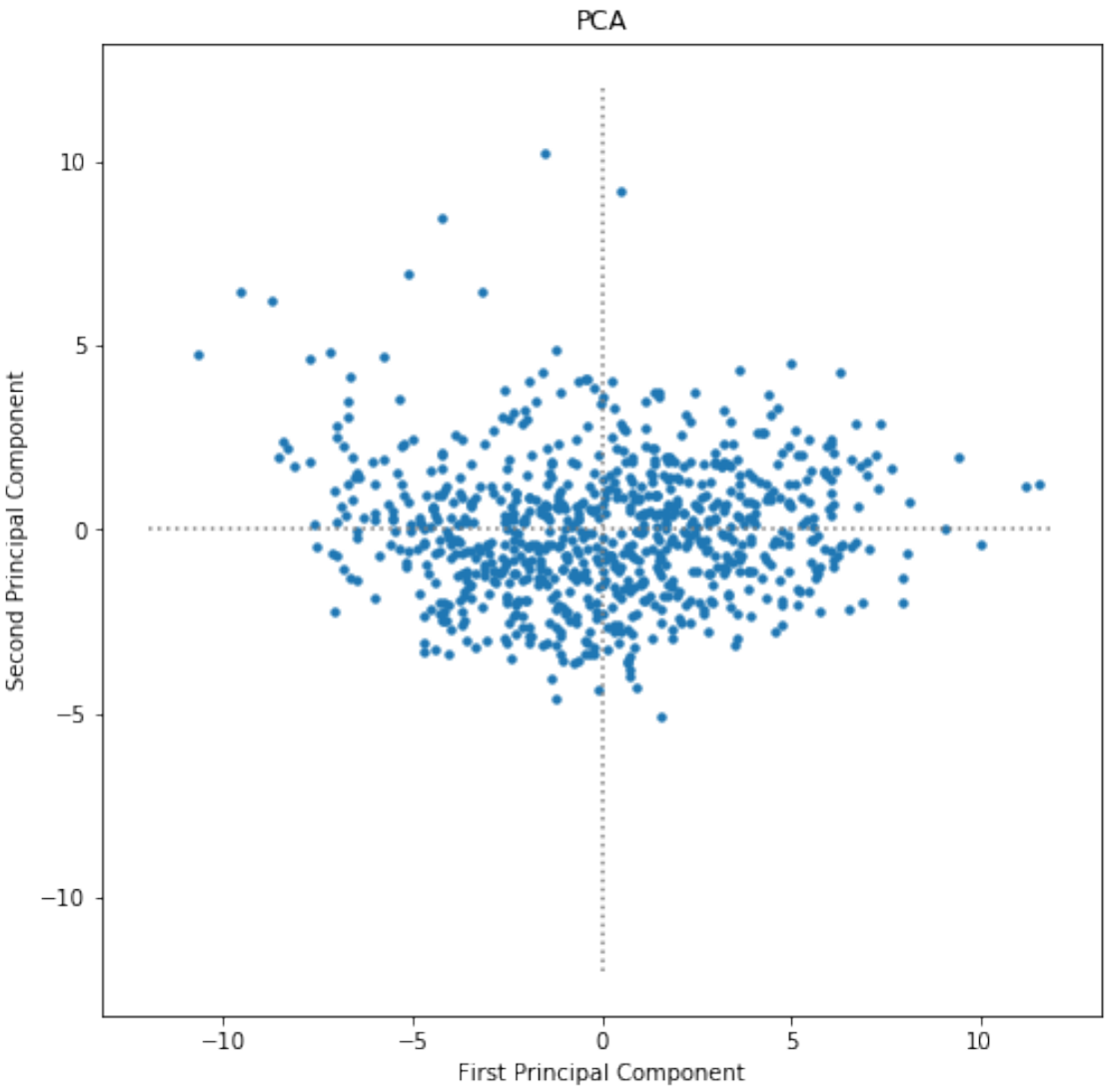
[92]:
```
pca = PCA(n_components=2)
PComponents = pca.fit_transform(wiki_std)
PCDF = pd.DataFrame(data = PComponents, columns = ['PC1', 'PC2'])
PCDF
```

[92]:

	PC1	PC2
0	-0.150216	-1.981942
1	-3.314020	-0.792198
2	-4.682484	-0.312170
3	1.774200	1.986144
4	7.254695	2.013318
..	…	…
795	0.227143	1.474476
796	4.434785	-0.931931
797	1.449455	-0.170549
798	-2.888282	2.720955
799	-7.000656	2.805160

[800 rows x 2 columns]

[93]:
```
plt.figure(figsize=(8, 8))
plt.scatter(PCDF['PC1'], PCDF['PC2'], s=11)
plt.xlabel('First Principal Component')
plt.ylabel('Second Principal Component')
plt.title('PCA')
plt.hlines(0,-12,12, linestyles='dotted', colors='grey')
plt.vlines(0,-12,12, linestyles='dotted', colors='grey');
```

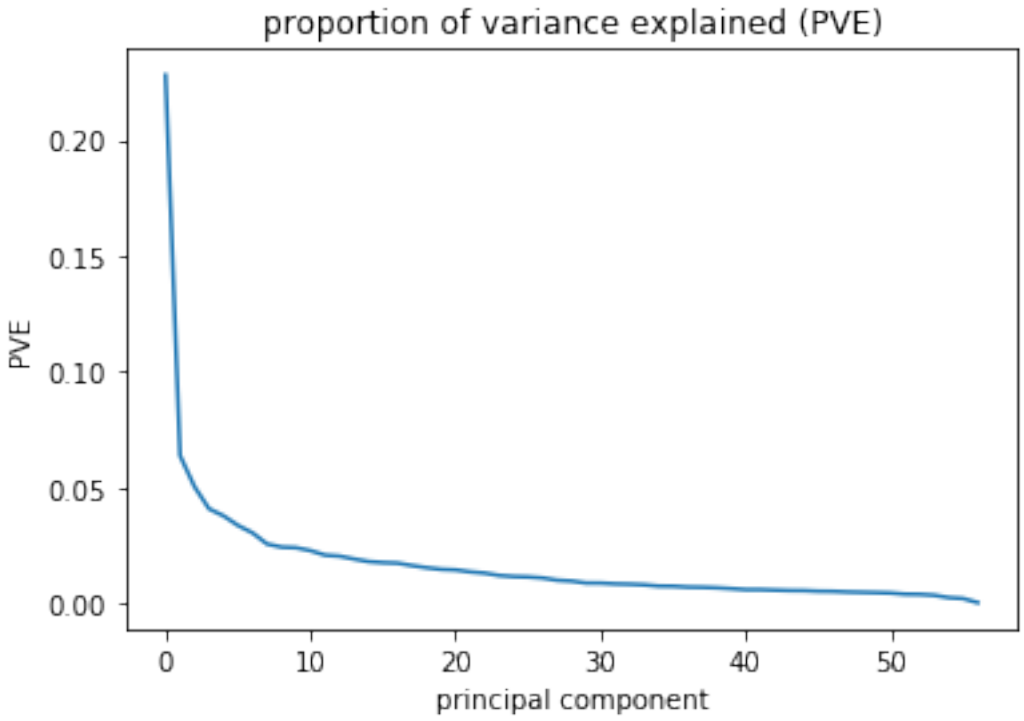


7. Calculate the proportion of variance explained (PVE) and the cumulative PVE for all the principal components. Approximately how much of the variance is explained by the first two principal components?

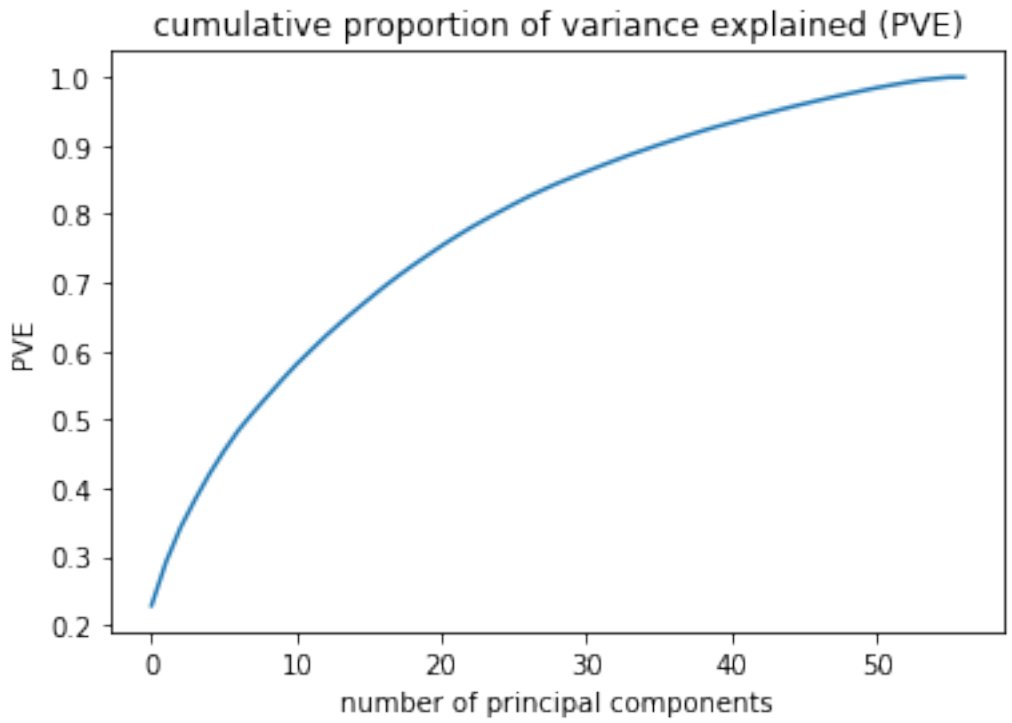
[96]:
```
wiki = wiki_data.values
wiki_trans = StandardScaler().fit(wiki).transform(wiki)
pca = PCA().fit(wiki_trans)
```

[97]:
```
plt.plot(range(len(pca.explained_variance_ratio_)), pca.
 ↪explained_variance_ratio_)
plt.title('proportion of variance explained (PVE)')
plt.xlabel('principal component')
plt.ylabel('PVE')
```

```
plt.show()
```



[98]:
```
plt.plot(range(len(pca.explained_variance_ratio_)), np.cumsum(pca.
 ↪explained_variance_ratio_))
plt.title('cumulative proportion of variance explained (PVE)')
plt.xlabel('number of principal components')
plt.ylabel('PVE')
plt.show()
```

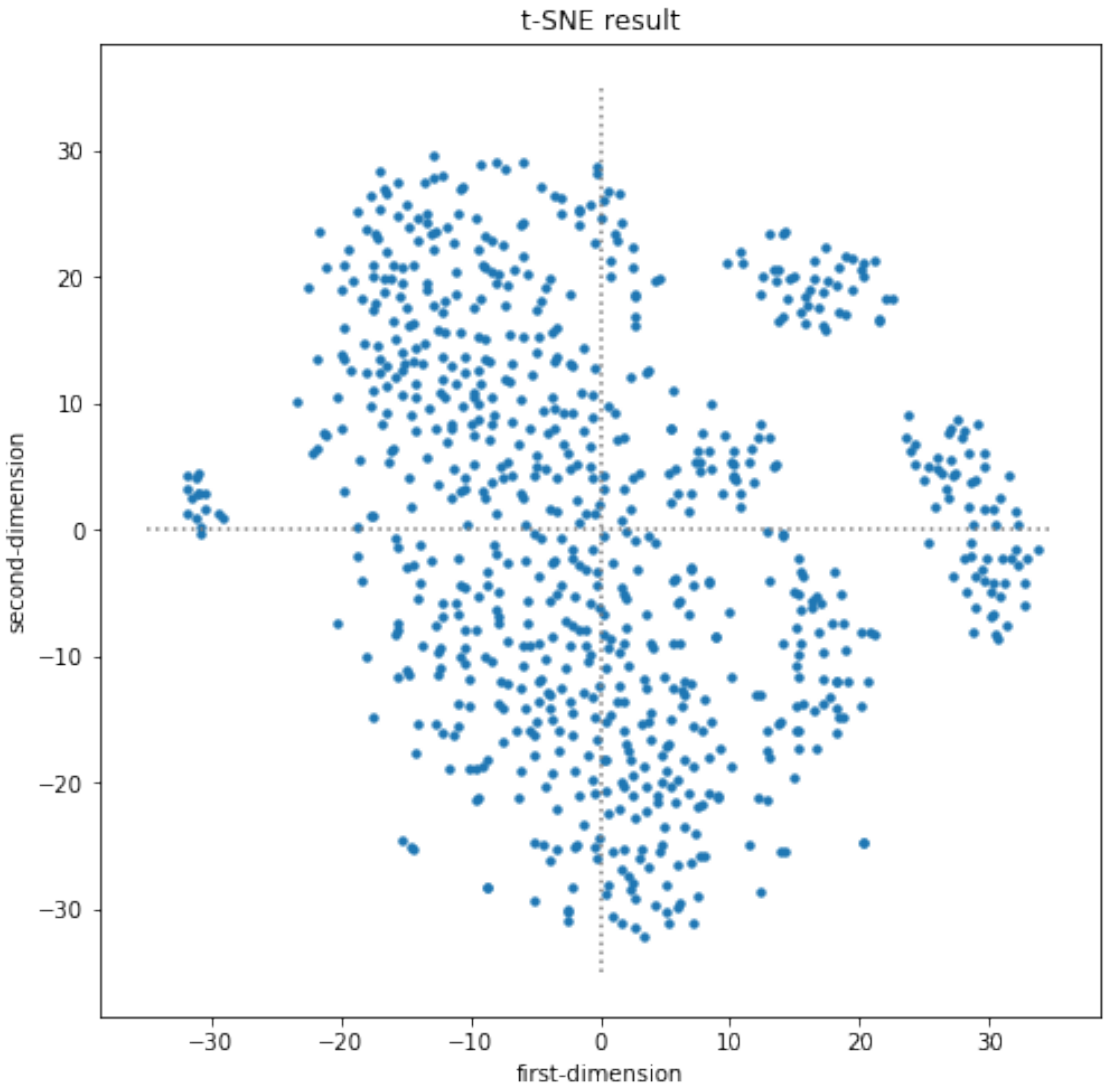


[99]:
```
print("The cimulative proportion of variance explained by the first two␣
↪components is: {:.2%}"
      .format(pca.explained_variance_ratio_[0]+pca.
↪explained_variance_ratio_[1]))
```

```
The cimulative proportion of variance explained by the first two components is:
29.18%
```

8. Perform t-SNE on the dataset and plot the observations on the first and second dimensions. Describe your results.

[48]:
```
wiki_embedded = TSNE(n_components=2).fit_transform(wiki_std)
fig = plt.figure(figsize=(8, 8))
pylab.scatter(wiki_embedded[:,0], wiki_embedded[:,1],s=11)
plt.xlabel('first-dimension')
plt.ylabel('second-dimension')
plt.title('t-SNE result')
plt.hlines(0,-35,35, linestyles='dotted', colors='grey')
plt.vlines(0,-35,35, linestyles='dotted', colors='grey')
plt.show()
```

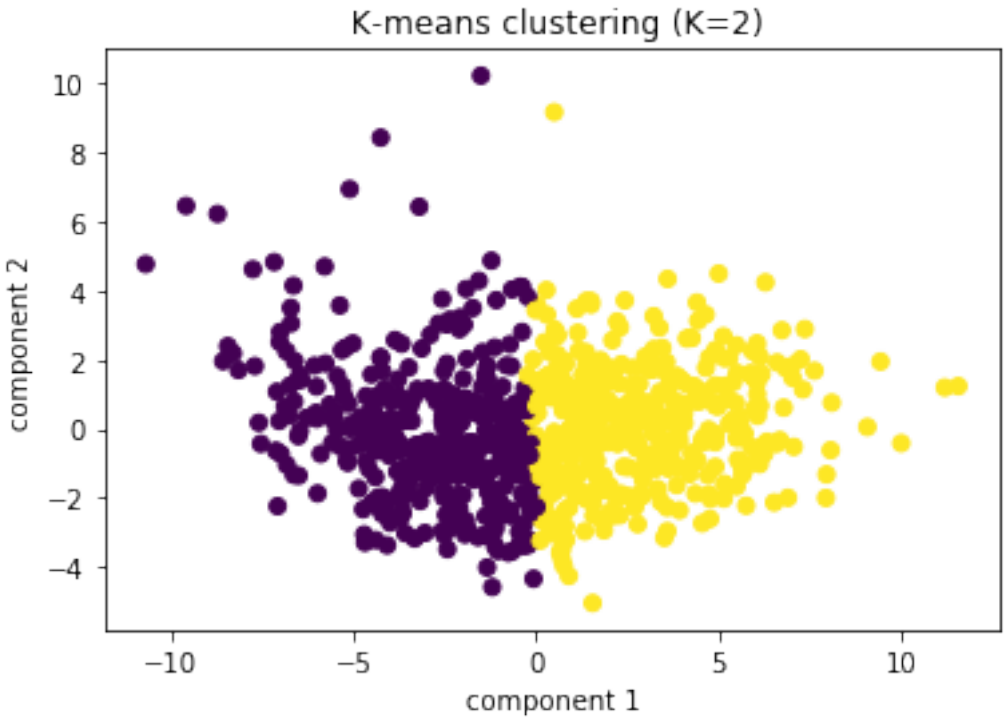


The t-SNE plot is scattered based on the first two principle components, showing a large group of points with smaller groups around. While in PCA graph, nearly all dots are in the same large group. This confirms t-SNE's advantage over PCA in terms of capturing relationships between features. Since t-SNE performs better dealing with non-linear relationships, we could make see their might be non-linear relationships between the variables.

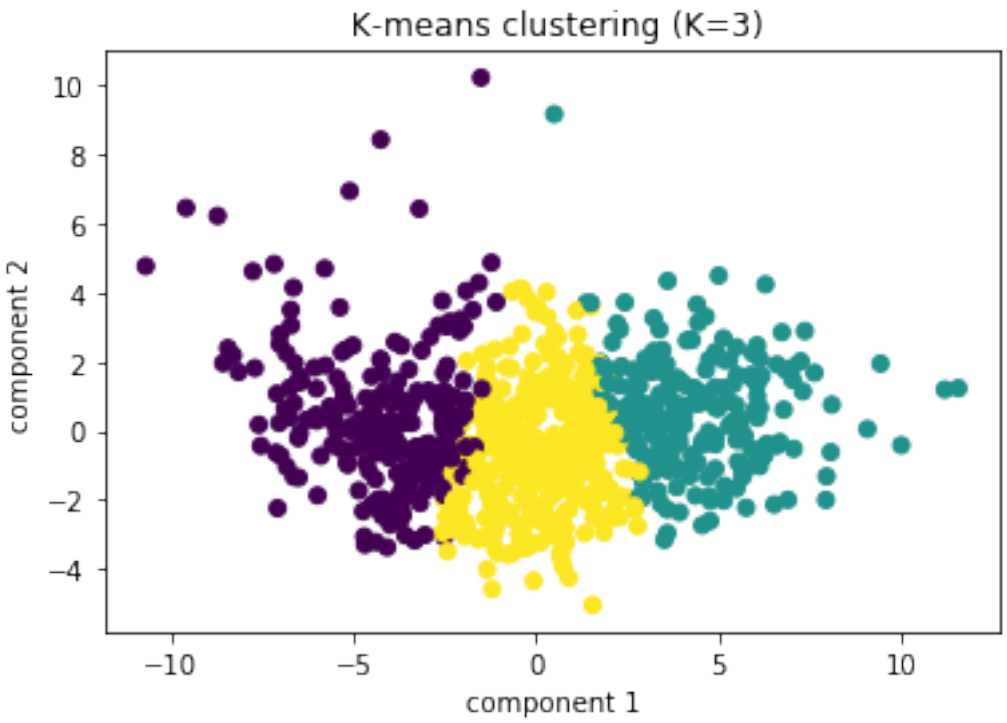
2.2 Clustering

9. Perform k-means clustering with $k = 2, 3, 4$. Be sure to scale each feature (i.e.,mean zero and standard deviation one). Plot the observations on the first and second principal components from PCA and color-code each observation based on their cluster membership. Discuss your results.

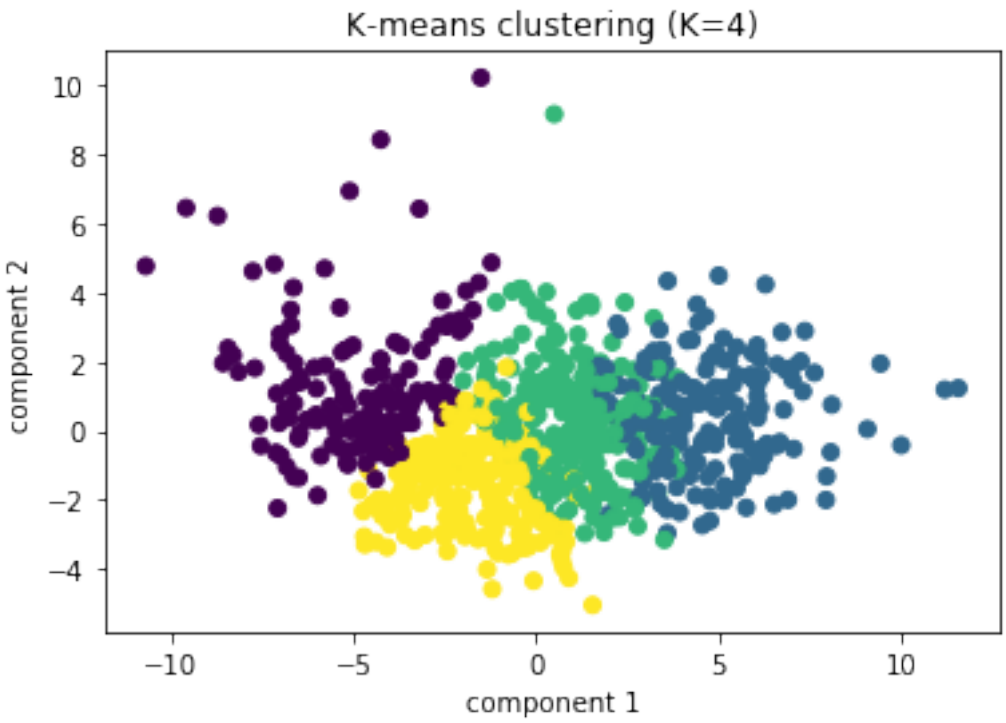
[119]:
```
score_vec = pca.fit_transform(wiki_trans)
kmeans = KMeans(n_clusters=2, random_state=0).fit(wiki_trans)
pylab.scatter(score_vec[:,0], score_vec[:,1], c=kmeans.predict(wiki_trans))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.title(' K-means clustering (K=2)')
plt.show()
```



[120]:
```
score_vec = pca.fit_transform(wiki_trans)
kmeans = KMeans(n_clusters=3, random_state=0).fit(wiki_trans)
pylab.scatter(score_vec[:,0], score_vec[:,1], c=kmeans.predict(wiki_trans))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.title(' K-means clustering (K=3)')
plt.show()
```



[121]:
```
score_vec = pca.fit_transform(wiki_trans)
kmeans = KMeans(n_clusters=4, random_state=0).fit(wiki_trans)
pylab.scatter(score_vec[:,0], score_vec[:,1], c=kmeans.predict(wiki_trans))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.title('K-means clustering (K=4)')
plt.show()
```



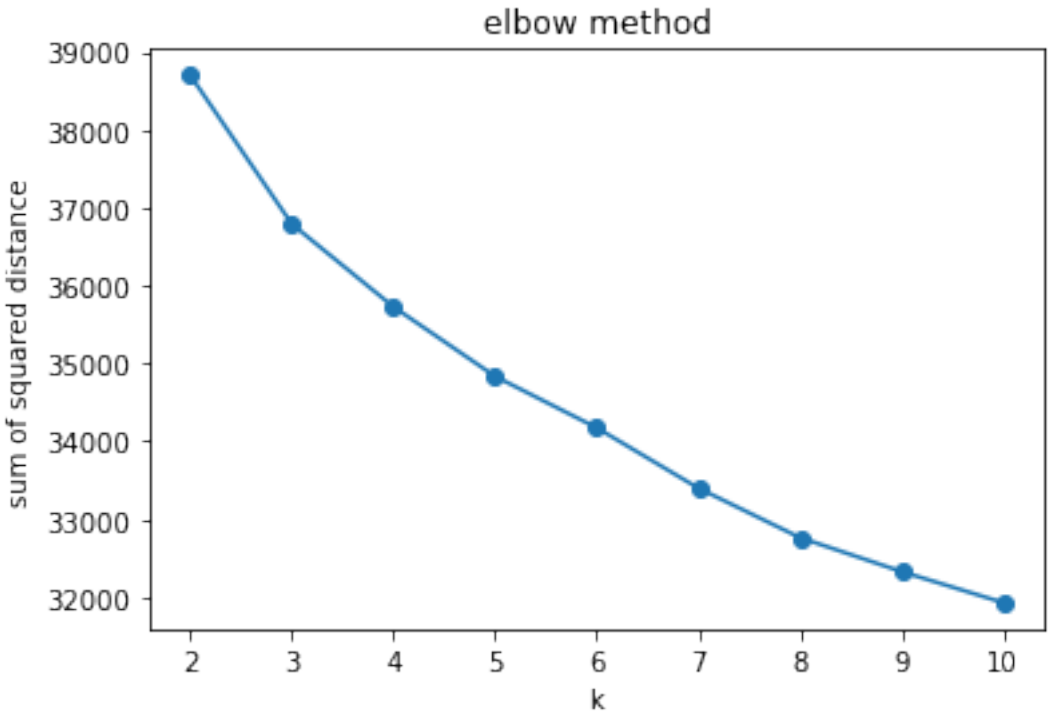
As we can see from the graphs above , the k-means clustering with k= 2 or 3 performs better to tease the data apart since they involve relatively clear boundary among each cluster, whereas when k=4, the clusters overlap and the boundaries among them becom fuzzy. Besides, the clustering is mainly based on the value of first component, while the second component only makes small contributions to the clustering.

10. Use the elbow method, average silhouette, and/or gap statistic to identify the optimal number of clusters based on k-means clustering with scaled features.

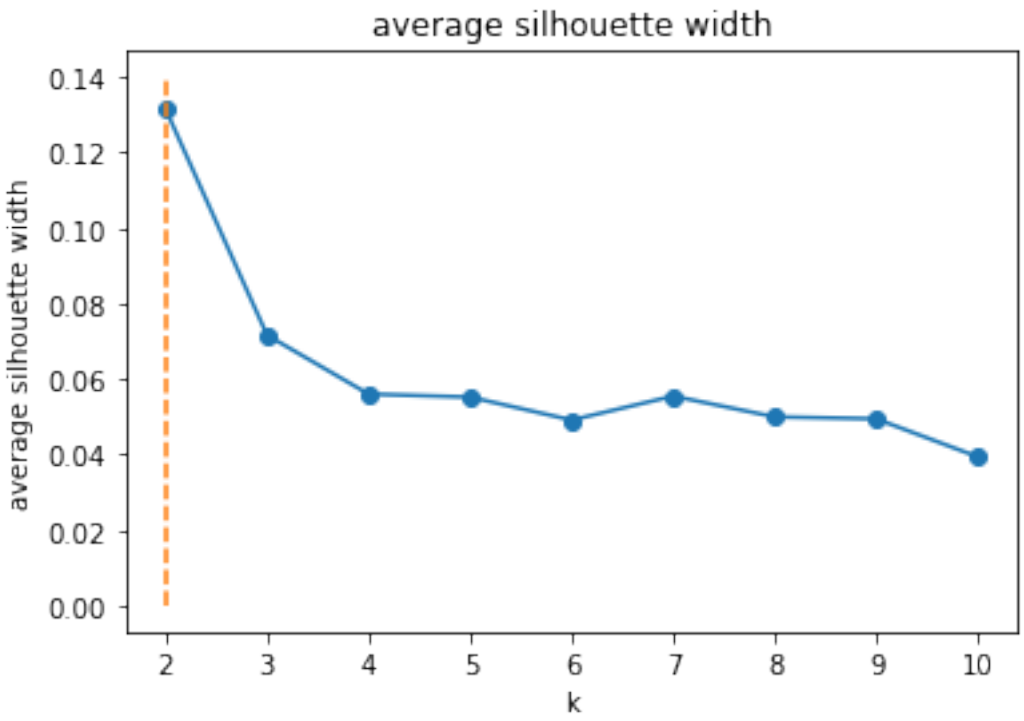
[122]:
```
k_lst = np.arange(2, 11, 1)
k_means = []
el_lst = []
silhouette_lst = []
for k in k_lst:
    km_tmp = KMeans(n_clusters=k, random_state=0).fit(wiki_trans)
    k_means.append(km_tmp)
    el_lst.append(km_tmp.inertia_)
    silhouette_lst.append(silhouette_score(wiki_trans, km_tmp.labels_,␣
 ↪metric='euclidean'))
```

[123]:
```
plt.plot(k_lst, el_lst, '-o')
plt.title('elbow method')
plt.xlabel('k')
plt.ylabel('sum of squared distance')
```

```
plt.show()
```



[124]:
```
plt.plot(k_lst, silhouette_lst, '-o')
plt.plot([k_lst[np.argmax(silhouette_lst)]]*2, [0, 0.14], linestyle = 'dashed')
plt.title('average silhouette width')
plt.xlabel('k')
plt.ylabel('average silhouette width')
plt.show()
```



Both the elbow method and average silhouette showed that K=2 is the optimal number of clusters.

[106]:
```
from gap_statistic import OptimalK
```

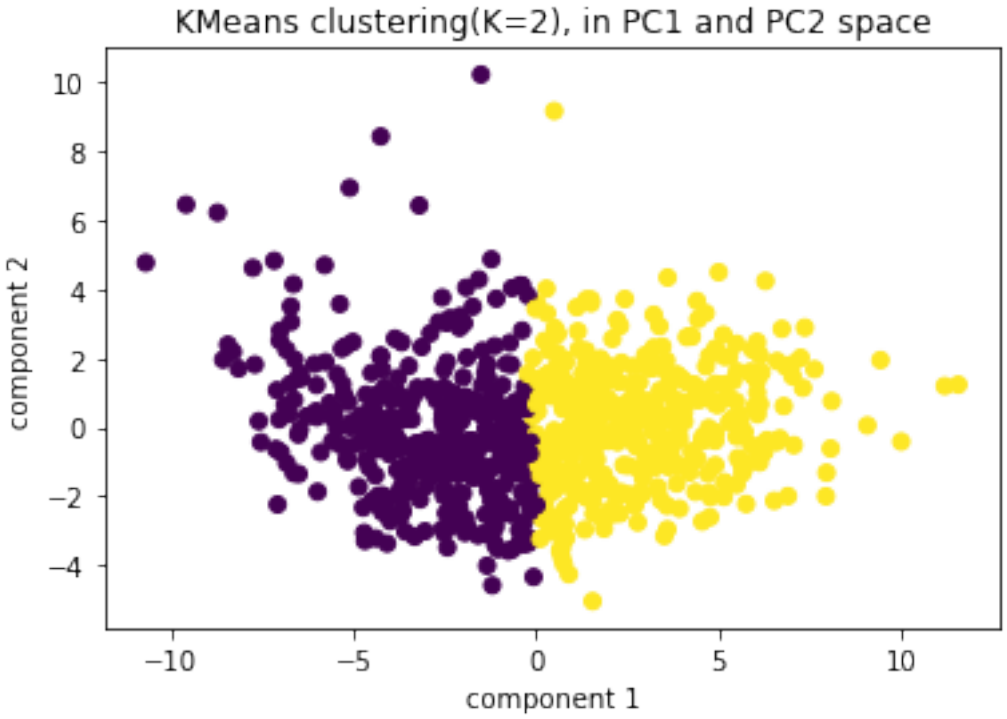
```
␣
↪---------------------------------------------------------------------------

       ModuleNotFoundError                       Traceback (most recent call␣
↪last)

       <ipython-input-106-edbf76be68db> in <module>
   ----> 1 from gap_statistic import OptimalK


       ModuleNotFoundError: No module named 'gap_statistic'
```

I cannot download gap_statistic in conda, so I didn't use it.

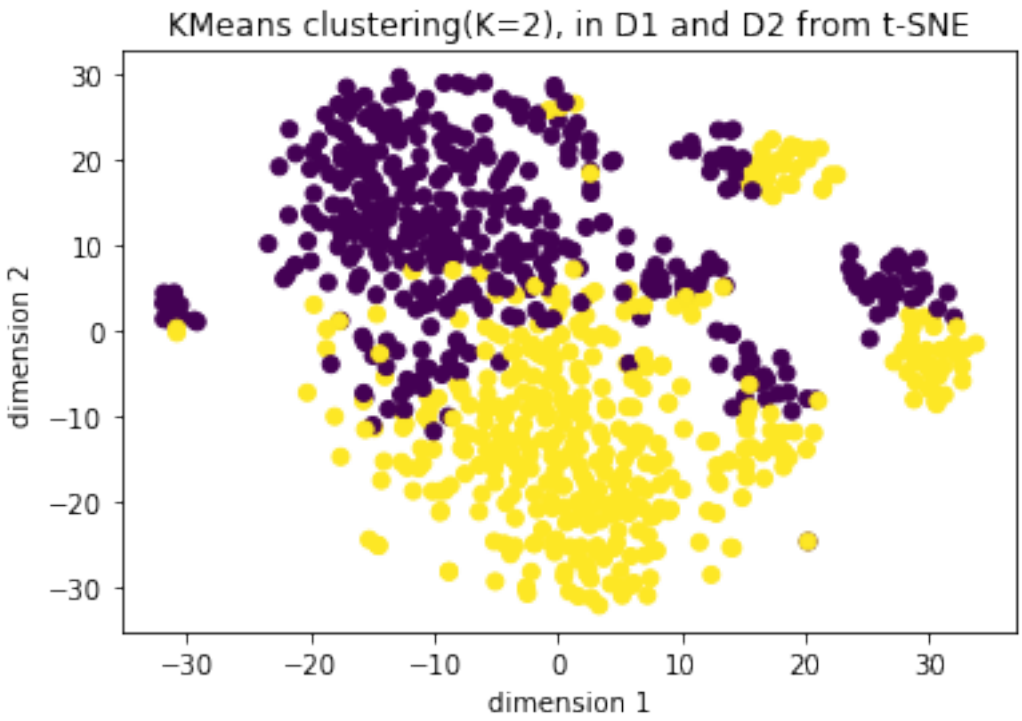
11. Visualize the results of the optimal $\hat{k}$-means clustering model. First use the first and second principal components from PCA, and color-code each observation based on their cluster membership. Next use the first and second dimensions from t-SNE, and color-code each observation based on their cluster membership. Describe your

results. How do your interpretations differ between PCA and t-SNE?

[117]:
```
pylab.scatter(score_vec[:,0], score_vec[:,1], c=k_means[0].predict(wiki_trans))␣
↪ #since we know the optimal k=2
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.title('KMeans clustering(K=2), in PC1 and PC2 space')
plt.show()
```



[118]:
```
pylab.scatter(wiki_embedded[:,0], wiki_embedded[:,1], c=k_means[0].
↪predict(wiki_trans))
plt.xlabel('dimension 1')
plt.ylabel('dimension 2')
plt.title('KMeans clustering(K=2), in D1 and D2 from t-SNE')
plt.show()
```



The PCA tends to cluster the data separated by a clearly linear boundary, whereas the t-SNE shape more complex structures, with a fuzze boundary and overlapping data. One possible explanation for the difference is the data follows a linear pattern, so PCA performs betterbetter. Besides, PCA preserves the large distances among points while t-SNE preserves the points close to each other, resulting in a main block in the middle and several satellite clusters dispersed around it, showing another disadvantage of t-SNE: it focuses too much on locality of data, leading to the ignorance of the global trends of the dataset.

[ ]: