Hu_Anqi_HW7

March 14, 2020

[1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.spatial import distance
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
```

0.1 k-means

[2]:
```python
np.random.seed(90210)

input_1 = [5,8,7,8,3,4,2,3,4,5]
input_2 = [8,6,5,4,3,2,2,8,9,8]
cats = [0, 1, 2]

inputs3 = pd.DataFrame({'input_1': input_1,
                        'input_2': input_2})
classes = np.random.choice(cats, 10)
inputs3['Class'] = classes

inputs3
```

[2]:

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

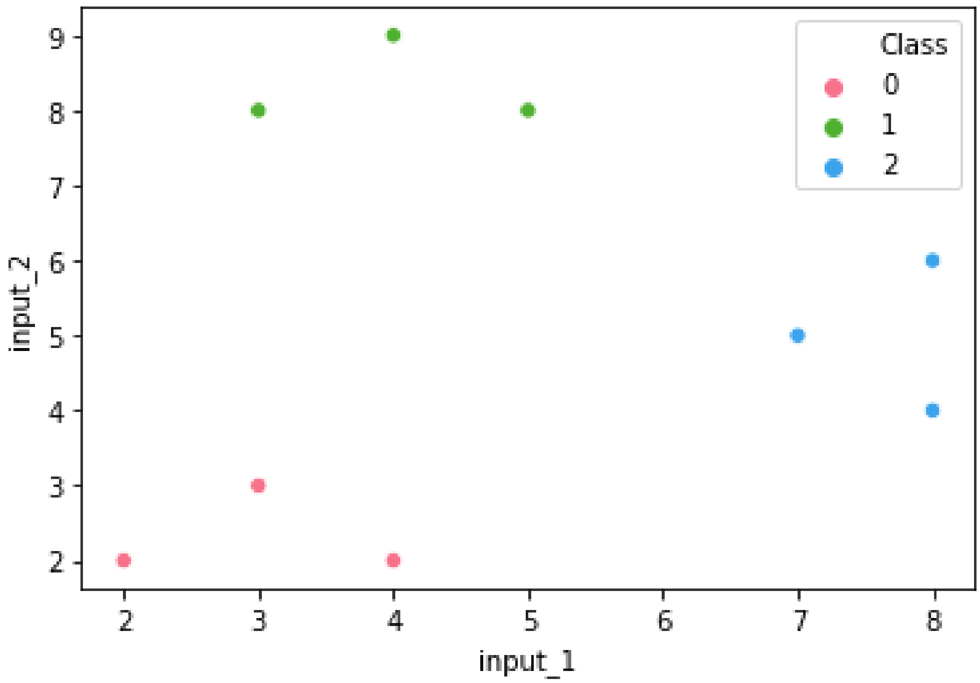
9	5	8	2

[3]:
```
for i in range(30):
    cent = {}
    for j in range(3):
        c1 = inputs3[inputs3['Class'] == j]['input_1'].mean()
        c2 = inputs3[inputs3['Class'] == j]['input_2'].mean()
        cent[j] = (c1, c2)
    new_class = []
    for val in inputs3.itertuples(index=False):
        min_dist = 50
        for key, c in cent.items():
            if distance.euclidean(c, (val[0], val[1])) < min_dist:
                min_dist = distance.euclidean(c, (val[0], val[1]))
                new_c = key
        new_class.append(new_c)

    if new_class == list(inputs3['Class']):
        break
    inputs3['Class'] = new_class

inputs3
```

[3]:

	input_1	input_2	Class
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

[4]:
```
sns.scatterplot('input_1', 'input_2',
                data=inputs3, hue = 'Class',
                palette=sns.color_palette("husl", 3), s=40);
```



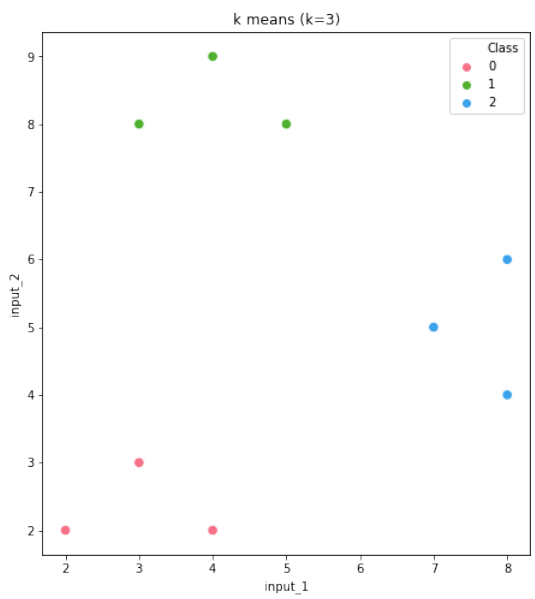
[5]:
```
input_1 = [5,8,7,8,3,4,2,3,4,5]
input_2 = [8,6,5,4,3,2,2,8,9,8]
cats = [0, 1]

inputs2 = pd.DataFrame({'input_1': input_1,
                        'input_2': input_2})
classes = np.random.choice(cats, 10)
inputs2['Class'] = classes

inputs2
```

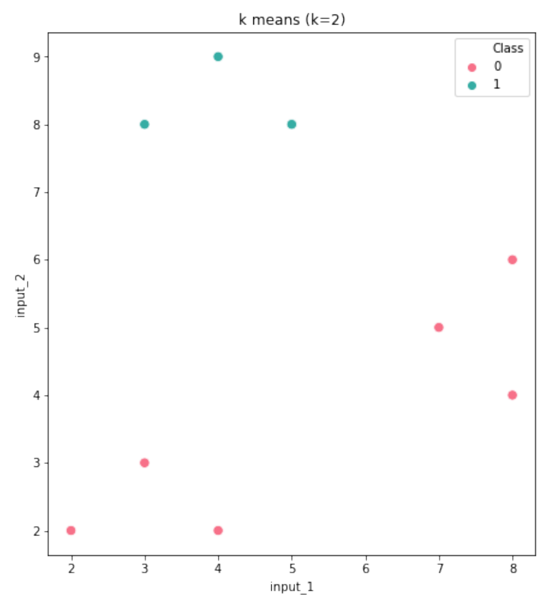
[5]:

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

[6]:
```
for i in range(30):
    cent = {}
    for j in range(2):
        c1 = inputs2[inputs2['Class'] == j]['input_1'].mean()
        c2 = inputs2[inputs2['Class'] == j]['input_2'].mean()
        cent[j] = (c1, c2)
    new_class = []
    for val in inputs2.itertuples(index=False):
        min_dist = 50
        for key, c in cent.items():
            if distance.euclidean(c, (val[0], val[1])) < min_dist:
                min_dist = distance.euclidean(c, (val[0], val[1]))
                new_c = key
        new_class.append(new_c)

    if new_class == list(inputs2['Class']):
        break
    inputs2['Class'] = new_class
```

[7]:
```
fig, axes=plt.subplots(1, 2, figsize=(16, 8))
sns.scatterplot('input_1', 'input_2',
                data=inputs3, hue = 'Class', ax=axes[0],
                palette=sns.color_palette("husl", 3), s=70)
axes[0].set_title('k means (k=3)')
sns.scatterplot('input_1', 'input_2',
                data=inputs2, hue = 'Class', ax=axes[1],
                palette=sns.color_palette("husl", 2), s=70)
axes[1].set_title('k means (k=2)');
```





Comparing the two k-means clusters, it seems like k=3 fitted the data points better, as the three groups of nodes with different colors belong in three distinct clusters, whereas the division when k=2 is less ideal and clear.

0.2 Application

0.2.1 Dimension reduction

[8]:
```
wiki = pd.read_csv('data/wiki.csv')
features = wiki.columns
num_fea = len(features)
wiki = StandardScaler().fit_transform(wiki)
```

[9]:
```
pca = PCA(n_components=num_fea)
pcs = pca.fit_transform(wiki)
colnames = ['PC' + str(i + 1) for i in range(num_fea)]
loadings = pd.DataFrame(pca.components_.T,
                        columns=colnames,
                        index=features)
```

[10]:
```
loadings.sort_values(by='PC1', ascending=False)[['PC1', 'PC2']][:5]
```

[10]:

	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

In the first principal component, `bi2`, `bi1`, `use3`, `use4`, and `pu3` are the top five strongly correlated variables.

[11]:
```
loadings.sort_values(by='PC2', ascending=False)[['PC1', 'PC2']][:5]
```

[11]:

	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

In the second principal component, `exp4`, `use2`, `use1`, `vis3`, and `domain_Engineering_Architecture` are the top five strongly correlated variables.

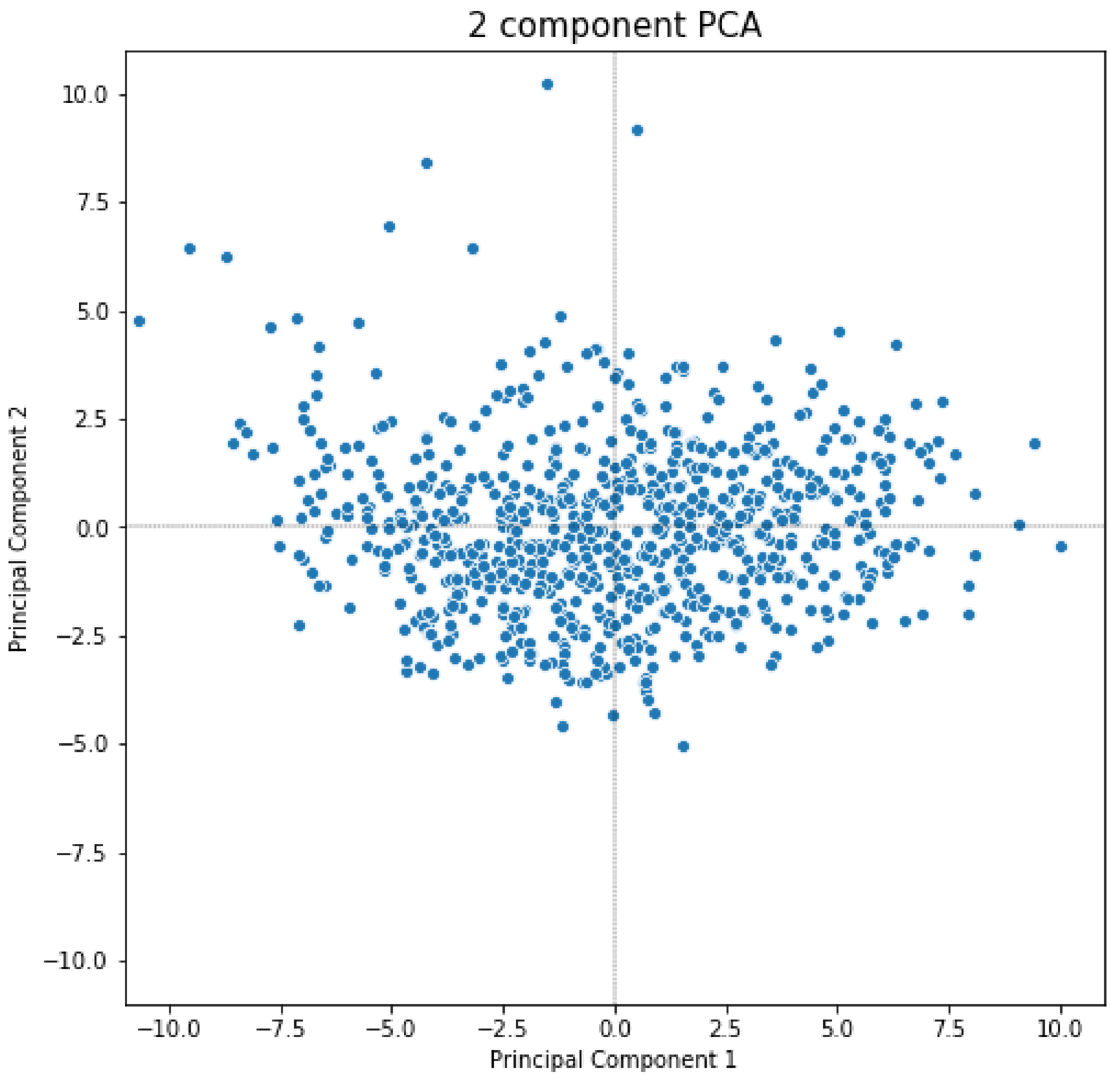
[12]:
```
pc_df = pd.DataFrame(data=pcs, columns=colnames)
pc_df[['PC1', 'PC2']]
```

[12]:

```
          PC1       PC2
0   -0.150216 -1.982012
1   -3.314020 -0.791963
2   -4.682484 -0.312449
3    1.774200  1.985882
4    7.254695  2.013041
..        …         …
795  0.227143  1.474271
796  4.434784 -0.931830
797  1.449455 -0.170542
798 -2.888282  2.721003
799 -7.000656  2.805396

[800 rows x 2 columns]
```

[13]:

```
fig, ax = plt.subplots(1, figsize=(8,8))
sns.scatterplot('PC1', 'PC2', data=pc_df)
plt.vlines(0, -11, 11, linestyles='dashed', linewidth=0.4)
plt.hlines(0, -11, 11, linestyles='dashed', linewidth=0.4)
plt.xlim(-11, 11)
plt.ylim(-11, 11)
ax.set_xlabel('Principal Component 1', fontsize = 10)
ax.set_ylabel('Principal Component 2', fontsize = 10)
ax.set_title('2 component PCA', fontsize = 15);
```



[14]:
```
pve = pca.explained_variance_ratio_
np.cumsum(pve)[:2]
```

[14]:
```
array([0.22810628, 0.29183102])
```

Between the first two components, about 29% of the variance in the data is explained.

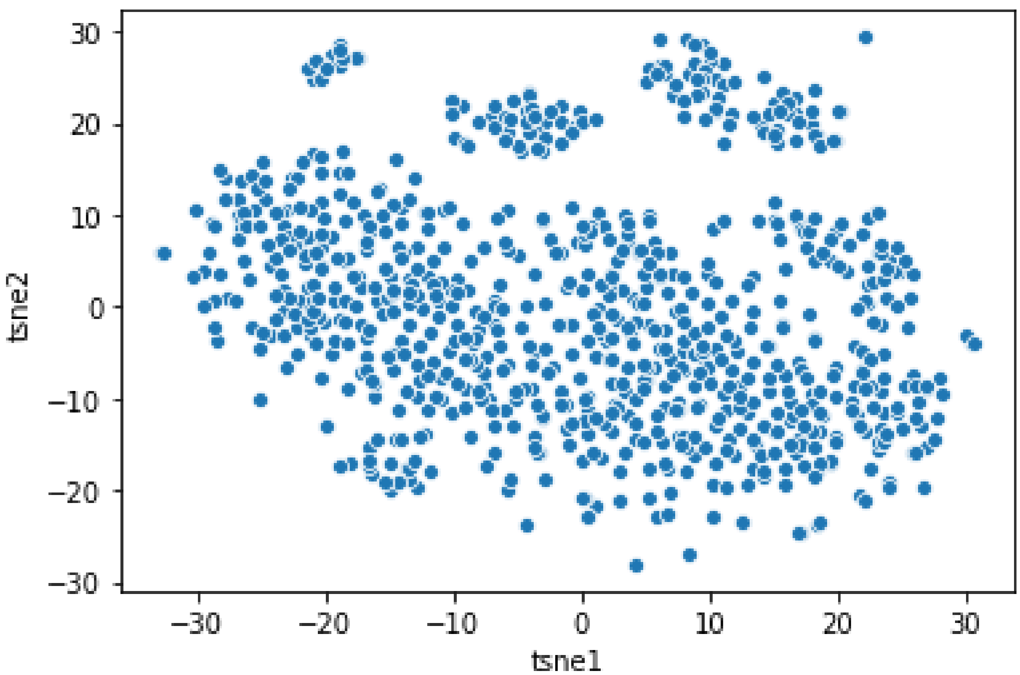
[15]:
```
tsne = TSNE(n_components=2).fit_transform(wiki)
tsne_df = pd.DataFrame(data=tsne, columns=['tsne1', 'tsne2'])

tsne_df
```

[15]:

	tsne1	tsne2
0	11.107976	17.774885

```
1     8.292616  25.501152
2    -9.281201  21.789764
3    -0.168578  20.641996
4    28.036818  -7.623133
..         …          …
795  13.022017   2.306484
796   5.907754 -13.526694
797   2.372066 -11.475722
798  -3.067989   9.576046
799 -21.863470  15.773821

[800 rows x 2 columns]
```

[16]:
```
sns.scatterplot('tsne1', 'tsne2', data=tsne_df);
```

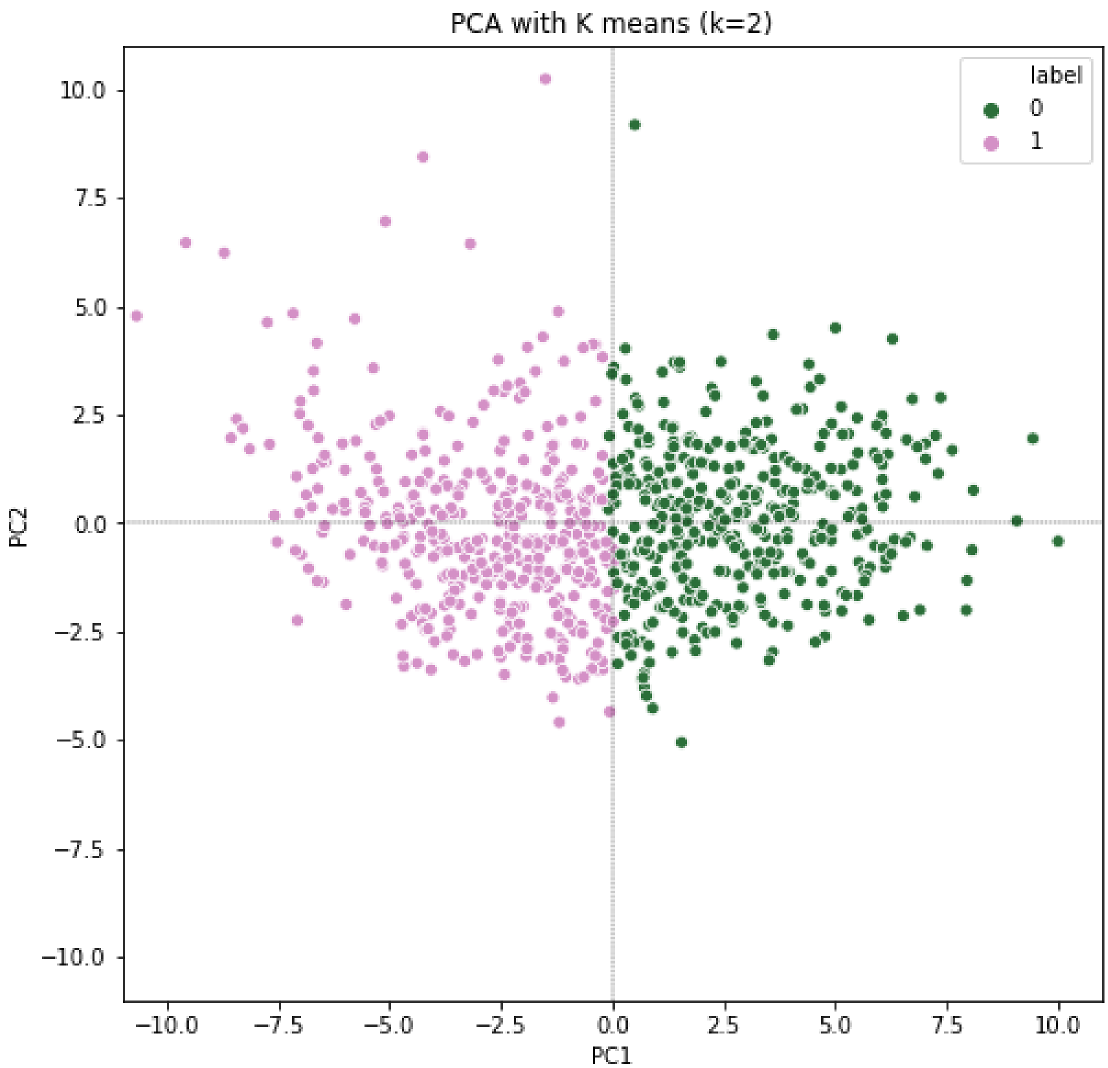


Compared to PCA, T-SNE seems to be better at capturing the complexity of higher dimensional models.

0.2.2 Clustering

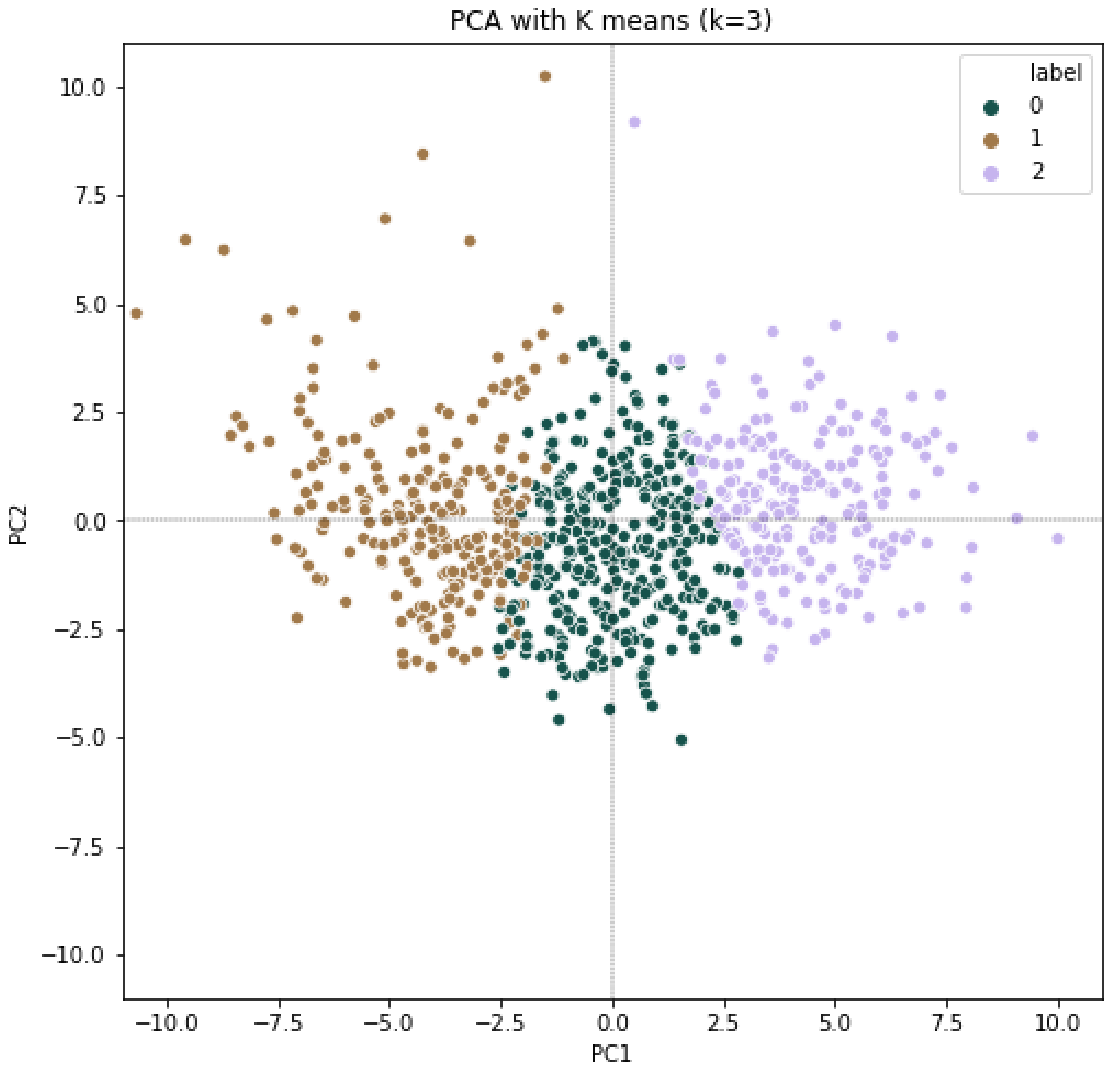
[17]:
```
kmeans2 = KMeans(n_clusters=2).fit(wiki)
new_df = pd.DataFrame({'PC1': pc_df['PC1'],
                       'PC2': pc_df['PC2'],
                       'label': kmeans2.labels_})
```

```
plt.figure(figsize=(8,8))
sns.scatterplot('PC1', 'PC2', data=new_df, hue='label',
                palette=sns.color_palette("cubehelix", 2))
plt.vlines(0, -11, 11, linestyles='dashed', linewidth=0.4)
plt.hlines(0, -11, 11, linestyles='dashed', linewidth=0.4)
plt.xlim(-11, 11)
plt.ylim(-11, 11)
plt.title('PCA with K means (k=2)');
```



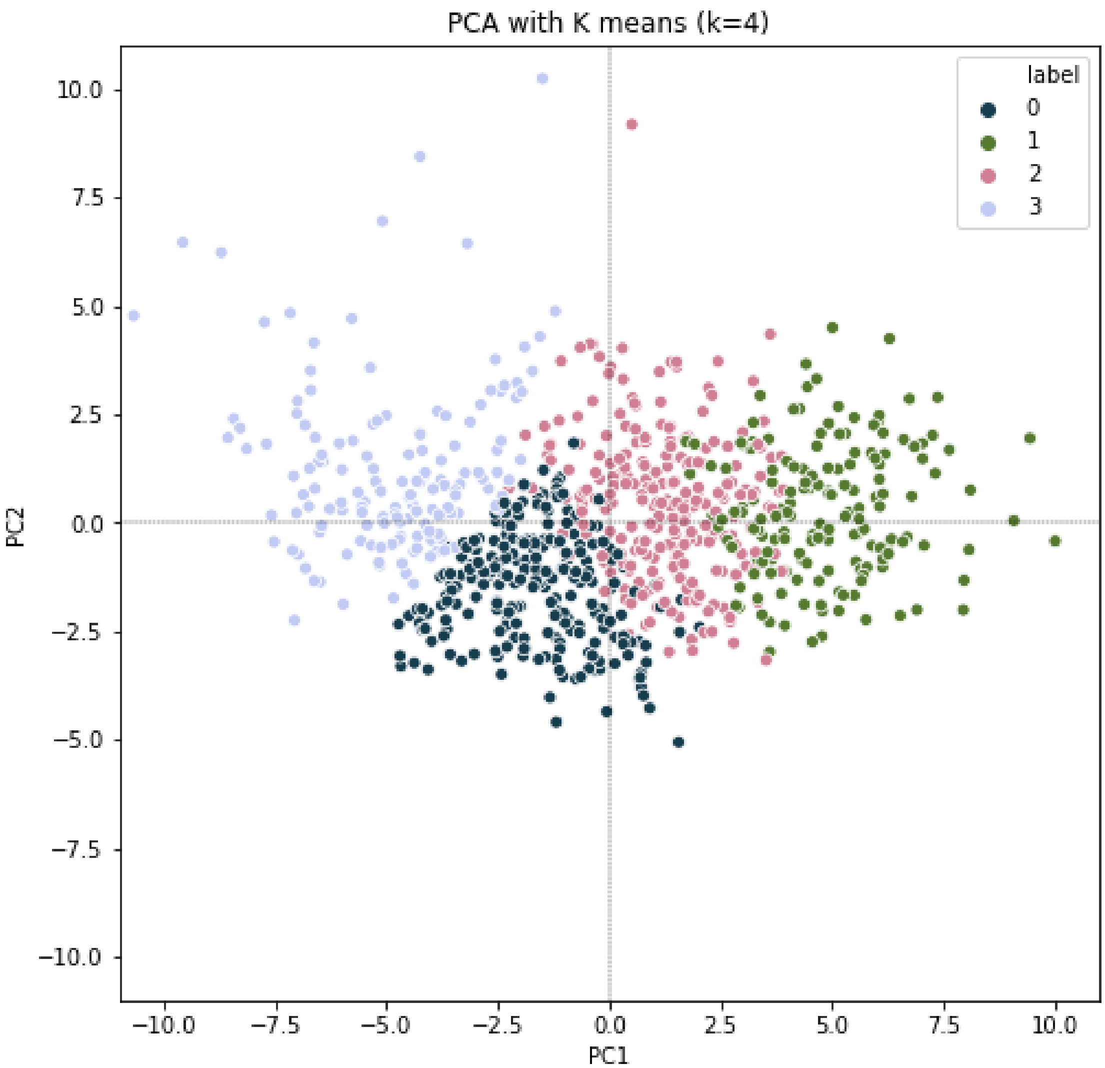
[18]:
```
kmeans3 = KMeans(n_clusters=3).fit(wiki)
new_df = pd.DataFrame({'PC1': pc_df['PC1'],
                       'PC2': pc_df['PC2'],
                       'label': kmeans3.labels_})
```

```
plt.figure(figsize=(8,8))
sns.scatterplot('PC1', 'PC2', data=new_df, hue='label',
                palette=sns.color_palette("cubehelix", 3))
plt.vlines(0, -11, 11, linestyles='dashed', linewidth=0.4)
plt.hlines(0, -11, 11, linestyles='dashed', linewidth=0.4)
plt.xlim(-11, 11)
plt.ylim(-11, 11)
plt.title('PCA with K means (k=3)');
```




```
[19]: kmeans4 = KMeans(n_clusters=4).fit(wiki)
new_df = pd.DataFrame({'PC1': pc_df['PC1'],
                       'PC2': pc_df['PC2'],
                       'label': kmeans4.labels_})
plt.figure(figsize=(8,8))
```

```
sns.scatterplot('PC1', 'PC2', data=new_df, hue='label',
                palette=sns.color_palette("cubehelix", 4))
plt.vlines(0, -11, 11, linestyles='dashed', linewidth=0.4)
plt.hlines(0, -11, 11, linestyles='dashed', linewidth=0.4)
plt.xlim(-11, 11)
plt.ylim(-11, 11)
plt.title('PCA with K means (k=4)');
```



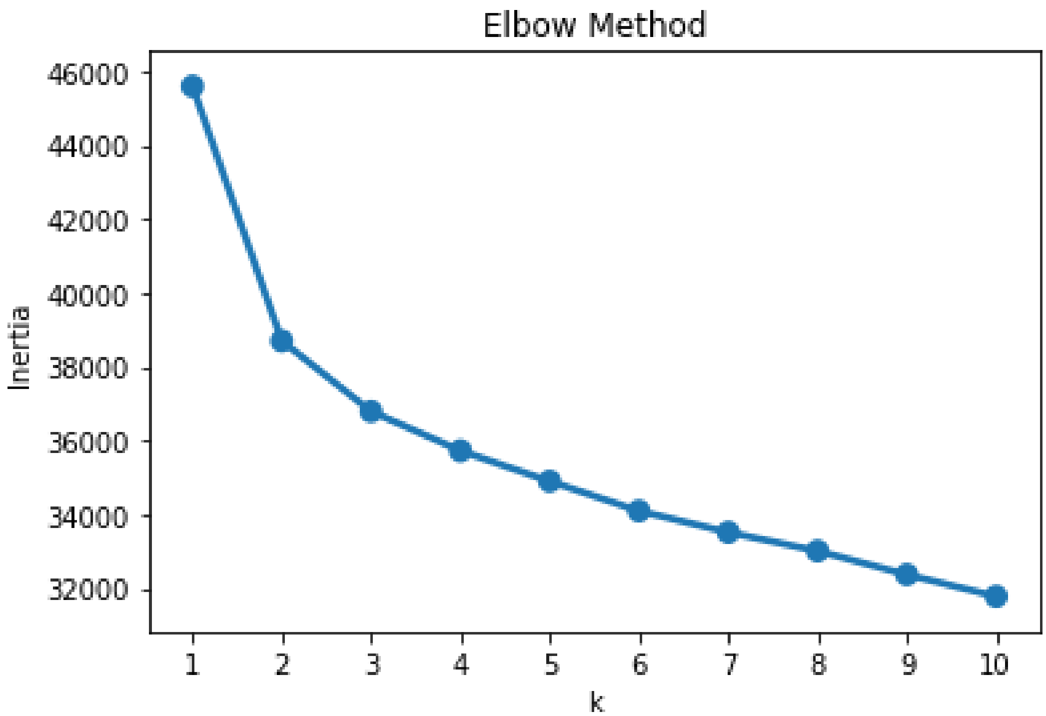
Using k=2,3,4 for k-means PCA, it is clear that as k increases, the clusters are overlapping more and more, and that the cluster boundaries are becoming less distinguishable.

[20]:
```
vals = []
for k in range(10):
    k_mod = KMeans(n_clusters=k+1).fit(wiki)
```

```
    vals.append(k_mod.inertia_)
```

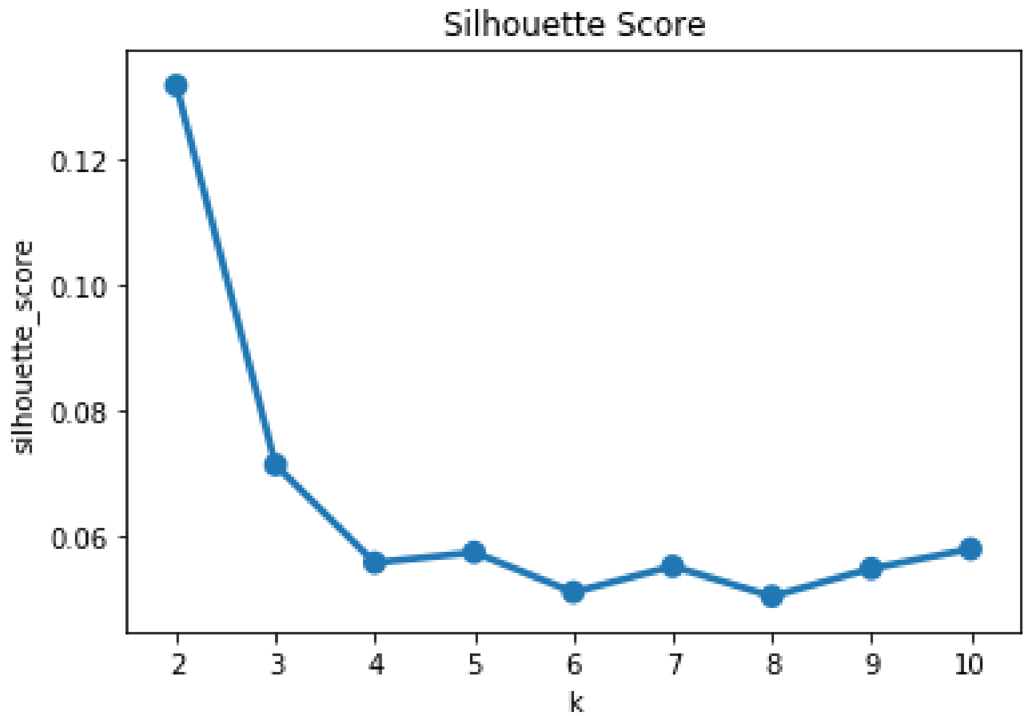
[21]:
```
vals_df = pd.DataFrame({'k': list(range(1, 11)),
                        'Inertia': vals})
```

[22]:
```
sns.pointplot('k', 'Inertia', data=vals_df)
plt.title('Elbow Method');
```



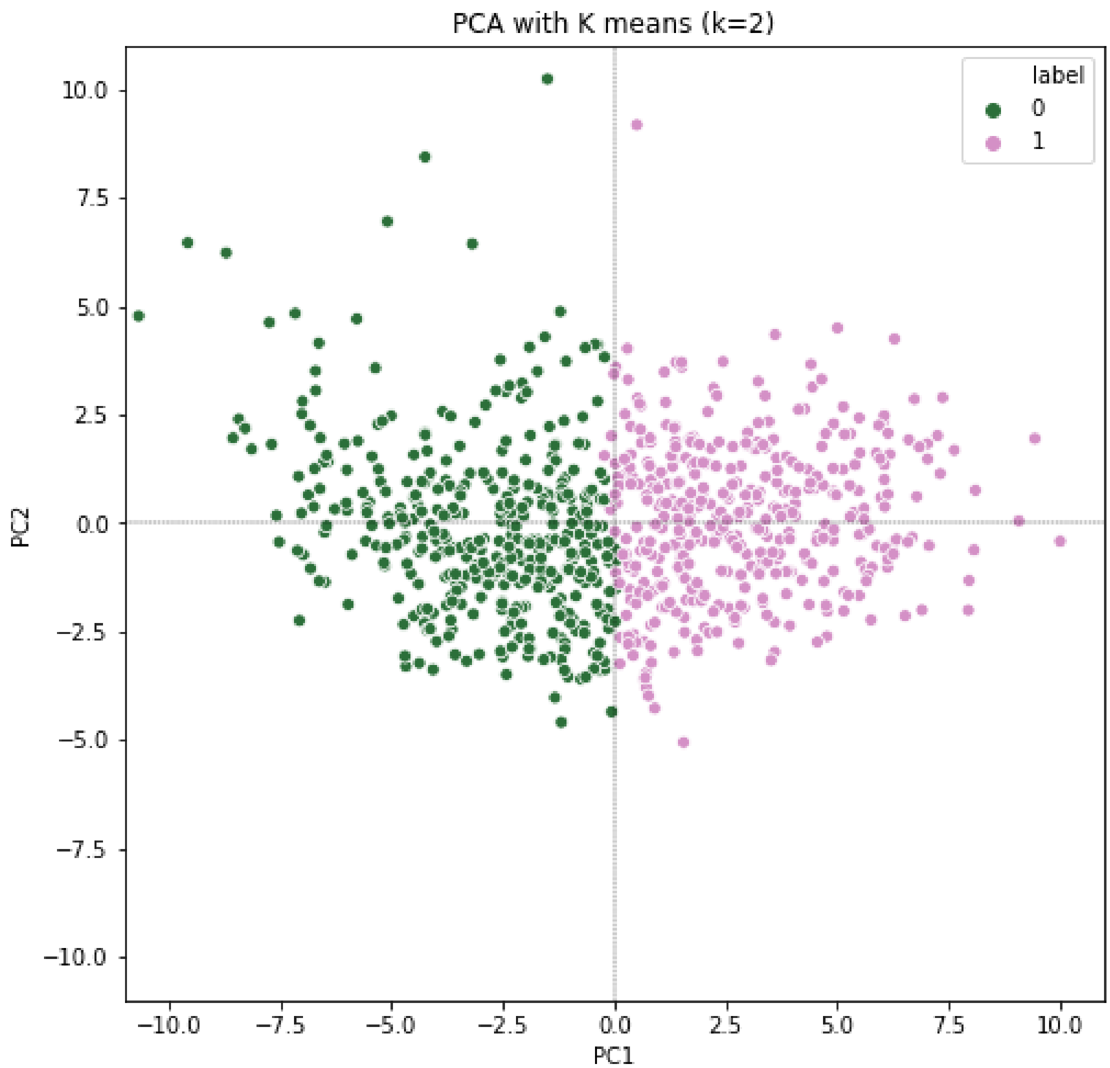
[23]:
```
sil_scores = []
for k in range(9):
    k_mod = KMeans(n_clusters=k+2).fit(wiki)
    sil_scores.append(silhouette_score(wiki, k_mod.labels_))
```

[24]:
```
vals_df = pd.DataFrame({'k': list(range(2, 11)),
                        'silhouette_score': sil_scores})

sns.pointplot('k', 'silhouette_score', data=vals_df)
plt.title('Silhouette Score');
```

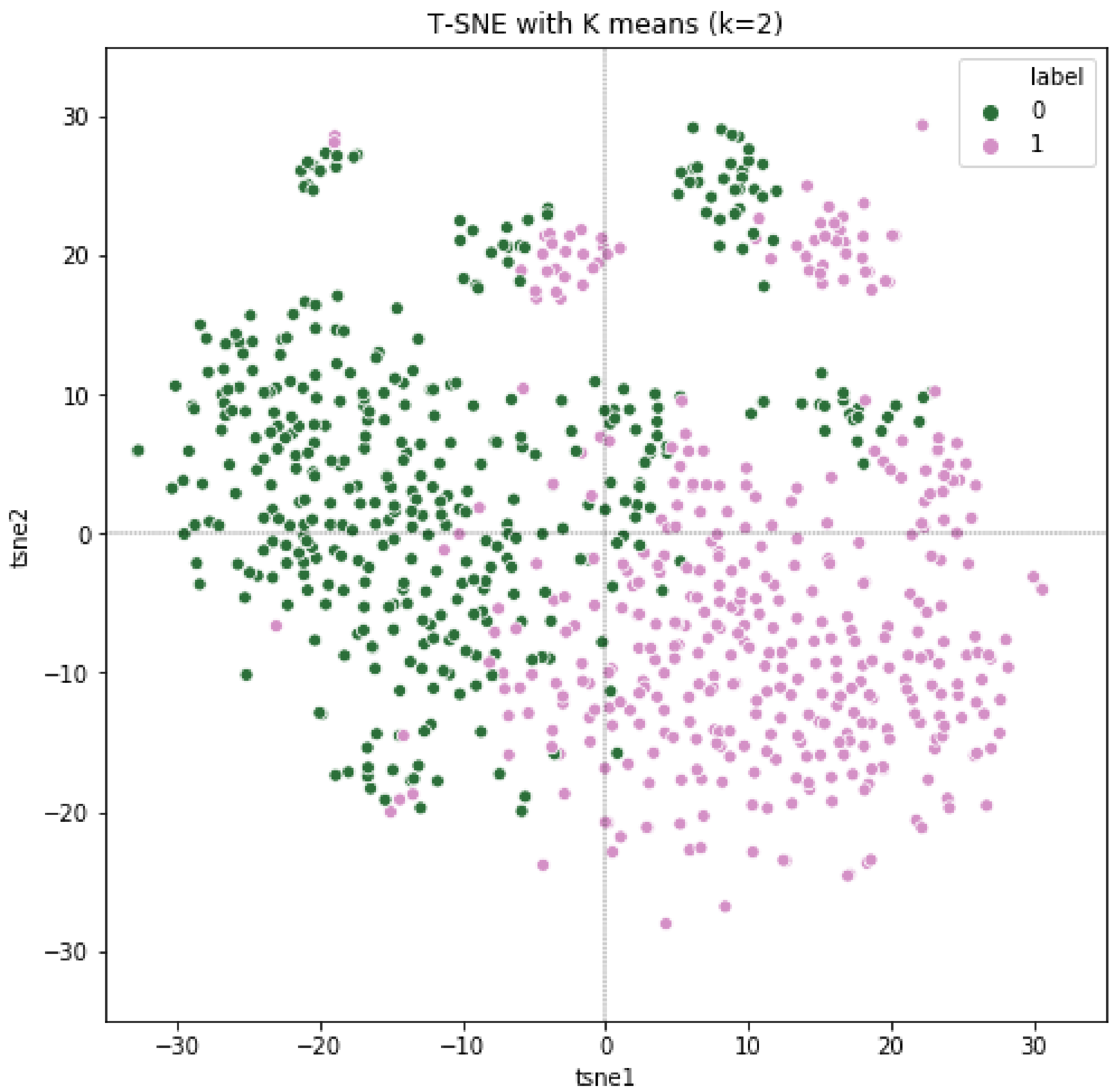


Using both the elbow method and average silhouette scores, we can see that k=2 is the most optimal number of clusters for this dataset. Using the elbow method, the inertia of the k-means model drops the greatest in magnitude when it shifts from k=1 to k=2. Judging by the silhouette scores, the score is the highest when k=2. Thus, k=2 should be the most ideal number of clusters.

[25]:
```
kmeans2 = KMeans(n_clusters=2).fit(wiki)
new_df = pd.DataFrame({'PC1': pc_df['PC1'],
                       'PC2': pc_df['PC2'],
                       'label': kmeans2.labels_})

plt.figure(figsize=(8,8))
sns.scatterplot('PC1', 'PC2', data=new_df, hue='label',
               palette=sns.color_palette("cubehelix", 2))
plt.vlines(0, -11, 11, linestyles='dashed', linewidth=0.4)
plt.hlines(0, -11, 11, linestyles='dashed', linewidth=0.4)
plt.xlim(-11, 11)
plt.ylim(-11, 11)
plt.title('PCA with K means (k=2)');
```



[26]:
```
tsne_df['label'] = kmeans2.labels_

plt.figure(figsize=(8,8))
sns.scatterplot('tsne1', 'tsne2', data=tsne_df, hue='label',
                palette=sns.color_palette("cubehelix", 2))
plt.vlines(0, -35, 35, linestyles='dashed', linewidth=0.4)
plt.hlines(0, -35, 35, linestyles='dashed', linewidth=0.4)
plt.xlim(-35, 35)
plt.ylim(-35, 35)
plt.title('T-SNE with K means (k=2)');
```



In PCA, it seems like the first principal component is a lot more important than the second one, as the cluster boundary is clearly a vertical one. On the other hand, T-SNE with k-means shows that the first and second dimensions are more equally important. The boundary is overlapping and much less clearcut.