Lab2

October 2, 2022

0.1 Lab 2: Clustering

0.1.1 Alice Wu. ID: qichaow

By applying different clustering methods, I think the **hierarchical method** is more flexible in trying different distance metrics. It is also easy to visualize and find the optimal hierachirial clusters by using the dendrogram. **K-means clustering** is good for numerical data, but for the categorical data, calulating the means of clusters might not be a good way to differntiate the data points.

Here, I also tried **K-modes clustering**, by finding the mismatch between variables and update the mode of these mismatches. I think this method makes more sense for categorical data by finidng the mismatch and updating the cluster with the modes.

Finally, I used **agglomerative clustering** with two different distance metrics: *Euclidean* and *Jaccard*. From the plots, we could observe that the Euclidean updates the clusters using distance, whiel Jaccard used the dissimilarity method (which is similar to k-modes).

[76]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA
from sklearn import manifold
from sklearn.cluster import AgglomerativeClustering
from scipy.cluster.hierarchy import dendrogram
from kmodes.kmodes import KModes

import scipy.cluster.hierarchy as shc
from matplotlib import pyplot
```

[77]:
```
df=pd.read_csv('/Users/aliceqichaowu/Desktop/38615/Lab2_clustering_dataset.csv')
df.head()
```

[77]:
```
                             ID  D_0  D_1  D_2  D_3  D_4  D_5  D_6  D_7  D_8  \
0  AAEAMMIUQZAASJ-MRXNPFEDSA-N    1    1    1    1    1    1    1    0    1
```

```
1  AAEFNWQXBPYXAC-UHFFFAOYSA-N    1    1    1    1    0    1    1    1    0
2  AAMHSIWFDKXUMZ-UHFFFAOYSA-N    1    1    1    1    1    1    0    1    1
3  AAPQXEOSVSLLMB-UHFFFAOYSA-N    1    1    1    1    0    1    1    1    1
4  AARXXEHXOBTROW-UHFFFAOYSA-N    1    1    1    1    1    1    1    0    1

   …  D_1014  D_1015  D_1016  D_1017  D_1018  D_1019  D_1020  D_1021  \
0  …       1       1       1       1       1       1       0       1
1  …       1       0       1       1       1       0       0       1
2  …       1       1       1       1       1       1       0       0
3  …       1       1       1       1       1       1       1       1
4  …       1       1       1       1       1       1       0       1

   D_1022  D_1023
0       1       1
1       1       1
2       1       1
3       1       1
4       1       1

[5 rows x 1025 columns]
```

[78]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 969 entries, 0 to 968
Columns: 1025 entries, ID to D_1023
dtypes: int64(1024), object(1)
memory usage: 7.6+ MB
```

[79]:
```python
# Check the df dimension and the missing values
print('Dataframe dimension: '+ str(df.shape) )
print('There are %i nan values in the dataframe' % df.isna().sum().sum())
```

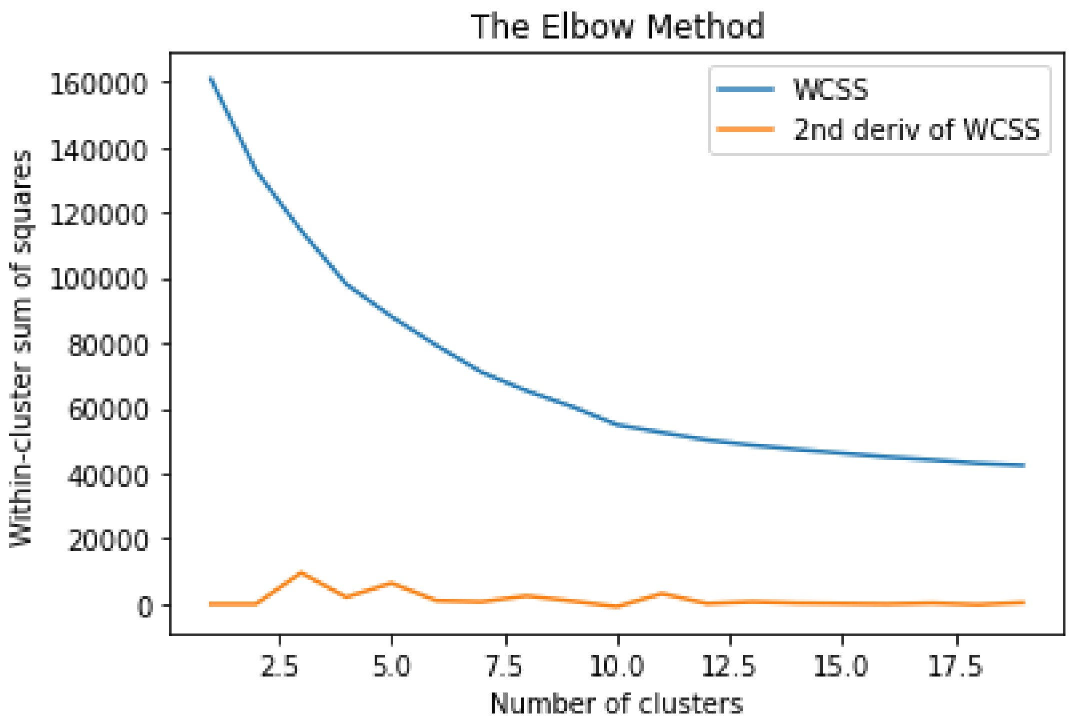
```
Dataframe dimension: (969, 1025)
There are 0 nan values in the dataframe
```

[80]:
```python
# Select data for analysis
X=df.iloc[:,1:]
```

0.2 K-means clustering

First, I use the elbow method to find the optimal number of clusters (k). I plot the Within-Cluster-Sum-of-Squares (WCSS) and its second derivative. The maximum of its 2nd derivative should be the optimal number k, which minimizes the cost most.

[81]:
```python
wcss = []
kcluster_list = range(1, 20)
for i in range(1, 20):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init␣
↪= 10, random_state = 0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

diff2 = np.append([0, 0], np.diff(np.diff(wcss)))

fig, ax = plt.subplots()

plt.plot(range(1, 20), wcss,label='WCSS')
plt.plot(kcluster_list, diff2, label='2nd deriv of WCSS')
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('Within-cluster sum of squares')

ax.legend()
plt.show()
```



From the graph and the calcualtion bewlow, we can see the optimal number k is 3.

[82]:
```
# Optimal number of clusters
k_opt = kcluster_list[np.argmax(diff2)]
print(k_opt)
```

```
3
```

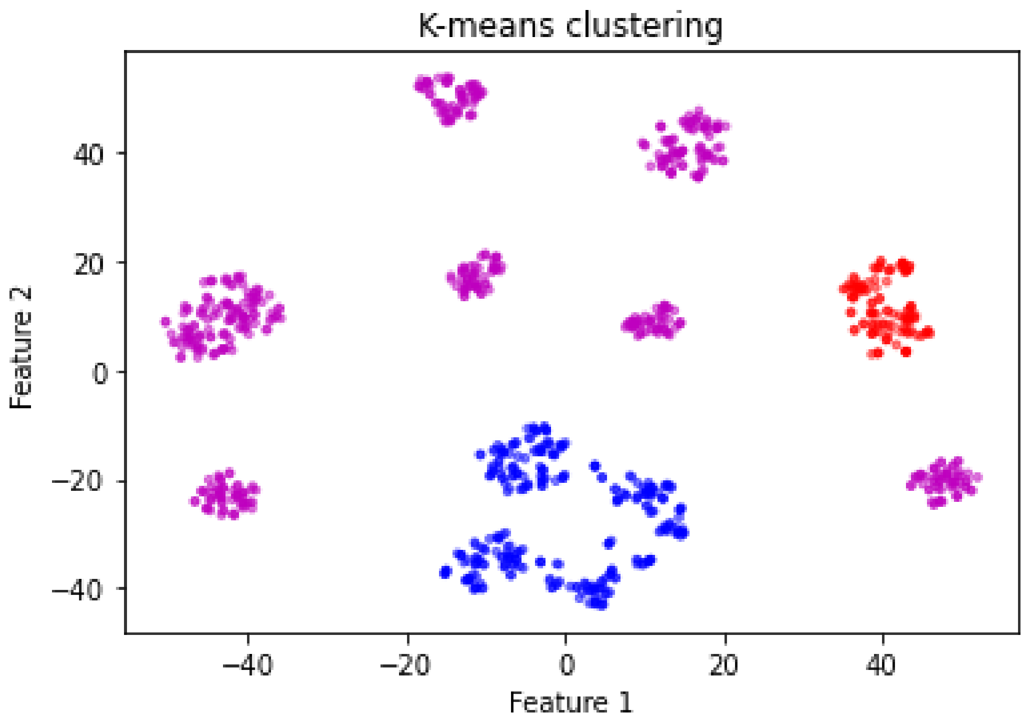
[83]:
```
# Apply K-means clustering with the optimal number of clusters
kmeans = KMeans(n_clusters=k_opt, random_state=0)
clusters = kmeans.fit(X)
labels = clusters.labels_  # predicted class (clusters)
```

0.3 Visualization

I use t-SNE to visualize clusters with colored labels from k-means clustering.

[85]:
```
# Implement tSNE to visualize the clusters
tsne = manifold.TSNE(random_state=42,n_components=2)
X_tsne = tsne.fit_transform(X)

# Plot clusters
df['labels'] = labels
colors = {0:'b', 1:'m',2:'r'}
fig1,ax = plt.subplots()
ax.scatter(X_tsne[:,0], X_tsne[:,1],c=df['labels'].map(colors), alpha=0.5,s=8)
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
plt.title('K-means clustering')
```

[85]:
```
Text(0.5, 1.0, 'K-means clustering')
```



0.4 K-modes clustering (for categorical data)

Next, I tried another clustering method–K-modes clustering for catergorical data. The implementation of this algorithm is illustrated below: 1) Use dissimilarity measure for categorical objects. The measurement is finding the total mismatches between the data points (i.e. 1 and 0 are mismatch); 2) Assign each observation to its closest cluster; 3) Calculate the modes of clusters and update new modes; 4) Repeat until there are is no re-assignment required.

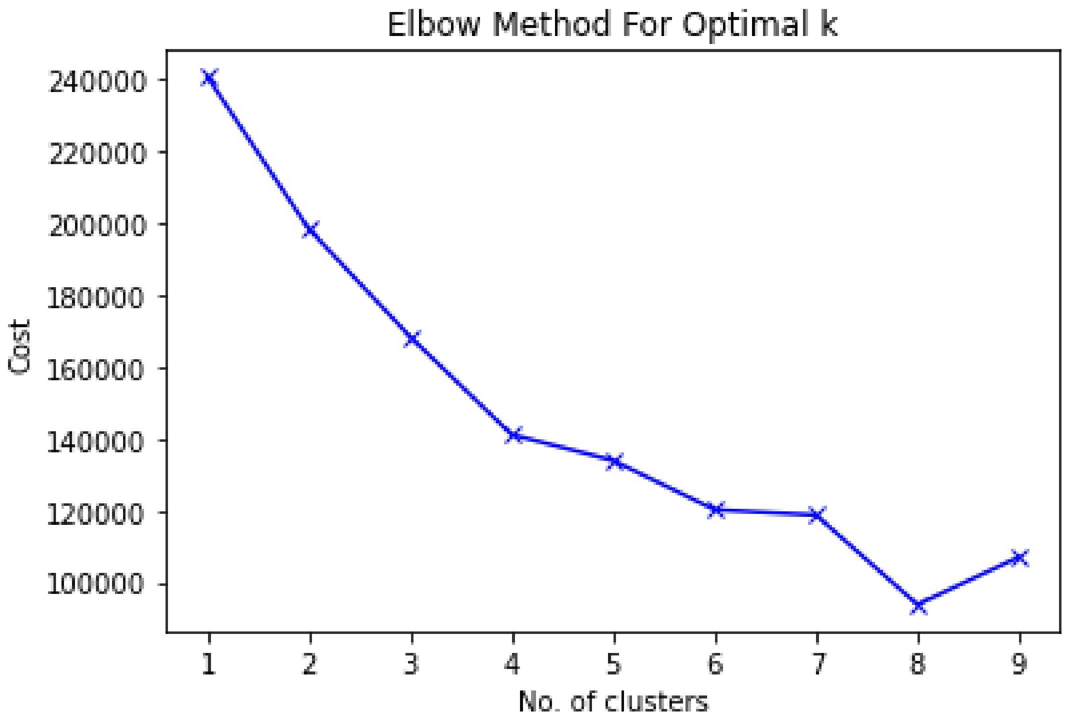
[86]:
```
cost = []
K = range(1,10)
for num_clusters in list(K):
    kmode = KModes(n_clusters=num_clusters, init = "random", n_init = 5,␣
 ↪verbose=1)
    kmode.fit_predict(X)
    cost.append(kmode.cost_)

plt.plot(K, cost, 'bx-')
plt.xlabel('No. of clusters')
plt.ylabel('Cost')
plt.title('Elbow Method For Optimal k')
plt.show()
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 0, cost: 240474.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 2, iteration: 1/100, moves: 0, cost: 240474.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 3, iteration: 1/100, moves: 0, cost: 240474.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 4, iteration: 1/100, moves: 0, cost: 240474.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 5, iteration: 1/100, moves: 0, cost: 240474.0
Best run was number 1
Init: initializing centroids
Init: initializing clusters
Starting iterations…
```

```
Run 1, iteration: 1/100, moves: 169, cost: 198349.0
Run 1, iteration: 2/100, moves: 21, cost: 198219.0
Run 1, iteration: 3/100, moves: 4, cost: 198209.0
Run 1, iteration: 4/100, moves: 0, cost: 198209.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 2, iteration: 1/100, moves: 75, cost: 209383.0
Run 2, iteration: 2/100, moves: 0, cost: 209383.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 3, iteration: 1/100, moves: 70, cost: 211896.0
Run 3, iteration: 2/100, moves: 0, cost: 211896.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 4, iteration: 1/100, moves: 58, cost: 212642.0
Run 4, iteration: 2/100, moves: 47, cost: 209419.0
Run 4, iteration: 3/100, moves: 4, cost: 209383.0
Run 4, iteration: 4/100, moves: 0, cost: 209383.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 5, iteration: 1/100, moves: 163, cost: 198228.0
Run 5, iteration: 2/100, moves: 13, cost: 198179.0
Run 5, iteration: 3/100, moves: 2, cost: 198177.0
Run 5, iteration: 4/100, moves: 0, cost: 198177.0
Best run was number 5
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 166, cost: 168254.0
Run 1, iteration: 2/100, moves: 0, cost: 168254.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 2, iteration: 1/100, moves: 291, cost: 170644.0
Run 2, iteration: 2/100, moves: 36, cost: 168263.0
Run 2, iteration: 3/100, moves: 3, cost: 168254.0
Run 2, iteration: 4/100, moves: 0, cost: 168254.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 3, iteration: 1/100, moves: 178, cost: 191444.0
Run 3, iteration: 2/100, moves: 54, cost: 190791.0
Run 3, iteration: 3/100, moves: 98, cost: 173125.0
Run 3, iteration: 4/100, moves: 73, cost: 168254.0
Run 3, iteration: 5/100, moves: 0, cost: 168254.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 4, iteration: 1/100, moves: 19, cost: 184532.0
Run 4, iteration: 2/100, moves: 0, cost: 184532.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 5, iteration: 1/100, moves: 64, cost: 181561.0
Run 5, iteration: 2/100, moves: 14, cost: 181381.0
Run 5, iteration: 3/100, moves: 0, cost: 181381.0
Best run was number 1
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 30, cost: 165142.0
Run 1, iteration: 2/100, moves: 7, cost: 164926.0
Run 1, iteration: 3/100, moves: 19, cost: 164168.0
Run 1, iteration: 4/100, moves: 4, cost: 164108.0
Run 1, iteration: 5/100, moves: 1, cost: 164108.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 2, iteration: 1/100, moves: 108, cost: 141062.0
Run 2, iteration: 2/100, moves: 6, cost: 141050.0
Run 2, iteration: 3/100, moves: 0, cost: 141050.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 3, iteration: 1/100, moves: 73, cost: 165920.0
Run 3, iteration: 2/100, moves: 0, cost: 165920.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 4, iteration: 1/100, moves: 81, cost: 168084.0
Run 4, iteration: 2/100, moves: 59, cost: 163414.0
Run 4, iteration: 3/100, moves: 17, cost: 163012.0
Run 4, iteration: 4/100, moves: 1, cost: 163009.0
Run 4, iteration: 5/100, moves: 0, cost: 163009.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 5, iteration: 1/100, moves: 376, cost: 177895.0
Run 5, iteration: 2/100, moves: 95, cost: 175665.0
Run 5, iteration: 3/100, moves: 57, cost: 173415.0
Run 5, iteration: 4/100, moves: 101, cost: 157685.0
Run 5, iteration: 5/100, moves: 18, cost: 157061.0
Run 5, iteration: 6/100, moves: 0, cost: 157061.0
Best run was number 2
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 371, cost: 159835.0
Run 1, iteration: 2/100, moves: 92, cost: 157058.0
Run 1, iteration: 3/100, moves: 0, cost: 157058.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 2, iteration: 1/100, moves: 99, cost: 138647.0
Run 2, iteration: 2/100, moves: 55, cost: 134685.0
Run 2, iteration: 3/100, moves: 24, cost: 133902.0
Run 2, iteration: 4/100, moves: 0, cost: 133902.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 3, iteration: 1/100, moves: 114, cost: 160344.0
Run 3, iteration: 2/100, moves: 39, cost: 159464.0
Run 3, iteration: 3/100, moves: 20, cost: 159188.0
Run 3, iteration: 4/100, moves: 4, cost: 159150.0
Run 3, iteration: 5/100, moves: 1, cost: 159150.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 4, iteration: 1/100, moves: 161, cost: 170864.0
Run 4, iteration: 2/100, moves: 2, cost: 170863.0
Run 4, iteration: 3/100, moves: 0, cost: 170863.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 5, iteration: 1/100, moves: 272, cost: 139399.0
Run 5, iteration: 2/100, moves: 24, cost: 139342.0
Run 5, iteration: 3/100, moves: 1, cost: 139342.0
Best run was number 2
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 224, cost: 130584.0
Run 1, iteration: 2/100, moves: 60, cost: 127906.0
Run 1, iteration: 3/100, moves: 14, cost: 127723.0
Run 1, iteration: 4/100, moves: 3, cost: 127721.0
Run 1, iteration: 5/100, moves: 0, cost: 127721.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 2, iteration: 1/100, moves: 139, cost: 151200.0
Run 2, iteration: 2/100, moves: 84, cost: 148497.0
Run 2, iteration: 3/100, moves: 6, cost: 148478.0
Run 2, iteration: 4/100, moves: 0, cost: 148478.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 3, iteration: 1/100, moves: 119, cost: 124878.0
Run 3, iteration: 2/100, moves: 7, cost: 124861.0
Run 3, iteration: 3/100, moves: 2, cost: 124860.0
Run 3, iteration: 4/100, moves: 0, cost: 124860.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 4, iteration: 1/100, moves: 124, cost: 120240.0
Run 4, iteration: 2/100, moves: 0, cost: 120240.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 5, iteration: 1/100, moves: 165, cost: 144605.0
Run 5, iteration: 2/100, moves: 49, cost: 142758.0
Run 5, iteration: 3/100, moves: 68, cost: 138106.0
Run 5, iteration: 4/100, moves: 74, cost: 130691.0
Run 5, iteration: 5/100, moves: 41, cost: 127425.0
Run 5, iteration: 6/100, moves: 33, cost: 125682.0
Run 5, iteration: 7/100, moves: 1, cost: 125679.0
Run 5, iteration: 8/100, moves: 0, cost: 125679.0
Best run was number 4
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 286, cost: 127325.0
Run 1, iteration: 2/100, moves: 109, cost: 122451.0
Run 1, iteration: 3/100, moves: 30, cost: 121715.0
Run 1, iteration: 4/100, moves: 1, cost: 121715.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 2, iteration: 1/100, moves: 164, cost: 121754.0
Run 2, iteration: 2/100, moves: 20, cost: 121721.0
Run 2, iteration: 3/100, moves: 2, cost: 121718.0
Run 2, iteration: 4/100, moves: 0, cost: 121718.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 3, iteration: 1/100, moves: 235, cost: 119746.0
Run 3, iteration: 2/100, moves: 84, cost: 118749.0
Run 3, iteration: 3/100, moves: 6, cost: 118738.0
Run 3, iteration: 4/100, moves: 0, cost: 118738.0
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 4, iteration: 1/100, moves: 113, cost: 126237.0
Run 4, iteration: 2/100, moves: 39, cost: 125479.0
Run 4, iteration: 3/100, moves: 7, cost: 125462.0
Run 4, iteration: 4/100, moves: 0, cost: 125462.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 5, iteration: 1/100, moves: 157, cost: 132955.0
Run 5, iteration: 2/100, moves: 5, cost: 132927.0
Run 5, iteration: 3/100, moves: 0, cost: 132927.0
Best run was number 3
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 103, cost: 143160.0
Run 1, iteration: 2/100, moves: 11, cost: 143085.0
Run 1, iteration: 3/100, moves: 19, cost: 142815.0
Run 1, iteration: 4/100, moves: 6, cost: 142805.0
Run 1, iteration: 5/100, moves: 0, cost: 142805.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 2, iteration: 1/100, moves: 201, cost: 120783.0
Run 2, iteration: 2/100, moves: 51, cost: 119019.0
Run 2, iteration: 3/100, moves: 5, cost: 119010.0
Run 2, iteration: 4/100, moves: 0, cost: 119010.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 3, iteration: 1/100, moves: 69, cost: 108340.0
Run 3, iteration: 2/100, moves: 5, cost: 108331.0
Run 3, iteration: 3/100, moves: 0, cost: 108331.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 4, iteration: 1/100, moves: 64, cost: 93864.0
Run 4, iteration: 2/100, moves: 1, cost: 93864.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 5, iteration: 1/100, moves: 81, cost: 113157.0
Run 5, iteration: 2/100, moves: 12, cost: 113135.0
Run 5, iteration: 3/100, moves: 1, cost: 113135.0
Best run was number 4
Init: initializing centroids
```

```
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 144, cost: 108991.0
Run 1, iteration: 2/100, moves: 49, cost: 108043.0
Run 1, iteration: 3/100, moves: 24, cost: 107874.0
Run 1, iteration: 4/100, moves: 5, cost: 107843.0
Run 1, iteration: 5/100, moves: 1, cost: 107843.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 2, iteration: 1/100, moves: 258, cost: 107088.0
Run 2, iteration: 2/100, moves: 4, cost: 107086.0
Run 2, iteration: 3/100, moves: 0, cost: 107086.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 3, iteration: 1/100, moves: 254, cost: 111505.0
Run 3, iteration: 2/100, moves: 88, cost: 110043.0
Run 3, iteration: 3/100, moves: 27, cost: 109660.0
Run 3, iteration: 4/100, moves: 18, cost: 109587.0
Run 3, iteration: 5/100, moves: 3, cost: 109585.0
Run 3, iteration: 6/100, moves: 0, cost: 109585.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 4, iteration: 1/100, moves: 178, cost: 107173.0
Run 4, iteration: 2/100, moves: 13, cost: 107097.0
Run 4, iteration: 3/100, moves: 1, cost: 107097.0
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 5, iteration: 1/100, moves: 117, cost: 130494.0
Run 5, iteration: 2/100, moves: 16, cost: 130439.0
Run 5, iteration: 3/100, moves: 10, cost: 129037.0
Run 5, iteration: 4/100, moves: 53, cost: 127570.0
Run 5, iteration: 5/100, moves: 7, cost: 127561.0
Run 5, iteration: 6/100, moves: 0, cost: 127561.0
Best run was number 2
```



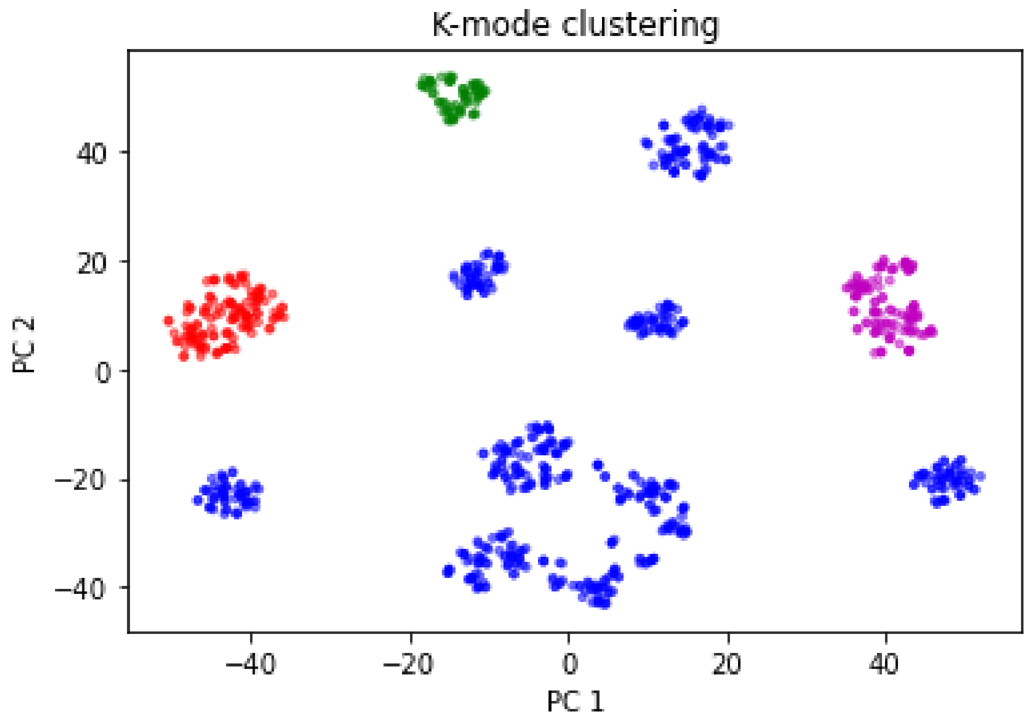
[87]:
```
# Building the model with 4 clusters
kmode = KModes(n_clusters=4, init = "Cao", n_init = 1, verbose=1)
clusters = kmode.fit_predict(X)
df['label_kmode'] =clusters
```

```
Init: initializing centroids
Init: initializing clusters
Starting iterations…
Run 1, iteration: 1/100, moves: 10, cost: 165926.0
Run 1, iteration: 2/100, moves: 1, cost: 165920.0
Run 1, iteration: 3/100, moves: 0, cost: 165920.0
```

0.5 Visualization

[88]:
```
# Plot clusters
colors = {0:'b', 1:'m',2:'r',3:'g'}
fig1,ax = plt.subplots()
ax.scatter(X_tsne[:,0], X_tsne[:,1],c=df['label_kmode'].map(colors), alpha=0.
↪5,s=8)
# ax.scatter(centers[:,0], centers[:,1],c='k', s=10)
ax.set_xlabel('PC 1')
ax.set_ylabel('PC 2')
plt.title('K-mode clustering')
```

```
[88]: Text(0.5, 1.0, 'K-mode clustering')
```

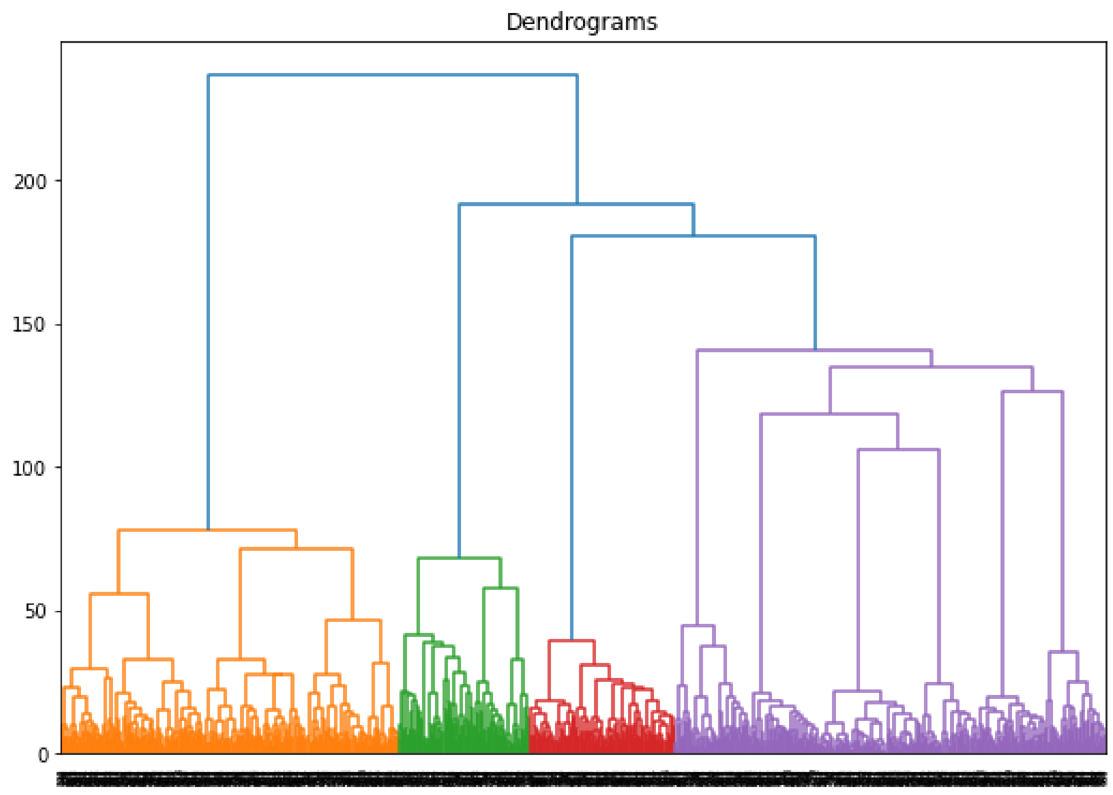


0.6 Agglomerative Clustering

The third clustering method I tried is agglomerative Clustering, which is a type of hierarchical clustering algorithm (bottom-up). It links pairs of data points and merges into clusters based on the linkage distance metrics. Here, I used two distance metrics Euclidean and jaccard. Euclidean distance is by calculating the distance between two points. Jaccard distance is by calculating the size of the intersection divided by the size of the union of two label sets. In simple words, it's used to find out disimilar two sets are.

0.7 Find the optimal munber of clusters with dendrogram: k=5

```
[90]: ## Find the optimal number k based on the hierarchical structure
pyplot.figure(figsize=(10, 7))
pyplot.title("Dendrograms")
dend = shc.dendrogram(shc.linkage(X, method='ward'))
```



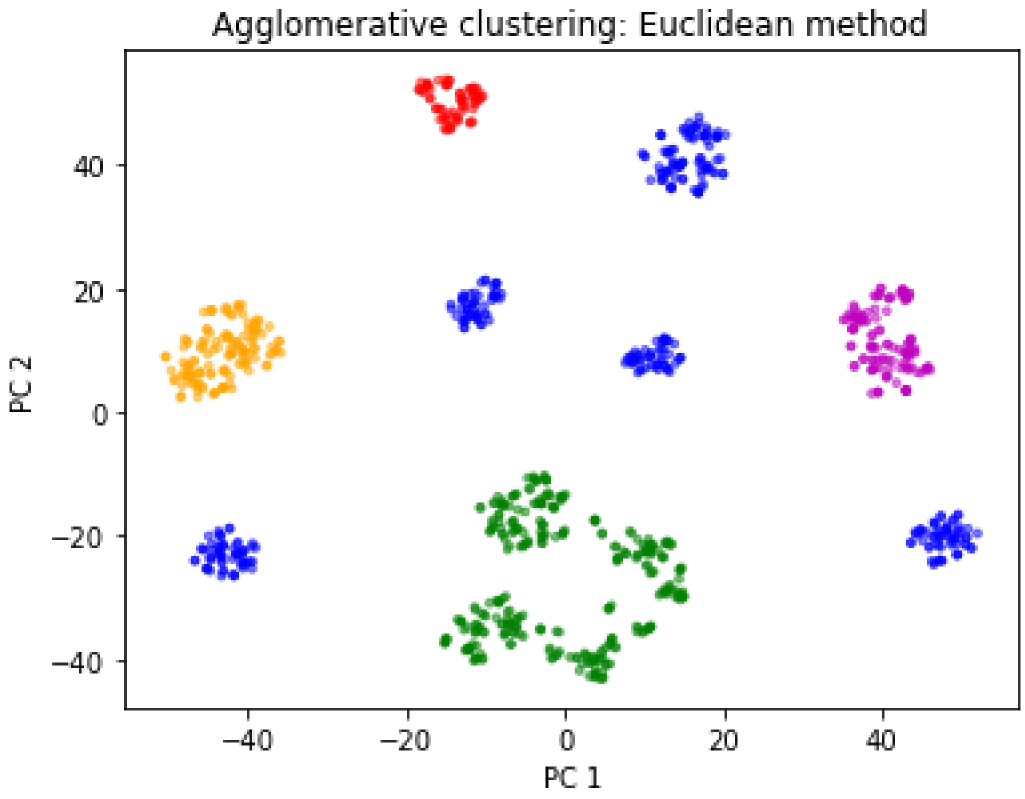
[91]:
```
clst1 =␣
 ↪AgglomerativeClustering(n_clusters=5,affinity='Euclidean',linkage='complete')
clst1.fit(X)
labels1 = clst1.labels_  # preicted class (clusters)
df['labels_Euclidean'] = labels1

clst2 =␣
 ↪AgglomerativeClustering(n_clusters=5,affinity='jaccard',linkage='complete')
clst2.fit(X)
labels2 = clst2.labels_  # preicted class (clusters)
df['labels_jaccard'] = labels2
```

0.8 Visualization

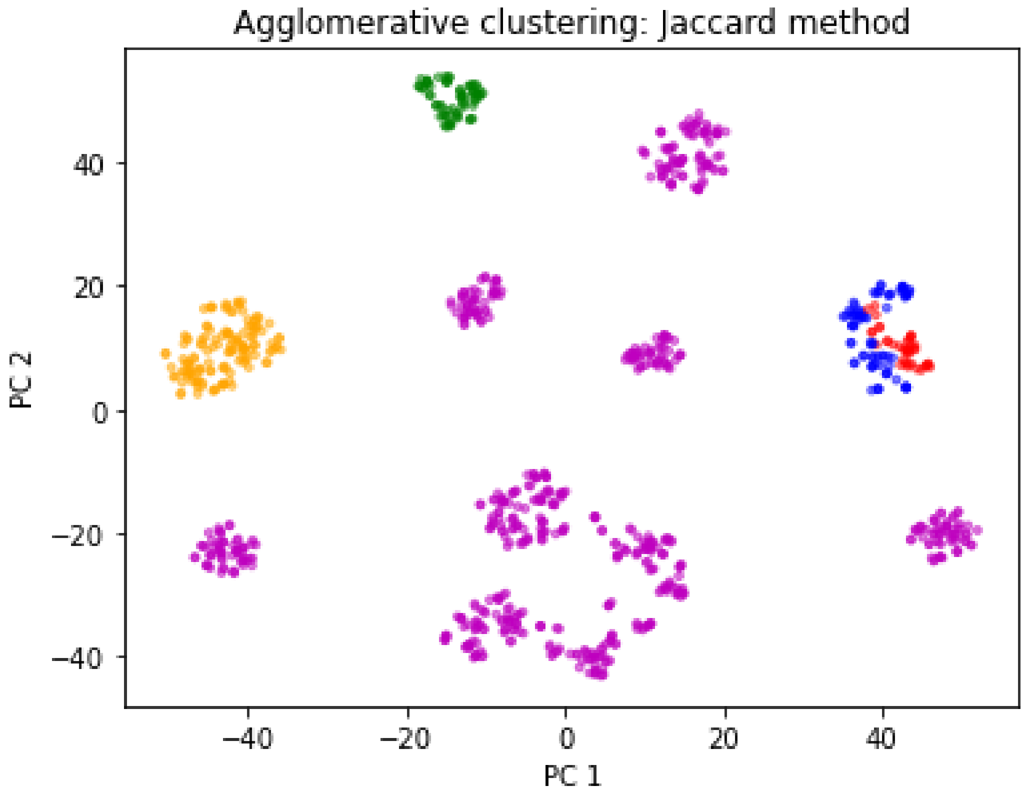
I use t-SNE to visualize clusters with colored labels from k-means clustering. From the plots of two different distance metrics method, their clusters are pretty similar. The only difference here is the cluster green and red clusters. What we can see from Euclidean method is that the clusters are far from each other, while the blue and red clusters in Jaccard are very close to each other indicating that it is not based on the distance itself.

[93]:
```
method=['Euclidean','Jaccard']
colors = {0:'b', 1:'m',2:'r',3:'g',4:'orange'}
```

```python
fig,ax = plt.subplots(2,1,figsize=(6, 10))

# fig.tight_layout()
ax[0].scatter(X_tsne[:,0], X_tsne[:,1],c=df['labels_Euclidean'].map(colors),␣
↪alpha=0.5,s=8)
ax[1].scatter(X_tsne[:,0], X_tsne[:,1],c=df['labels_jaccard'].map(colors),␣
↪alpha=0.5,s=8)
for i in range(0,2):
    ax[i].set_xlabel('PC 1')
    ax[i].set_ylabel('PC 2')
    ax[i].set_title('Agglomerative clustering: '+method[i]+' method')
```



Agglomerative clustering: Euclidean method
PC 2
PC 1
40
20
0
−20
−40
−40
−20
0
20
40



Agglomerative clustering: Jaccard method
PC 2
PC 1
40
20
0
−20
−40
−40
−20
0
20
40

```
[ ]:
```