30100HW7

March 14, 2020

Yanjie Zhou HW7

MACS 30100

Dr. Waggoner

2020 Mar 14

[4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
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
```

1 1

[7]:
```
x1 = [5,8,7,8,3,4,2,3,4,5]
x2 = [8,6,5,4,3,2,2,8,9,8]
np.random.seed(1234)
df_k3 = pd.DataFrame({'x1': x1, 'x2': x2})
k3_labels = np.random.choice(3, 10, replace=True)
df_k3['k_label'] = k3_labels
df_k3
```

[7]:
```
   x1  x2  k_label
0   5   8        2
1   8   6        1
2   7   5        0
3   8   4        0
4   3   3        0
5   4   2        1
6   2   2        1
7   3   8        1
8   4   9        2
9   5   8        2
```

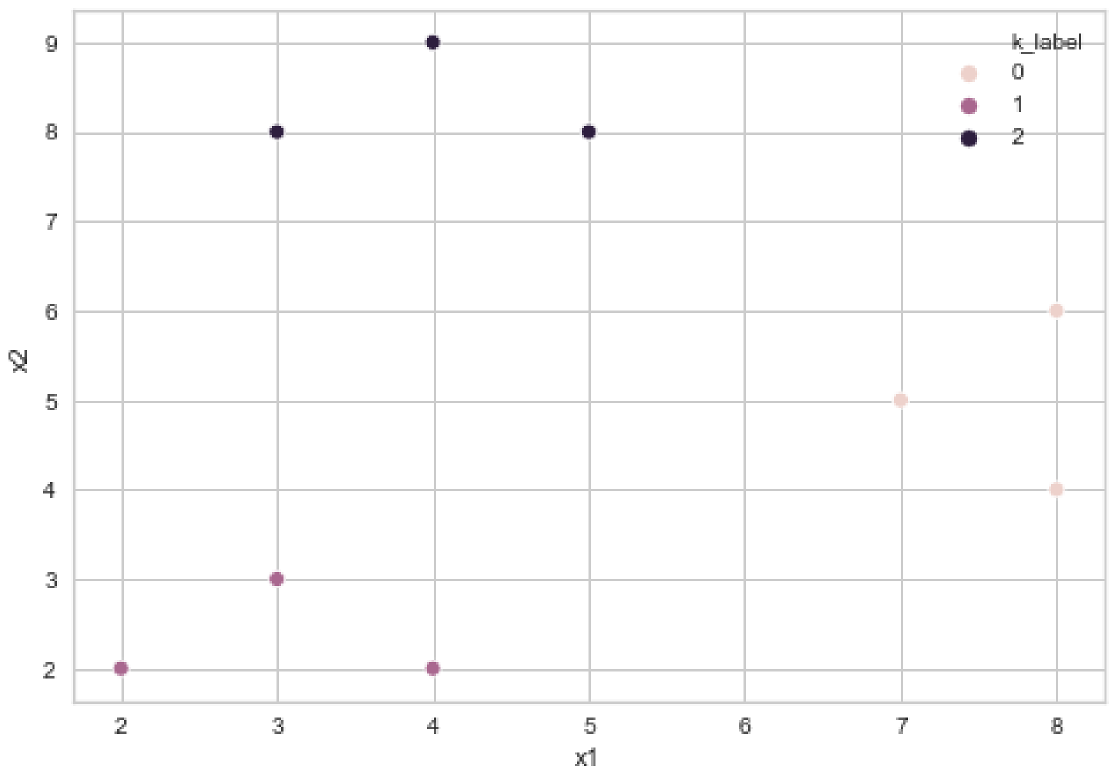
2 2

[8]:
```
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]['x1'].mean()
            cent2 = fit_df[fit_df['k_label'] == k]['x2'].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['x1'] - v[0]) ** 2 + (row['x2'] -␣
 ↪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

fit_k3 = fit_kmeans(3, df_k3)
fit_k3
```

[8]:
```
   x1  x2  k_label
0   5   8        2
1   8   6        0
2   7   5        0
3   8   4        0
4   3   3        1
5   4   2        1
6   2   2        1
7   3   8        2
8   4   9        2
9   5   8        2
```

3 3

[10]:
```
sns.scatterplot(x='x1', y='x2', hue='k_label', data=fit_k3);
```



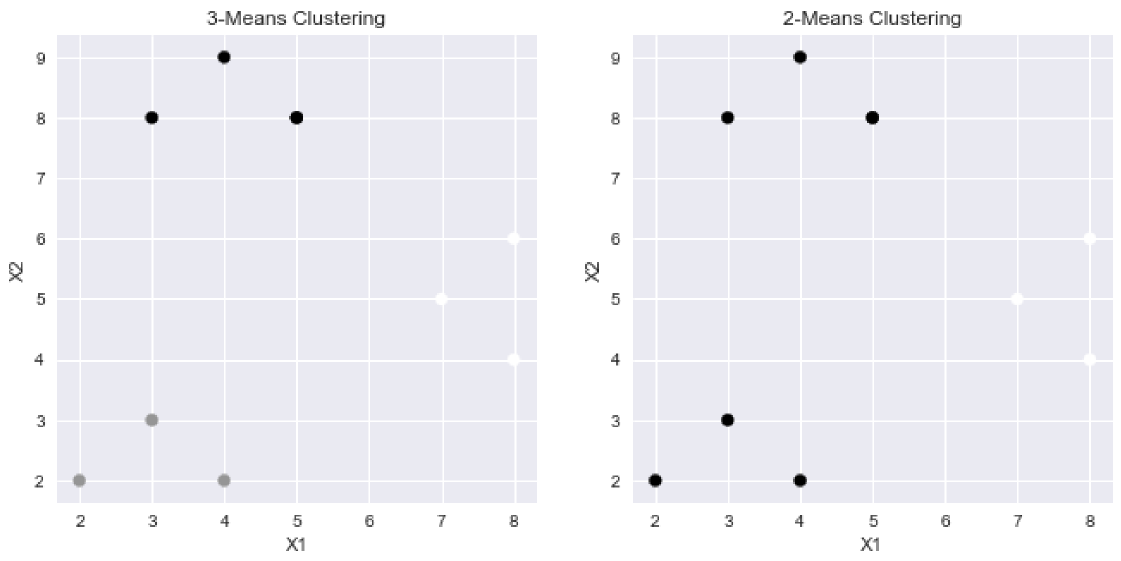
4 4

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

```
[13]:    x1  x2  k_label
      0   5   8        1
      1   8   6        0
      2   7   5        0
      3   8   4        0
      4   3   3        1
      5   4   2        1
      6   2   2        1
      7   3   8        1
      8   4   9        1
      9   5   8        1
```

```
[33]: fig, axs = plt.subplots(1, 2, figsize=(11,5))
      axs[0].scatter(fit_k3['x1'], fit_k3['x2'], c=fit_k3['k_label'], s=50)
      axs[0].set_xlabel('X1')
      axs[0].set_ylabel('X2')
```

```
axs[0].set_title('3-Means Clustering')
axs[1].scatter(fit_k2['x1'], fit_k2['x2'], c=fit_k2['k_label'], s=50)
axs[1].set_xlabel('X1')
axs[1].set_ylabel('X2')
axs[1].set_title('2-Means Clustering');
```



5 5

k=3 is better for this dataset, because if k=2, the distances between points within the cluster in black are still very high. As is shown in the right plot, three points in the black cluster are close together albeit away with another trio. When k=3, the distances become much smaller, which makes it a good choice of clustering for this dataset.

6 6

[34]:
```
df_wiki = pd.read_csv('C:/Users/zyj/Downloads/wiki.csv')
```

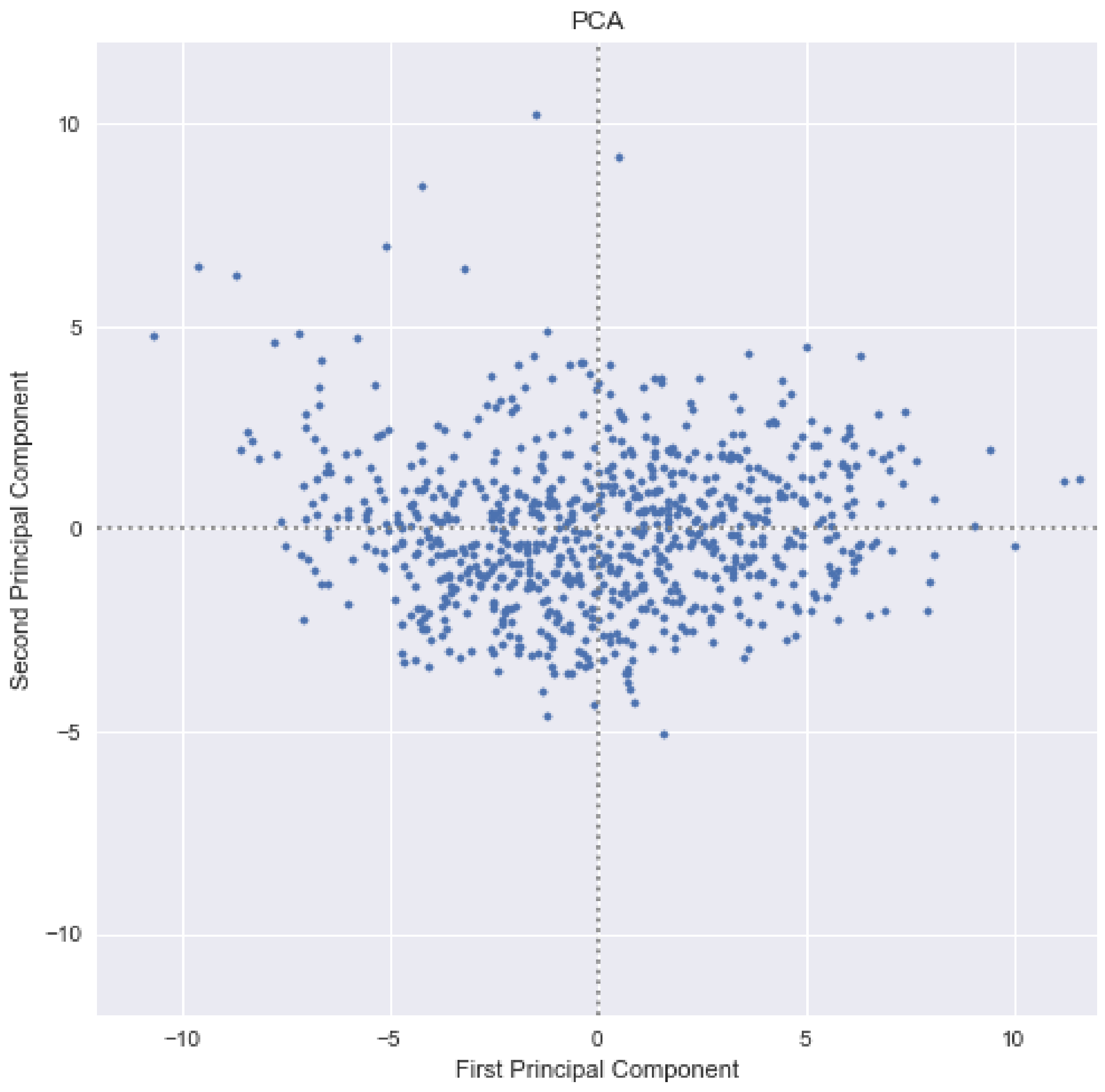
[36]:
```
x = df_wiki.values
x = StandardScaler().fit_transform(x)
x
```

[36]:
```
array([[-0.28718866, -0.86413245,  1.14257407, …, -0.15171652,
        -0.05006262, -2.1618878 ],
       [-0.02204045, -0.86413245,  1.14257407, …, -0.15171652,
        -0.05006262, -2.1618878 ],
       [-0.68491098, -0.86413245,  1.14257407, …, -0.15171652,
        -0.05006262, -2.1618878 ],
       …,
       [ 0.9059783 ,  1.15723001,  1.14257407, …, -0.15171652,
        -0.05006262,  0.46255869],
       [-0.02204045,  1.15723001, -0.87521678, …, -0.15171652,
        -0.05006262,  0.46255869],
       [ 0.37568187,  1.15723001,  1.14257407, …, -0.15171652,
        -0.05006262,  0.46255869]])
```

[57]:
```
pca = PCA(n_components=2, random_state=1234)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['PC1', 'PC2'])
principalDf
```

[57]:
```
          PC1       PC2
0   -0.150216 -1.982557
1   -3.314020 -0.793374
2   -4.682484 -0.312333
3    1.774200  1.986924
4    7.254695  2.014038
..        …         …
795  0.227143  1.475131
796  4.434784 -0.931281
797  1.449455 -0.170083
798 -2.888282  2.721318
799 -7.000657  2.806714

[800 rows x 2 columns]
```

[43]:
```
plt.figure(figsize=(8, 8))
plt.xlim(-12,12)
plt.ylim(-12,12)
plt.scatter(principalDf['PC1'], principalDf['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');
```



```
[50]: pc1_corr = [(n, np.corrcoef(c, principalDf['PC1'])[0][1]) for n, c in df_wiki.
       ↪iteritems()]
      sorted(pc1_corr, key=lambda x: x[1], reverse=True)[:5]
```

```
[50]: [('bi2', 0.8326740884020257),
       ('bi1', 0.8156152224336632),
       ('use3', 0.7889903915787533),
       ('use4', 0.7736625286437459),
       ('pu3', 0.7603359671593874)]
```

```
[51]: pc2_corr = [(n, np.corrcoef(c, principalDf['PC2'])[0][1]) for n, c in df_wiki.
       ↪iteritems()]
      sorted(pc2_corr, key=lambda x: x[1], reverse=True)[:5]
```

```
[51]: [('exp4', 0.43546743753444545),
       ('use2', 0.4166861010644903),
       ('use1', 0.37708617914786796),
       ('vis3', 0.37668447514654674),
       ('domain_Engineering_Architecture', 0.3267114204703721)]
```

According to the above analyses, the variables ‘bi2’, ‘bi1’, ‘use3’, ‘use4’, ‘pu3’ are strongly correlated with the first principal component, while ‘exp4’, ‘use2’, ‘use1’, ‘vis3’ are strongly correlated with the second one.

7 7

```
[55]: print(f'{pca.explained_variance_ratio_[0]} of the variance is explained by the␣
       ↪first principal component')
      print(f'{pca.explained_variance_ratio_[1]} of the variance is explained by the␣
       ↪second principal component')
      print(f'{sum(pca.explained_variance_ratio_)} of the variance is explained by␣
       ↪the first two principal components')
```

```
0.22810627785663376 of the variance is explained by the first principal
component
0.06372473871135148 of the variance is explained by the second principal
component
0.2918310165679852 of the variance is explained by the first two principal
components
```

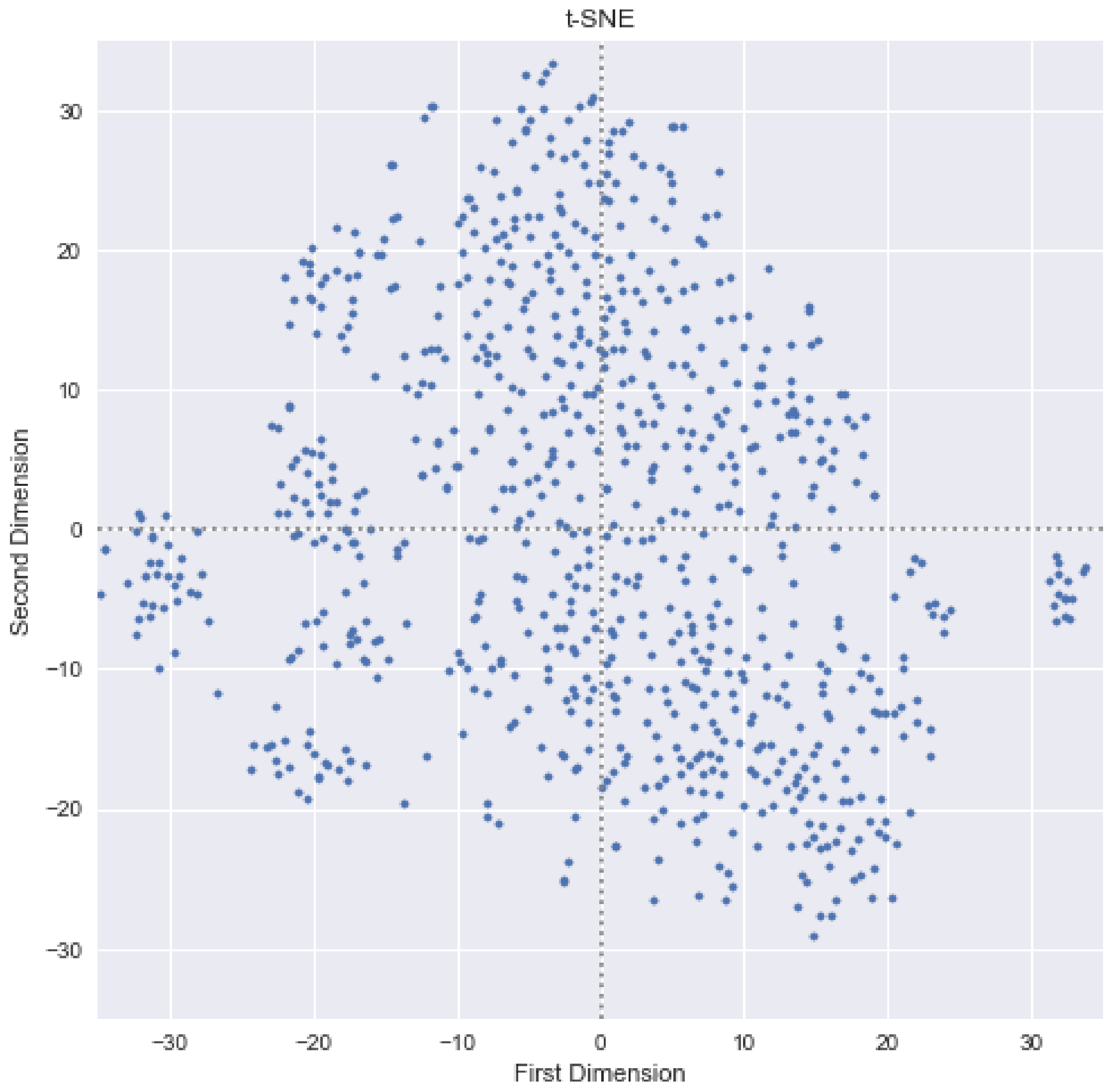
8 8

```
[58]: tsne = TSNE(n_components=2, random_state=1234)
      tsneDim = tsne.fit_transform(x)
      tsneDf = pd.DataFrame(data = tsneDim
                   , columns = ['dim1', 'dim2'])
      tsneDf
```

```
[58]:          dim1       dim2
      0   -22.647652 -12.732190
      1   -19.262894 -16.672525
      2    -7.856101 -20.482122
      3   -20.132755   5.491135
      4    -7.237445  29.417648
      ..         …          …
      795  11.679301  18.673101
      796 -11.296926  15.385149
      797  -3.649632   4.755620
      798  16.311146  -1.265161
      799  16.426849 -26.509228
```

```
[800 rows x 2 columns]
```

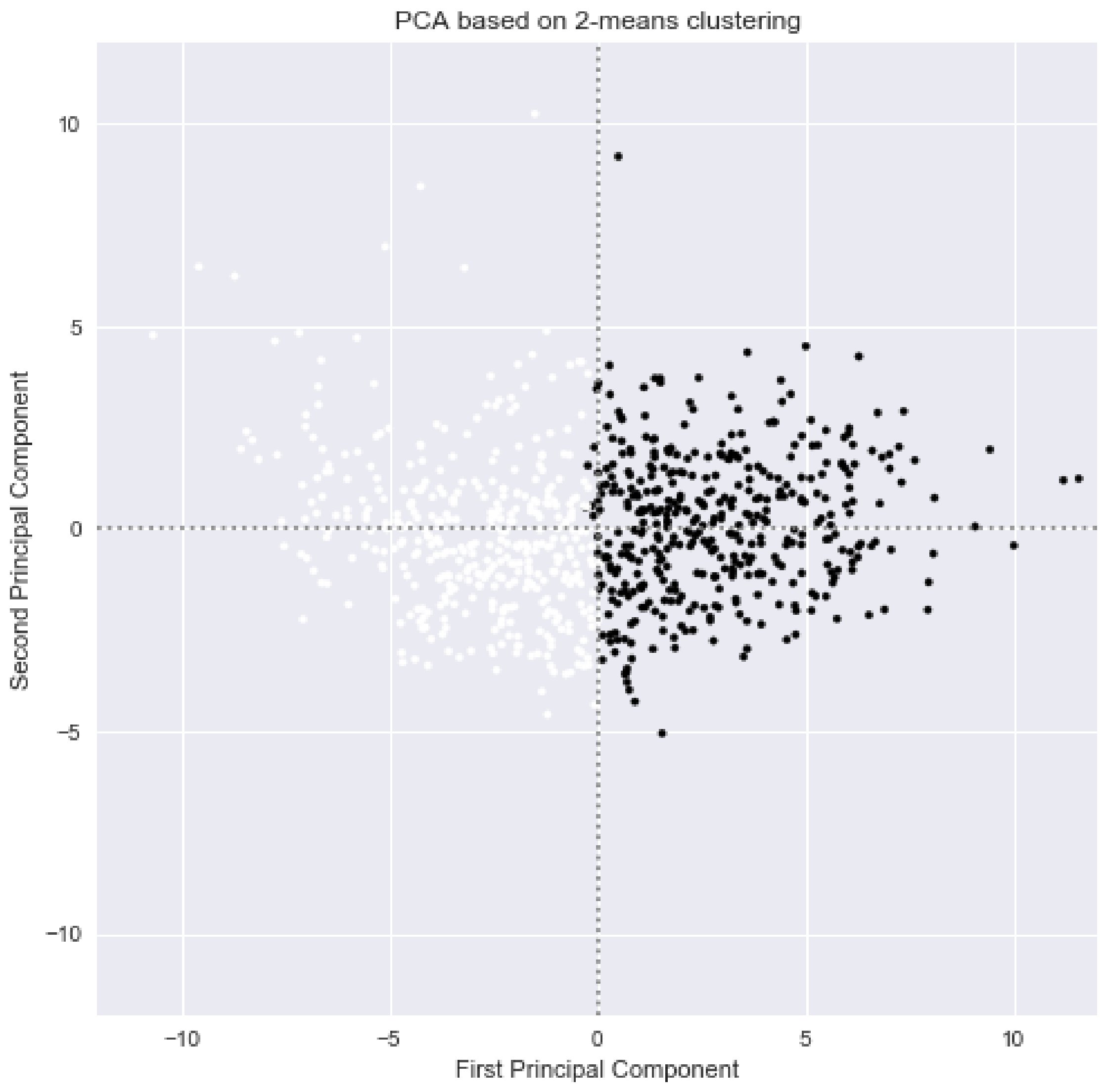
[60]:
```
fig = plt.figure(figsize=(8, 8))
plt.xlim(-35,35)
plt.ylim(-35,35)
plt.hlines(0,-35,35, linestyles='dotted', colors='grey')
plt.vlines(0,-35,35, linestyles='dotted', colors='grey')
plt.scatter(tsneDf['dim1'], tsneDf['dim2'], s=11)
plt.xlabel('First Dimension')
plt.ylabel('Second Dimension')
plt.title('t-SNE');
```



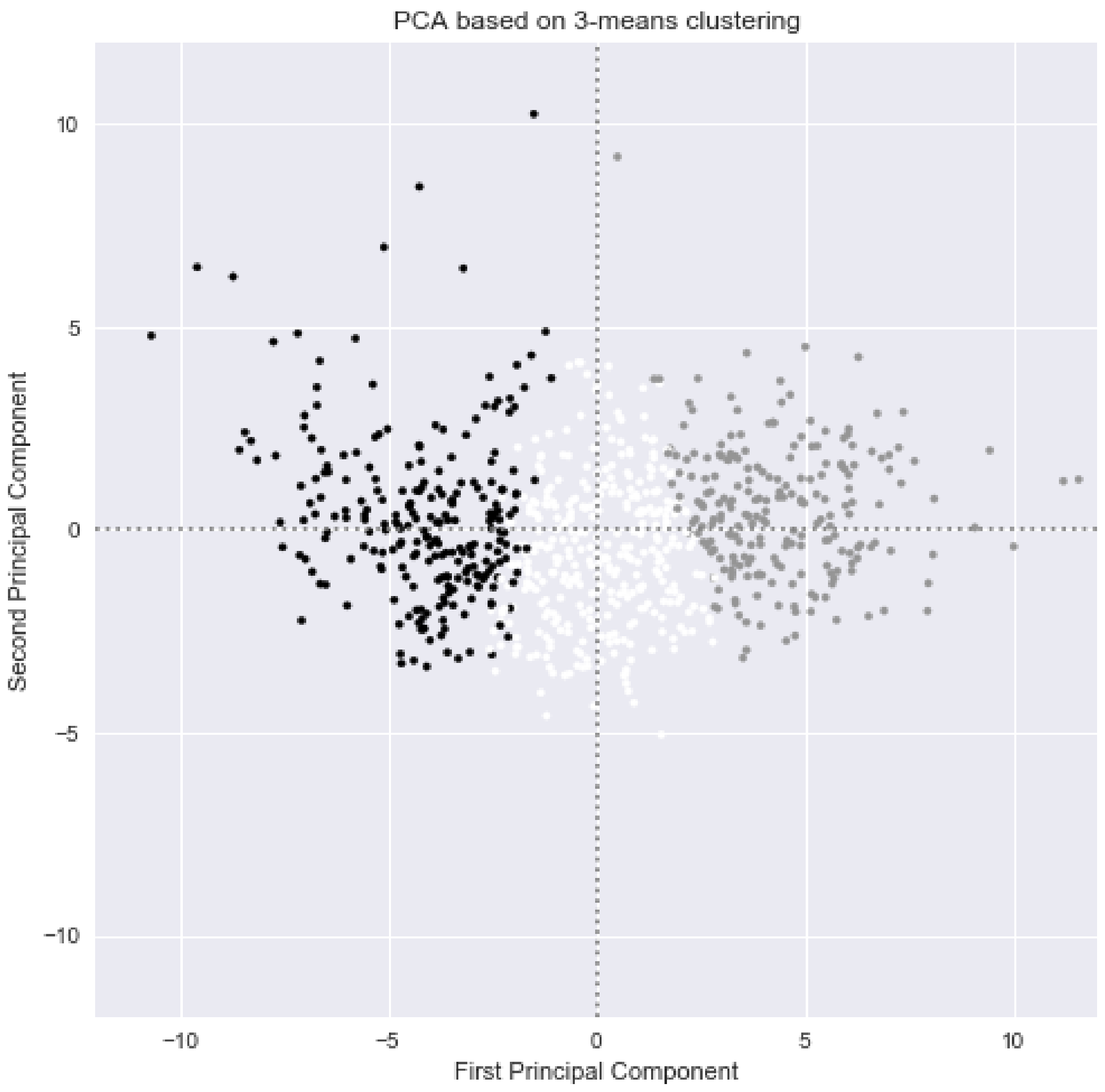
As the plot shows, t-SNE plot is composed of a large group of points with several small groups of points scattered around, which shows the advantage of t-SNE in contrast to PCA that it preserves the local structure of data. Considering that t-SNE is better at dealing with non-linear relationships, the plot suggests that the relationships between variables may be non-linear.

9 9

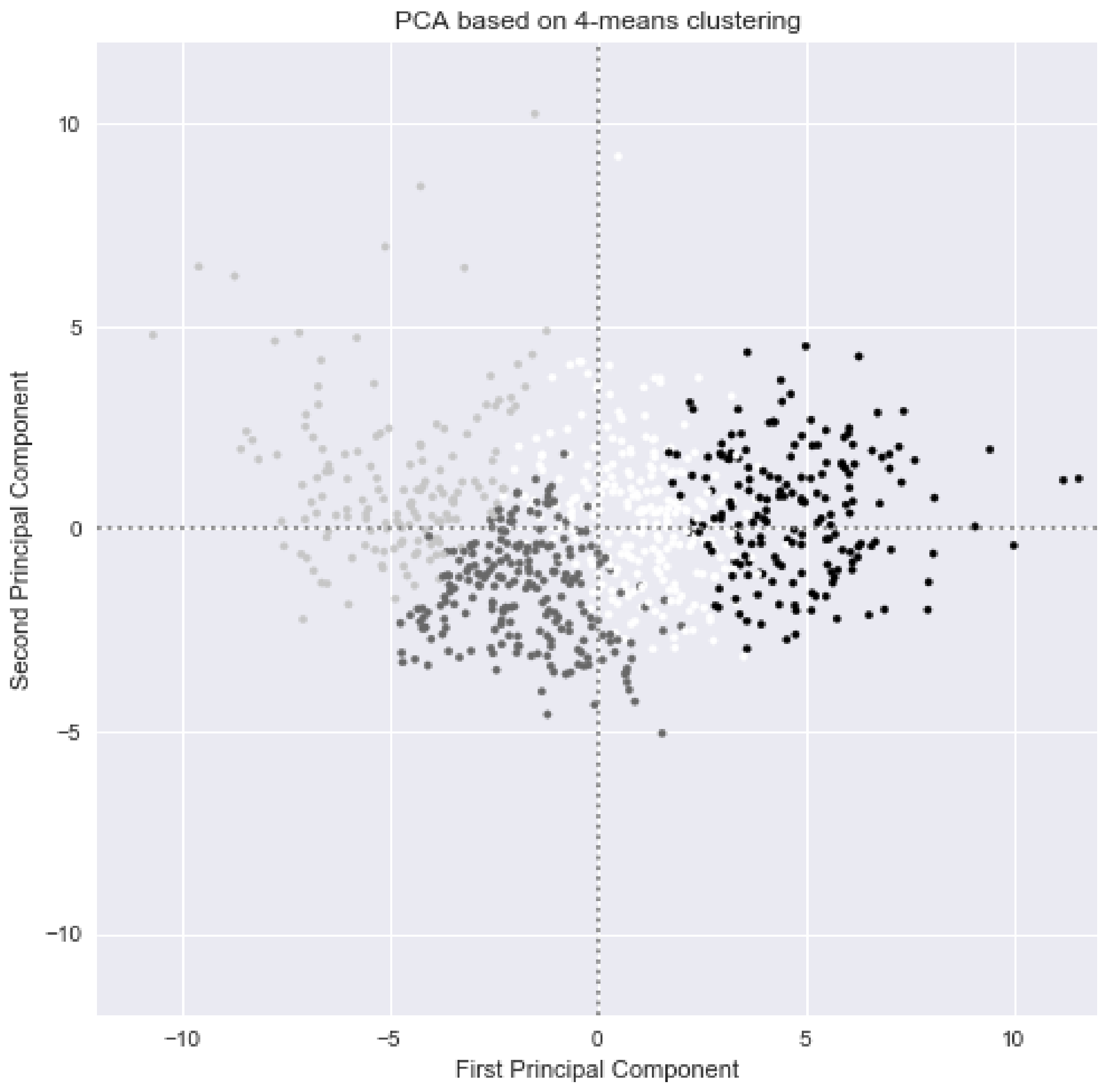
[68]:
```
k2 = KMeans(n_clusters=2, random_state=1234).fit(x)

def kmeans_plot(k):
    plt.figure(figsize=(8, 8))
    plt.xlim(-12,12)
    plt.ylim(-12,12)
    plt.scatter(principalDf['PC1'], principalDf['PC2'], s=11, c=k.labels_)
    plt.xlabel('First Principal Component')
    plt.ylabel('Second Principal Component')
    num_cluster = k.get_params()['n_clusters']
    plt.title(f'PCA based on {num_cluster}-means clustering')
    plt.hlines(0,-12,12, linestyles='dotted', colors='grey')
    plt.vlines(0,-12,12, linestyles='dotted', colors='grey');

kmeans_plot(k2)
```



```
[69]: k3 = KMeans(n_clusters=3, random_state=1234).fit(x)
      kmeans_plot(k3)
```



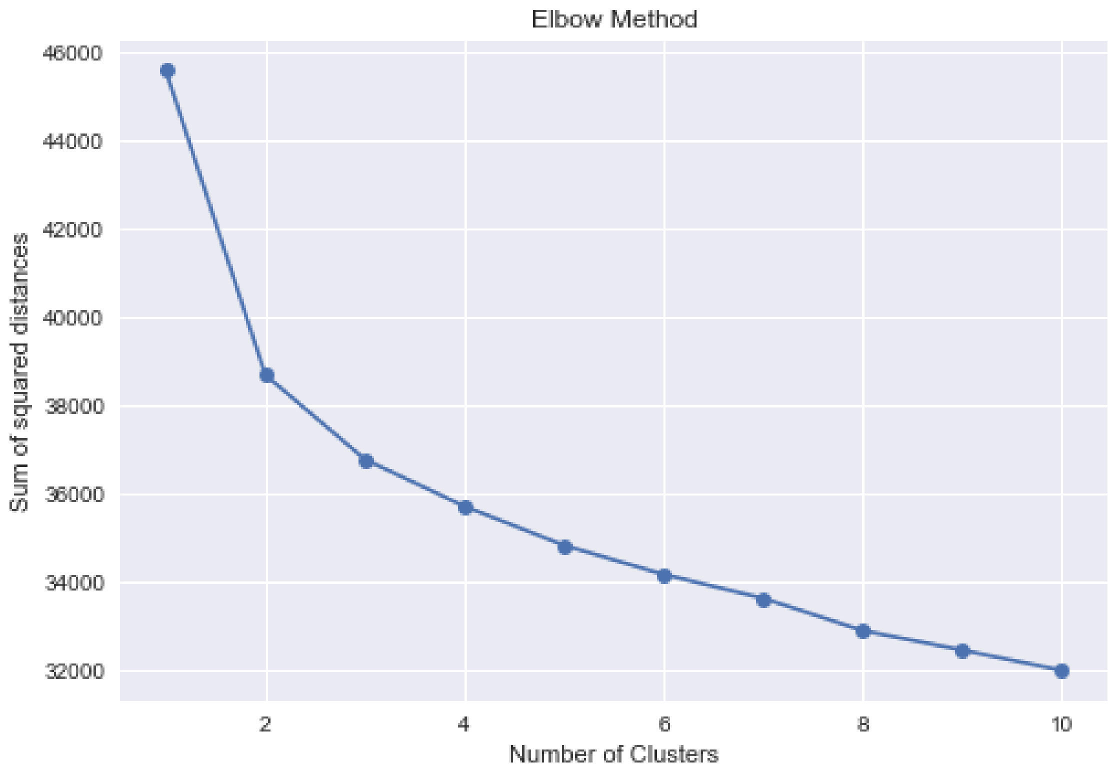
```
[70]: k4 = KMeans(n_clusters=4, random_state=1234).fit(x)
      kmeans_plot(k4)
```



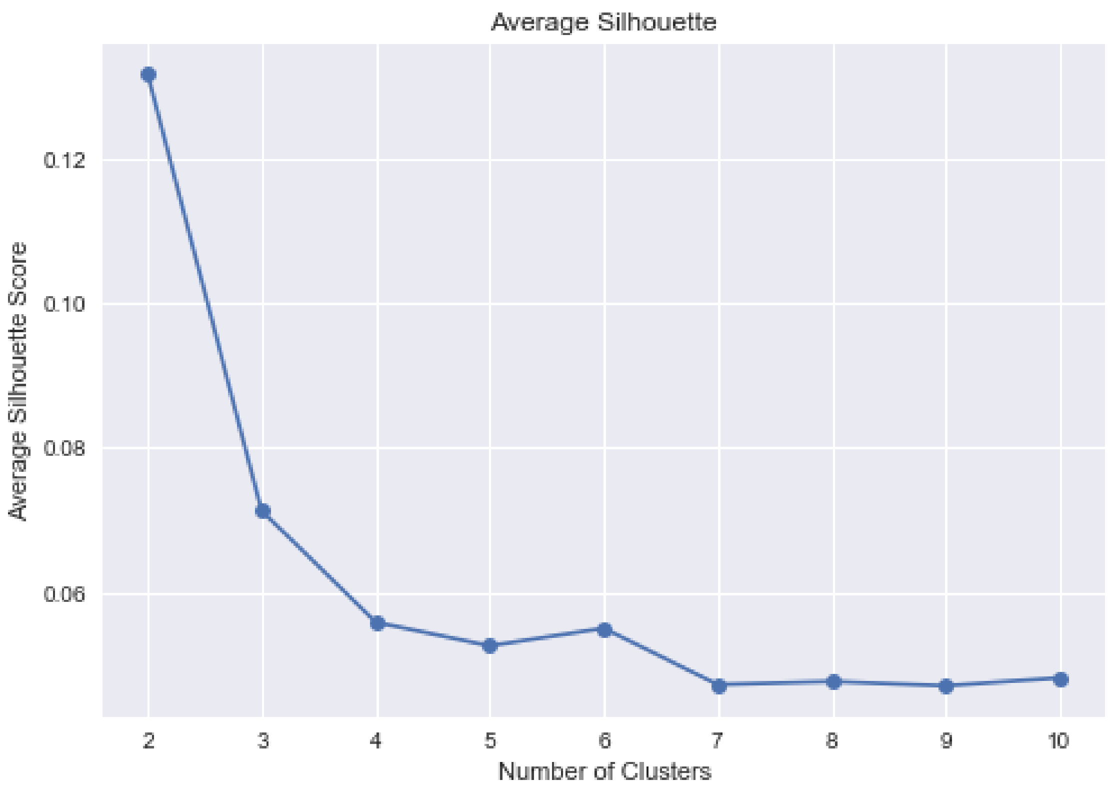
According to the above plots, k-means clustering with 2 clusters performs the best because the points in each cluster are seperated far away from each other, while 3 clusters and 4 clusters produce the relatively close clusters.

10   10

[78]:
```
inertias = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=1234).fit(x)
    inertias.append(kmeans.inertia_)
plt.plot(range(1, 11), inertias, '-o')
plt.ylabel('Sum of squared distances')
plt.xlabel('Number of Clusters')
plt.title('Elbow Method');
```



[80]:
```
silhouettes = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=1234)
    silhouettes.append(silhouette_score(x, kmeans.fit_predict(x)))
plt.plot(range(2, 11), silhouettes, '-o')
plt.ylabel('Average Silhouette Score')
plt.xlabel('Number of Clusters')
plt.title('Average Silhouette');
```



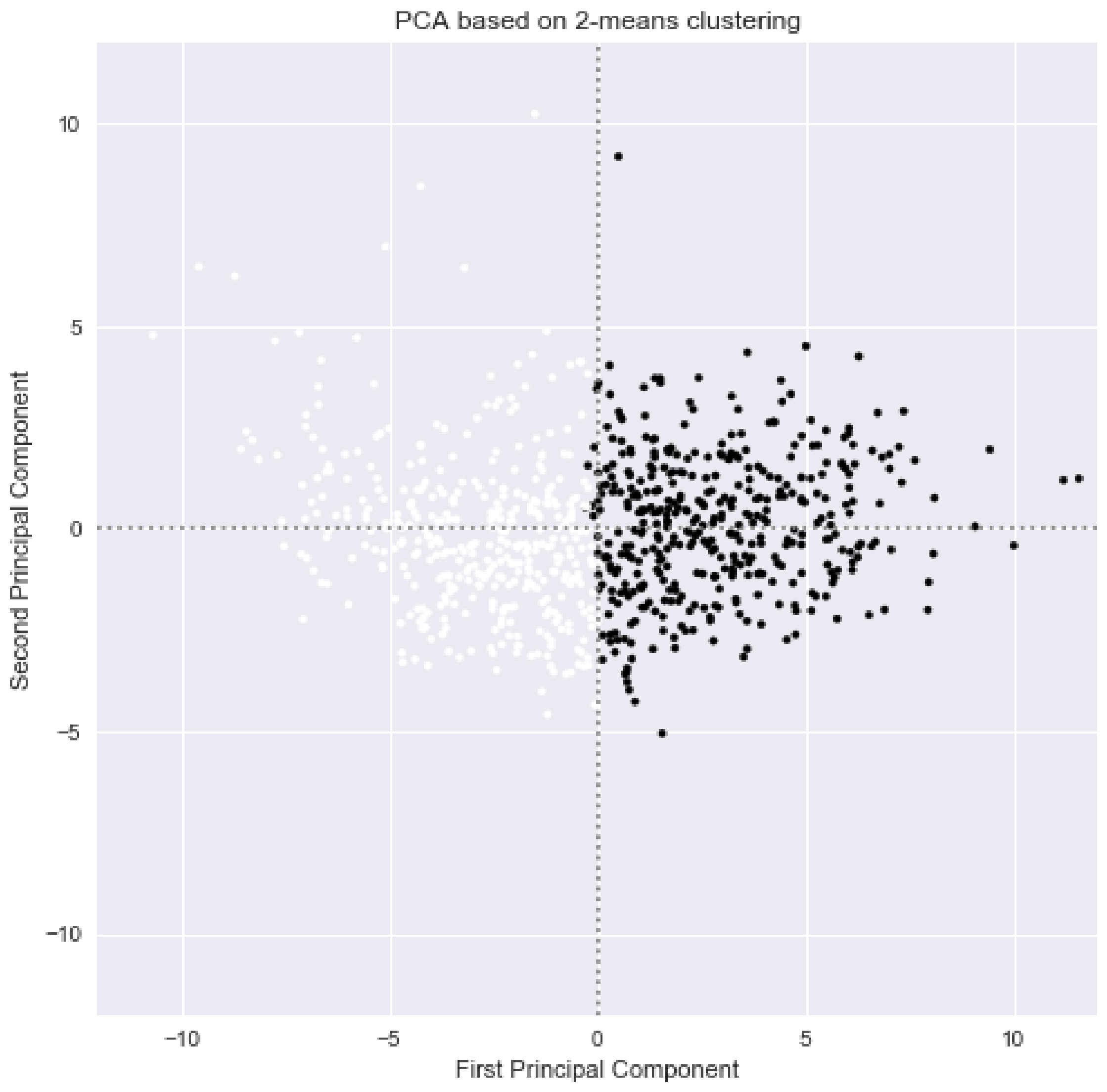
[95]:
```
from gap_statistic import optimalK
np.random.seed(1234)
opt = optimalK.OptimalK()
n_clusters = opt(x, cluster_array=np.arange(2, 11))
print('the optimal number of clusters based on gap statistic: ', n_clusters)
```

```
the optimal number of clusters based on gap statistic:  8
```

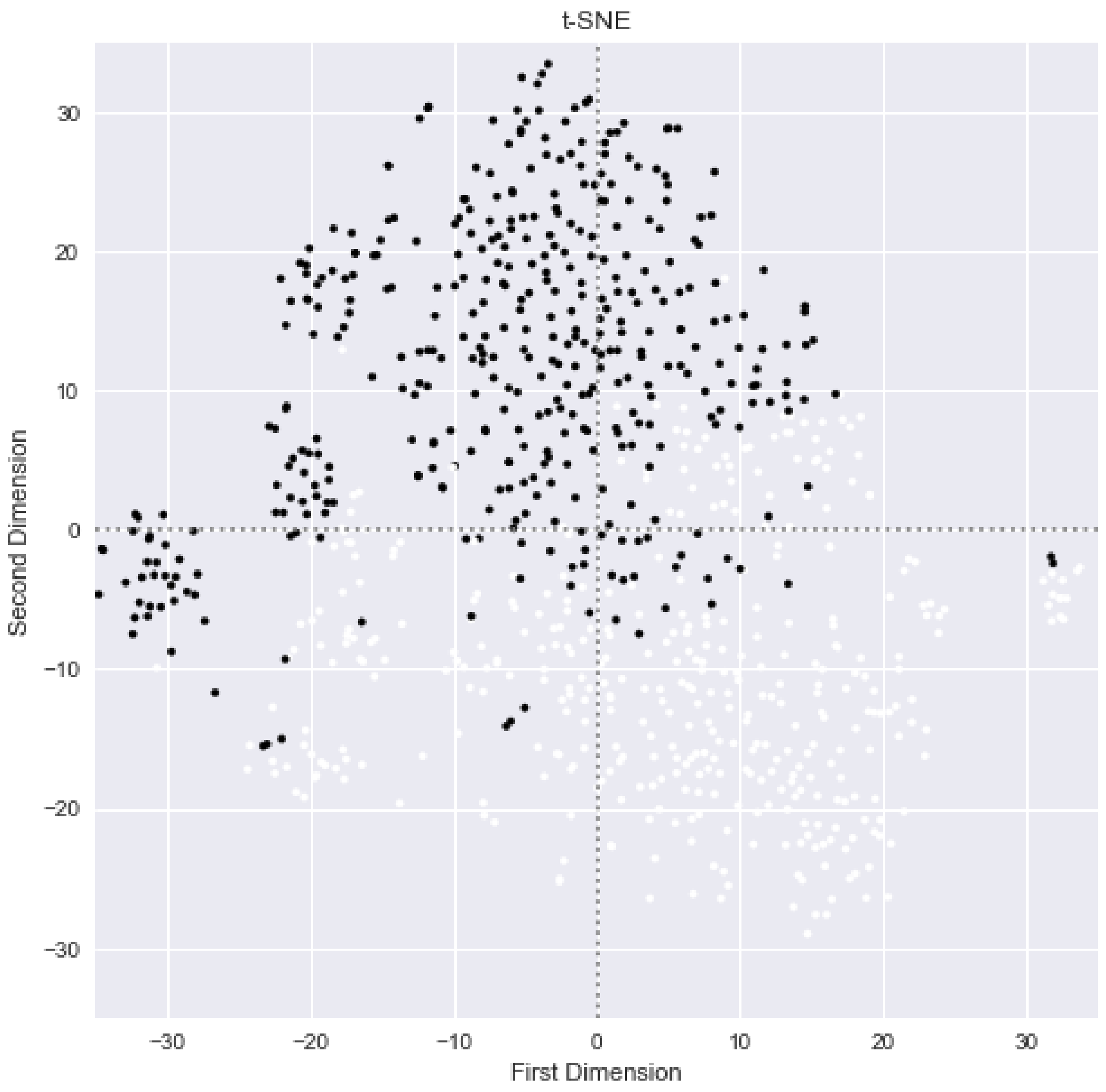
According to both the silhoutte score and the elbow method, the optimal number of clusters is 2, while based on the gap statistic, the optimal number is 8. So we choose 2 as the optimal number of clusters.

11 11

[96]:
```
kmeans_plot(k2)
```



[97]:
```
fig = plt.figure(figsize=(8, 8))
plt.xlim(-35,35)
plt.ylim(-35,35)
plt.hlines(0,-35,35, linestyles='dotted', colors='grey')
plt.vlines(0,-35,35, linestyles='dotted', colors='grey')
plt.scatter(tsneDf['dim1'], tsneDf['dim2'], s=11, c=k2.labels_)
plt.xlabel('First Dimension')
plt.ylabel('Second Dimension')
plt.title('t-SNE');
```



According to the above plots, PCA gives us a better visualization of the data, while for t-SNE, the clusters are overlapped with a fuzzy boundary. PCA has the assumption that the data follows a linear pattern, which may account for why its performance is better. Furthermore, PCA preserves the large distances between points while t-SNE preserves the points which are close to each other, which leads to a plot with a main block in the middle and several satellite clusters dispersed around it. The disadvantage of t-SNE is that it only focuses on neighbors, but is unable to accurately represent the global trends, thus performing worse.