Problem II: K-means Clustering

1452669, Yang LI, April 7

Data Preprocessing

Since the raw trade data is high dimensional, I use Principal Component Analysis (PCA) to reduce them to a few principal components, in particular, I use 2 compared with the LSH method using Euclidean Distance. Hence it improve the average silhouette obviously.

K-means

K-means is the basic and comprehensive clustering algorithm, it accepts an initial set of k means $m_1^{(1)}, m_2^{(1)}, \ldots, m_k^{(1)}$, and proceeds by alternating between two steps:

- Assignment: Assign each observation to the cluster whose mean has the least squared Euclidean distance.

  $S_i^{(t)} = \{x_p : ||x_p - m_i^{(t)}||^2 \leq ||x_p - m_j^{(t)}||^2 \ \forall j, 1 \leq j \leq k\}$

  where each x_p is assigned to exactly one $S^{(t)}$, even if it could be assigned to two or more of them.

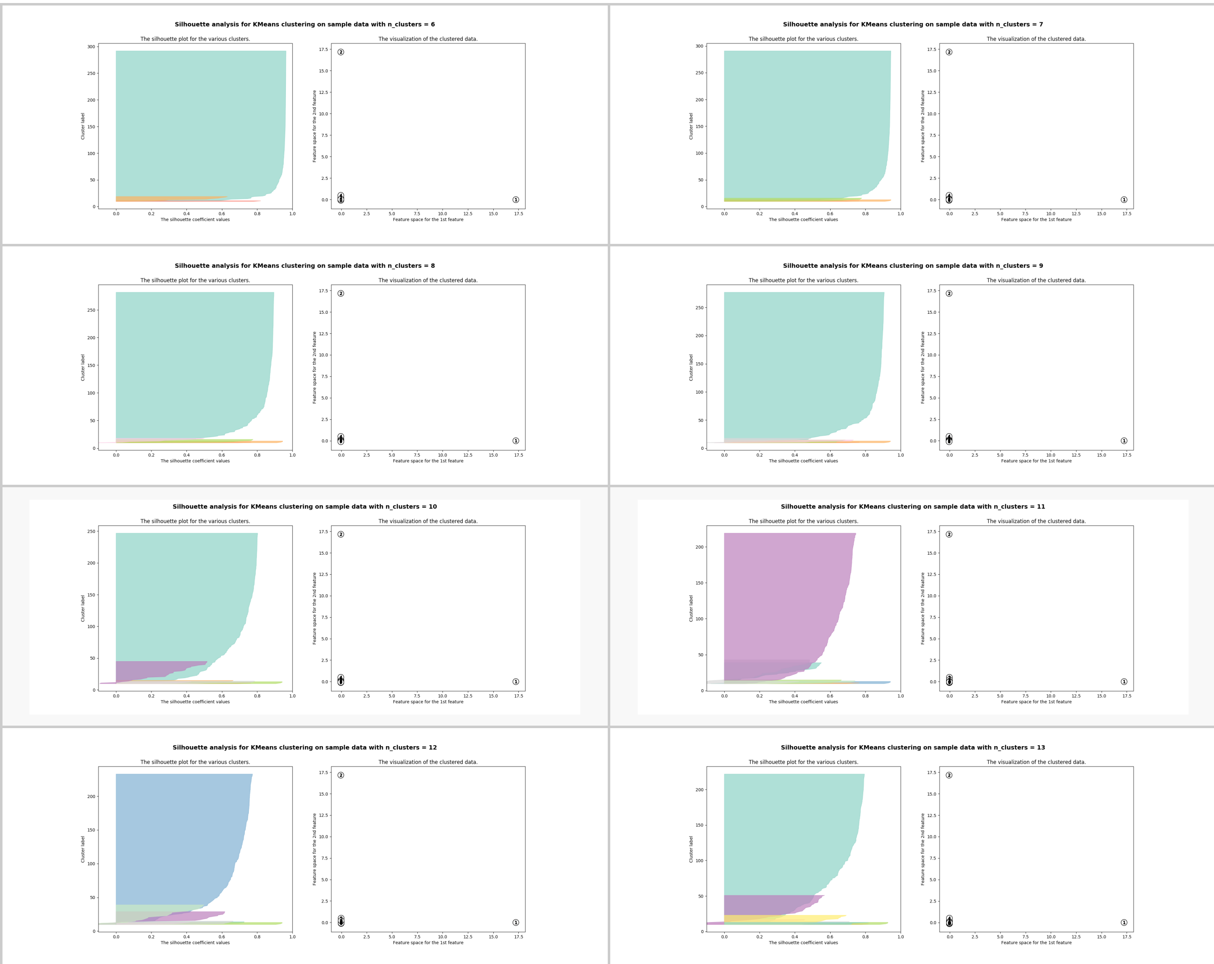
- Update: Calculate the new means to be the centroids of the observations in the new clusters.

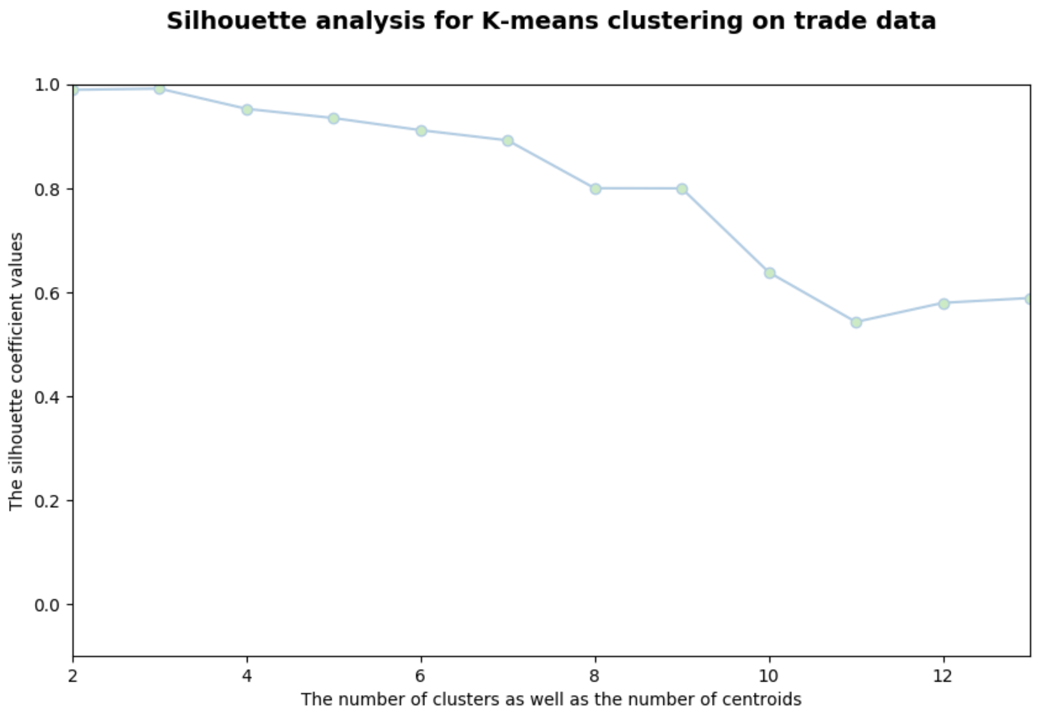
  $m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \Sigma_{x_j \in S_i^{(t)}} x_j$

```
def kmeans(df, random_vip, knns):
    k = int(math.sqrt(df.shape[1] / 2))
    silhouette_avgs = []

    for n_clusters in range(2, k + 2):
        logging.debug("K-means: n_clusters = {}".format(n_clusters))
        clusterer = KMeans(n_clusters=n_clusters)
        X = PCA(n_components=2, whiten=True).fit_transform(df.T)
        cluster_labels = clusterer.fit_predict(X)
        silhouette_avg = silhouette_score(X, cluster_labels)
        silhouette_avgs.append(silhouette_avg)
        print("For n_clusters =", n_clusters,
              "The average silhouette_score in K-means is :",
silhouette_avg)

        # if n_clusters >= k / 2:
        #     plot_silhouette(X, cluster_labels, n_clusters, clusterer)

        res = 0
        no = cluster_labels[df.columns.get_loc(random_vip)]
        for neighbor in knns:
```

```python
            if cluster_labels[df.columns.get_loc(neighbor)] == no:
                res += 1
            else:
                logging.info(
                    "K-means: vipno: {} is not in the same cluster.".format(
                        neighbor))
        print(
            "For k = {} in kNN, there has {} in the same cluster in K-means.".format(
                len(knns), res))

    # plot_kmeans_clusterno(k, silhouette_avgs)

    return silhouette_avgs.index(max(silhouette_avgs)) + 2
```

The silhouette analysis results for different clusters show as follows:





Above figures intuitively indicate the number of clusters in the trade dataset, like 9 for example. The exact silhouette score are in the following table.

number of clusters	silhouette score
2	0.9898142095520185
3	0.9921749295434477
4	0.9532367585544059
5	0.9354793995672006
6	0.912177772818011
7	0.8925583120466012
8	0.8003854540541915
9	0.8001215931883985
10	0.6280774320073931
11	0.5553540967764518
12	0.6433141132870174
13	0.6210252967296004

Compared with the clustering of kNN using LSH, the result are in the following table (duplicate 100 times):

k	correctness
2	1.0
3	0.99
4	0.98
5	0.948
6	0.938
7	0.938
8	0.928
9	0.926
10	0.798
11	0.79
12	0.75
13	0.766

Performance

Time & Space Complexity in Theory

Let t_{dist} be the time to calculate the distance between two objects, K stands for the number of clusters (centroids) and n stands for the number of objects. Thus, given bound number of iterations I.

- time complexity: $O(IKnt_{dist})$ = $O(IKnm)$ where m means for m-dimensional vectors.
- space complexity: $O((n+K)m)$

Benchmark in Practice

Using the common algorithm for K-means clustering, it is a polynomial even though finding the optimal solution to the k-means problem is NP-hard. As we analyzed above, the Lloyd's algorithm is not the global optimum, it always gets a local maximum which differs from the random chosen at the beginning.

In practice, given a bound number of iteration, it cost both in data preprocessing and the finding and calculation of centroids parts.

```
Timer unit: 1e-06 s

Total time: 0.753593 s
File: /Users/Yang/Developer/420235DataMining/hw1/q2/kmeans.py
Function: kmeans at line 11

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
    11                                           @profile
    12                                           def kmeans(df, random_vip, knns):
    13         1        176.0    176.0      0.0      k = int(math.sqrt(df.shape[1] / 2))
    14         1          2.0      2.0      0.0      silhouette_avgs = []
    15
    16        13         16.0      1.2      0.0      for n_clusters in range(2, k + 2):
    17        12       1981.0    165.1      0.3          logging.debug("K-means: n_clusters = {}".format(n_clusters))
    18        12        132.0     11.0      0.0          clusterer = KMeans(n_clusters=n_clusters)
    19        12     294449.0  24537.4     39.1          X = PCA(n_components=2, whiten=True).fit_transform(df.T)
    20        12     414335.0  34527.9     55.0          cluster_labels = clusterer.fit_predict(X)
    21        12      41378.0   3448.2      5.5          silhouette_avg = silhouette_score(X, cluster_labels)
    22        12         30.0      2.5      0.0          silhouette_avgs.append(silhouette_avg)
    23        12         14.0      1.2      0.0          print("For n_clusters =", n_clusters,
    24        12        368.0     30.7      0.0                "The average silhouette_score in K-means is :", silhouette_avg)
    25
    26                                                   # if n_clusters >= k / 2:
    27                                                   #     plot_silhouette(X, cluster_labels, n_clusters, clusterer)
    28
    29        12         12.0      1.0      0.0          res = 0
    30        12        208.0     17.3      0.0          no = cluster_labels[df.columns.get_loc(random_vip)]
    31        72         66.0      0.9      0.0          for neighbor in knns:
    32        60        217.0      3.6      0.0              if cluster_labels[df.columns.get_loc(neighbor)] == no:
    33        60         51.0      0.8      0.0                  res += 1
    34                                                       else:
    35                                                           logging.info(
    36                                                               "K-means: vipno: {} is not in the same cluster.".format(
    37                                                                   neighbor))
    38        12          9.0      0.8      0.0          print(
    39        12         11.0      0.9      0.0              "For k = {} in kNN, there has {} in the same cluster in K-means.".format(
    40        12        134.0     11.2      0.0                  len(knns), res))
    41
    42                                               # plot_kmeans_clusterno(k, silhouette_avgs)
    43
    44         1          4.0      4.0      0.0      return silhouette_avgs.index(max(silhouette_avgs)) + 2
```

```
Line #    Mem usage    Increment   Line Contents
================================================
    11  137.504 MiB  137.504 MiB   @profile
    12                             def kmeans(df, random_vip, knns):
    13  137.504 MiB    0.000 MiB       k = int(math.sqrt(df.shape[1] / 2))
    14  137.504 MiB    0.000 MiB       silhouette_avgs = []
    15
    16  147.027 MiB    0.000 MiB       for n_clusters in range(2, k + 2):
    17  147.023 MiB    0.000 MiB           logging.debug("K-means: n_clusters = {}".format(n_clusters))
    18  147.023 MiB    0.000 MiB           clusterer = KMeans(n_clusters=n_clusters)
    19  147.027 MiB    8.465 MiB           X = PCA(n_components=2, whiten=True).fit_transform(df.T)
    20  147.027 MiB    0.258 MiB           cluster_labels = clusterer.fit_predict(X)
    21  147.027 MiB    0.781 MiB           silhouette_avg = silhouette_score(X, cluster_labels)
    22  147.027 MiB    0.000 MiB           silhouette_avgs.append(silhouette_avg)
    23  147.027 MiB    0.000 MiB           print("For n_clusters =", n_clusters,
    24  147.027 MiB    0.020 MiB                 "The average silhouette_score in K-means is :", silhouette_avg)
    25
    26                                     # if n_clusters >= k / 2:
    27                                     #     plot_silhouette(X, cluster_labels, n_clusters, clusterer)
    28
    29  147.027 MiB    0.000 MiB           res = 0
    30  147.027 MiB    0.000 MiB           no = cluster_labels[df.columns.get_loc(random_vip)]
    31  147.027 MiB    0.000 MiB           for neighbor in knns:
    32  147.027 MiB    0.000 MiB               if cluster_labels[df.columns.get_loc(neighbor)] == no:
    33  147.027 MiB    0.000 MiB                   res += 1
    34                                         else:
    35                                             logging.info(
    36                                                 "K-means: vipno: {} is not in the same cluster.".format(
    37                                                     neighbor))
    38  147.027 MiB    0.000 MiB           print(
    39  147.027 MiB    0.000 MiB               "For k = {} in kNN, there has {} in the same cluster in K-means.".format(
    40  147.027 MiB    0.000 MiB                   len(knns), res))
    41
    42                                 # plot_kmeans_clusterno(k, silhouette_avgs)
    43
    44  147.027 MiB    0.000 MiB       return silhouette_avgs.index(max(silhouette_avgs)) + 2
```

Screenshot

```
/usr/local/bin/python3.6 /Users/Yang/Developer/420235DataMining/hw1/main.py
INFO:root:DataFrame shape: (2635, 298)
<class 'pandas.core.frame.DataFrame'>
Index: 2635 entries, 10000004 to 40000700
Columns: 298 entries, 13205496418 to 6222021615015662822
dtypes: float64(298)
memory usage: 6.0+ MB
DEBUG:root:DataFrame info: None
INFO:root:random vipno: 1590140606433
vipno in ranked order using kNN(k = 5):
1590160754770
1591011326672
1594140467704
1592140611301
1591020667889
DEBUG:root:K-means: n_clusters = 2
DEBUG:root:K-means: n_clusters = 3
For n_clusters = 2 The average silhouette_score in K-means is : 0.9898142095574196
For k = 5 in kNN, there has 5 in the same cluster in K-means.
DEBUG:root:K-means: n_clusters = 4
For n_clusters = 3 The average silhouette_score in K-means is : 0.9921749295445973
For k = 5 in kNN, there has 5 in the same cluster in K-means.
For n_clusters = 4 The average silhouette_score in K-means is : 0.9532367586043914
For k = 5 in kNN, there has 5 in the same cluster in K-means.
DEBUG:root:K-means: n_clusters = 5
For n_clusters = 5 The average silhouette_score in K-means is : 0.9354793989646298
For k = 5 in kNN, there has 5 in the same cluster in K-means.
DEBUG:root:K-means: n_clusters = 6
For n_clusters = 6 The average silhouette_score in K-means is : 0.9121777730153087
For k = 5 in kNN, there has 5 in the same cluster in K-means.
DEBUG:root:K-means: n_clusters = 7
DEBUG:root:K-means: n_clusters = 8
For n_clusters = 7 The average silhouette_score in K-means is : 0.8925583105475965
For k = 5 in kNN, there has 5 in the same cluster in K-means.
For n_clusters = 8 The average silhouette_score in K-means is : 0.8003854367303654
For k = 5 in kNN, there has 5 in the same cluster in K-means.
DEBUG:root:K-means: n_clusters = 9
DEBUG:root:K-means: n_clusters = 10
For n_clusters = 9 The average silhouette_score in K-means is : 0.8054482598322832
For k = 5 in kNN, there has 5 in the same cluster in K-means.
For n_clusters = 10 The average silhouette_score in K-means is : 0.6352370948494438
DEBUG:root:K-means: n_clusters = 11
For k = 5 in kNN, there has 5 in the same cluster in K-means.
For n_clusters = 11 The average silhouette_score in K-means is : 0.6411969201951703
DEBUG:root:K-means: n_clusters = 12
For k = 5 in kNN, there has 5 in the same cluster in K-means.
For n_clusters = 12 The average silhouette_score in K-means is : 0.636311067583867
For k = 5 in kNN, there has 5 in the same cluster in K-means.
DEBUG:root:K-means: n_clusters = 13
For n_clusters = 13 The average silhouette_score in K-means is : 0.6167497931869516
For k = 5 in kNN, there has 5 in the same cluster in K-means.

Process finished with exit code 0
```

```
/usr/local/bin/python3.6 /Users/Yang/Developer/420235DataMining/hw1/main.py
<class 'pandas.core.frame.DataFrame'>
Index: 2635 entries, 10000004 to 40000700
Columns: 298 entries, 13205496418 to 6222021615015662822
dtypes: float64(298)
memory usage: 6.0+ MB
WARNING:root:For n_clusters = 2, the average correctness of K-means is 1.0
WARNING:root:For n_clusters = 3, the average correctness of K-means is 0.99
WARNING:root:For n_clusters = 4, the average correctness of K-means is 0.9800000000000001
WARNING:root:For n_clusters = 5, the average correctness of K-means is 0.9480000000000001
WARNING:root:For n_clusters = 6, the average correctness of K-means is 0.9380000000000001
WARNING:root:For n_clusters = 7, the average correctness of K-means is 0.9380000000000001
WARNING:root:For n_clusters = 8, the average correctness of K-means is 0.9279999999999999
WARNING:root:For n_clusters = 9, the average correctness of K-means is 0.9259999999999999
WARNING:root:For n_clusters = 10, the average correctness of K-means is 0.798
WARNING:root:For n_clusters = 11, the average correctness of K-means is 0.79
WARNING:root:For n_clusters = 12, the average correctness of K-means is 0.75
WARNING:root:For n_clusters = 13, the average correctness of K-means is 0.766

Process finished with exit code 0
```