Question3 报告

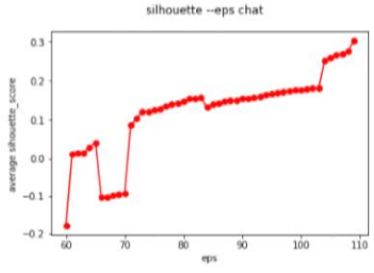
一、结果

a) 第一问的结果：

为了能更清晰的看到结果，这里我把当 eps∈(60, 110)的 silhouette 结果逐条打印了出来，这里选定 min_samples = 2

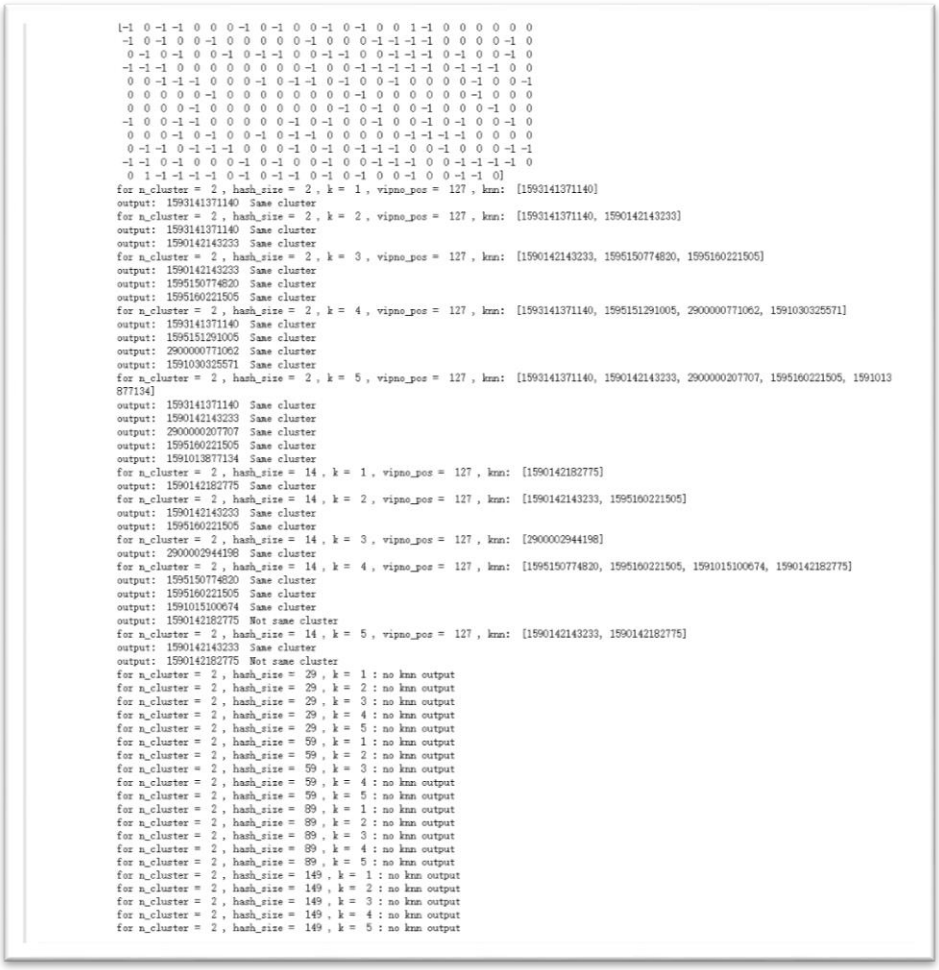
```
[6]: if __name__ == '__main__':
        data, data_array, vipno_num, vipno_len = pre_data()
        #发现当且仅当min_samples=2时，才能够得出两个以上的聚类，
        #当eps小于100时，能得出两个以上聚类
        #silhouette在[1, 100]区间内随eps递增
        #当eps=103时，得出2个簇，silhouette_score=0.18
        question_a(data_array)
        question_b(data, data_array, vipno_num, vipno_len, eps = 103)
```

```
For eps= 60 The number of clusters is: 2 The average silhouette_score is: -0.178271264861
For eps= 61 The number of clusters is: 1 The average silhouette_score is: 0.0104240899559
For eps= 62 The number of clusters is: 1 The average silhouette_score is: 0.014039844931
For eps= 63 The number of clusters is: 1 The average silhouette_score is: 0.014039844931
For eps= 64 The number of clusters is: 1 The average silhouette_score is: 0.0281371828833
For eps= 65 The number of clusters is: 1 The average silhouette_score is: 0.0412105143044
For eps= 66 The number of clusters is: 2 The average silhouette_score is: -0.103694985381
For eps= 67 The number of clusters is: 2 The average silhouette_score is: -0.102268573101
For eps= 68 The number of clusters is: 2 The average silhouette_score is: -0.0993383322543
For eps= 69 The number of clusters is: 2 The average silhouette_score is: -0.0962247346922
For eps= 70 The number of clusters is: 2 The average silhouette_score is: -0.0927013048853
For eps= 71 The number of clusters is: 1 The average silhouette_score is: 0.0856427448906
For eps= 72 The number of clusters is: 1 The average silhouette_score is: 0.102469636219
For eps= 73 The number of clusters is: 1 The average silhouette_score is: 0.120685347081
For eps= 74 The number of clusters is: 1 The average silhouette_score is: 0.120685347081
For eps= 75 The number of clusters is: 1 The average silhouette_score is: 0.124364680007
For eps= 76 The number of clusters is: 1 The average silhouette_score is: 0.128091313184
For eps= 77 The number of clusters is: 1 The average silhouette_score is: 0.135554267885
For eps= 78 The number of clusters is: 1 The average silhouette_score is: 0.139259484897
For eps= 79 The number of clusters is: 1 The average silhouette_score is: 0.142967212773
For eps= 80 The number of clusters is: 1 The average silhouette_score is: 0.146577191124
For eps= 81 The number of clusters is: 1 The average silhouette_score is: 0.153816497939
For eps= 82 The number of clusters is: 1 The average silhouette_score is: 0.153816497939
For eps= 83 The number of clusters is: 1 The average silhouette_score is: 0.157164685298
For eps= 84 The number of clusters is: 2 The average silhouette_score is: 0.131467385811
For eps= 85 The number of clusters is: 2 The average silhouette_score is: 0.139350061409
For eps= 86 The number of clusters is: 2 The average silhouette_score is: 0.141966582249
For eps= 87 The number of clusters is: 2 The average silhouette_score is: 0.146182515127
For eps= 88 The number of clusters is: 2 The average silhouette_score is: 0.148498791335
For eps= 89 The number of clusters is: 2 The average silhouette_score is: 0.148498791335
For eps= 90 The number of clusters is: 2 The average silhouette_score is: 0.15328849904
For eps= 91 The number of clusters is: 2 The average silhouette_score is: 0.15528197333
For eps= 92 The number of clusters is: 2 The average silhouette_score is: 0.15774729805
For eps= 93 The number of clusters is: 2 The average silhouette_score is: 0.160209151923
For eps= 94 The number of clusters is: 2 The average silhouette_score is: 0.164886695094
For eps= 95 The number of clusters is: 2 The average silhouette_score is: 0.167292206511
For eps= 96 The number of clusters is: 2 The average silhouette_score is: 0.169655982594
For eps= 97 The number of clusters is: 2 The average silhouette_score is: 0.172045099018
For eps= 98 The number of clusters is: 2 The average silhouette_score is: 0.174384245737
For eps= 99 The number of clusters is: 2 The average silhouette_score is: 0.176484946327
For eps= 100 The number of clusters is: 2 The average silhouette_score is: 0.176484946327
For eps= 101 The number of clusters is: 2 The average silhouette_score is: 0.178381161654
For eps= 102 The number of clusters is: 2 The average silhouette_score is: 0.180361278231
For eps= 103 The number of clusters is: 2 The average silhouette_score is: 0.180361278231
For eps= 104 The number of clusters is: 1 The average silhouette_score is: 0.251571567333
For eps= 105 The number of clusters is: 1 The average silhouette_score is: 0.258824616745
For eps= 106 The number of clusters is: 1 The average silhouette_score is: 0.265948563235
For eps= 107 The number of clusters is: 1 The average silhouette_score is: 0.269630934976
For eps= 108 The number of clusters is: 1 The average silhouette_score is: 0.276834283372
For eps= 109 The number of clusters is: 1 The average silhouette_score is: 0.302105382088
```



b) 第二问的结果：

这里将 DBSCAN 算法聚类之后得到的标签数组打印了出来，eps=103，min_samples=2



```
[-1  0 -1 -1  0  0  0 -1  0 -1  0  0 -1  0 -1  0  0  1 -1  0  0  0  0  0  0
 -1  0 -1  0  0 -1  0  0  0  0  0 -1  0  0  0 -1 -1 -1 -1  0  0  0  0 -1  0
  0 -1  0 -1  0  0 -1  0 -1 -1  0  0 -1 -1  0  0 -1 -1 -1  0 -1  0  0 -1  0
 -1 -1 -1  0  0  0  0  0  0  0  0 -1  0  0 -1 -1 -1 -1 -1  0 -1 -1 -1  0  0
  0  0 -1 -1 -1  0  0  0 -1  0 -1 -1  0 -1  0  0  0 -1  0  0  0  0 -1  0  0 -1
  0  0  0  0  0 -1  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0 -1  0  0  0
  0  0  0  0 -1  0  0  0  0  0  0  0  0  0 -1  0 -1  0  0 -1  0  0  0 -1  0  0
 -1  0  0 -1 -1  0  0  0  0  0 -1  0 -1  0  0 -1  0  0 -1  0 -1  0  0 -1  0
  0  0  0 -1  0 -1  0  0 -1  0 -1 -1  0  0  0  0  0 -1 -1 -1 -1  0  0  0  0
  0 -1 -1  0 -1 -1 -1  0  0  0 -1  0 -1  0 -1 -1 -1  0  0 -1  0  0  0 -1 -1
 -1 -1  0 -1  0  0  0 -1  0 -1  0  0 -1  0  0 -1 -1 -1  0  0 -1 -1 -1 -1  0
  0  1 -1 -1 -1 -1  0 -1  0 -1 -1  0 -1  0 -1  0  0 -1  0  0 -1 -1  0]
for n_cluster =  2 , hash_size =  2 , k =  1 , vipno_pos =  127 , knn:  [1593141371140]
output:  1593141371140  Same cluster
for n_cluster =  2 , hash_size =  2 , k =  2 , vipno_pos =  127 , knn:  [1593141371140, 1590142143233]
output:  1593141371140  Same cluster
output:  1590142143233  Same cluster
for n_cluster =  2 , hash_size =  2 , k =  3 , vipno_pos =  127 , knn:  [1590142143233, 1595150774820, 1595160221505]
output:  1590142143233  Same cluster
output:  1595150774820  Same cluster
output:  1595160221505  Same cluster
for n_cluster =  2 , hash_size =  2 , k =  4 , vipno_pos =  127 , knn:  [1593141371140, 1595151291005, 2900000771062, 15910303255571]
output:  1593141371140  Same cluster
output:  1595151291005  Same cluster
output:  2900000771062  Same cluster
output:  15910303255571  Same cluster
for n_cluster =  2 , hash_size =  2 , k =  5 , vipno_pos =  127 , knn:  [1593141371140, 1590142143233, 2900000207707, 1595160221505, 1591013
877134]
output:  1593141371140  Same cluster
output:  1590142143233  Same cluster
output:  2900000207707  Same cluster
output:  1595160221505  Same cluster
output:  1591013877134  Same cluster
for n_cluster =  2 , hash_size =  14 , k =  1 , vipno_pos =  127 , knn:  [1590142182775]
output:  1590142182775  Same cluster
for n_cluster =  2 , hash_size =  14 , k =  2 , vipno_pos =  127 , knn:  [1590142143233, 1595160221505]
output:  1590142143233  Same cluster
output:  1595160221505  Same cluster
for n_cluster =  2 , hash_size =  14 , k =  3 , vipno_pos =  127 , knn:  [2900002944198]
output:  2900002944198  Same cluster
for n_cluster =  2 , hash_size =  14 , k =  4 , vipno_pos =  127 , knn:  [1595150774820, 1595160221505, 1591015100674, 1590142182775]
output:  1595150774820  Same cluster
output:  1595160221505  Same cluster
output:  1591015100674  Same cluster
output:  1590142182775  Not same cluster
for n_cluster =  2 , hash_size =  14 , k =  5 , vipno_pos =  127 , knn:  [1590142143233, 1590142182775]
output:  1590142143233  Same cluster
output:  1590142182775  Not same cluster
for n_cluster =  2 , hash_size =  29 , k =  1 : no knn output
for n_cluster =  2 , hash_size =  29 , k =  2 : no knn output
for n_cluster =  2 , hash_size =  29 , k =  3 : no knn output
for n_cluster =  2 , hash_size =  29 , k =  4 : no knn output
for n_cluster =  2 , hash_size =  29 , k =  5 : no knn output
for n_cluster =  2 , hash_size =  59 , k =  1 : no knn output
for n_cluster =  2 , hash_size =  59 , k =  2 : no knn output
for n_cluster =  2 , hash_size =  59 , k =  3 : no knn output
for n_cluster =  2 , hash_size =  59 , k =  4 : no knn output
for n_cluster =  2 , hash_size =  59 , k =  5 : no knn output
for n_cluster =  2 , hash_size =  89 , k =  1 : no knn output
for n_cluster =  2 , hash_size =  89 , k =  2 : no knn output
for n_cluster =  2 , hash_size =  89 , k =  3 : no knn output
for n_cluster =  2 , hash_size =  89 , k =  4 : no knn output
for n_cluster =  2 , hash_size =  89 , k =  5 : no knn output
for n_cluster =  2 , hash_size =  149 , k =  1 : no knn output
for n_cluster =  2 , hash_size =  149 , k =  2 : no knn output
for n_cluster =  2 , hash_size =  149 , k =  3 : no knn output
for n_cluster =  2 , hash_size =  149 , k =  4 : no knn output
for n_cluster =  2 , hash_size =  149 , k =  5 : no knn output
```

二、分析

DBSCAN 算法的基本思路：

1. 给定数据点集合 D，邻域半径 eps，以及最小数目 min_samples。 生成簇标识 C
2. 选取一个未被标记的数据点 p，判断该数据点是否为核心对象，若是核心对象则跳到步骤 3，否则 4
3. 将数据点 p 密度可达的所有数据点标记为 C，表明这些数据点已被分类且属于簇 C
4. 若数据点 p 是噪声点，将 p 标记为噪声点，否则标记为非噪声点的非核心对象
5. 更新簇标识 C，跳回 2，直到所有数据点被标记

本题中 eps 和 min_samples 的值都非常难确定，在调参数的过程中，我发现，当 min_samples 的值确定的前提下，其 silhouette 参数会随着 eps 的增加而增加，所以我试着把生成的簇的个数打印出来之后，发现当且仅当 min_samples = 2 时，才有可能出现两个以上的簇。而假设 min_samples=2，且簇的个数必须为两个以上，从打印出的结果来看当 eps=103 时，silhouette 参数能够取到最大值 0.18。

(一) 取 KNN 的结果进行验证分析

当 eps=102,min_samples=2，簇的个数为 2 时，取 KNN 的结果验证，由最上面第二图可知验证通过。但是 silhouette 参数比较小，而且考虑到第二题中的出现问

题，为了进一步分析，我将此时样本的标签打印出来：

```
[-1  0 -1 -1  0  0  0 -1  0 -1  0  0 -1  0 -1  0  0  1 -1  0  0  0  0  0  0
 -1  0 -1  0  0 -1  0  0  0  0  0 -1  0  0  0 -1 -1 -1 -1  0  0  0  0 -1  0
  0 -1  0 -1  0  0 -1  0 -1 -1  0  0 -1 -1  0  0 -1 -1 -1  0 -1  0  0 -1  0
 -1 -1 -1  0  0  0  0  0  0  0  0 -1  0  0 -1 -1 -1 -1 -1  0 -1 -1 -1  0  0
  0  0 -1 -1 -1  0  0  0 -1  0 -1 -1  0 -1  0  0 -1  0  0  0  0 -1  0  0 -1
  0  0  0  0  0 -1  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0 -1  0  0  0
  0  0  0  0 -1  0  0  0  0  0  0  0  0 -1  0 -1  0  0 -1  0  0  0 -1  0  0
 -1  0  0 -1 -1  0  0  0  0  0 -1  0 -1  0  0 -1  0  0 -1  0 -1  0  0 -1  0
  0  0  0 -1  0 -1  0  0 -1  0 -1 -1  0  0  0  0  0 -1 -1 -1 -1  0  0  0  0
  0 -1 -1  0 -1 -1 -1  0  0  0 -1  0 -1  0 -1 -1 -1  0  0 -1  0  0  0 -1 -1
 -1 -1  0 -1  0  0  0 -1  0 -1  0  0 -1  0  0 -1 -1 -1  0  0 -1 -1 -1 -1  0
  0  1 -1 -1 -1 -1  0 -1  0 -1 -1  0 -1  0 -1  0  0 -1  0  0 -1 -1  0]
```

可以看到样本中出现了大量的噪点，结合之前第二题的结论，可能是由于数据源中大量数据相似，只有少量噪点数据。为了验证这一点，我取 k=4,eps=300（k=4 时簇的个数为 1，eps=300 保证 silhouette 足够大）并查看结果：

```
[ 0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0 -1  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0 -1  0
  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0
  0  0  0  0 -1 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
```

可知此时只有少量的噪点，簇个数为 1，所以数据源中大量数据相似，只有少量噪点数据这个猜想是正确的。

(二) 和 KMeans 的最优结果进行比较

基于前面的分析，当簇的个数为 1 时，DBSCAN 不存在最优情况；所以我就拿簇为 2 的时候的最优解和 KMeans 最优解进行比较。

由第二题可知：

- KMeans_silhouette = 0.94 > DBSCAN_silhouette = 0.18
- KMeans_time = 0.275 < DBSCAN_time = 0.8

而且 KMeans 中样本标签的情况如图：

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
```

很明显，在相同条件下，DBSCAN 的结果没有 KMeans 的效果好，原因可能是因为：DBSCAN 算法需要通过比较样本与样本之间的距离来确定聚类的，对于二维空间的点，可以使用欧几里得距离来计算，而对于很高维度的数据集，对其距离的定义会非常困难，而这次使用的数据正是维度很高很高，因此导致聚类效果差。

三、性能

可以看出 DBSCAN 花了大量的时间，可能是由于数据量较大，而且数据维度高，DBSCAN 要求较大的内存支持，I/O 消耗也很大。