Assignment 4: Data Analysis

Junzhu Xiang, Haomin Lin

Kmeans Part:

(1) K must be a positive integer. If I input K which is less than or equal to 0, then throw errors.

```
C:\Users\jzhsi\Desktop\CSE6010\A4>A4 faithful-Kmeans.txt -10 out.txt
This program is: A4
Error! K must be a positive integer!

C:\Users\jzhsi\Desktop\CSE6010\A4>A4 faithful-Kmeans.txt 0 out.txt
This program is: A4
Error! K must be a positive integer!
```

(2) K must be less than or equal to data size in the input file. If K is larger than that, then throw errors. Now, data size is 250, if I input K which is 251:

```
C:\Users\jzhsi\Desktop\CSE6010\A4>A4 faithful-Kmeans.txt 251 out.txt
This program is: A4
Input file name is: faithful-Kmeans.txt
The number of clusters is: 251
Output file name is: out.txt
Error! K is 251, data_size is 250!
```

(3) I input K = 250, it should show that every data is a cluster centroid and every cluster has only one item if every data is different from each other. But in input file, some data are same, so the result will be different. As you can see the picture below, some clusters have only 1 item, some have 0, and some have two. This can be verified in the input file(left) and output file(right). There are two same data, which is (1.750000,47.0000).

```
size of each cluster:
1 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 1 1 1 1 0 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1 1 1 1 2 1 1 2 1 1 2 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1
1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 0 1 1 1 0 1 1 1
```

1.833000 54.000000	1.833000 54.000000 10
3.917000 84.000000	3.917000 84.000000 11
4.200000 78.000000	4.200000 78.000000 12
1.750000 47.000000	1.750000 47.000000 13
4.700000 83.000000	4.700000 83.000000 14
2.167000 52.000000	2.167000 52.000000 15
1.750000 62.000000	1.750000 62.000000 16
4.800000 84.000000	4.800000 84.000000 17
1.600000 52.000000	1.600000 52.000000 18
4.250000 79.000000	4.250000 79.000000 19
1.800000 51.000000	1.800000 51.000000 20
1.750000 47.000000	1.750000 47.000000 13

(4) If I choose K = 2, this program can output results in a proper manner. The picture below is the final result after 6 round training. You can see, the second picture, three values in the first line mean the cluster number, the data size, and its dimension. From second line, these three values mean the value of dimension_1, the value of dimension_ 2 , and it label.

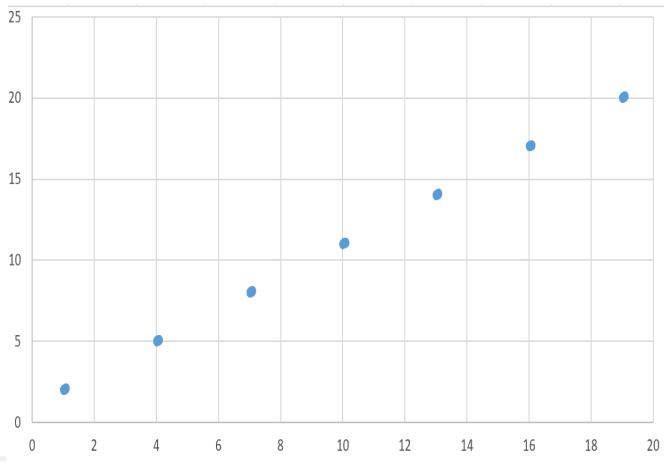
```
size of each cluster:
157 93
coordinates of each center:
4.298146 80.343949
2.098742 54.967742
root mean square = 8106.532487

After 2 round training, Root Mean Square doesn't change!
```

```
2 250 2
3.600000 79.000000 0
1.800000 54.000000 1
3.333000 74.000000 0
2.283000 62.000000 1
4.533000 85.000000 0
2.883000 55.000000 1
4.700000 88.000000 0
3.600000 85.000000 0
1.950000 51.000000 1
4.350000 85.000000 0
1.833000 54.000000 1
3.917000 84.000000 0
4.200000 78.000000 0
1.750000 47.000000 1
```

(5) I created a testing data file to verify my program is correct. This data file contains 70 two-dimension data, and every 10 data is a group. In a group, every data is different from each and very close to each data and its distribution. The first 7 data are centroids, which are from seven group. The picture below on the right shows the result. As you can see, the output shows 7 clusters, and each contains 10 data, and each centroid is the average of 10 data in the group. So, through this, we can verify the program is correct. If you want to verify K-means correctness, you can use Kmeans_data_test.txt as the input file to run this program.

File Edit Format View Help

```
70 2
1.01    2.01
4.01    5.01
7.01    8.01
10.01   11.01
13.01   14.01
16.01   17.01
19.01   20.01
1.02    2.02
1.03    2.03
1.04    2.04
1.05    2.05
1.06    2.06
1.07    2.07
1.08    2.08
1.09    2.09
1.1     2.1
4.02    5.02
4.03    5.03
4.04    5.04
4.05    5.05
4.06    5.06
4.07    5.07
```




```
size of each cluster:
10 10 10 10 10 10 10
coordinates of each center:
1.055000 2.055000
4.055000 5.055000
7.055000 8.055000
10.055000 11.055000
13.055000 14.055000
16.055000 17.055000
19.055000 20.055000
root mean square = 0.115500
```

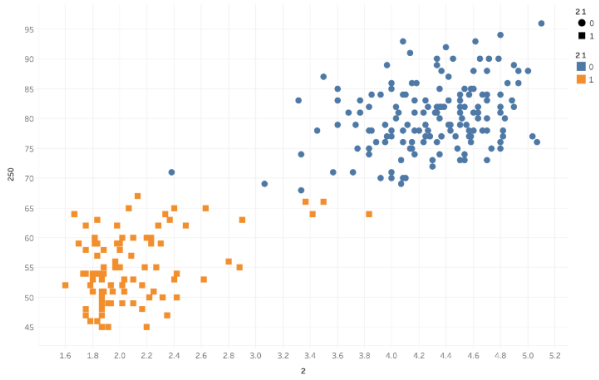
(6) In Kmeans program, we choose a method to decide when to stop training. If in the current round root mean square(RMS) is equal to the root mean square of previous round, which means RMS does not change, so training completes. So in this case, we should stop training. And also, we set a max round, which is 100, if after 100 rounds training, RMS is still changing, we also should stop.

(7) In our program, we have considered the situation that the data has more than 2 dimensions, e.g. 4 dimensions. The picture below shows the result after processing 4 dimensions data. If you want to test 4 dimensions data, you can use faithful-Kmeans-4.txt and faithful-KNN-4.txt in the zip file. (Here, K =7)

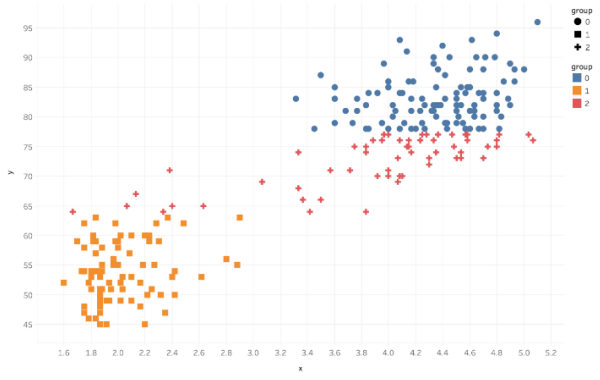
```
size of each cluster:
54 34 37 23 44 37 21
coordinates of each center:
4.327426 78.870370 4.327426 78.870370
1.962824 49.000000 1.962824 49.000000
4.121108 73.459459 4.121108 73.459459
2.463000 63.260870 2.463000 63.260870
4.330273 83.613636 4.330273 83.613636
2.030568 55.648649 2.030568 55.648649
4.513429 90.000000 4.513429 90.000000
root mean square = 2018.597495
```

```
7 250 4
3.600000 79.000000 3.600000 79.000000 0
1.800000 54.000000 1.800000 54.000000 5
3.333000 74.000000 3.333000 74.000000 2
2.283000 62.000000 2.283000 62.000000 3
4.533000 85.000000 4.533000 85.000000 4
2.883000 55.000000 2.883000 55.000000 5
4.700000 88.000000 4.700000 88.000000 6
3.600000 85.000000 3.600000 85.000000 4
1.950000 51.000000 1.950000 51.000000 1
4.350000 85.000000 4.350000 85.000000 4
1.833000 54.000000 1.833000 54.000000 5
3.917000 84.000000 3.917000 84.000000 4
4.200000 78.000000 4.200000 78.000000 0
1.750000 47.000000 1.750000 47.000000 1
4.700000 83.000000 4.700000 83.000000 4
```

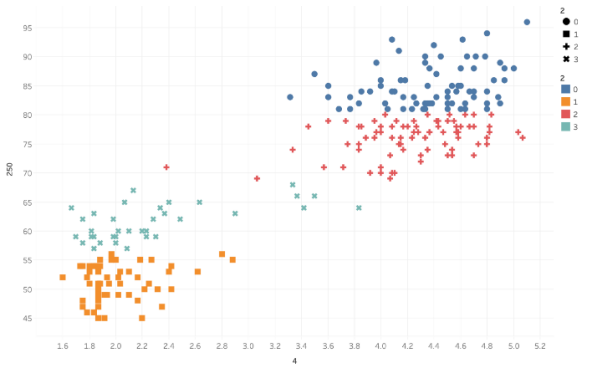
(8) By choosing several K (number of clusters), we got several different results. We selected K from 2 to 7 and plotted 6 drawings to demonstrate the 6 results. After discussing and analyzing, we think when K equals to 2, we can get a reasonable result. Below are 6 images of 6 testing results.



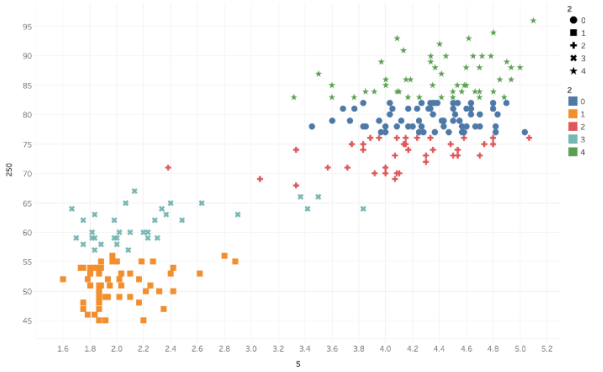
K = 2



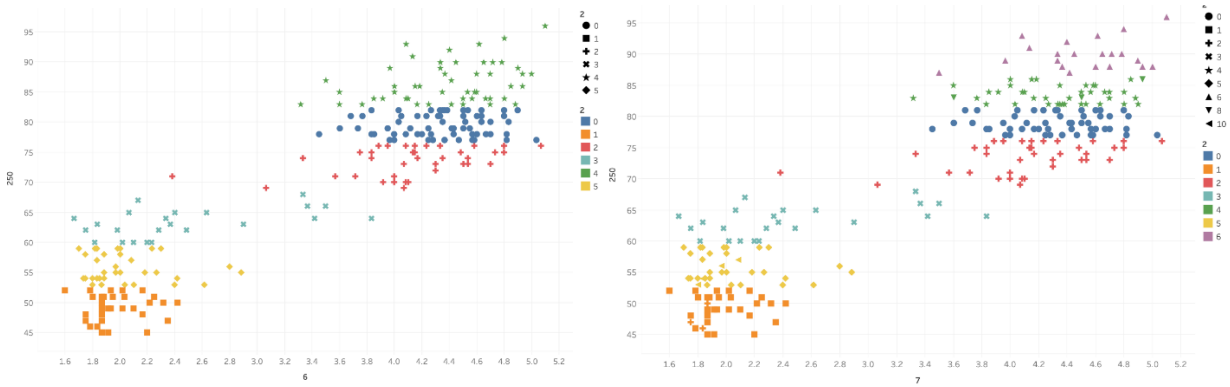
K = 3



K = 4



K = 5



K = 6 K = 7

As you can see images above, X axis represents the duration of the eruption in minutes, and Y axis represents the time duration in minutes from the previous eruption until this eruption occurred. Generally, these data can be divided to two parts: first, long duration of eruption and long waiting duration; second, short duration of eruption and short waiting duration. There are few sporadic data between those two parts, but it doesn't matter. Normally, if the waiting duration is longer, the energy in the spring will be larger. As a result, its eruption duration will be longer correspondingly. Therefore, when we set K as 2, it can illustrate the realistic meaning of these data. However, with K increasing, the number of clusters will increase, and the shape of every cluster becomes flatter and flatter. And we cannot its realistic meanings. So, this is the reason why we set K as 2.

KNN Part:

The KNN program can work correctly by reading unlabeled data and label them with KNN algorithm.

Through file I/O in C, the program import data produced by the K-means program called "trainer" and data to be classified called "trainee". And it also defines the name of output file and the k:

```
[hlin374@coc-ice ~]$ ./main o.txt T.txt R.txt 11
This program is: ./main
Trainer file name is: o.txt
Trainee file name is: T.txt
The number of clusters is: 11
Output file name is: R.txt
```

Since K must be positive, when we enter a negative k or 0, the program will return error:

```
[hlin374@coc-ice ~]$ ./main o.txt T.txt R.txt -1
This program is: ./main
Error! Input should be positive!
```

```
[hlin374@coc-ice ~]$ ./main o.txt T.txt R.txt 0
This program is: ./main
Error! Input should be positive!
```

Then the KNN algorithm is implemented as follows:

(1) Calculate distances and then store them: As we start the program, we read the sizes of trainer and trainee, dimension of the data and the number of labels.

```
raw data size:18
trainer data size:250
raw data dimenion:2
trainer data dimenion:2
number of labels:10
```

Then the program starts to calculate the distances between the first trainee and all the trainers, using priority queue to store the distances with the trainer's label in one priority queue, with smaller distance having higher priority:

```
6.072107 : 0
6.079991 : 0
6.168983 : 1
7.000321 : 1
7.002857 : 1
```

(2) Find the nearest neighbors: Then the program will move on to find the nearest neighbors of the first trainee, with k set, the program will pick out k smallest distances and record their labels. In this way, we can know in k nearest neighbors, how many members each cluster has.

```
label 1 now has one more nearest neighbor, total 1
label 1 now has one more nearest neighbor, total 2
label 1 now has one more nearest neighbor, total 3
label 1 now has one more nearest neighbor, total 4
label 1 now has one more nearest neighbor, total 5
label 1 now has one more nearest neighbor, total 6
label 1 now has one more nearest neighbor, total 7
label 1 now has one more nearest neighbor, total 8
label 1 now has one more nearest neighbor, total 9
label 1 now has one more nearest neighbor, total 10
label 1 now has one more nearest neighbor, total 11
```

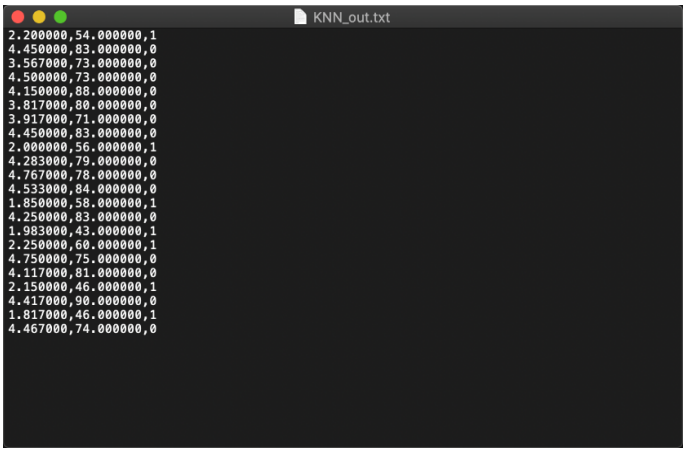
(3) Label the trainee and write it to the output file: After figuring out how many nearest neighbors one cluster has, we can compare which cluster has more nearest neighbors and label the data (we use trainer data with 2 clusters to give an example.

```
node #17:
The cluster 0 has 3 members
The cluster 2 has 8 members
```

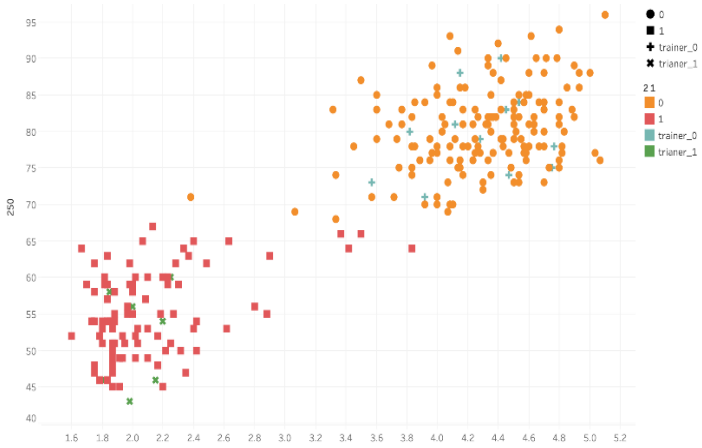
And we can store the result in one .txt file:

```
The following information has been written in the output file:
2.200000,54.000000,1
4.450000,83.000000,0
3.567000,73.000000,0
4.500000,73.000000,0
4.150000,88.000000,0
3.817000,80.000000,0
3.917000,71.000000,0
4.450000,83.000000,0
2.000000,56.000000,1
4.283000,79.000000,0
4.767000,78.000000,0
4.533000,84.000000,0
1.850000,58.000000,1
4.250000,83.000000,0
1.983000,43.000000,1
2.250000,60.000000,1
4.750000,75.000000,0
4.117000,81.000000,0
2.150000,46.000000,1
4.417000,90.000000,0
1.817000,46.000000,1
4.467000,74.000000,0
```

As is in the output file:



With data labeled, we can plot a graph to see the result:



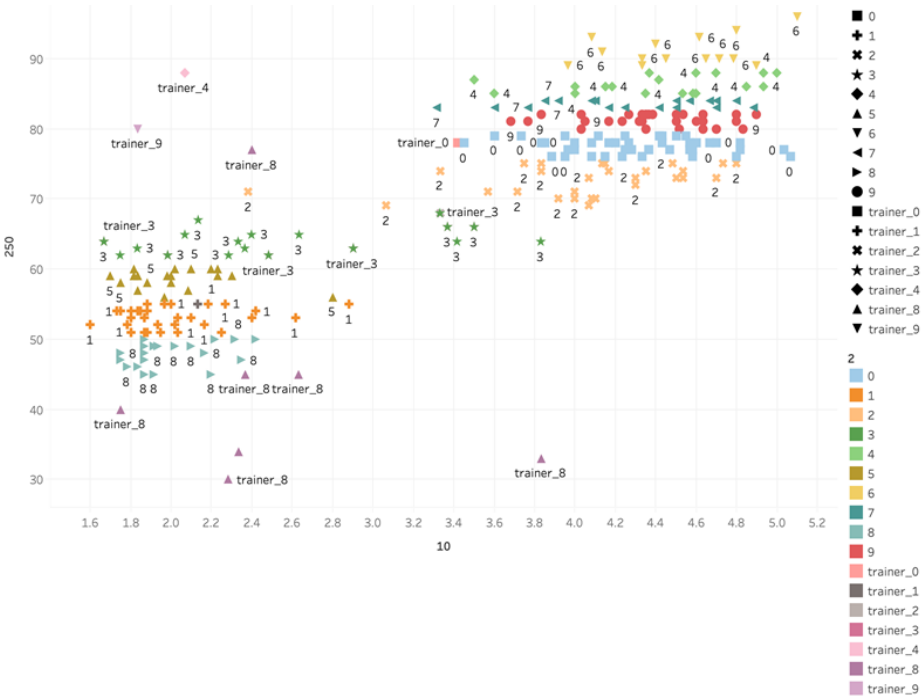
It turns out that with 2 trainer clusters, 22 unclassified data can be labeled and assigned to the suitable cluster. We also see results with different k:

k = 3:	k = 7:	k = 11:	k = 15:	k = 19:
2.200000,54.000000,1	2.200000,54.000000,1	2.200000,54.000000,1	2.200000,54.000000,1	2.200000,54.000000,1
4.450000,83.000000,0	4.450000,83.000000,0	4.450000,83.000000,0	4.450000,83.000000,0	4.450000,83.000000,0
3.567000,73.000000,0	3.567000,73.000000,0	3.567000,73.000000,0	3.567000,73.000000,0	3.567000,73.000000,0
4.500000,73.000000,0	4.500000,73.000000,0	4.500000,73.000000,0	4.500000,73.000000,0	4.500000,73.000000,0
4.150000,88.000000,0	4.150000,88.000000,0	4.150000,88.000000,0	4.150000,88.000000,0	4.150000,88.000000,0
3.817000,80.000000,0	3.817000,80.000000,0	3.817000,80.000000,0	3.817000,80.000000,0	3.817000,80.000000,0
3.917000,71.000000,0	3.917000,71.000000,0	3.917000,71.000000,0	3.917000,71.000000,0	3.917000,71.000000,0
4.450000,83.000000,0	4.450000,83.000000,0	4.450000,83.000000,0	4.450000,83.000000,0	4.450000,83.000000,0
2.000000,56.000000,1	2.000000,56.000000,1	2.000000,56.000000,1	2.000000,56.000000,1	2.000000,56.000000,1
4.283000,79.000000,0	4.283000,79.000000,0	4.283000,79.000000,0	4.283000,79.000000,0	4.283000,79.000000,0
4.767000,78.000000,0	4.767000,78.000000,0	4.767000,78.000000,0	4.767000,78.000000,0	4.767000,78.000000,0
4.533000,84.000000,0	4.533000,84.000000,0	4.533000,84.000000,0	4.533000,84.000000,0	4.533000,84.000000,0
1.850000,58.000000,1	1.850000,58.000000,1	1.850000,58.000000,1	1.850000,58.000000,1	1.850000,58.000000,1
4.250000,83.000000,0	4.250000,83.000000,0	4.250000,83.000000,0	4.250000,83.000000,0	4.250000,83.000000,0
1.983000,43.000000,1	1.983000,43.000000,1	1.983000,43.000000,1	1.983000,43.000000,1	1.983000,43.000000,1
2.250000,60.000000,1	2.250000,60.000000,1	2.250000,60.000000,1	2.250000,60.000000,1	2.250000,60.000000,1
4.750000,75.000000,0	4.750000,75.000000,0	4.750000,75.000000,0	4.750000,75.000000,0	4.750000,75.000000,0
4.117000,81.000000,0	4.117000,81.000000,0	4.117000,81.000000,0	4.117000,81.000000,0	4.117000,81.000000,0
2.150000,46.000000,1	2.150000,46.000000,1	2.150000,46.000000,1	2.150000,46.000000,1	2.150000,46.000000,1
4.417000,90.000000,0	4.417000,90.000000,0	4.417000,90.000000,0	4.417000,90.000000,0	4.417000,90.000000,0
1.817000,46.000000,1	1.817000,46.000000,1	1.817000,46.000000,1	1.817000,46.000000,1	1.817000,46.000000,1
4.467000,74.000000,0	4.467000,74.000000,0	4.467000,74.000000,0	4.467000,74.000000,0	4.467000,74.000000,0

Since the two clusters are pretty far from each other, when k<20, there won't be too much change in the results. However, when there are multiple clusters that are close to each other, the results will vary with k goes larger than 20, some far neighbors may be counted, when they become the majority, they may fool the KNN algorithm :

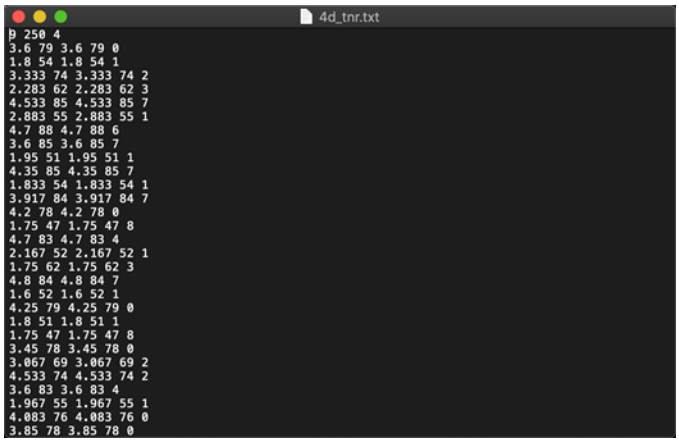
k = 11:	k = 31:	k = 101:
2.200000,54.000000,1	2.200000,54.000000,1	2.200000,54.000000,1
4.450000,83.000000,4	4.450000,83.000000,4	4.450000,83.000000,4
3.567000,73.000000,2	3.567000,73.000000,2	3.567000,73.000000,0
4.500000,73.000000,2	4.500000,73.000000,2	4.500000,73.000000,0
4.150000,88.000000,6	4.150000,88.000000,7	4.150000,88.000000,4
3.817000,80.000000,4	3.817000,80.000000,4	3.817000,80.000000,0
3.917000,71.000000,2	3.917000,71.000000,2	3.917000,71.000000,0
4.450000,83.000000,4	4.450000,83.000000,4	4.450000,83.000000,4
2.000000,56.000000,5	2.000000,56.000000,1	2.000000,56.000000,1
4.283000,79.000000,0	4.283000,79.000000,0	4.283000,79.000000,0
4.767000,78.000000,0	4.767000,78.000000,3	4.767000,78.000000,0
4.533000,84.000000,7	4.533000,84.000000,4	4.533000,84.000000,4
1.850000,58.000000,5	1.850000,58.000000,5	1.850000,58.000000,1
4.250000,83.000000,4	4.250000,83.000000,4	4.250000,83.000000,4
1.983000,43.000000,8	1.983000,43.000000,8	1.983000,43.000000,1
2.250000,60.000000,5	2.250000,60.000000,5	2.250000,60.000000,1
4.750000,75.000000,2	4.750000,75.000000,0	4.750000,75.000000,0
4.117000,81.000000,4	4.117000,81.000000,4	4.117000,81.000000,4
2.150000,46.000000,8	2.150000,46.000000,8	2.150000,46.000000,1
4.417000,90.000000,6	4.417000,90.000000,6	4.417000,90.000000,4
1.817000,46.000000,8	1.817000,46.000000,8	1.817000,46.000000,1
4.467000,74.000000,2	4.467000,74.000000,2	4.467000,74.000000,0

And this proves that with our own data, this program can still works as expected. We can also draw a graph to show the result of the classification, since some nodes are too far away in y-direction, this graph cannot reflect the actual distance between the nodes :

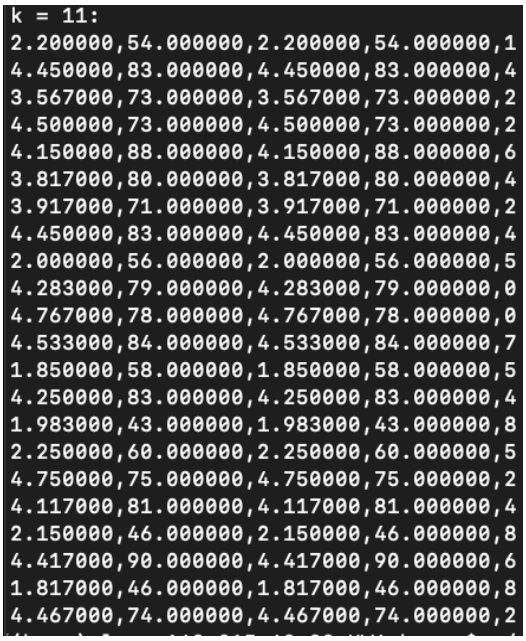


(4) Read multidimensional data: For multidimensional data, we can also read it and use it to calculate the distances, labeling all the unclassified data:

With command: . /main 4d_tnr.txt 4d_tre.txt R.txt 11, 4d_tnr.txt as follows:



We get the results:



It proves that we can handle multidimensional data.

Cooperation:

In this assignment, Junzhu Xiang is responsible for Kmeans part, and Haomin Lin is responsible for KNN part. During this cooperation, we discussed about how to pass data from Kmeans to KNN, how to choose cluster number, and how to solve difficulties we met, etc.