CSE 6010 - Computational Problem Solving Assignment 3 - K-Means Clustering

Ting Liao / GTID: 903278773

Slide 1

First, I structured my Cluster program by breaking down the whole executing program into small, simple function such as “get_cluster_centroid” and “assign_cluster”. In other words, I can focus on the current function without being interrupted by other functions.

For example, in the “get_cluster_centroid” function, I used a nest for-loop to help me get each of the value from the original and normalized data array, and put them into my “cluster_points” and “normalized_cluster_points” array. Here I simply dealt with this process and I wouldn’t be distracted by any other function.

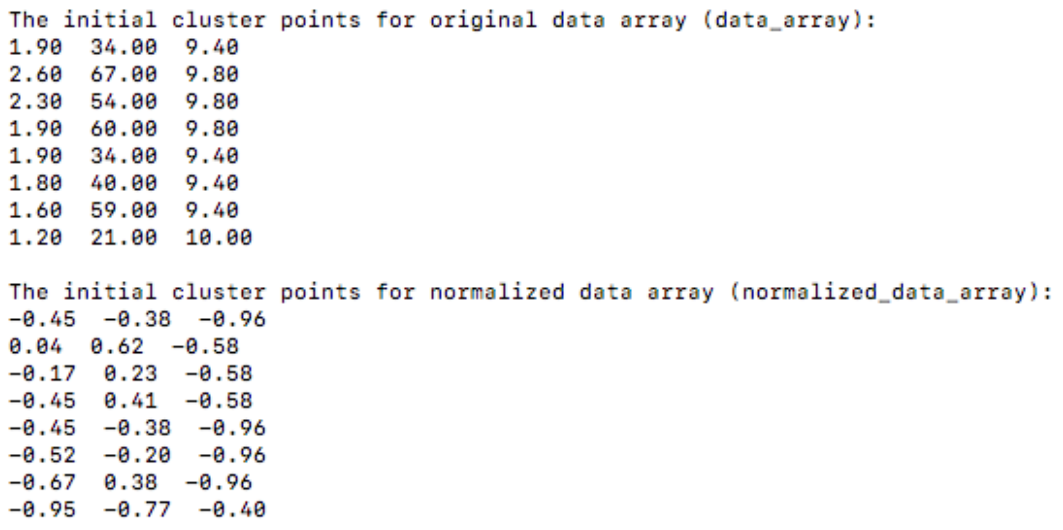
Second, I separate each item in the data_array (and normalized_data_array) to find out which cluster point has the minimum distance with it by using nested for-loop. To be more specific, The first loop would loop through every row, then the second loop would loop through every “K cluster” (K1, K2, …) to compare the distance and the last loop would calculate the square distance for each item in data array. After that, I assigned the index into the “cluster_group” array mentioning that which group this data item should be.

Third, now I could start to find the new cluster points. In the “get_new_cluster_points” function, I used nested for-loop to help me get the sum of the data item in each cluster_group and get their average value. Then, I assigned these average values into cluster_points array to replace the old one. However, I have already made an old_cluster_points array in order to enable me check whether I found the final cluster_points.

After that, I used the “final_cluster” function to help me integrate all functions together and do the iteration. I used an if-statement to help me check that if old_cluster_points is equal to cluster_points, break. That means I got the final cluster_points and no need for any iteration anymore.

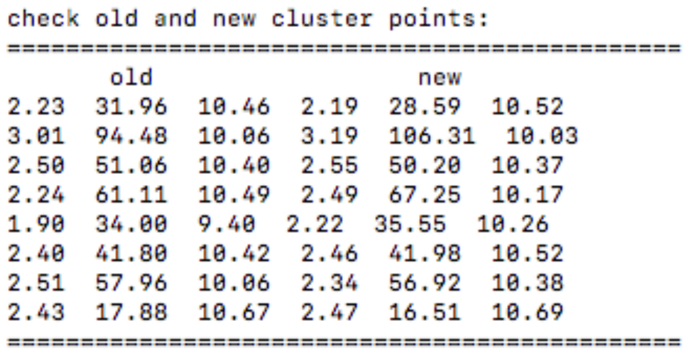
What’s more, the normalized part is only different from the original data in the initial data setting. Then, I created “create_normalized_data” function to complete those settings. After that, I just repeat the above process to get the normalized final cluster_points.

Slide 2



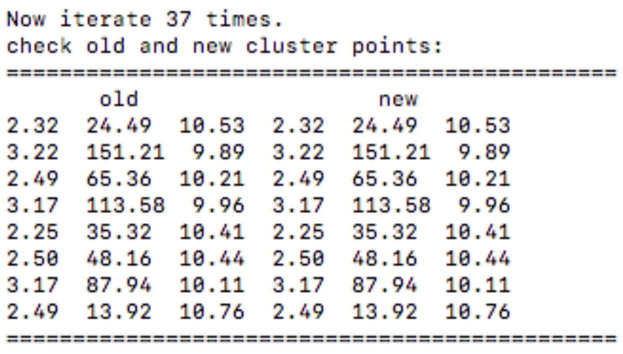
```
The initial cluster points for original data array (data_array):
1.90  34.00  9.40
2.60  67.00  9.80
2.30  54.00  9.80
1.90  60.00  9.80
1.90  34.00  9.40
1.80  40.00  9.40
1.60  59.00  9.40
1.20  21.00  10.00

The initial cluster points for normalized data array (normalized_data_array):
-0.45  -0.38  -0.96
0.04  0.62  -0.58
-0.17  0.23  -0.58
-0.45  0.41  -0.58
-0.45  -0.38  -0.96
-0.52  -0.20  -0.96
-0.67  0.38  -0.96
-0.95  -0.77  -0.40
```

(Fig 1.)



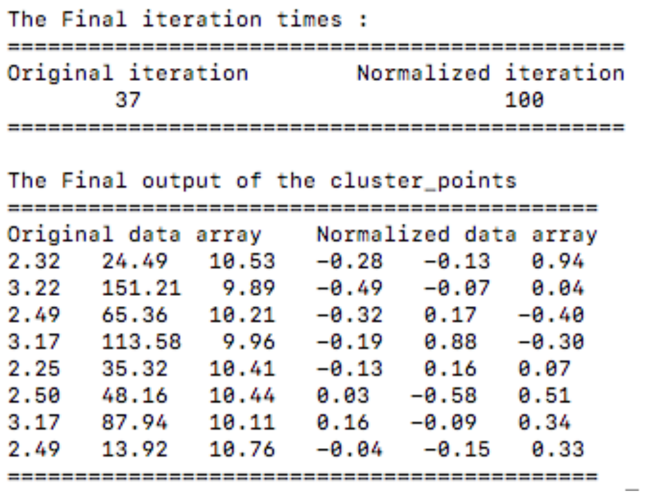
```
check old and new cluster points:
==================================================
       old                     new
2.23  31.96  10.46  2.19  28.59  10.52
3.01  94.48  10.06  3.19  106.31  10.03
2.50  51.06  10.40  2.55  50.20  10.37
2.24  61.11  10.49  2.49  67.25  10.17
1.90  34.00  9.40  2.22  35.55  10.26
2.40  41.80  10.42  2.46  41.98  10.52
2.51  57.96  10.06  2.34  56.92  10.38
2.43  17.88  10.67  2.47  16.51  10.69
==================================================
```

(Fig 2.)



```
Now iterate 37 times.
check old and new cluster points:
==============================================
      old                    new
2.32  24.49  10.53  2.32  24.49  10.53
3.22  151.21  9.89  3.22  151.21  9.89
2.49  65.36  10.21  2.49  65.36  10.21
3.17  113.58  9.96  3.17  113.58  9.96
2.25  35.32  10.41  2.25  35.32  10.41
2.50  48.16  10.44  2.50  48.16  10.44
3.17  87.94  10.11  3.17  87.94  10.11
2.49  13.92  10.76  2.49  13.92  10.76
==============================================
```

(Fig 3.)



```
The Final iteration times :
==============================================
Original iteration          Normalized iteration
       37                            100
==============================================

The Final output of the cluster_points
============================================
Original data array    Normalized data array
2.32   24.49   10.53   -0.28    -0.13   0.94
3.22   151.21   9.89   -0.49    -0.07   0.04
2.49   65.36   10.21   -0.32    0.17   -0.40
3.17   113.58   9.96   -0.19    0.88   -0.30
2.25   35.32   10.41   -0.13    0.16   0.07
2.50   48.16   10.44   0.03    -0.58   0.51
3.17   87.94   10.11   0.16    -0.09   0.34
2.49   13.92   10.76   -0.04    -0.15   0.33
============================================
```

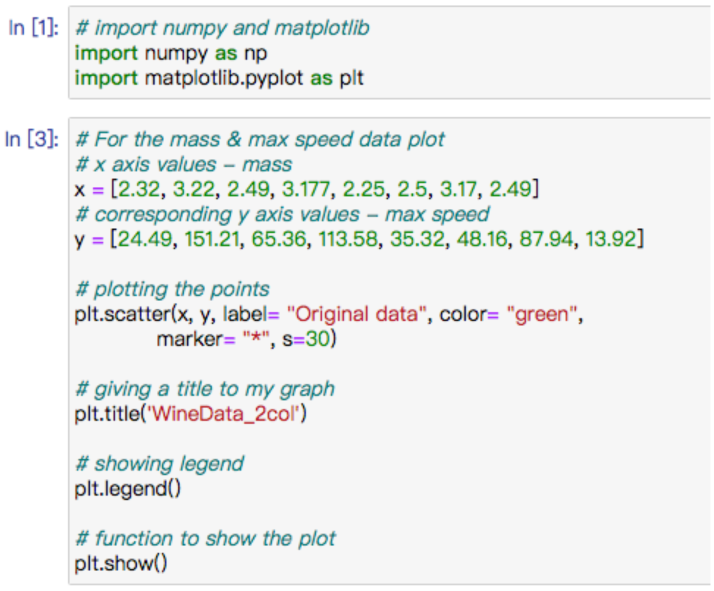
(Fig 4.)

I believe that my program works successfully and the result is correct because of the above screen shot. As you can see, first, I print out the initial cluster points (Fig 1.), which means it picks up the cluster points successfully. Then, I check whether the new cluster points is same as the old one (Fig 2. & Fig 3.). If no, then continue the iteration. However, if yes, then the iteration would shut down. Finally, the print out results show the detail info for my clustering program (Fig 4.).

Slide 3

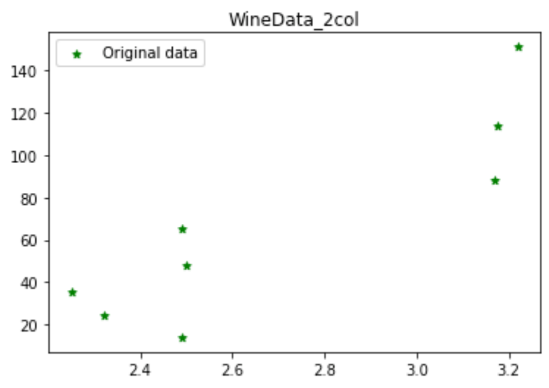
Implementing normalized data value may improve the results of K mean clustering but not always. In most cases, normalization can provide the same importance to all the variables in your input data sets.

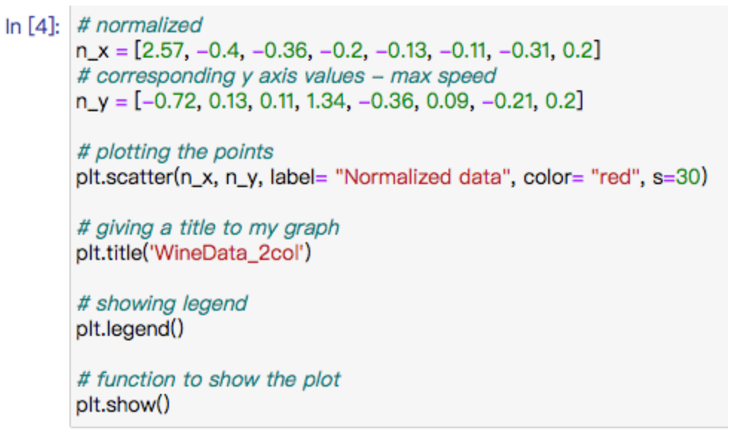
For example, in our WineData_2col.txt file, the value of the first column is way smaller than the second column (x1 = 1.9, y1 = 34), which means that the second column will definitely have more importance while doing the k-means clustering computation with respect to the first column. If it’s still not pretty clear, think about x = 1 vs. y = 1000,000.



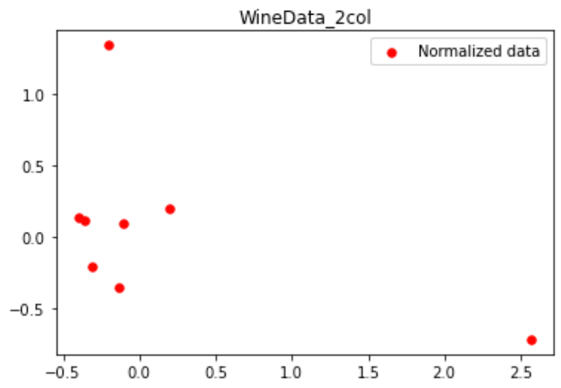
```
In [1]: # import numpy and matplotlib
        import numpy as np
        import matplotlib.pyplot as plt
```

```
In [3]: # For the mass & max speed data plot
        # x axis values - mass
        x = [2.32, 3.22, 2.49, 3.177, 2.25, 2.5, 3.17, 2.49]
        # corresponding y axis values - max speed
        y = [24.49, 151.21, 65.36, 113.58, 35.32, 48.16, 87.94, 13.92]

        # plotting the points
        plt.scatter(x, y, label= "Original data", color= "green",
                    marker= "*", s=30)

        # giving a title to my graph
        plt.title('WineData_2col')

        # showing legend
        plt.legend()

        # function to show the plot
        plt.show()
```





```
In [4]: # normalized
        n_x = [2.57, -0.4, -0.36, -0.2, -0.13, -0.11, -0.31, 0.2]
        # corresponding y axis values - max speed
        n_y = [-0.72, 0.13, 0.11, 1.34, -0.36, 0.09, -0.21, 0.2]

        # plotting the points
        plt.scatter(n_x, n_y, label= "Normalized data", color= "red", s=30)

        # giving a title to my graph
        plt.title('WineData_2col')

        # showing legend
        plt.legend()

        # function to show the plot
        plt.show()
```



From the figure, I know that after normalizing the data, the output cluster points will be much closer to the median 0. Also, there are still some points locate far away from others. I guess this is because if we first pick that point as one of our cluster point. It will permanently stand as one of the cluster points due to the distance with itself will be zero. Therefore, I guess the extra topic for picking random cluster points as the initial cluster points will improve this situation.

Slide 4

In this program, I didn’t have time to do the extra part. However, I can briefly guess what will picking the random points as the initial cluster points benefit our program. From the slide 3, we knew that if we pick a far away point as the initial cluster point, it will never change. Therefore, using the random points can definitely avoid this problem.

Slide 5.

For the original datasets, I found that using k = 7 is most useful. This is because, the iteration time was less than k = 6 and k = 8. With faster speed to get the final cluster points, I think it means that every point was put in the right cluster group sooner than the k value next to it (k = 6, k = 8).

However, for the normalized data sets, I couldn't truly derive the true final cluster points before the iteration limit was reached. I guess that the data after normalized, they are distributed more evenly, so any change made on the cluster points will significantly make every point be sent to a different cluster group. Due to that, every time we will find a new cluster point, but those points which originally belonged to it would join the group. Therefore, the iteration never stop until our initial iteration limit was reached.

I consider that the algorithm produced meaningful clusters for these two datasets because by repeatedly assigned points to the cluster group with the minimum distance will eventually let every point to find the right cluster point. However, the accuracy might have changed due to the setting of the k value and the method I did to prevent any duplicate cluster point or the situation I mentioned many times that the initial cluster points contain a far away single point. I believed that solve all these problems appropriately, so the results should be correct and thus the algorithm is meaningful.