Project4 Report

Xueying Sui 1001682442

Problem

We need to fulfill the K-means clustering algorithm. We have data sets, we need to cluster the data, that is, using unsupervised learning. For K-means clustering, the value of K is not fixed, you can assign a value to K. If you make K=3, then the K-means algorithm will eventually divide the data set into 3 categories. So the K-means algorithm needs to determine the value of K in advance.

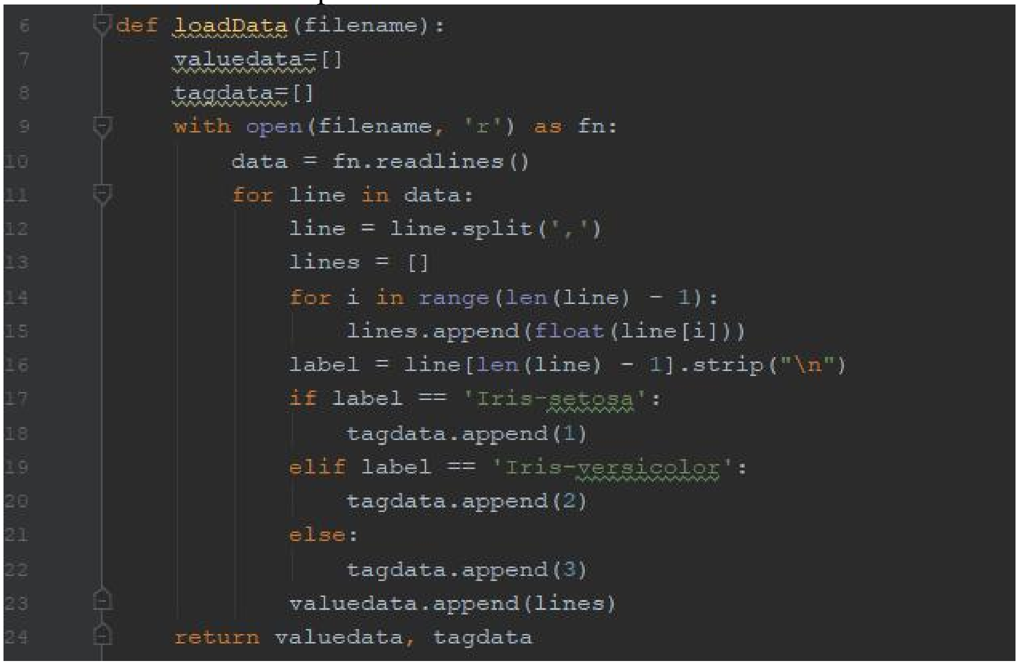
Data

We use the data set of iris (http://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data).
This data set include 150 data items. Actually, this data set contains 3 irises, these data items have their labels, but if we fulfill the K-means clustering algorithm, we don't need these labels. We could use these labels as an evaluation method. But the evaluation only applies to the case of K=3.

Method

Step1 Load data

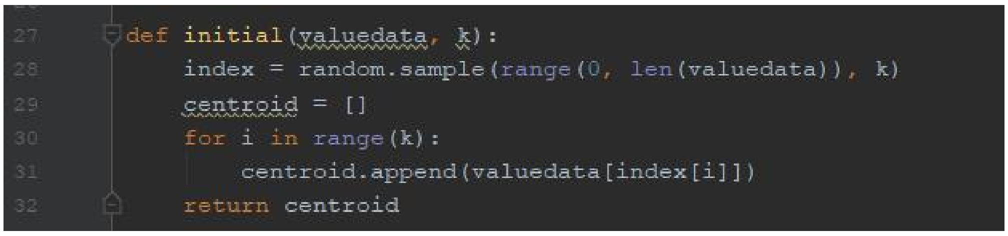
We have a data set that include 150 data items. We need to read there data items into a list. Because these data items have their labels, we use another list to save the labels. Although we don't need these labels when we do K-means cluster, we can use labels list to evaluate the algorithm if the K=3.
The core code of this step:



```
def loadData(filename):
    valuedata=[]
    tagdata=[]
    with open(filename, 'r') as fn:
        data = fn.readlines()
        for line in data:
            line = line.split(',')
            lines = []
            for i in range(len(line) - 1):
                lines.append(float(line[i]))
            label = line[len(line) - 1].strip("\n")
            if label == 'Iris-setosa':
                tagdata.append(1)
            elif label == 'Iris-versicolor':
                tagdata.append(2)
            else:
                tagdata.append(3)
            valuedata.append(lines)
    return valuedata, tagdata
```

Step2 Initialize the centroids

Initialize the first group of centroids, we used the random function to initialize. The number of centroids equals to the value of K. We randomly take K data points from the data set obtained in the previous step as initial centroids.
The core code of this step:

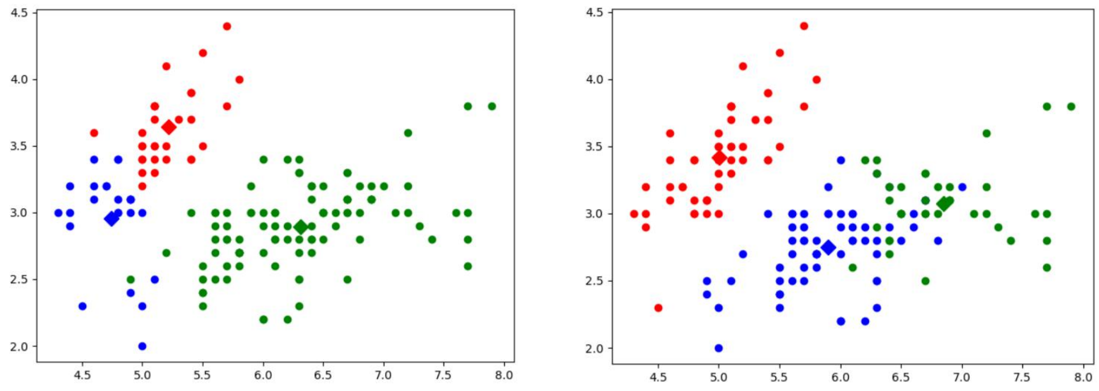


```python
def initial(valuedata, k):
    index = random.sample(range(0, len(valuedata)), k)
    centroid = []
    for i in range(k):
        centroid.append(valuedata[index[i]])
    return centroid
```

Step3 Iterate centroids until they no longer change or meet the convergence conditions

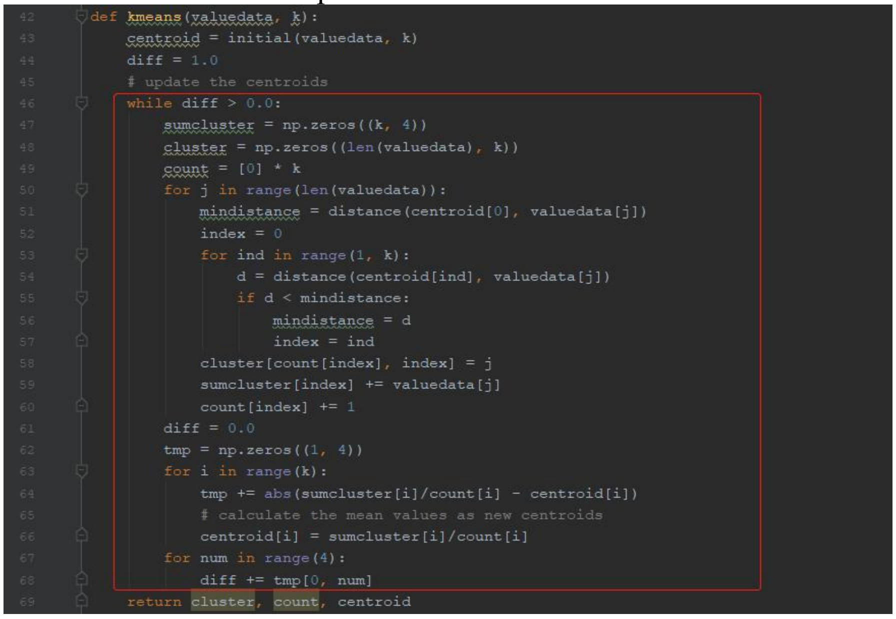
This step is the core step of K-means algorithm. Take K=3 as an example. First, use 3 centroids as the standard, for every other data items, calculate the distance between the data item and each centroid, the data item belongs to the class that it’s centroid has the nearest distance. Then we use the mean of the points that are currently in the same class as the new centroids. And we iterate centroids as this way until they no longer change or meet the convergence conditions.

But there is still a problem, K-means algorithm converges to local optimum, that means when we finish the iterate, what we get is not necessarily the global optimal result. For example, the following two figures are the results obtained when K=3(For the convenience of display, I **only painted two dimensions**):

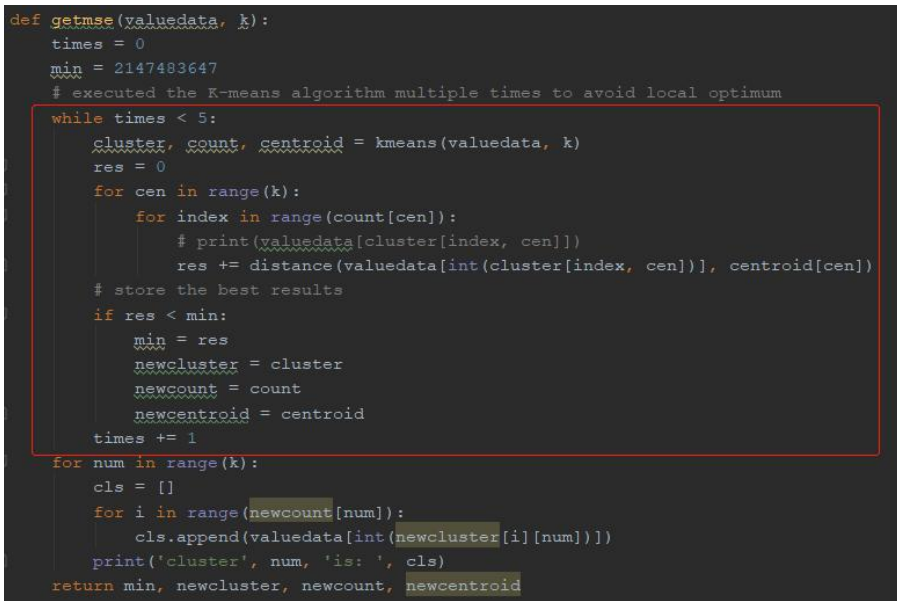


In order to solve this problem, I executed the K-means algorithm multiple times in the code, evaluated the result by calculating the distance value from each point to the centroid, and selected the result with the smallest sum of distances as the final result.This method has achieved good results.

The core code of this step:



```python
def kmeans(valuedata, k):
    centroid = initial(valuedata, k)
    diff = 1.0
    # update the centroids
    while diff > 0.0:
        sumcluster = np.zeros((k, 4))
        cluster = np.zeros((len(valuedata), k))
        count = [0] * k
        for j in range(len(valuedata)):
            mindistance = distance(centroid[0], valuedata[j])
            index = 0
            for ind in range(1, k):
                d = distance(centroid[ind], valuedata[j])
                if d < mindistance:
                    mindistance = d
                    index = ind
            cluster[count[index], index] = j
            sumcluster[index] += valuedata[j]
            count[index] += 1
        diff = 0.0
        tmp = np.zeros((1, 4))
        for i in range(k):
            tmp += abs(sumcluster[i]/count[i] - centroid[i])
            # calculate the mean values as new centroids
            centroid[i] = sumcluster[i]/count[i]
        for num in range(4):
            diff += tmp[0, num]
    return cluster, count, centroid
```



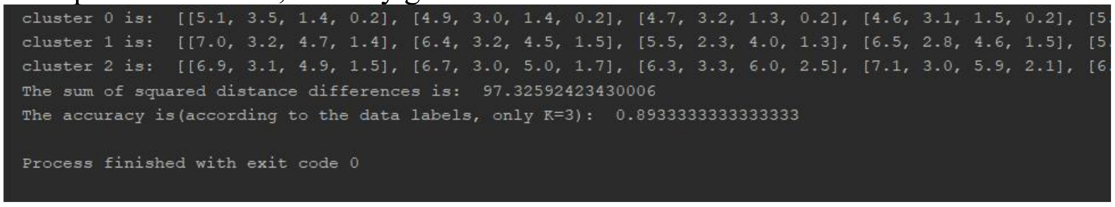
```
def getmse(valuedata, k):
    times = 0
    min = 2147483647
    # executed the K-means algorithm multiple times to avoid local optimum
    while times < 5:
        cluster, count, centroid = kmeans(valuedata, k)
        res = 0
        for cen in range(k):
            for index in range(count[cen]):
                # print(valuedata[cluster[index, cen]])
                res += distance(valuedata[int(cluster[index, cen])], centroid[cen])
        # store the best results
        if res < min:
            min = res
            newcluster = cluster
            newcount = count
            newcentroid = centroid
        times += 1
    for num in range(k):
        cls = []
        for i in range(newcount[num]):
            cls.append(valuedata[int(newcluster[i][num])])
        print('cluster', num, 'is: ', cls)
    return min, newcluster, newcount, newcentroid
```

Step4 Get the final clustering result

From previous step we can get the final clustering result, we can plot the scatter according to this result.

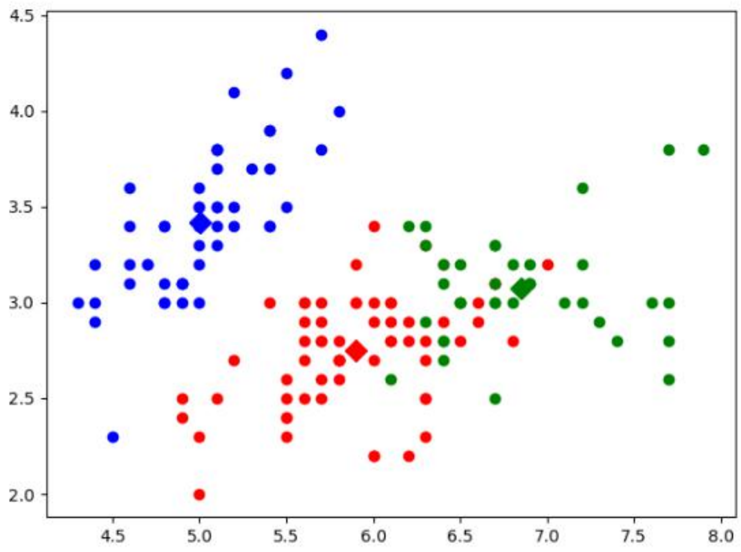
Results

If we input K=3, we can get the cluster result and the accuracy of our cluster result. If we input other values, we only get the cluster result.



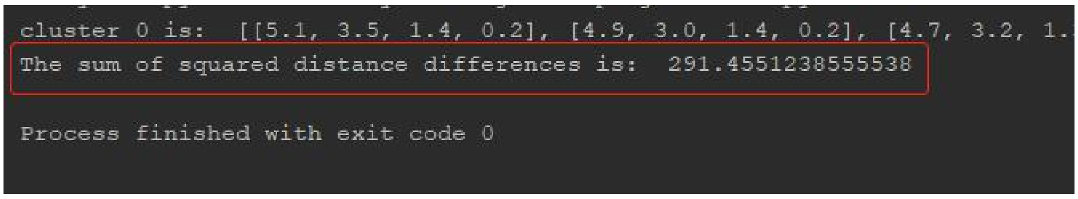
```
cluster 0 is:  [[5.1, 3.5, 1.4, 0.2], [4.9, 3.0, 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [4.6, 3.1, 1.5, 0.2], [5
cluster 1 is:  [[7.0, 3.2, 4.7, 1.4], [6.4, 3.2, 4.5, 1.5], [5.5, 2.3, 4.0, 1.3], [6.5, 2.8, 4.6, 1.5], [5
cluster 2 is:  [[6.9, 3.1, 4.9, 1.5], [6.7, 3.0, 5.0, 1.7], [6.3, 3.3, 6.0, 2.5], [7.1, 3.0, 5.9, 2.1], [6
The sum of squared distance differences is:  97.32592423430006
The accuracy is(according to the data labels, only K=3):  0.8933333333333333

Process finished with exit code 0
```

For the convenience of display, I **only painted two dimensions**:

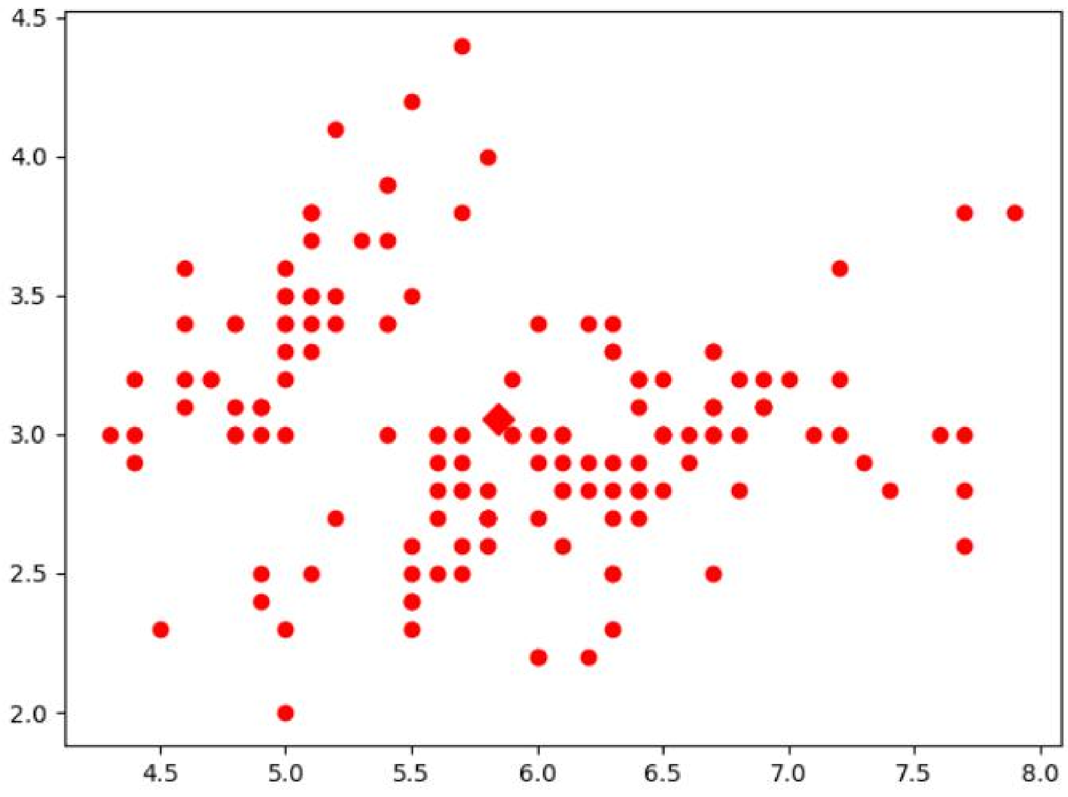


In this experiment, we picked K=3, not only because if K=3 we can evaluate the result, but also the sum of squared distance difference of K=3 lower than the sum of squared distance difference of K=1 and K=2, the result of K=1 and K=2 are as following:

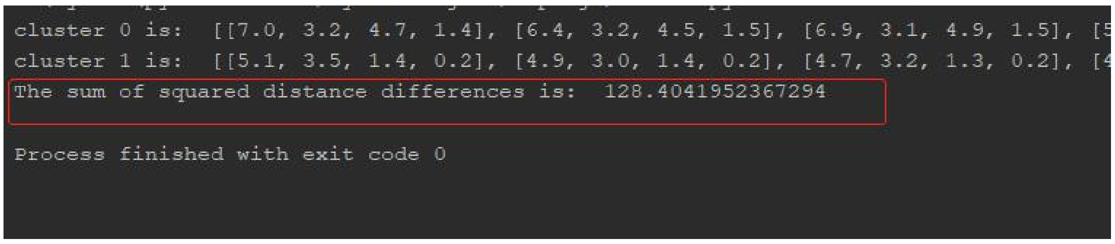
K=1



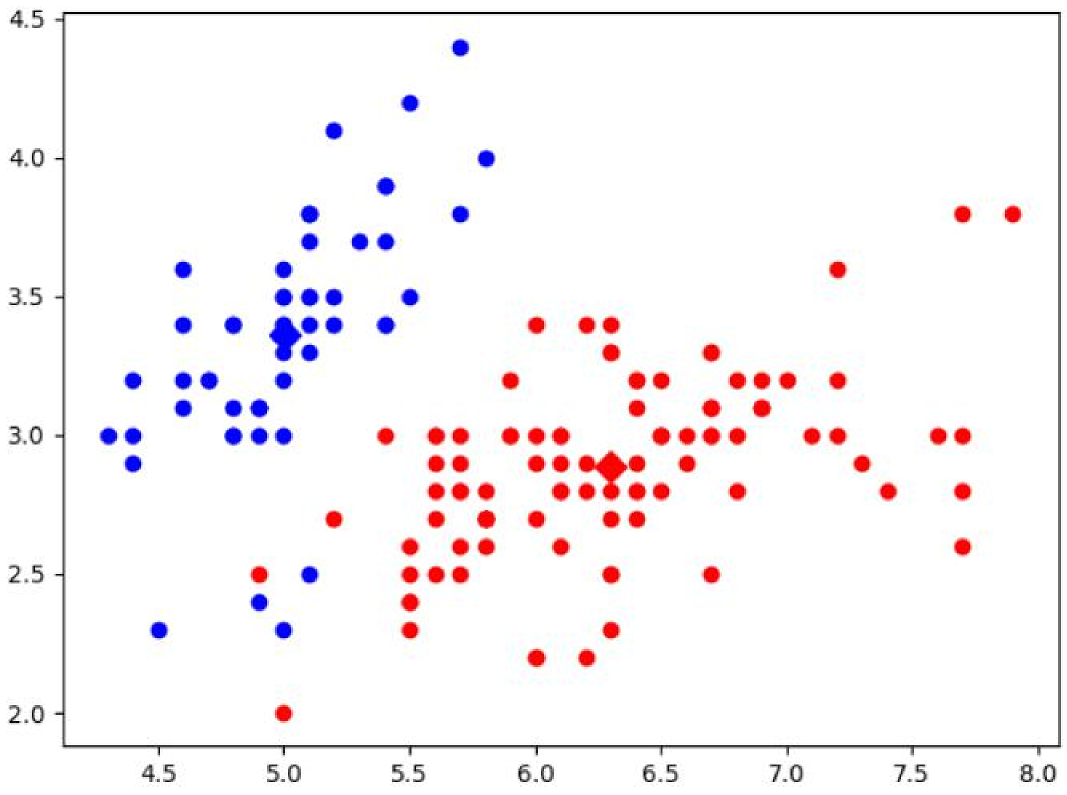
```
cluster 0 is:  [[5.1, 3.5, 1.4, 0.2], [4.9, 3.0, 1.4, 0.2], [4.7, 3.2, 1.
The sum of squared distance differences is:  291.4551238555538

Process finished with exit code 0
```



K=2



```
cluster 0 is:  [[7.0, 3.2, 4.7, 1.4], [6.4, 3.2, 4.5, 1.5], [6.9, 3.1, 4.9, 1.5], [5
cluster 1 is:  [[5.1, 3.5, 1.4, 0.2], [4.9, 3.0, 1.4, 0.2], [4.7, 3.2, 1.3, 0.2], [4
The sum of squared distance differences is:  128.4041952367294

Process finished with exit code 0
```



According to the above results, we picked K=3

Summary

Through this project, I understand the principle of the K-means algorithm. I have a better understanding of how to update the centroids. I think that implementing the K-means algorithm is far less simple than originally thought. In the process of implementation, I have encountered problems such as local optimization and solved it. I also realized the advantages and disadvantages of the K-means algorithm. I think this is a rare experience.