CS470 Homework 3

Jaekyum Kim

2020-02-27

Collaboration Statement THIS CODE IS MY OWN WORK, IT WAS WRITTEN WITHOUT CONSULTING A TUTOR OR CODE WRITTEN BY OTHER STUDENTS - Jaekyum Kim

1 Introduction

The program is executable with 3 parameters: the name of the input dataset file, the value of K, and the name of the output file. The two kinds of dataset I used in this project was "iris.data" and "wine.data". The output contains numerical class labels (formatted as one number per row) for all the records in the test dataset and reports the sum squared error (SSE) and silhouette coefficient in the last row.

2 Cluster Initialization

A conventional K-means algorithm first arbitrarily chooses whatever a user input's K is and randomly chooses K-number of clusters among the dataset. The code to choose K-number of datapoint is following (this function was commented out in the code):

```
for i in range(0,k):
    index = random.randrange(0, len(dataset), 3)
    initialization.append(dataset[index])
```

However randomly choosing initialization points for clusters can possibly cause a problem later when the algorithm finishes. While experimenting this issue, I have noticed that this naive approach can produce a negative Silhouette coefficient. According to the textbook on the section of coefficient points, when the silhouette coefficient value is negative (i.e., b(o) ¡ a(o)), this means that, in expectation, o is closer to the objects in another cluster than to the objects in the same cluster as o. In many cases, this is a bad situation and should be avoided. Therefore to fix the problem, I have implemented the algorithm to pick K-number of farthest point within each other, so the K-means algorithm is guaranteed to find the most optimal cluster points before running the iteration. Because the algorithm chooses fixed K-number of points, the silhouette point will never hit the negative number. The code for farthest point initialization is the following:

```
def spitDist(firstcentroid, my_list):
    distDict.clear()

    for eachline in dataset:
        difference = map(operator.sub, firstcentroid, eachline)
        squared = [i ** 2 for i in difference]
        listsummed = sum(squared)
        euclidean = math.sqrt(listsummed)

        #print("calculating distance between: ",eachline,"and",
                                            firstcentroid)
```

```
        #print("calculated euclidean for this edge: ", euclidean)

        eachline = tuple(eachline)
        if eachline not in initialization:
            distDict[tuple(eachline)] = euclidean #node that is
                                                  farthest out
            maxlength = euclidean

    return distDict

if k >= 2:
    for eachcluster in initialization:
        tmp=spitDist(eachcluster, my_list)
        a = sorted(tmp.items(), key=lambda x: x[1],reverse=True)

        candidateslist = []

        for eachvector in a:
            candidateslist.append(eachvector[0])#make list of
                                                  candidates in sorted
                                                  order

        if len(initialization) != 1:
            ans = []
            for eachItemI in candidateslist:
                for eachItemJ in result:
                    if eachItemI == eachItemJ:
                        ans.append(eachItemI)
            result = list(ans)

            initialization.append(ans[0])
        elif len(initialization) == 1:
            initialization.append(candidateslist[0])
            result = list(candidateslist)

        if(len(initialization) == k):
            break
```

To briefly explain the algorithm for farthest point cluster selection, The algorithm first choose a random first point to begin with because you need a point to start measuring points that are farthest away from each other. This routine would have been easy if there was a built in collection for sets that keep order, then first i could just find a set of distance from a chosen cluster point to every other points and get the point that are farthest away. Then, I intersect those two sets of sorted distance to find the common longest distance. If I keep running the routine, then I would be able to find K-number of cluster points that are farthest away from each other.

3 Elbow method

k	SSE	Silhouette
2	152.3687065	-0.29707146
3	78.94506583	0.091685706
4	57.34540932	0.350636952
5	49.74079031	0.315489025
6	38.93096305	0.253268829
7	37.38677029	0.284576335
8	35.66954806	0.586158855
9	35.5466871	0.177780621
10	35.5466871	0.183995076

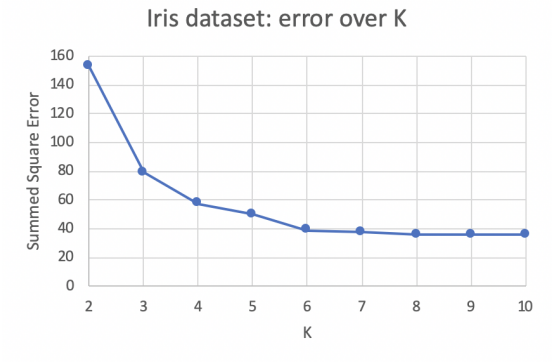

Figure 1: SSE vs K for Iris dataset

Figure 1 As shown in the dataset, only three types of Iris species occur. The relationship graphed displays that the desirable K is in between 2 and 4. The bend in the trendline is at k=3.

k	SSE	Silhouette
2	4545800.9	-0.144968347
3	2633614.5	0.138533573
4	1333194.1	0.417803787
5	1341434.2	0.372554368
6	1012188.3	0.411805641
7	708491.82	0.484160727
8	641783.09	0.500179189
9	641783.09	0.50761599
10	628232.76	0.492309576

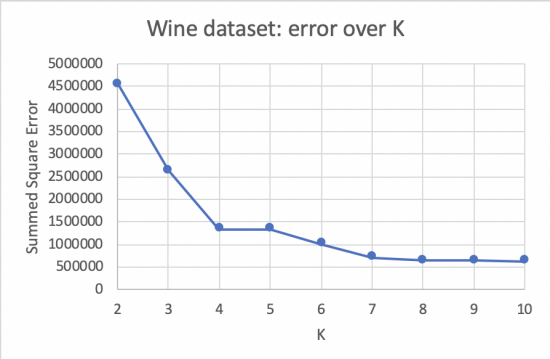

Figure 2: SSE vs K for Wine dataset

Figure 2 As shown in the dataset, only three types of wine variations can occur. The relationship graphed displays that the desirable K is in between 2 and 4. The bend in the trendline is at k=3.