28 Nov 2018

# CS550 Machine Learning - HW2 Report

H. Alperen Cetin

## 1_____

**Implementation**

Here are the steps of the k-means algorithm that I implemented:

1- Find the mean and the standard deviations (std) of the given data (im2 in our case).
2- Generate K random points. For each random point r, choose a random initial centroid as: "r * std + mean"
3- Find all distances between data points and centroids and label all points with the closest centroid.
4- With labeled points, find mean values for each label (cluster). Keep old centroids in a temp variable and assign these new mean values to real centroid list.
5- Calculate **error** as the norm of difference vector between new and old centroids. If error is zero, end the process, and return label array and centroid array. Else, go to 3rd step.

| | K=2 | | | K=4 | | | K=8 | | | K=16 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **R** | **G** | **B** | **R** | **G** | **B** | **R** | **G** | **B** | **R** | **G** | **B** |
| **1** | 214.29 | 166.81 | 196.05 | 212.96 | 38.36 | 123.15 | 180.23 | 185.67 | 180.65 | 189.63 | 15.70 | 99.75 |
| **2** | 186.00 | 82.29 | 94.97 | 159.79 | 119.92 | 47.42 | 175.08 | 121.78 | 150.80 | 135.34 | 81.38 | 15.23 |
| **3** | | | | 202.07 | 133.51 | 172.71 | 235.49 | 140.04 | 202.16 | 229.60 | 103.12 | 18.01 |
| **4** | | | | 224.15 | 199.49 | 217.98 | 234.53 | 214.72 | 230.72 | 172.40 | 118.06 | 152.46 |
| **5** | | | | | | | 201.35 | 22.60 | 101.99 | 150.42 | 171.17 | 165.06 |
| **6** | | | | | | | 214.96 | 139.40 | 32.83 | 190.01 | 203.05 | 206.04 |
| **7** | | | | | | | 23.10 | 84.16 | 80.01 | 242.78 | 160.72 | 219.70 |
| **8** | | | | | | | 234.14 | 74.16 | 157.43 | 240.29 | 226.22 | 236.23 |
| **9** | | | | | | | | | | 14.43 | 118.72 | 116.17 |
| **10** | | | | | | | | | | 216.50 | 164.28 | 36.20 |
| **11** | | | | | | | | | | 12.60 | 59.06 | 65.42 |
| **12** | | | | | | | | | | 240.01 | 106.18 | 179.70 |
| **13** | | | | | | | | | | 197.90 | 146.50 | 186.54 |
| **14** | | | | | | | | | | 194.36 | 81.72 | 126.76 |
| **15** | | | | | | | | | | 240.01 | 32.24 | 136.26 |
| **16** | | | | | | | | | | 220.76 | 199.85 | 115.18 |

**Table 1: Clustering vector (centroid) values for K=2, 4, 8 and 16**

| K | Distortion | K | Distortion | K | Distortion | K | Distortion | K | Distortion |
|---|---|---|---|---|---|---|---|---|---|
| **2** | 101169 | **6** | 61520 | **10** | 50207 | **14** | 44334 | **18** | 40157 |
| **3** | 89876 | **7** | 58170 | **11** | 48405 | **15** | 42685 | **19** | 39585 |
| **4** | 81545 | **8** | 55424 | **12** | 47506 | **16** | 41977 | **20** | 38946 |
| **5** | 67791 | **9** | 52462 | **13** | 45389 | **17** | 41121 | | |

**Table 2: Distortion (clustering error) values for K=2, 3, ..., 20**

K=2

K=4

K=8

K=16

**Figure 1: K-means results for K=2, 4, 8 and 16**

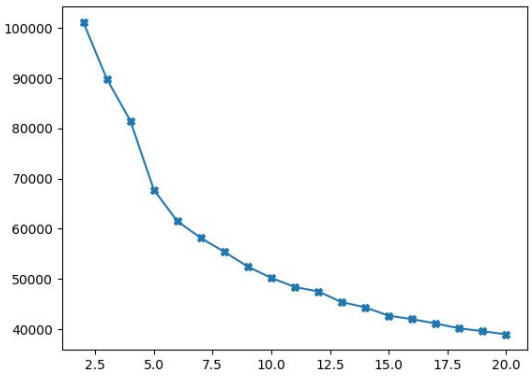


**Figure 2: Distortion graphic for K=2, 3, ... 20**

**Figure 3: K-means result for K=6**

The implementation given above is used to cluster "sample.jpg" with the K values 2, 4, 8 and 16. Table 1 shows the clustering vectors (centroids). Sub images in Figure 1 shows clustered images respectively 2, 4, 8 and 16.

I defined the **distortion** (clustering error) as the summation of distances between points and their own cluster centroids, and I ran the algorithm for K = 2,3, ..., 20 to find optimum K value for the same image. Table 2 shows the distortion values. Figure 2 shows a graphic of the distortions according to K values. I chose **6** for the **optimum K value**, because there are relatively small decreases after K equals 6 (Elbow Method). Clustering error(distortion) value for K=6 is in Table 2.

## 2

### Implementation

I used centroid-linked agglomerative clustering. I just kept the centroid and the item count of each cluster. Here are the steps of the agglomerative clustering algorithm that I implemented:

1- Define each data point as a different cluster, and the point itself as the centroid of the cluster. Also, assign 1 to item count, and initialize a label array showing centroid indexes in the centroid array.
2- Find the closest two centroids (minimum distance, maximum similarity).
3- Add the cluster with high index to the cluster with the low index combining centroids with average mean and summing item counts.
4- Replace all labels of the cluster with the higher index with labels of the cluster with the low index in the labels array.
5- If the length of the centroids array is equal to K, end the process, and return label array and centroid array. Else, go to 2nd step.

#### Speeding Up the Algorithm

To speed up the algorithm, I didn't give the whole data points at the beginning. I gave the points to the agglomerative clustering algorithm as the total number of clusters be 100 initially at every iteration. 100 is a number that I found experimentally. If I use 1000, it is too slow for agglomerative clustering algorithm, or if I use 30, it increases the iteration count. Here are the steps for speeding up part:

1- Run the agglomerative clustering algorithm with 100 data points initially. It will produce K clusters.
2- Keeping the cluster information from previous step, run the agglomerative clustering algorithm with new (100-K) data points.
3- If data points array comes to the end, finish the process. Else, go to 2nd step.

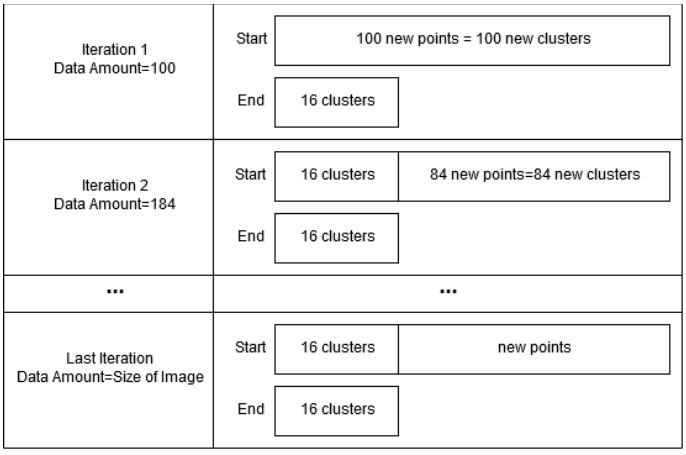


**Figure 4: The logic behind speed-up the algorithm for K = 16**

For a better understanding, Figure 4 shows iterations for K=16. In this situation, the speeding up algorithm sends the data to agglomerative clustering algorithm 84 by 84.

I need to say that my approach is order dependent, and a good clustering algorithm should be order independent. The data order can be randomized. Because the result images are good enough, I didn't randomize the image data while calling Agglomerative Algorithm. It would increase the total runtime. The algorithm finds 16 clusters in approximately two and a half hours. It is already much. Also, additional arrangements are required for the code. So, I didn't randomize data order, because of good enough clustered images, time issues and code simplicity.

This implementation is used to cluster “sample.jpg” with the K values 2, 4, 8 and 16. At the beginning, I clustered data for K=16. Keeping the data of K=16 clustering, I changed the K value to 8, and ran the agglomerative clustering algorithm with an empty data array. Then, the algorithm produces clusters for 8. I did the same thing for 4 and 2 respectively. So, running the algorithm for K=16 took most of the time, after that, for the K values smaller than 16, it is easy to calculate clustering centroids and labels.

Figure 5 shows clustered images respectively 2, 4, 8 and 16. Figure 6 shows distortions of agglomerative clustering. It is not like the distortion graphic in part 1. It is not telling us anything meaningful about optimal K value.

I can’t share clustering vectors and exact distortion values, because report is already 4-page. Distortion values can be seen in Figure 6 approximately.

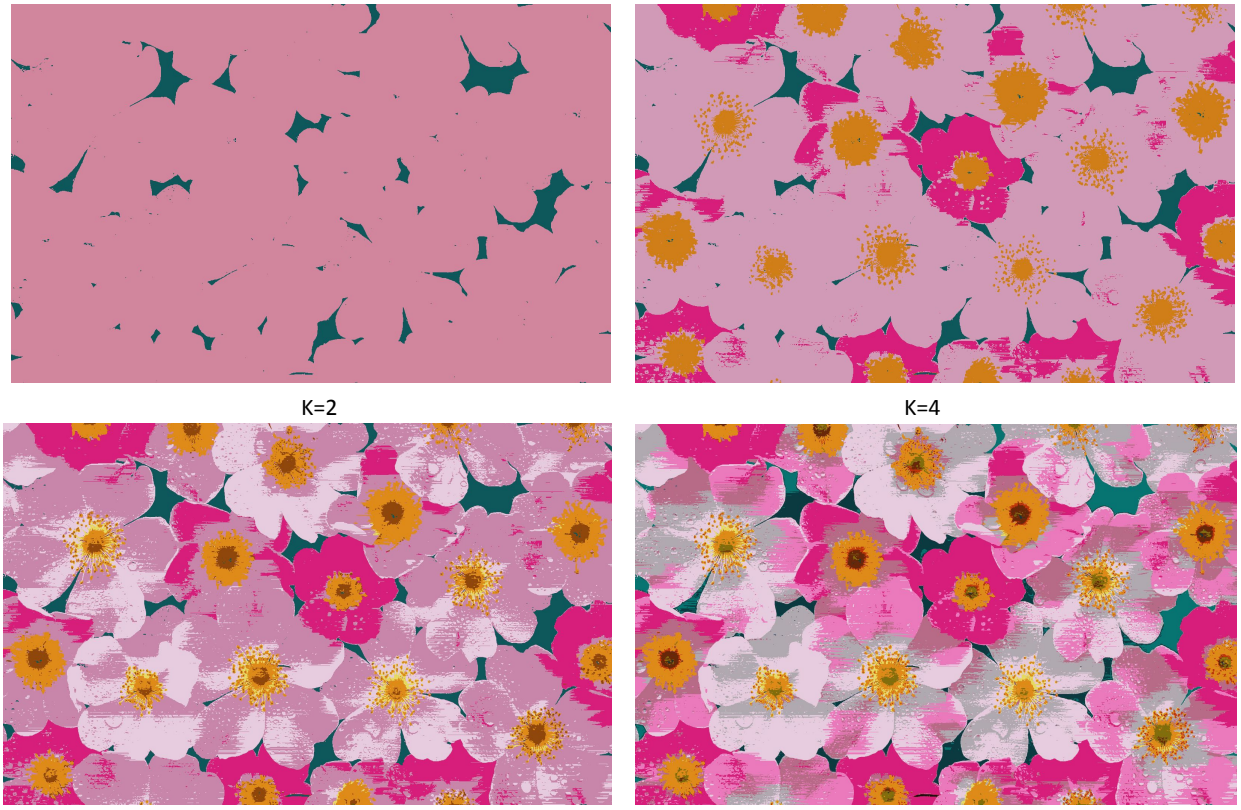


**Figure 5: Centroid linked agglomerative clustering results for K=2, 4, 8 and 16**

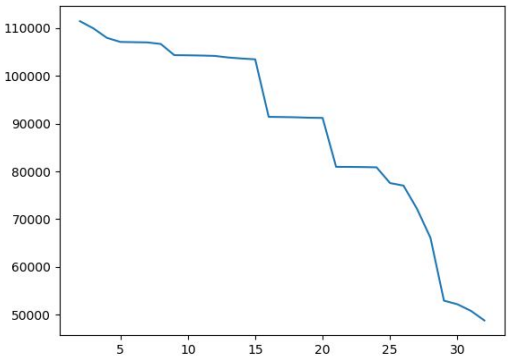


**Figure 6: Distortion graphic for K=2, 3, … 32**

**<u>Comparison of Part 1 and Part 2</u>**

K-means is faster than agglomerative clustering for small number of K values. But it needs to be fitted again and again for different K values. Agglomerative clustering fits once and can be used for each K value. Also, K-means implementation is easier and memory need is less than agglomerative. Because of the size of similarity matrix, agglomerative clustering needs more memory. When we look at the clustered image results, K-means has better results for the eye.