# CSCI 720 - Big Data Analytics

**Nihal Surendra Parchand**

**np9603@rit.edu**

**Homework 6 K-Means**

### For cluster k=1

The initial centroid selected for k=1 would be a random data point out of the 1304 data points given in the original data. Based on that centroid, the center of mass would be calculated and the new centroids will be calculated till the centroids in two consecutive iterations are not equal.

```
The number of data points for centroid  [-0.3  9.8  1.5] = 1304
The center of mass = ['[-0.3  9.8  1.5]']
The best sse for 1 clusters is: 128158.58
Total time to complete 500 iterations for 1 clusters = 143.25 seconds
```

### For cluster k=2

```
The number of data points for centroid  [ 6.8 11.7  1.2] = 622
The number of data points for centroid  [-6.7  8.1  1.8] = 682
The center of mass = ['[ 6.8 11.7  1.2]', '[-6.7  8.1  1.8]']
The best sse for 2 clusters is: 64663.11
Total time to complete 500 iterations for 2 clusters = 430.37 seconds
```

### For cluster k=3

```
The number of data points for centroid  [ 6.  11.1  1. ] = 723
The number of data points for centroid  [-10.4   7.2  -2.7] = 313
The number of data points for centroid  [-5.5  9.5  7.7] = 268
The center of mass = ['[ 6.  11.1  1. ]', '[-10.4   7.2  -2.7]', '[-5.5  9.5  7.7]']
The best sse for 3 clusters is: 40588.45
Total time to complete 500 iterations for 3 clusters = 464.96 seconds
```

### For cluster k=4

```
The number of data points for centroid  [13.9 10.4  0.2] = 230
The number of data points for centroid  [-1.8 11.9  5.5] = 497
The number of data points for centroid  [1.9 8.5 0. ] = 274
The number of data points for centroid  [-10.5   7.2  -2.7] = 303
The center of mass = ['[13.9 10.4  0.2]', '[-1.8 11.9  5.5]', '[1.9 8.5 0. ]', '[-10.5   7.2  -2.7]']
The best sse for 4 clusters is: 28868.03
Total time to complete 500 iterations for 4 clusters = 613.53 seconds
```

## For cluster k=5

```
The number of data points for centroid  [14.2 10.3  0.1] = 224
The number of data points for centroid  [ 2.2 14.3  2.8] = 253
The number of data points for centroid  [2.  8.5 0. ] = 268
The number of data points for centroid  [-5.6  9.6  7.9] = 255
The number of data points for centroid  [-10.6   7.2  -2.7] = 304
The center of mass = ['[14.2 10.3  0.1]', '[ 2.2 14.3  2.8]', '[2.  8.5 0. ]', '[-5.6  9.6  7.9]', '[-10.6   7.2  -2.7]']
The best sse for 5 clusters is: 13730.36
Total time to complete 500 iterations for 5 clusters = 831.27 seconds
```

## For cluster k=6

Since in some cases the consecutive cluster centers were not getting the same values, I decided to put another stopping criteria where it would come out of the loop if the number of iterations for the inner loop goes beyond 100.

```
The number of data points for centroid  [14.4 10.3  0.1] = 223
The number of data points for centroid  [ 2.6 14.4  2.8] = 246
The number of data points for centroid  [2.  8.5 0. ] = 268
The number of data points for centroid  [-5.3  9.5  8. ] = 245
The number of data points for centroid  [-14.3  10.8   4.7] = 25
The number of data points for centroid  [-10.4   7.1  -2.8] = 297
The center of mass = ['[14.4 10.3  0.1]', '[ 2.6 14.4  2.8]', '[2.  8.5 0. ]', '[-5.3  9.5  8. ]', '[-14.3  10.8   4.7]', '[-10.4   7.1  -2.8]']
The best sse for 6 clusters is: 9682.53
Total time to complete 500 iterations for 6 clusters = 921.96 seconds
```

## For cluster k=7

```
The number of data points for centroid  [14.5 10.3 -0. ] = 220
The number of data points for centroid  [ 2.6 14.4  2.8] = 245
The number of data points for centroid  [2.  8.5 0. ] = 267
The number of data points for centroid  [-5.6  9.4  7.9] = 238
The number of data points for centroid  [-14.3  10.8   4.7] = 25
The number of data points for centroid  [-10.4   7.1  -2.8] = 297
The number of data points for centroid  [ 4.9 13.   9.5] = 12
The center of mass = ['[14.5 10.3 -0. ]', '[ 2.6 14.4  2.8]', '[2.  8.5 0. ]', '[-5.6  9.4  7.9]', '[-14.3  10.8   4.7]', '[-10.4   7.1  -2.8]', '[ 4.9 13.   9.5]']
The best sse for 7 clusters is: 9053.76
Total time to complete 500 iterations for 7 clusters = 1179.93 seconds
```

## For cluster k=8

```
The number of data points for centroid  [14.5 10.3  0. ] = 219
The number of data points for centroid  [ 2.6 14.4  2.8] = 244
The number of data points for centroid  [ 2.1  8.7 -0.1] = 260
The number of data points for centroid  [-5.7 10.   7.9] = 152
The number of data points for centroid  [-5.   8.   7.4] = 97
The number of data points for centroid  [-10.4   7.1  -2.8] = 298
The number of data points for centroid  [-15.   10.9   4.6] = 22
The number of data points for centroid  [ 4.9 13.   9.5] = 12
The center of mass = ['[14.5 10.3  0. ]', '[ 2.6 14.4  2.8]', '[ 2.1  8.7 -0.1]', '[-5.7 10.   7.9]', '[-5.   8.   7.4]', '[-10.4   7.1  -2.8]', '[-15.   10.9   4.6]', '[ 4.9 13.   9.5]']
The best sse for 8 clusters is: 8886.38
Total time to complete 500 iterations for 8 clusters = 1543.12 seconds
```

## For cluster k=9

```
The number of data points for centroid  [14.4 10.3  0.1] = 223
The number of data points for centroid  [ 2.6 14.4  2.8] = 245
The number of data points for centroid  [ 2.2  8.7 -0.1] = 258
The number of data points for centroid  [-5.7 10.   7.8] = 150
The number of data points for centroid  [-5.   8.   7.4] = 97
The number of data points for centroid  [-10.3   6.3  -2.2] = 121
The number of data points for centroid  [-10.5   7.7  -3.2] = 178
The number of data points for centroid  [-15.   10.9   4.6] = 22
The number of data points for centroid  [ 0.9 13.1 11.2] = 10
The center of mass = ['[14.4 10.3  0.1]', '[ 2.6 14.4  2.8]', '[ 2.2  8.7 -0.1]', '[-5.7 10.   7.8]', '[-5.   8.   7.4]', '[-10.3   6.3  -2.2]', '[-10.5   7.7  -3.2]', '[-15.   10.9   4.6]', '[ 0.9 13.
The best sse for 9 clusters is: 8608.06
Total time to complete 500 iterations for 9 clusters = 1732.64 seconds
```

**For cluster k=10**

```
The number of data points for centroid  [14.3 11.  -0.5] = 111
The number of data points for centroid  [14.4  9.6  0.6] = 115
The number of data points for centroid  [ 2.9 13.6  2.9] = 144
The number of data points for centroid  [ 2.1 15.4  2.6] = 102
The number of data points for centroid  [ 2.1  8.7 -0.1] = 255
The number of data points for centroid  [-5.7 10.   7.8] = 150
The number of data points for centroid  [-5.   8.   7.4] = 97
The number of data points for centroid  [-10.4   7.1  -2.8] = 298
The number of data points for centroid  [-15.   10.9   4.6] = 22
The number of data points for centroid  [ 0.9 13.1 11.2] = 10
The center of mass = ['[14.3 11.  -0.5]', '[14.4  9.6  0.6]', '[ 2.9 13.6  2.9]', '[ 2.1 15.4  2.6]', '[ 2.1  8.7 -0.1]', '[-5.7 10.   7.8]', '[-5.   8.   7.4]', '[-10.4   7.1  -2.8]', '[-15.   10.9
The best sse for 10 clusters is: 8375.96
Total time to complete 500 iterations for 10 clusters = 2093.25 seconds
```

**For cluster k=11**

Since it was taking more than half an hour for 10 clusters, I decided to update the loop iteration limit to 50.

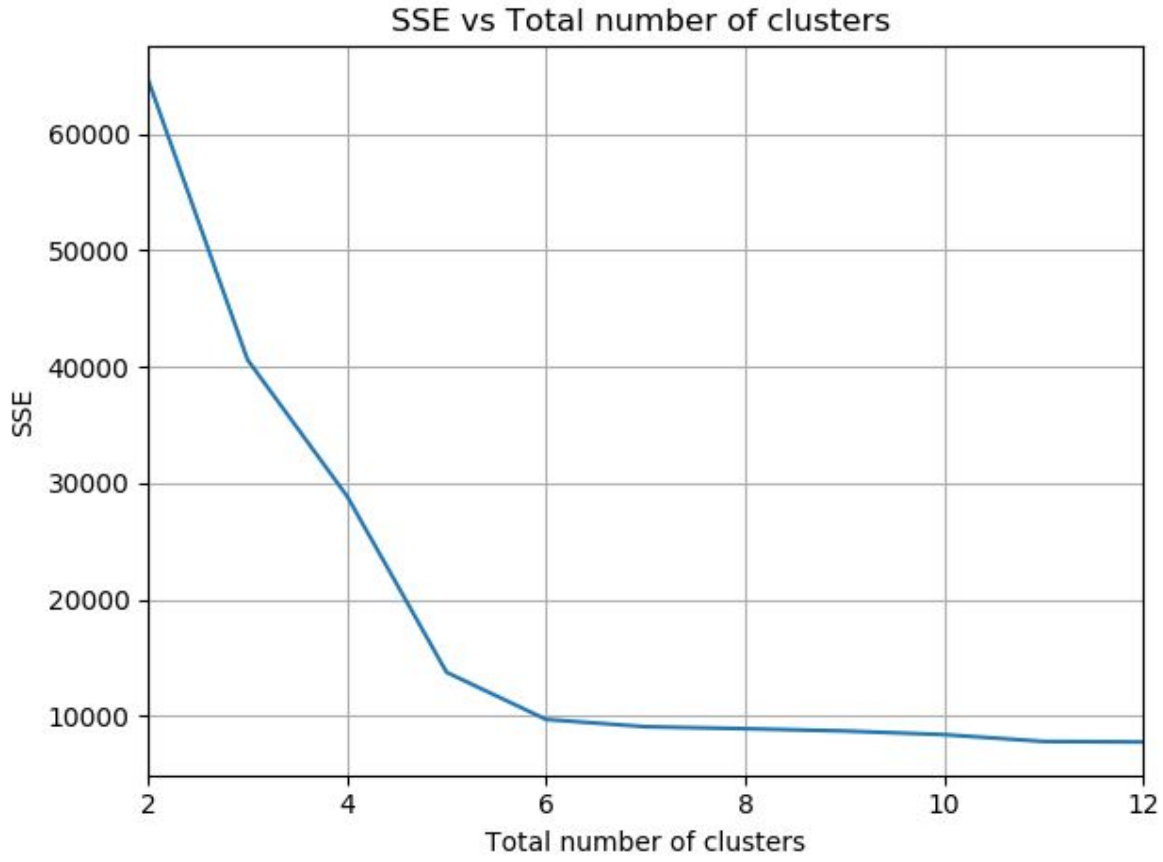
```
The number of data points for centroid  [14.5 10.3  0. ] = 219
The number of data points for centroid  [ 3.  13.5  2.9] = 133
The number of data points for centroid  [ 2.1 15.3  2.7] = 113
The number of data points for centroid  [ 2.2  8.7 -0.1] = 256
The number of data points for centroid  [-5.7 10.   7.9] = 152
The number of data points for centroid  [-5.   8.   7.4] = 97
The number of data points for centroid  [-10.2   6.2  -2.3] = 115
The number of data points for centroid  [-10.5   7.7  -3.2] = 182
The number of data points for centroid  [-20.2   8.7   3.8] = 10
The number of data points for centroid  [ 4.9 13.   9.5] = 12
The number of data points for centroid  [-11.1  12.    4.6] = 15
The center of mass = ['[14.5 10.3  0. ]', '[ 3.  13.5  2.9]', '[ 2.1 15.3  2.7]', '[ 2.2  8.7 -0.1]', '[-5.7 10.   7.9]', '[-5.   8.   7.4]', '[-10.2   6.2  -2.3]', '[-10.5   7.7  -3.2]', '[-20.2
The best sse for 11 clusters is: 7787.71
Total time to complete 500 iterations for 11 clusters = 2485.08 seconds
```

**For cluster k=12**

I plugged in the power cord for this iteration. (The time taken would seem a bit weird). But it was approximately ~3000 seconds when I tried to run it without plugging in the cord.

```
The number of data points for centroid  [14.2 11.1 -0.6] = 104
The number of data points for centroid  [14.6  9.7  0.6] = 120
The number of data points for centroid  [ 3.  13.5  3.2] = 116
The number of data points for centroid  [ 2.2 15.1  2.6] = 128
The number of data points for centroid  [ 2.1  9.2 -0.2] = 186
The number of data points for centroid  [1.8 7.  0.6] = 83
The number of data points for centroid  [-5.7 10.1  7.7] = 132
The number of data points for centroid  [-5.4  8.5  8.1] = 106
The number of data points for centroid  [-10.3   6.3  -2.3] = 123
The number of data points for centroid  [-10.5   7.7  -3.2] = 176
The number of data points for centroid  [-15.   10.9   4.6] = 22
The number of data points for centroid  [ 2.3 13.6 11.2] = 8
The center of mass = ['[14.2 11.1 -0.6]', '[14.6  9.7  0.6]', '[ 3.  13.5  3.2]', '[ 2.2 15.1  2.6]', '[ 2.1  9.2 -0.2]', '[1.8 7.  0.6]', '[-5.7 10.1  7.7]', '[-5.4  8.5  8.1]', '[-10.3   6.3  -2.3]'
The best sse for 12 clusters is: 7741.31
Total time to complete 500 iterations for 12 clusters = 2455.99 seconds
```

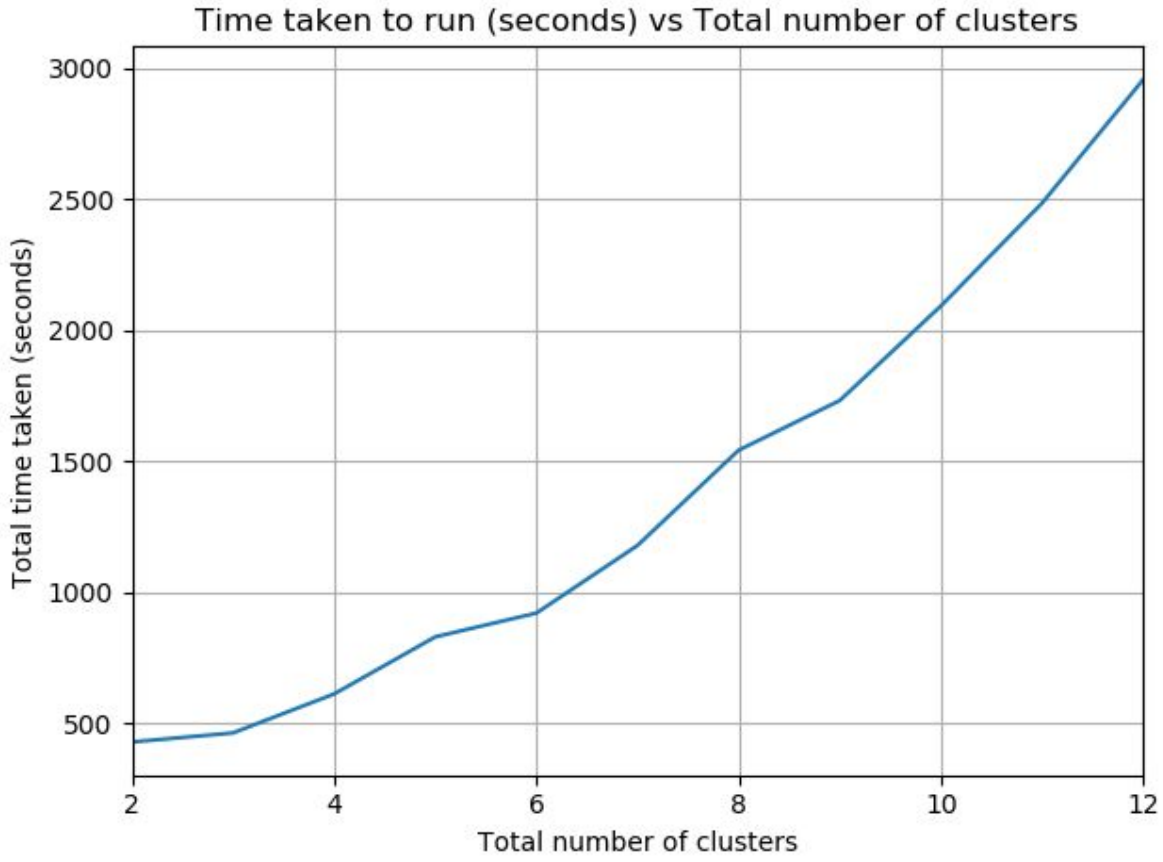
**b)** SSE vs K (Number of clusters)




**c)** After observing the graph/plot of SSE vs K, I could observe that as the value of k increases, the SSE drops rapidly. This means that as we cluster the original data using more number of centers, we can reduce the SSE and have a better clustering.

I believe that K=6 would be the optimal solution as it is the proper knee point in the graph and after this point the SSE in the following K clusters till 12 drops off gradually.

**d)** Time required for completion vs K (Number of clusters)




As the number of clusters increases, the complexity of the calculations increase. It is a very time-consuming task to perform 500 iterations for more than 5 clusters.

**e)** The basic flow of the program is that for each k cluster, we iterate 500 times to select k random centroids and then we assign the data points to these centroids. Next, we recalculate the new centroids based on the values/data points assigned to the randomly selected clusters and assign this new calculated centroids as our current centroids for the next iteration. We do this until we hit the stopping condition.

The stopping condition that I used to break the inner loop had a check on whether the current centroids are equal to the newly calculated centroid. If they are not equal, I assign the newly calculated centroids as the current centroids and then I loop again till the current centroids become equal to the next newly calculated centroids.

For 6<K<10 clusters I applied another check to avoid situations where the centroids are getting assigned again and again and they never become equal. If the loop iterations go beyond 100, then I stop and break out of the loop. This avoids situations where the cluster centroids selected. Similarly, for 10<K<12 I updated the value of loop iterations to 50.

I tested my stopping criteria by printing out statements for every iteration so that it is clear that it is not going in an infinite loop.

The data structures that I used are lists and dictionaries. Some lists contained basic floating point integers for storing centroids. Some lists contained numpy arrays as list elements. The dictionaries consisted of key value pairs where each key was a centroid that was converted from a numpy array/list to a string because dictionary keys can not store mutable objects. The values were a list of numpy arrays where each numpy array refers to a data point that belongs to the cluster key (centroid).

There were some issues at the beginning in deciding the correct way and data structure to store the centroids and their corresponding data points. I came to know that lists can not be stored as dictionary keys (mutable).

**(g)** The hardest part of getting this assignment finished was the time constraints and the complexity of the code. The code was not that difficult but debugging it on a medium specifications laptop made it time-consuming. At first, I made the mistake of storing the same information in too many redundant dictionaries and lists which made it more complex and increase the time complexity. I figured out the redundant code while I was waiting for the output which took a lot of time.

**(h)** Doing this assignment cleared my concepts of k-Means and I am pretty sure that I don’t need to mug up the algorithm anymore. One interesting point that I read on the web while browsing about Mahalanobis distance is that choosing between the Mahalanobis distance and the Euclidean distance depends on the shape of the clusters. If the clusters have circular shape, Euclidean distance is preferred. But if the clusters are elliptical then Mahalanobis is a better option.