# CS 478

## Scott Leland Crossen

## Clustering (KMeans) Project

### Part 1

The code that implements the kmeans clustering algorithm is included with the submission of this project. It is written in Scala and follows the functional paradigm. The output for k=4 clusters on the sponge dataset is shown below:

#### Expand for Full Output

#### Final Output

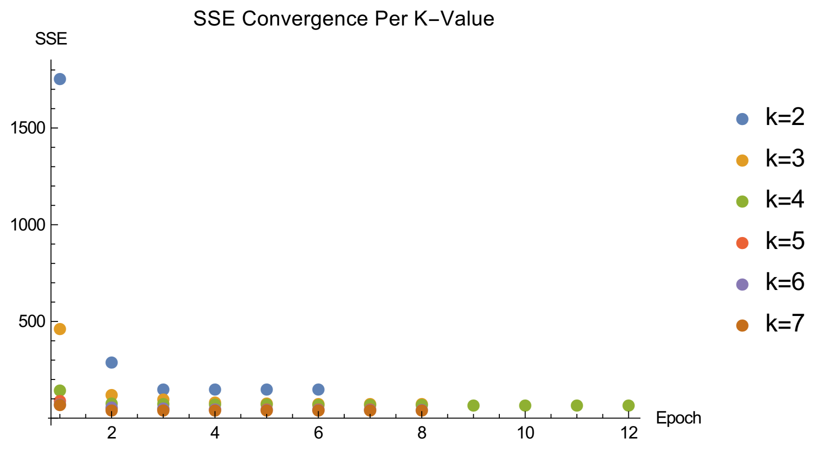
```
Centroid 0 = 1_CAPA, SIN_CAPA_INTERNA_DEL_CORTEX, SI, 2.444, SIN_TILOSTILOS_ADI-
CIONALES, SIN_ESPICULA_PRINCIPAL_ESTILO, 2.611, 0.444, OTROS
Centroid 1 = SIN_CORTEX, SIN_CAPA_INTERNA_DEL_CORTEX, NO, 0.000, SIN_TILOSTILOS_ADI-
CIONALES, SIN_ESPICULA_PRINCIPAL_ESTILO, 2.000, 0.000, OTROS
Centroid 2 = SIN_CORTEX, SIN_CAPA_INTERNA_DEL_CORTEX, NO, 0.000, SIN_TILOSTILOS_ADI-
CIONALES, SIN_ESPICULA_PRINCIPAL_ESTILO, 0.500, 1.250, OTROS
Centroid 3 = 2_CAPAS, TANGENCIAL, SI, 3.040, INTERMEDIARIOS, SIN_ESPICULA_PRINCIPAL_ES-
TILO, 2.520, 2.480, OTROS
Making Assignments
        0=0 1=1 2=2 3=2 4=2 5=2 6=3 7=1 8=1 9=1
        10=1 11=1 12=0 13=3 14=3 15=3 16=3 17=3 18=3 19=3
        20=3 21=3 22=0 23=3 24=3 25=3 26=0 27=0 28=3 29=3
        30=3 31=3 32=3 33=1 34=1 35=1 36=3 37=1 38=1 39=0
        40=0 41=0 42=1 43=1 44=1 45=1 46=0 47=3 48=3 49=0
        50=1 51=1 52=0 53=0 54=1 55=1 56=1 57=1 58=1 59=1
        60=0 61=0 62=1 63=0 64=0 65=1 66=1 67=1 68=1 69=1
        70=0 71=3 72=3 73=0 74=3 75=3
Amount of instances: Cluster 0: 18, Cluster 1: 29, Cluster 2: 4, Cluster 3: 25
Individual SSE: Cluster 0: 61.167, Cluster 1: 42.000, Cluster 2: 5.750, Cluster 3: 95.440
Total SSE: 204.357
```

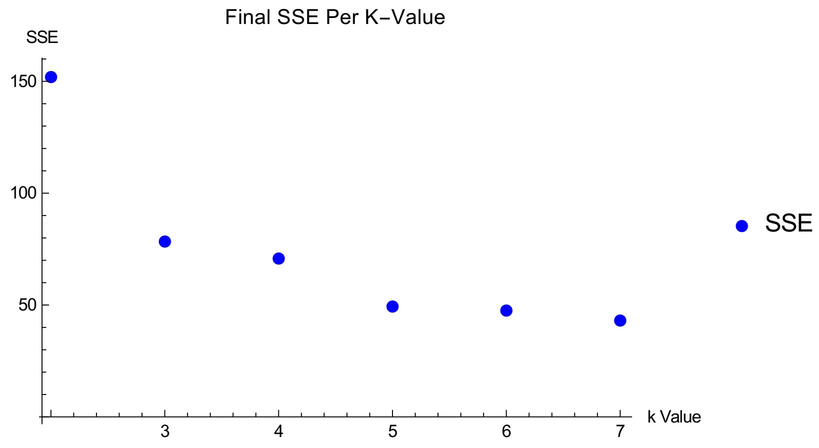
The algorithm converged on the final SSE after 11 iterations

### Part 2

For this problem, unnormalized data was used. For each trial, the convergence on the final SSE is shown first. This plot shows SSE vs the Epoch that it takes to converge. The next plot shows the final

SSE value against what k-value was used. Understandably, the greater the amount of clusters, the smaller the SSE is. This is because adding more centroids will never take away from the overall algorithm. More centroids will always reduce the overall SSE because there is more opportunity for the individual instances to lie next to a centroid. At the very least more centroids will just split the previous centroids. For the case of the Iris data set, the clustering mainly depends on where the initial centroids are created. For the most part, the clusters will form along the label boundaries but this is only the case when a centroid exists with that label.





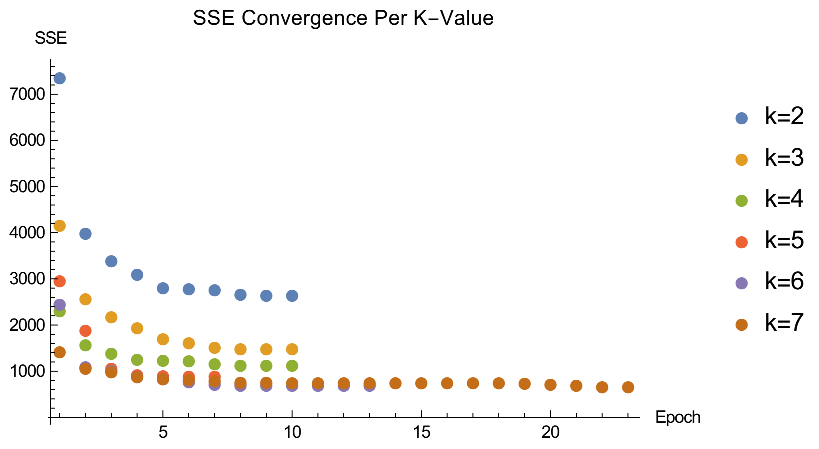
Adding the instance label produces much more reliable results. Almost always each cluster will have around 49-51 instances in it. Interestingly enough, clusters above k=3 are just divisions of the k=3 clusters. at k=4, there will be two clusters of size ~50 and two clusters of size ~25. Upon investigation, the former clusters are just divisions of one of the k=3 clusters. The clusters also reliably fall along label boundaries and almost always assign the same instances to the same label-cluster.

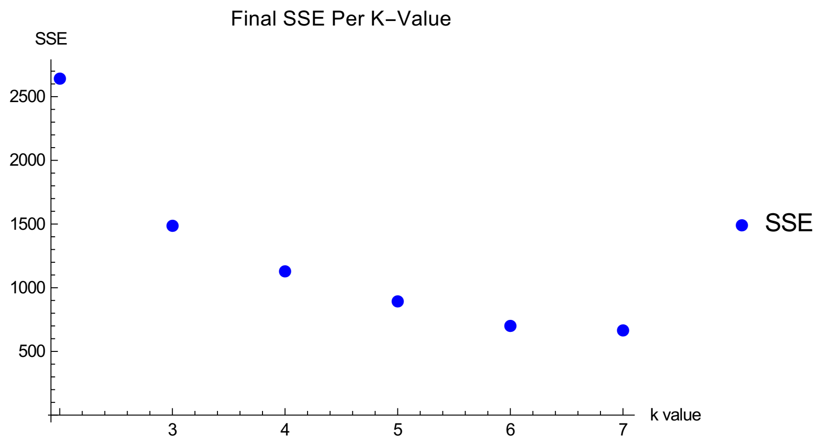
## Part 3

The final feature in the rings data is treated continuously below. This is likely because because the amount of possible values for this feature is around 29. Because there are so many possible values, it makes more sense to use a euclidian distance-type metric rather than a mode of the features. Assuming that there is an inherent physical meaning to the final feature, a distance metric will more accurately capture the cluster than a mode because it considers more cluster instances than a mode when recalculating the cluster centroid.

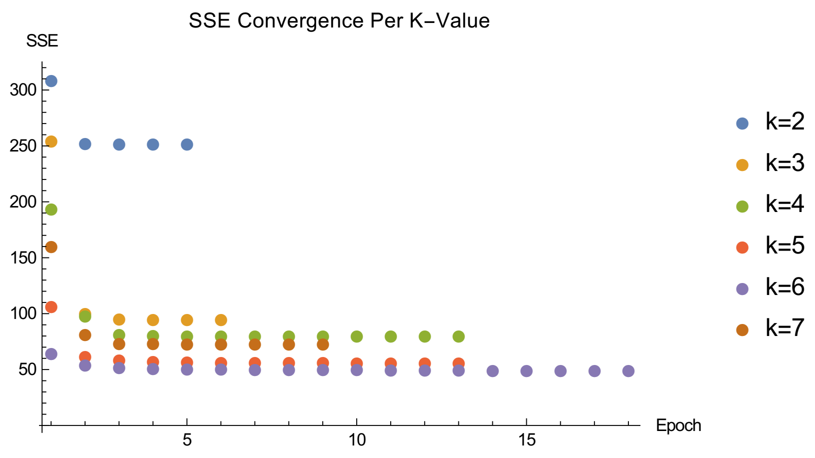
As before, the convergence per k-value is shown below. It took the k=7 data set the longest to converge probably because instances in the cluster kept switching between clusters. The final SSE values are shown in the next plot. As always, a greater k-value creates clusters with instances closer to the centroid.Interestingly enough, there was very little difference between the k=6 and k=7 trials.

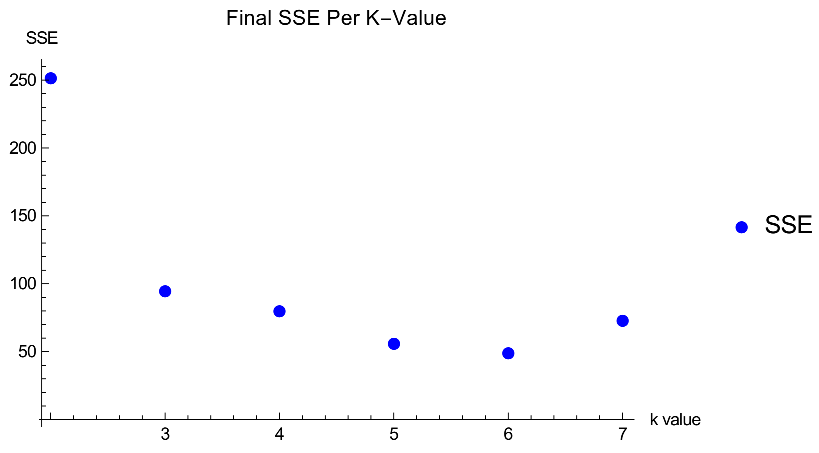
The unnormalized data is shown first.





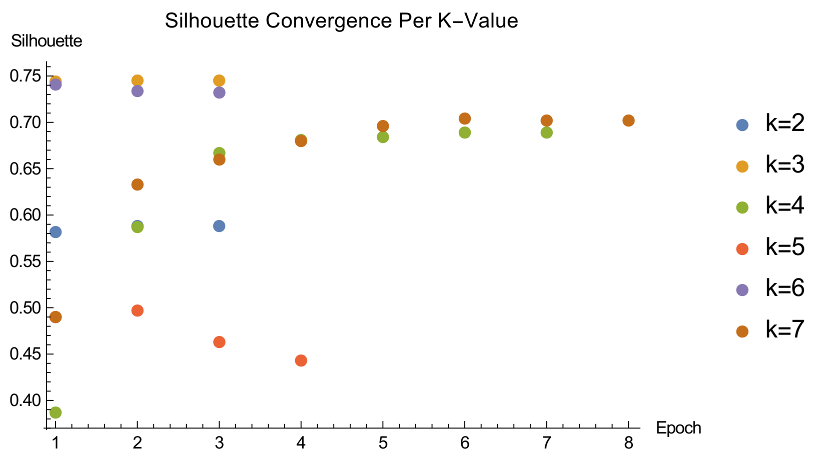
The normalized data is shown next. Notice that there are some really weird behaviors with the k=2 data. This is obviously not a good fit on the data. On/after k=3, the algorithm seems to approach a more reasonable fit. In fact, after k=3 the SSE does not improve very dramatically at all. The k=3 and k=7 points are very close to each other.

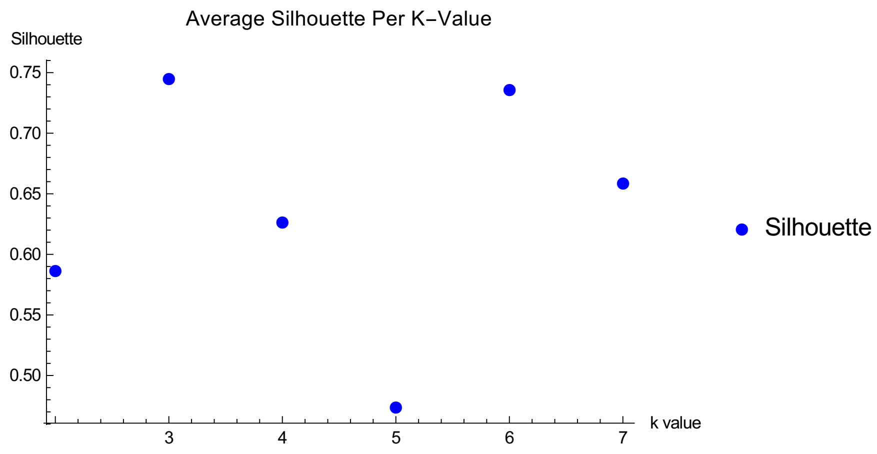




## Part 4

Below are the requested plots showing the silhouette metric on the abalone data set. The first plot shows how many epochs it took for each k-value to converge. The second plot shows the final average silhouette per k-value. higher silhouettes were found at k values of 3 and 6. The lowest silhouette score was found at 5. It makes sense that the highest silhouette scores were found at multiples of each other. This is probably the case with every multiple of the natural cluster. The data set is probably naturally clustered into 3 clusters and the silhouette score reflects that with higher silhouettes at 3 and 6. Splitting 3 clusters into 2 each will still result in a good silhouette because of the even division. In this way, the silhouette score is much better than anything else to decide which cluster size to use. It shows which k-values are optimal quite easily.

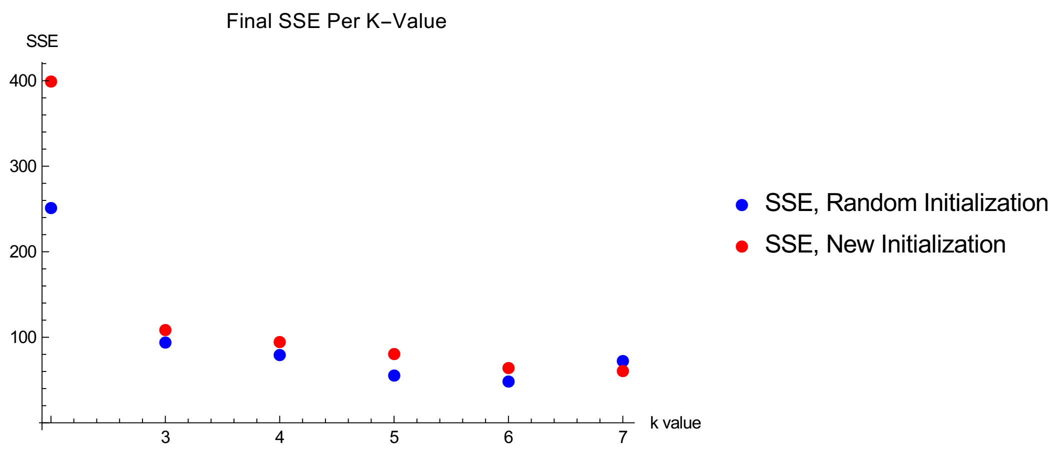




## Part 4

For my expansion project, I designed ways to select starting centroids. The function is applied to the entire data set one column (feature) at a time. The next few paragraphs describe this algorithm assuming a cluster amount of k:
For continuous features the set of all possible values is sorted and divided into k+1 portions. Each boundary instance is then sequestered to be used as a centroid.
For nominal features, the set of features is sorted and the features with more instances are shown first. the data is then divided into k+1 even portions again and the boundary instances are used as centroids.
For unknown values, features are discarded unless more than 2/(k+1) times the total size of the data set is unknown. At this point, some centroids will be calculated as unknown until there centers are re-assigned.

What I found is that there was no improvement using this method over a normal randomization Initialization. Below is a comparison between the randomization and this new method using the un-normalized abalone data set:



The reason this showed no improvement was likely because the centroids it calculated were not actually part of the dataset. The combination of columns were so wrong that it was no better than random assignment. It took no consideration of associated features and whether the resulting centroid was even near the rest of the data.

In an attempt to make an initialization algorithm that worked, I next created an algorithm that chose instances based on how far they were from each other. Under this method, the program would take the entire dataset and calculate the distance between each instance and every other instance. It would then pick the two furthest-away points. This worked great for two clusters, but I couldn’t figure out what to do for more clusters. Somehow I would need to calculate the instance that maximized the distance from the initial two, but I couldn’t figure out a good way to do this. Besides, it was already decently slow for two clusters.