Clustering Lab Report

1.0

For this lab, we were tasked with implementing 2 clustering algorithms, HAC and KMeans. HAC is a process where you start with N clusters, one for each point in your dataset. At each step, you merge the two closest clusters according to some distance metric (single or complete for this lab). You repeat this process until you have the desired number of clusters remaining. This process is deterministic and does a fair job of consistently clustering unlabeled data. KMeans is a clustering process which is a bit more stochastic. To begin, you select K random points from the dataset and use those as your initial 'centroids.' At each stage you first assign points to the cluster with the nearest centroid, then you recalculate the centroids based on the center of the points in the cluster. For example, if I had the data [1,2,3,4] and my initial centroids were [1,4], then my initial clusterings would be [1,2] and [3,4] and the new centroids would be [1.5,3.5].

1.1

There were two main hurdles I had to overcome in this part of the lab. The first issue I had was getting my output to exactly match the debug output. After quite a while of debugging, I was confident my core algorithm was correct, so I started to look at my normalization. I was initially using the sklearn normalize preprocessor with 'l1' normalization. This normalizer defaults to normalizing each data sample instead of each feature, which seems very strange to me and lead to lots of debugging frustration. After poking around quite a bit, I figured out I needed to use the sklearn MinMaxScaler to calculate the (x-xmin)/(xmax-xmin) formula described in the lab specs. After I had figured out the correct normalization approach, my sample numbers and centroids were matching exactly with the debug output.

Once I had figured out the normalization and centroids, I began to work on the silhouette score portion of the lab. I felt I had a pretty solid understanding of silhouette scoring, so I thought this would be fairly straight forward. I was definitely wrong. After a few hours of rewrites, manual stepping through, and hand solving smaller problems, I was left very confused as to why my larger outputs weren't exactly right. My output values were very close, but they just didn't quite match sklearns. Usually, my output differed by around .02 which was just incredibly frustrating. Eventually, I just gave up on getting the full 5% extra credit and just hoped my half-finished approach would maybe get me a few extra points. In hindsight, I should have just enjoyed my thanksgiving evening and left the silhouette score calculations to sklearn.

Once I had given up on silhouette score, the section was pretty straight forward. My debug output exactly matched what it should, so I am reasonably confident my evaluation output will as well. The outputs for each of the models is a bit verbose, so hopefully my formatting is good enough. My evaluation output for complete link was:

```
Num clusters: 5

Silhouette Score 0.2930
sse  43.737292816168946
( 0.1656,0.2226,0.3509,0.2328,0.1520,0.3424,0.1918,0.0000 )
count 18 sse 6.103276601086827
```

```
( 0.7918,0.8268,0.6060,0.7669,0.7787,0.4634,0.7877,1.0000 )
count 49 sse 16.4629953844541
( 0.5326,0.6002,0.4536,0.5531,0.5144,0.5401,0.6266,1.0000 )
count 21 sse 5.68996386485991
( 0.3636,0.4107,0.5609,0.3857,0.3937,0.2808,0.3155,0.0000 )
count 49 sse 14.690217666282146
( 0.3471,0.3622,0.7897,0.2790,0.4410,0.8568,0.2234,0.0000 )
count 3 sse 0.7908392994859765
```

And for single link:

```
Num clusters: 5

Silhouette Score 0.3971
sse  51.446327922953806
( 0.7140,0.7588,0.5603,0.7028,0.6994,0.4864,0.7393,1.0000 )
count 70 sse 27.650392389558274
( 0.3159,0.3648,0.5145,0.3490,0.3338,0.3077,0.2868,0.0000 )
count 67 sse 23.795935533395546
( 0.3065,0.3333,0.6979,0.2792,0.3787,1.0000,0.2373,0.0000 )
count 1 sse 0.0
( 0.2774,0.2597,1.0000,0.1224,0.4505,0.7549,0.1290,0.0000 )
count 1 sse 0.0
( 0.0884,0.1818,0.0000,0.1450,0.1538,0.1245,0.0000,0.0000 )
count 1 sse 0.0
```

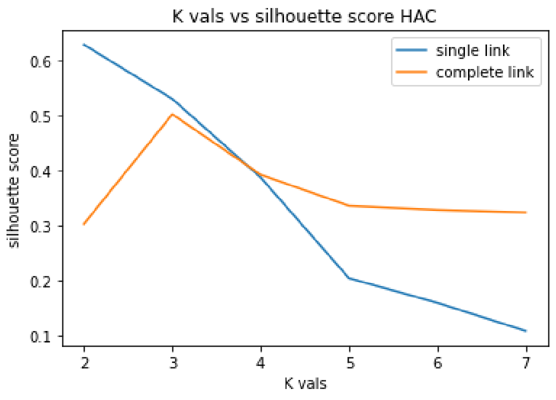
1.2

For this segment, I didn't run into any major issues. I was able to just program the KMeans algorithm and have it work pretty much right off the bat. There were a few small issues I had, but none are worth mentioning and they were all able to be resolved without much trouble. As previously described, K-means is as simple as assigning points to groups, then recalculating the groups so there isn't a lot to discuss here. After getting my debug to match exactly, my evaluation output is as follows:

```
K val 5
Silhouette Score  0.2769
SSE 42.20068633746149

( 0.5349,0.6013,0.4617,0.5541,0.5232,0.5544,0.6248,1.0000 )
count: 22 sse 6.138029145693975

( 0.3684,0.4338,0.4356,0.4388,0.3573,0.2988,0.3673,0.0000 )
count: 31 sse 8.109084617255881

( 0.3658,0.3854,0.7563,0.3093,0.4659,0.3691,0.2413,0.0000 )
count: 20 sse 6.539395366004818

( 0.1634,0.2137,0.3968,0.2178,0.1555,0.3078,0.1775,0.0000 )
count: 19 sse 5.626176224546659

( 0.7961,0.8310,0.6054,0.7709,0.7802,0.4553,0.7919,1.0000 )
count: 48 sse 15.78800098396018
```

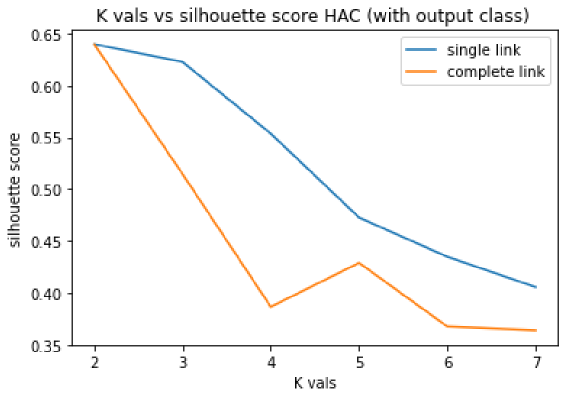
2

In Section 2, we began to use our models to solve an actual problem, namely the Irtis classification problem. Besides just using HAC and KMeans, we also tried different values of K and different distance metrics to see how they would affect our silhouette score. For all problems in part 2, I decided to use the MinMaxScaler to normalize my data. When doing my analysis, I was surprised at how similar single and complete links were. Usually in this class, one approach just handles problems significantly better, but in this case these two distance metrics produced similar quality results. Below is my chart showing how the silhouette score changes when the distance metric and K values change.



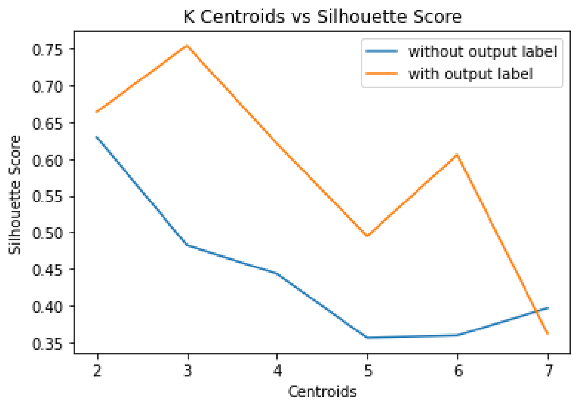
After looking through the output of this section's code, I believe complete link forms more evenly sized clusters, while single tends to form a few mega clusters that span larger areas. While neither one of these approaches is always better than the other, there are cases where each is better than the other. It seems for low K values, single link tends to cluster better than complete. However, when you increase the K values, complete link starts to get ~triple the silhouette score of single link. While I do not believe this is a general trend that holds for all problems, it does show that different algorithms may do better or worse on specific sections of the same problem.

After running the numbers without allowing the output as an input feature, I tried adding the output class as a feature to see how that affected clustering. Below is the silhouette score vs. k vals with the output feature included as one input



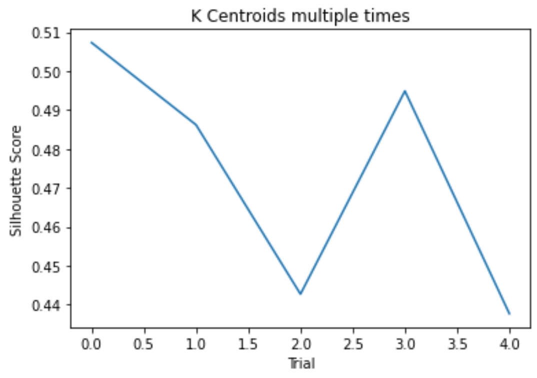
Note the difference in the scale on this graph compared to the other. When we didn't include the output feature, our silhouette scores got as low as .1 in some cases. When we did include the output feature, our silhouette scores were generally higher by a good margin than we got for output-less graphing. By including the output as a feature, HAC was able to see instances with the same output as being more similar which led to more optimal groupings. In this case, single link was able to effectively traverse similar instances which led to it maintaining higher silhouette scores.

After running analysis on HAC, I went on to do a similar analysis on K Means. Below is the image showing my results.



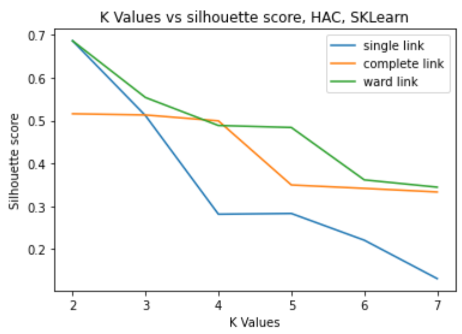
When including the output label, my silhouette scores were generally significantly higher than without including them. I believe this is because the centroids were able to more easily identify the regions in space where lots of points were gathered. This allowed the KMeans to place centroids at these high density areas which improves the silhouette score.

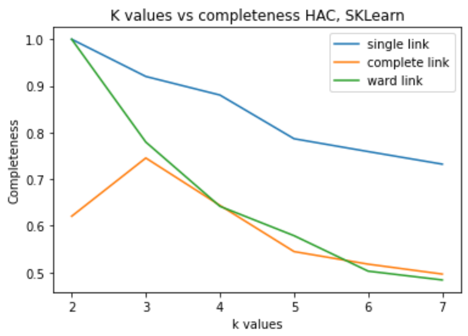
For the final part of this section, I just ran the KMeans algorithm multiple times with the same hyperparameters to compare the silhouette scores. As previously mentioned, KMeans works by randomly selecting K points to be the initial centroids. This means that KMeans is not deterministic in the same way HAC is. Therefore, run to run, the silhouette score will change. Below is the silhouette score over 5 runs. Note that there is no significant pattern in this graph.



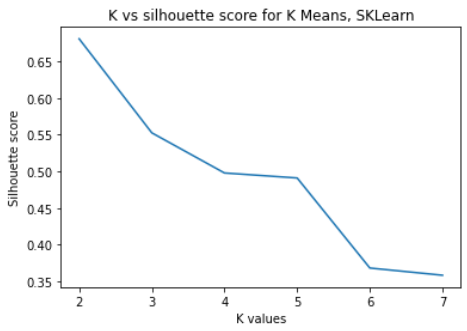
3

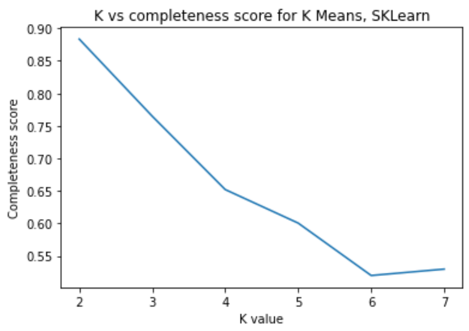
In this section, we did similar experiments as before, but we instead used sklearn models to do the clustering. Besides only using silhouette score, I also used the sklearn completeness metrics which rates how similar the cluster's targets are. For example, if a cluster all had the same target, the completeness would be high. Below are the silhouette scores and completeness scores for SKLearn's HAC. In these diagrams, we can see single link does a better job of splitting along output classes, while complete link does a better job of splitting by distance. This new insight granted by completeness can help to inform our distance metric decision. While doing this, I also experimented with different hyperparameters. I tried using different ‘affinities’ parameters, but none of them improved my results. I also added a





After doing this analysis using HAC, I went on to do a similar analysis using KMeans. Below are the silhouette and completeness scores for various K values from SKLearn’s KMeans implementation. Just like with HAC, I decided to use completeness as the augmenting scoring function. I tried using sklearn’s ‘k-means++’ centroid selection algorithm, but I wasn’t able to notice any difference in speed or performance from the ‘random’ selection algorithm.





4

As previously explained, this section was a bit of a failure. I tried and tried to get my silhouetteScore function to work correctly, but after a few hours of trying, it was still consistently incorrect. I’m not sure why I was getting different results, but I ended up giving up.