Alex Heiner
Clustering Lab

1 Algorithm Analysis

HAC

When the fit() function is called each point is wrapped in a numpy array, which initializes each point to be its own cluster. Then the overall while loop starts, which runs until the size of the clusters array is the same value as k.

The main functionality of the HAC algorithm happens in the update_clusters() function. This function is used by both single and complete link clustering. In order to combine clusters, the function loops through each cluster, and finds the distance to every other cluster. The first iteration, each point is in its own cluster so we are essentially finding the distance between each point. As clusters are combined the algorithm has to handle the case of finding the distance between clusters with single points and clusters with multiple points, as well as two clusters with multiple points. I handled these cases by checking the length of the cluster array in each iteration. If the cluster is made up of multiple points, I loop through each point in that cluster, and find all the distances.

When there is a cluster with multiple points, the distance that is chosen to represent the distance between two clusters depends on the link type of the algorithm. If we are doing single link, we want the smallest distance to represent the distance between the clusters. If we are doing complete link, it is the largest distance out of the options to represent the distance. This is handled in my get_should_update() function, which returns true if we should update the value that we want.

After looping in the update_clusters() function, I find the best overall distance, find the index of that point in the cluster, add that point to the cluster, and delete the point from the previous cluster.

The update_clusters() function gets called from the overall while loop until the number of clusters is the same size as k. Then my score function calculates the centroid and finds the SSE for that cluster.

K-Means

The K-Means implementation was very straightforward. The difference between this algorithm and HAC is that the clusters are created based on the centroids. At the beginning of the training, the initial centroids are set to either the first k points, or k random points from the data. Since the clusters are determined by the centroids, each iteration of the overall while loop clears the previous clusters and creates new ones from the updated centroids. The K-Means training continues until the centroids have not been updated.

Again, I implemented an update_clusters() function that does most of the work. This function loops through the points, then loops through the centroids, and figures out which centroid it is closest to. After the closest centroid is found, that point is added to a cluster that the centroid is in.

After each point has been added to a cluster, I loop through the clusters, calculate a new cluster value, and use numpy.array_equal() to see if the centroids have changed. If a centroid has been updated, the algorithm will continue to run. Otherwise we are done training.

Handling Nominal Features or Unknown Values

For nominal features I think I would use a 0,1 output as a distance measurement. This is similar to what I did in the KNN lab. Distance would be 0 if they are the same/alike, and 1 if they are not the same/different.

The way I would handle unknown values would depend on the dataset values. If the dataset values were continuous, I would replace the unknown value with the mean from that column. If the dataset values were continuous, I would replace the value with mode from that column.

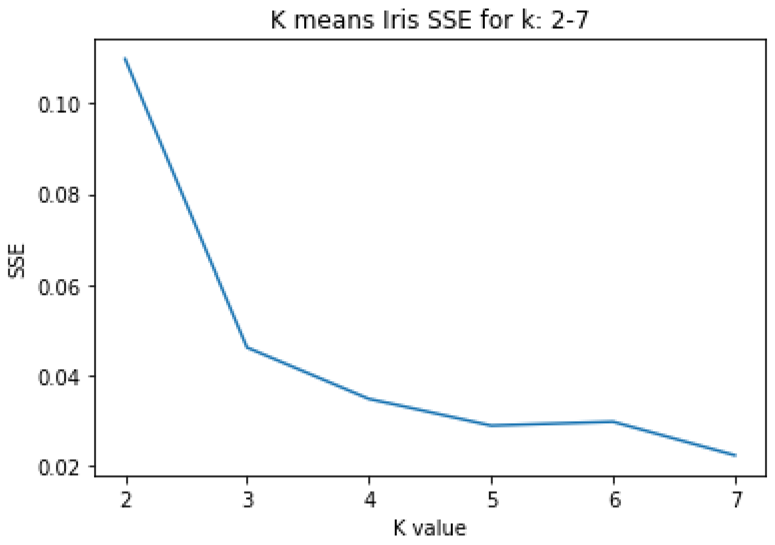
Experience

OverallI thought the implementation for this lab was pretty straightforward. One of the hardest parts was figuring out how to combine clusters for both HAC and K-Means. This was just a result of trying to figure out how to work with multi-dimensional numpy arrays.

2 Run HAC & K-Means on Full Iris Data

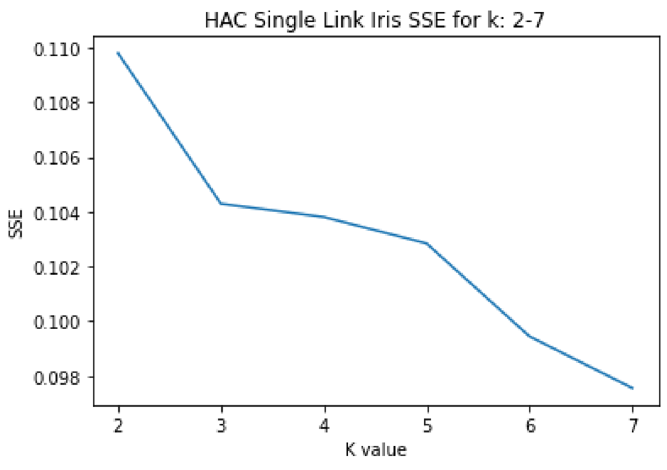
I chose to normalize while training on the Iris dataset.

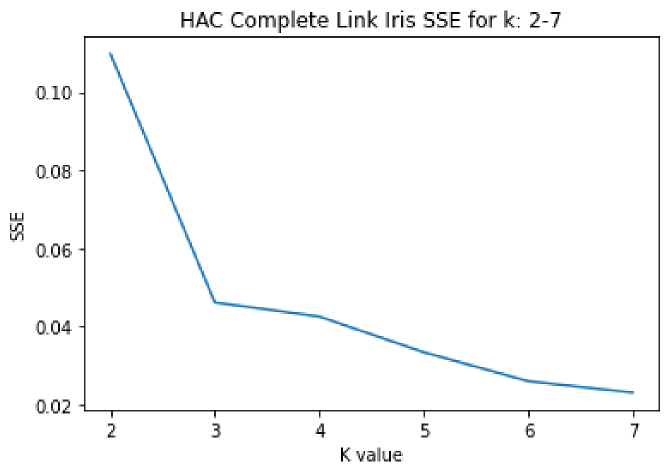
Training K-Means model on Iris dataset



This graph shows that generally as the number of clusters increases, the SSE decreases. I don’t think this would be true for all values of k, but that is what it shows for k=1-7. This makes sense because if there are more clusters, there will be fewer points in them. This means that the points in the clusters will generally be closer to their centroid.

Training Single and Complete Link HAC on full Iris dataset





Similar to above, the number of points in a cluster will be smaller if there are more clusters. Since the centroid is at the center of the cluster, if there are fewer points, it seems like they will be closer to the centroid. This doesn't mean that the clustering is correct, or makes the most sense, but it will help the points to be closer to their centroid.

It is also interesting that the SSE for the complete link is smaller than the SSE for the single link. Since the single link chooses the minimum distance from cluster to cluster, it seems like it would have a smaller SSE.

Training K-Means 5 times with k=4

There was a little bit of variation as I trained with the Iris data 5 times with k=4. The SSE usually stayed at a value around 0.03-0.04, but sometimes there was one iteration where it jumped up to a value of around 0.1. I'm guessing this could be a result of picking k random centroids to start. This didn't happen every time, but it did happen while I tested this multiple times.

3 SK versions of K-Means and HAC

I did not write my own code to get the silhouette score.

Discussion on impact of silhouette score when selecting best clustering

It was pretty helpful. I think if I were to print out the silhouette graph it would've been more helpful. In general, the closer the silhouette is to 1 the better it is. As I trained with the HAC and K-Means models with the k value ranging from 2-7, it seems like they all did better with lower k values.

This is especially true with HAC single link. When k=2 the silhouette score is 0.69, then by the time k=4 it drops to 0.17. It continues to drop through the rest of the training, and ends at a value of 0.01 when k=7. The other models did not drop as fast, but their highest silhouette score was when k=2.

Discussion on different hyper-parameters for new dataset

I chose to use a breast cancer dataset for the hyper-parameter experiment

For AgglomerativeClustering I got my best silhouette score with linkage set to “single” and the number of clusters set to 2. I tried all types of linkages and many different cluster sizes.
For the K-Means I had a harder time getting the silhouette score to improve. It seemed to reach a peak of 0.697. I got this score with the number of clusters set to 2, and the algorithm set to auto. I also tried changing the number of times the algorithm will be run with different centroid seeds, max iterations, and different tolerance levels.