Ben Zamboldi
William & Mary
B.S., CS & CAMS ‘22
3 April 2022

K-Means README

The Python packages used for this part of the problem set include Pandas, Math, t-SNE, and Seaborn. I used Pandas to import and store the data, as well as to manipulate the data for use in the K-mean algorithm. I used the Math package for the square root function. Finally, I used the t-SNE and Seaborn package to plot the high-dimensional data to visualize the training clusters.

First, a function to calculate the Euclidean distance between two points was implemented. The function takes in two lists of points of the same dimension. The function loops through each dimension of the points and adds the squared difference to the total sum and finally returns the square root of the sum. The Math package was used to calculate the square root. Because the dimensions of the points are constant, this method runs in constant O(1) time.

To implement the K-means algorithm, I created a class K_means. The init method initializes the data, K value, and the initial centroids and clusters. The data initialized is the training dataset and the K is set to a positive integer value. According to the algorithm, the centroids are to be assigned to random points in the dataset. However, to keep the project consistent, I used the suggested initialized centroids posted in the class Piazza. To initialize the clusters, I iterated through each row of the training set (representing each point) and calculated the distance to each of the cluster centroids. Then, the point would be initialized to the cluster of minimum distance. Because this method loops through the training dataset once, the runtime is O(n) where n is the number of entries in the training dataset.

The recompute_center method serves the purpose of calculating the mean point of each cluster. For each cluster, the method averages the values across all points in the cluster for each dimension. This provides a new, updated centroid for each cluster that represents the average point of all points in the cluster. The method updates the self.centroids list to reflect the updated centroid points. Because the method is iterating across every point in the dataset, the method is of O(n) runtime where n is the number of entries in the dataset.

The assign_clusters method recursively assigns the training data to clusters. First, the recompute_center is called to get updated, averaged centroids for each cluster. Then, for each entry in the training dataset, the Euclidean distance is calculated to each updated centroid and that point is assigned to the cluster of minimum distance. At this point, the cluster centroids have been recomputed and the points have been assigned to the nearest cluster. The check_difference method is used to check if the clusters have changed. If the clusters have not changed, the algorithm has converged, and the method returns. If the clusters have changed, then assign_clusters() is recursively called to recompute the centroids and update the clusters until convergence. When the algorithm converges, the method will return the centroids. The runtime of this method is linear O(n) where n is the number of items in the training set.

With the K_means class implementation, I created an instance with the K_means training data and initialized K=3. I called the assign_clusters method on the instance to recursively calculate the clusters which were returned and stored in a list. Then, with the testing data, I iterated through each point in the set to make predictions. For each point, the Euclidean distance between the point and each of the clusters was calculated and the point was assigned to the

cluster of minimum distance. This provided the predictions for the unlabeled testing data based off the training data points. Overall, this algorithm is of linear time, O(n). The predicted results can be seen in the table below.

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	labels
0	1	4.6	3.2	1.4	0.2	cluster_2
1	2	5.3	3.7	1.5	0.2	cluster_2
2	3	5.0	3.3	1.4	0.2	cluster_2
3	4	5.7	3.0	4.2	1.2	cluster_1
4	5	5.7	2.9	4.2	1.3	cluster_1
5	6	6.2	2.9	4.3	1.3	cluster_1
6	7	6.3	2.5	5.0	1.9	cluster_1
7	8	6.5	3.0	5.2	2.0	cluster_3
8	9	6.2	3.4	5.4	2.3	cluster_3
9	10	6.3	3.7	5.0	2.1	cluster_3

I additionally plotted the clustering results of training for K-means using the sklearn t-SNE and Seaborn packages. I used the TSNE.fit_transform method on the training dataset to calculate the t-SNE embedding. Then, with the features ‘SepalLengthCm’ as x and ‘SepalWidthCm’ as y and the assigned clusters as the hue, I plotted the t-SNE embedding to a Seaborn scatterplot. The results of the scatterplot can be seen below.