Project 2 - K-Means Algorithm Part 1

Rishikesh Iyer ID: 1227881670

February 2024

1 Project Summary

This project involved understanding and implementing the K-Means algorithm, which is considered an unsupervised learning algorithm as it receives a random set of data points with no labels. The algorithm is designed such that it will take an input of a dataset in order to eventually create k number of clusters from the dataset and then produce the closest centers to those clusters. These centers are called centroids. This means if there are 2 centers or centroids given into the algorithm, there will be 2 sets of clusters after the algorithm is finished, each with its own corresponding centroid.

The basic summary of k-Means is that it will take in a set of data samples, initialize a random set of center vectors, then keep iterating through the data samples and classify those samples to whichever center vector it is closest to through checking their Euclidean distances. The input centers are then replaced by the mean of the clusters, and then the algorithm will take these newly created centers and repeats the process. The algorithm stops when the centers stop changing. This can also be checked by the objective function or the loss function, which is represented by $\sum_{i=1}^{k}\sum_{xD}\|\text{x-}\mu\|^2$, where k is the number of clusters, x is each of the data samples in set D which is taken from the cluster, and μ is the associated center vector for cluster k. When the centers stop updating, the loss function should also be 0 or close to 0.

2 Strategy 1

The first strategy involved being given an already initiated set of centers based off our student ID and 300 sample data points. This part involved implementing the k-Means algorithm to update the given center values until the centers are not changing after a full iteration. This means that the given centers are as close to clusters in the data sets as possible.

I created the algorithm by creating a dictionary to hold the clusters that reset through each iteration. Then I iterated through each of the sample data points by calling a function that I created called mindistindex that takes in the cluster value, the sample of data, the input centers, and an index number. The function created a list of distances for each of the data points in the cluster

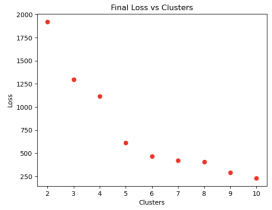

Figure 1: Final Loss Values vs Clusters

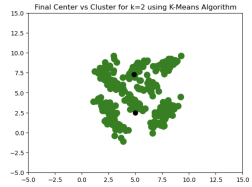

Figure 2: k=2 Final Center vs Final Cluster

to each data point in the sample , and then return the index of the minimum distance in the list. The index would be keyed in the cluster dictionary to the data point that was being iterated on. I set the algorithm to stop iterating at 1000 as suggested in class if the centers kept updating, but this was obviously nowhere near close to iterating that many times.

The idea behind the loss value is that the k-Means algorithm will try to minimize it as much as possible to track how close the center vectors are to the clusters. As we can see from the loss values, the larger the cluster value is, the closer the center values are to the middle of clusters. This makes sense, as with more center vectors to go around, they will naturally be able to find the middles of clusters, especially for larger amounts of data. Interestingly, the loss value takes a sharp downturn when the number of clusters is 5.

Below are the images of the final centers vs final clusters to compare. I decided to compare cluster 2 and cluster 10 as they were the smallest and largest and see how they affect the centroid placements.

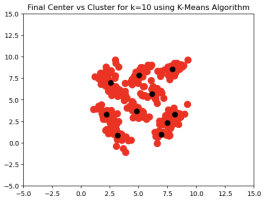

Figure 3: k=10 Final Center vs Final Cluster