CSC 466 Project Report

Our project focuses on comparing two unsupervised clustering algorithms: DBSCAN and K Means Clustering. We wanted to compare both using their average Silhouette Score. We generated a synthetic data set in R in the shape of a donut.

DBSCAN works by accepting a list of data points (normalized), an epsilon, and an int minimum number of points. It initializes an empty hashmap of clusters and an empty list of core points, then it searches through all points to see if they are a core point. It recursively classifies these core points by picking a random core point and classifying it and all its neighbors as a new cluster. Then it recurses on all neighbors that are core points. It stops when all points are either in a cluster or classified as noise. Because silhouette score isn't accurate with concave data like this, we had to resort to other types of filtering to find the best hyperparameters. We determined that with this data we wanted two groupings, so we determined values of minimum points and epsilon by trying different values for each and only considering the results that created two clusters. Our best Silhouette score for DBSCAN was 0.04246.

The Silhouette Score measures how well a data point fits within its assigned cluster. It works by comparing the average distance between that point and all other points in the same cluster, as well as the average minimum distance between that point and all other points in any other cluster. It doesn't work as well with concave clusters. Average scores closer to 1 indicate a good cluster assignment. The silhouette score especially doesn't work with our DBSCAN grouping for this data because points in the outer ring are pretty far away from each other, but all are relatively close to the other center grouping.

K Means Clustering: We implemented K-Means clustering in Java from scratch, so it starts off by grouping point objects into k clusters chosen by the hyperparameter tuning function that we created that calculates the silhouette score for and interval of k values. The core method is algo() which runs the clustering loop, and stops when either: a maximum number of iterations

is reached or centroids no longer move. The algorithm follows the same one found within the slides from class and was implemented nearly the same way.

We have the constructor KMeansCluster, which accepts a list of data points (normalized), desired number of clusters (k), and iteration limit, Initializes an empty list of clusters, Step 1: Random Initialization Randomly selects k unique points as starting centroids and Initializes Cluster objects with those centroids. Step 2: the main loop (algo), runs up to maxIter times, and stops early if centroids do not change.

For the results of our KMeansCluster we only used silhouette score as our evaluation metric on our synthetic dataset, and our results were shown in class and in the slides, where running our hyperparam tuning function found that k=6 gave the highest and most optimal silhouette score, leading the 6 clusters as seen in the graph from the slides.