Group31 Project 2 Report

Rylan Abraham, Yen Pham, Alexis Phu, Thong Nguyen

Project Description

In this project we read in information about points and clusters, then we clustered the points through two different methods. The first method is k-means clustering during which the number of clusters is defined in the input file. The second method is centroid linkage hierarchical clustering, in which every point is its own cluster and each level they cluster together reducing the number of clusters by half.

During the clustering we calculated the Dunn's index of the points to determine the largest value and output it with its k value and the number of clusters with its Dunn. More information will be shown in this report in each functionality's section.

Reading in and Storing Data and Configuration (Alexis Phu)

For this portion of the project we created two classes, the config class and the point class. The point class stores an x-value, a y-value, and an ID. The x and y coordinates are stored as floats to make the necessary calculations function properly, but the IDs were fine to be stored as integers for simplicity. The centroids, since they are given as points, are stored as points. Most of the data is stored in vectors because they are a great way to store large amounts of data that can be accessed at any index, will allocate memory automatically, and is a familiar concept for all of the group members. The config class stores the information from the configuration file, including the name of the points file, the min and max k-values, and a vector of the initial centroid point ID's. Also stored in the config class to make the requested output easier to retrieve, were the number of k-values to be tested and the number of points in the points file. Within the config class are the functions for reading in the configuration file and for reading in the points file.

The readConfig function takes in the file name as a parameter and reads the file information using file i/o, storing it in the appropriate variable. The readPoints function takes in the parameters of a vector of points to store all the points, another vector of points to store the centroids, and a 2D vector of points to store the centroids in separate vectors according to their k-value. The reason there are two different ways of storing the centroids is because the k-means clustering and hierarchical clustering parts of the project were designed to use the centroids differently, so there are now two ways of storing them.

The readPoints function opens the points file that was specified in the configuration file and reads in all of the points, populating the points vector that was passed in with all of the data. Then, it iterates through all of the points looking for the point ID's that match the centroid ID's, and populates the centroid vector that was passed in with the points that match. I made sure the centroid vector did not have repeats by comparing the ID of each point that matched a centroid ID to the elements already in the centroid vector and did not add them if they were already there. For the 2D vector of centroids, I used the previously populated centroid vector in order to give the information to the other vector. For example, if the first k-value was 2, the first 2 elements of the centroids vector would be copied into the first vector element of the 2D vector. In main, the user is prompted for the name of the configuration file that is passed into the readConfig function. The points vector and centroid vectors are declared in main and then passed into the readPoint function in order to be populated.

K-Means Clustering (Thong Nguyen)

When doing K-Means clustering, the K-Means class has to have the k-value which is the number of clusters there will be, centroids, and an empty vector for clustering. When looping through, as long as clustering is not finished, continue to loop in which points will be assigned to the closest centroid to them (if equal distance to 2 centroids, choose one with a lower ID) and assigned to the corresponding cluster. Do this until all points are clustered and calculate and sign the new X and Y for all the centroids. The loop will stop if and only if a cluster remains unchanged of its points after the clustering checks.

When looping through, the first loop check all points and assign them to their designated closest cluster by using simple Pathagrean Identity using the given X and Y coordinates. If two or more centroids distance to the point is equal, assign the point to the centroid with the lowest ID. After all points are assigned, calculate for the new coordinates for the centroid by adding all X of all points in a cluster dividing by the cluster size (i.e the number of points the cluster contains), do the same for Y. Assigned the new X and Y values to all centroids, then check to see if the previous centroids matches the new centroids, if not, the loop is not finished, if yes, the clustering has been completed.

Centroid Linkage Hierarchical Clustering (Yen)

For this portion of the project we calculated the Centroid Linkage Hierarchical clustering. First, we consider each given point is a cluster that has only one member, which is itself. Then, we calculated the distance between two clusters by calculating the distance between two centroid of two clusters using Euclidean formula. If the distance between two centroids of any pair of clusters is the smallest, we group them together using smaller group ID. These newly formed clusters are linked to each other to create bigger clusters. This process is iterated until all the objects in the original data set are linked together in a hierarchical tree.

To do my part, first, I created a first scenario when all the points have not been grouped yet and stand by itself as an individual cluster. I ran a loop to go through all the points in the file Points, then I pushed each point to a temporary cluster, considering each temporary cluster only contain one point, then I pushed each cluster into another vector which contains all the clusters for a k-value. At level 0, the number of clusters is as same as the number of points in the provided file.

For the next step, I ran a nested loop to calculate the distance between two centroids of two clusters. The first loop will first go over all the clusters, then the second loop will go over all the clusters with the indexes larger than the current cluster index from the first loop; this will prevent we from calculating the distance between the same centroid of the same cluster (which will always be 0). To know which clusters to group together, or to know which cluster pair that their distance is the smallest,outside the loop I created a minLink that saves two cluster indexes while those two clusters' distance to each other is the smallest and needs to be grouped together. Inside the loop, I created a temporary link that saves two cluster indexes to identify which clusters I am calculating the centroids. After I calculated the distance between two centroids of two clusters, I compared that distance to the smallest distance we have so far (smallest distance was set as a temporary variable); if the current calculated distance is less than the current smallest distance, then I reset the smallest distance and assign the temporary link to the minLink.

After getting the distance and the indexes of two clusters which have the smallest distance, I grouped those two clusters by “pushing” all the points of the cluster with larger index to the cluster with smaller index, then I deleted the cluster with larger index, and I also need to delete the centroid of the cluster with larger index (as the cluster itself has been gone).

After the cluster with larger index be deleted, I updated the centroid of the cluster with the smaller index, as it just welcomed all the points from the cluster with larger index. To do that I just ran over all the points in the cluster with smaller index to sum up all the X and Y coordinates, then outside the loop I took average of the sum to get the new centroid coordinates of X and Y. After getting a new centroid coordinates I set them as a new centroid for that cluster.

And after the cluster of larger index has been removed, I updated the number of clusters by assigning the updated size of the vector of clusters to a temporary numofClusters variable.

Dunn Index Calculation (Rylan)

For this portion of the project we calculated the Dunn's index for both clustering algorithms and analyzed the results. For the analysis we compared the values of the dunn for each k in k-means, and for every number of clusters in hierarchical to find the largest value.

To calculate the Dunn we first calculated the smallest distance between any two points in different clusters. Next we calculated the largest distance between any two points that are in the same cluster. To calculate the distance between two points we found the square root of the second x minus the first x squared plus the second y minus the first y squared (distance = sqrt$((x_2 - x_1)^2+(y_2 - y_1)^2)$). Lastly we divided the largest distance between any two points in the same cluster by the smallest cluster distance.

To find both the largest distance between any two points in the same cluster, and smallest distance between any two points in different cluster we used a vector of vector of point objects that contain the x, y, and ID values of the point, and which vector the point object was stored in determines the cluster it is part of. To find both distances we did a four level to go through each cluster, and compare the distance between each point in the same cluster or in different clusters depending on if we're finding the intercluster distance or intracluster distance.