Final Project Program Description

When approaching the problem of clustering, I knew that for each implementation I would need to have the same data to measure distance. I followed the guidelines given in the lecture on text mining. First, I went through each document and turned it into the bag of words representation and added it to a list. Then I went through each list and added unique words to a corpus. In the next step I created a list the size of the corpus for each document. I went through the document and placed the count of the words at the index of the dictionary in the matching index of the list representing the document. This creates the list of term frequencies. I then converted the term frequencies into weighted log frequencies by adding one plus the log base 10 of the count if the count is greater than 0.

The next major step was using the lists of weighted log frequencies to calculate the cosine similarity between two documents. I did this by using the method given in the lecture. I went through the list and summed the product of the values at each index. This provides a score of how similar the documents are. The higher the score, the more similar the documents. With this working, I was ready to approach k-means clustering.

In k-means clustering, k random midpoints are generated. Then, all the points are assigned to the closest midpoint. After this, a new median is calculated for each cluster, and points are reassigned. This is done repeatedly until either the midpoint does not change or barely changes. I made a method to choose k random points from the list of all points and made sure that the same point cannot be chosen more than once. I made another method that assigns a point to a cluster based on the closest midpoint of each cluster. I use the cosine similarity function to select the closest point.

I then wrote code that goes from 1 to k, and for each number, makes a cluster. Random midpoints are generated and then all the remaining points are assigned to the nearest cluster. The average of each cluster is then calculated with a method. The method makes a new average point by averaging each index of every point in the cluster and then adding that to the new point. The old midpoint and the new average are then compared. If they’re different, then the program enters a while loop until either they are equivalent or a set number of repetitions is reached. Within the while loop, points are reassigned, and the average is recalculated. Once the while loop ends, the points have successfully been assigned to clusters.

The dbscan implementation uses the same methods to read in data, calculate the average for a cluster, and calculate the cosine similarity. It differs in that it goes through every point and removes the noise points that don’t have a set number of neighbors within a predetermined range. Then it sees if the point has been assigned to a cluster. If it hasn’t, it increases the index of the current cluster and assigns it. Then it assigns any unassigned points in the set range to that cluster as well. I determined the set range that would work best by using the guidelines detailed in the presentation. I graphed the fourth nearest neighbor cosine similarity versus the points sorted according to the fourth nearest neighbor distance. From the graph I found the noise point threshold.

The last thing I needed to do was determine the most effective clustering implementation. I did this by using a form of SSE. For each cluster, I calculated the average and then added the cosine similarity of every point and the average to a final sum. The higher the final sum, the better the cluster. According to my results the implementation with the best performance was k-means. The best k values were 10, 12, 15, 7, and 5, respectively. All the k-means runs performed better than the dbscan run. On average, k-means performed 48 percent better than dbscan.