Austin Bailey

Dr. Claudia Pearce

CMSC 476

4 May 2018

Homework 5: Agglomerative Clustering

In this final assignment, the idea was to create an agglomerative clustering algorithm with the average link method. This method, also known as group average link method, is designed around the idea of the similarity correlating with the average. The average would be calculated from the similarity matrix, and it would be recalculated every time there was a merge. From there, it would be compared to different scores where the most similar would be the 2 clusters merged, followed up by a restructuring of the matrix.

In my script, I built off my previous assignments preprocessor. This meant that it needed to be modified to calculate the cosine similarity score of each document in the collection, rather than a query and a specific document. This required some tweaking but was overall already structured to calculate the cosine similarity score, so it was just a matter of changing it slightly. This was the idea behind how I implemented the agglomerative clustering algorithm. I would first calculate each similarity score between all documents and create a similarity matrix. Afterwards, the program would then go through and find the closest score to the average. It would then perform the merging after a series of conditional statements.

Elaborating on the first part of the algorithm, each document needed to be calculated with cosine similarity to each other. This was done by creating a nested loop. The outer loop would be the first document, and the inner loop would be the second document. Inside the inner loop, the similarity calculations would occur, and this would go on for all documents (1:2, 1:3, etc). When done, the first document counter would increment, and the same thing would occur. There was no need to calculate the diagonal of the matrix or the bottom triangular portion since the Sim(I, i) = 1.0 and Sim(I, j) = Sim(j, i). To create the matrix itself, I created a list called simMatrix, which was appended with 503 extra lists. This was to simulate a 2d array. For each cosine similarity score, there would be condition to check if the comparison was being done for the first time, which it would then append a 1 to that list (simulating the 1.0 on the diagonal). It would append to the list regularly, and then append to the opposite as well (sim(I,j) = sim(j, i)). It would do this for all 503 html files.

In the second half, the actual clustering algorithm was implemented. This algorithm was split up into three parts. The first part was to go through the similarity matrix and find the similarity score that was closest to the average calculated beforehand. The documents would be recorded along with the score. It would continually check to see if there exists a similarity score above 0.4, which it would use to end the overall loop if that was the case. Afterwards, the documents that were selected would be merged. I simulated "deleting" the higher entry by setting all of their values to 0 in the row and column, and would subtract the values from the top of the average as well as the bottom to recalculate. The final part consisted of concatenating a merge message with the two merged clusters. This would repeat until the condition of the 0.4 score was met.

According to the program, the two most similar documents were listed as 102 and 130. The most dissimilar documents were 1 and 13. The document that was closest to the corpus centroid was 371. As for efficiency, it could have been faster. The major problem of trying to implement the clustering algorithm is that the runtime for average link, max link, and min link are O(n^2). This means that by implementing this, the runtime slowed significantly. The runtime went from roughly 50 seconds last assignment to around 150 seconds for this assignment. This was mostly due to the construction of the similarity matrix. If the matrix was outputted to the disk, and it was read from beforehand then the runtime for the program would be significantly better. The problem is that the preprocessing and the construction of the matrix slowed the program down significantly. Perhaps by using different data structures more effectively, the runtime could be improved.