Information Retrieval (CMSC 476/676)

Homework 5 – Report

Done by – Anushka Dhekne (CMSC VD19739)

The objective of the assignment is to perform analysis on the 503 HTML documents corpus for performing document merging and clustering. The homework includes constructing a similarity matrix, in which the entry (i,j) is the similarity between documents i and j computed using cosine similarity. A document is perfectly similar to itself if the entries on the main diagonal are all 1. Similarity is all symmetric if sim(i,j) = sim(j,i), that is, the similarity matrix is in the upper triangular form. For performing the clustering algorithm, the similarity matrix will be needed. The hierarchical agglomerative clustering algorithm merges the 2 objects in consideration if their centroids are closer to each other than the centroids of any other pair of objects. An object could be a single document or an existing cluster. If a document is not yet a part of any cluster, then it acts as a centroid of its own singleton cluster. Here, we assume that the intersection between 2 clusters is null. The hierarchical agglomerative clustering uses the group average linkage method, which will be explained in detail in this report. The clustering of objects stops when no 2 clusters (or documents) have similarity greater than a threshold value - 0.4. This report also discusses which is the most similar pair of documents and which is the most dissimilar pair of documents as well as which document is closest to the calculated centroid.

I have used Python programming language to complete this assignment and Visual Studio Code as the code editor. This report discusses the implementation, the working of each function, the algorithm used, and explains the output and the files used for input. In this homework, I have used the output files that I got from homework 2, which are the .wts files. Each of the original .html files has an individual .wts file of its own, which contains all the unique terms or words and their respective term weights calculated using the TF-IDF (term frequency-inverted document frequency) algorithm. The report also includes screenshots of the output.

The first function – 'reading_wts_files' basically reads all the .wts files containing the term and its weight in that document, calculated using the tf-idf algorithm. The function has a loop that iterates over the files in .wts format in the specified folder and reads the contents of each file. For each of the file, the function extracts the word and the tf-idf weight associated with that term. The function ensures the validity of each file's format as well as handles any exceptions met while reading the files and finally, returns the weights dictionary.

The next function – 'computing_similarity_matrix' constructs a similarity matrix based on the cosine similarity between the tf-idf weights of the words in the different .wts document. It considers each pair of words and calculates their cosine similarity using the tf-idf weights stored in the 'weights' dictionary from the previous function. This ensures that each element of the matrix represents a similarity score between 2 of the documents. This similarity matrix is then used for the clustering algorithm.

The function – 'cosine_similarity' is called in the above-given function for computing the similarity matrix. This considers the tf-idf weights for word1 and word2 from the document by retrieving the respective weights of these 2 words from the 'weights' dictionary and calculates the cosine similarity between those weights using a standard formula. This includes a logic for computing the core similarity which is essential for constructing a similarity matrix. This also allows an efficient clustering analysis.

The next function – 'algo_hierarchical_clustering' essentially performs the hierarchical agglomerative clustering algorithm. The clustering criterion used in this is average linkage. The algorithm iteratively merges clusters based on their average linkage distance until a specified threshold value is reached. For this code, the threshold value to be considered is 0.4. That is, the clusters are merged until the average linkage value between them is at least 0.4. This function uses the nested loop structure for the task of comparison of similarity scores between the clusters and then decides the optimal merges based on it. The function progressively merges these clusters with the closest average linkage values, while considering the threshold value constraint at the same time.

The next function – 'cluster_distance_avg_linkage' computes the average linkage between the 2 merged clusters of documents. This method calculates the average similarity score between all the pairs of documents which are present in these 2 clusters. This provides a clear idea of the overall similarity between the 2 clusters of documents. This is the basis of the hierarchical clustering algorithm.

The next function – 'print_merges' is used to print all the cluster merges done through the hierarchical clustering algorithm and function along with their corresponding similarity scores. It is also used to find the document pairs having the most similar (highest) score and the most dissimilar (lowest) score.

The next function – 'most_similar_dissimilar_pairs' is used to return the most similar and most dissimilar pairs to the above-given function – 'print_merges'. This function identifies the most similar and most dissimilar pairs of documents based on the similarity matrix which was derived from the tf-idf term weights. The function traverses the upper triangular matrix of the similarity matrix and then it compares the cosine similarity scores between the document pairs. It then identifies the most similar and the most dissimilar pairs of documents based on this comparison of similarity scores.

The next function – 'closest_to_centroid' finds the documents which is closest to the centroid of the dataset. This computes the average similarity of each of the documents to all other documents in the dataset. It then identifies the document with the maximum average value of the cosine similarity score as the document closest to the centroid of the dataset. Finding the centroid point in the cluster of documents helps to analyze the central reference point within the entire cluster of documents as well as in characterizing the document clusters based on their proximity and similarity to the dataset's central document.

Finally, the main function of the program brings together all these functions and their components to successfully run it and give the desired output. The main function is also the section where the file folder containing all the .wts files necessary to build the similarity matrix is accessed. Overall, it serves as the part of the program where data processing, similarity matrix construction, document clustering and displaying of the results takes place.

In the given program, the clusters are initially names using the indices of each of the individual documents. That is, each document is initially treated as an individual cluster and the clusters dictionary is used to map each document index to a list containing only that index. Eventually, during the hierarchical clustering step, the clusters are merged together based on their similarity scores until the threshold value (0.4) is attained. The merged clusters are then assigned the name of one the original individual clusters. Typically, the name of the cluster having a lower index value is given to the merged cluster. This method of naming is quite simple and consistent throughout the entire clustering process.

The implementation of the similarity matrix uses the numpy python library as the matrix is implemented as a two-dimensional numpy array. This matrix stores the pair-wise cosine similarity scores between each pair of documents. The matrix is initialized with zeros using the numpy function and then these zeros are

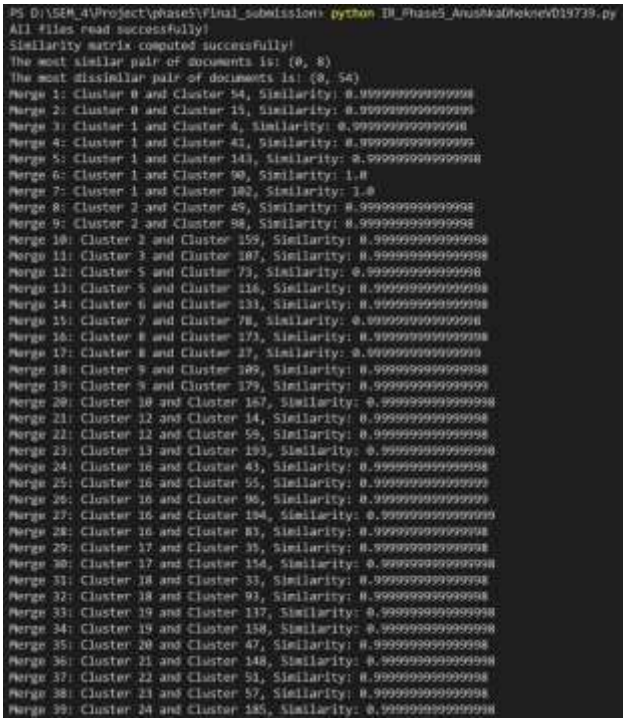
replaced by the calculated similarity scores. Each cell in the similarity matrix represents the similarity between the pair of documents and the diagonal cells usually show a similarity score of 1.

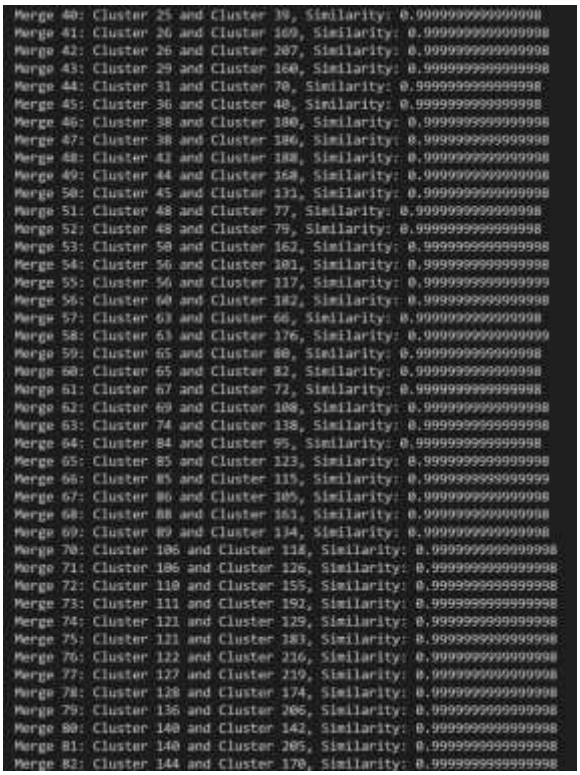
The main data structures used in the program are dictionaries to store the tf-idf weights and to store the clusters of documents during the hierarchical clustering algorithm. The clustering dictionary's key-value pair represents a cluster, where the key is the index, and the value is a list of document indices belonging to that cluster. In the weights dictionary, the key represents the word in the document and the value represents the tf-idf weight of that word. Another major data structure considered in this program is the 'merges' list which stores the information about the merges performed during the clustering step. Each element in the list consists of document indices from the merged clusters and the similarity scores between these documents.

The most similar pair of documents, having the highest similarity score, as determined by the cosine similarity scores in the similarity matrix computed by the program is that of document 0 (001.wts) and document 8 (009.wts). The most dissimilar pair of documents, having the lowest similarity score, as determined by the cosine similarity scores in the similarity matrix computed by the program is that of document 0 (001.wts) and document 54 (055.wts). The document closest to the centroid of the cluster of documents is found based on the average similarity of an individual document to all other documents in the cluster of documents. The document index with the maximum average similarity score is then said to be the closest document to the centroid. In the program, the first document (001.wts) is found to be the document that is closest to the centroid of the dataset.

Overall, the code is quite complex and requires a variety of parameters to compute everything. The overall time complexity of the code is O(n^3), where 'n' is the number of documents. The overall space complexity of the code is O(n^2), where 'n' is the number of documents.

The screenshots of the output are given below.





```
Merge 83: Cluster 149 and Cluster 203, Similarity: 0.9999999999999998
Merge 84: Cluster 189 and Cluster 201, Similarity: 0.9999999999999998
Merge 85: Cluster 0 and Cluster 157, Similarity: 1.0
Merge 86: Cluster 3 and Cluster 104, Similarity: 1.0
Merge 87: Cluster 5 and Cluster 76, Similarity: 1.0
Merge 88: Cluster 6 and Cluster 103, Similarity: 0.9999999999999999
Merge 89: Cluster 7 and Cluster 177, Similarity: 1.0
Merge 90: Cluster 10 and Cluster 61, Similarity: 1.0
Merge 91: Cluster 13 and Cluster 202, Similarity: 1.0
Merge 92: Cluster 17 and Cluster 24, Similarity: 1.0
Merge 93: Cluster 20 and Cluster 100, Similarity: 1.0
Merge 94: Cluster 21 and Cluster 151, Similarity: 1.0
Merge 95: Cluster 22 and Cluster 145, Similarity: 1.0
Merge 96: Cluster 23 and Cluster 171, Similarity: 0.9999999999999998
Merge 97: Cluster 25 and Cluster 64, Similarity: 1.0
Merge 98: Cluster 29 and Cluster 132, Similarity: 1.0
Merge 99: Cluster 31 and Cluster 8, Similarity: 1.0
Merge 100: Cluster 42 and Cluster 156, Similarity: 1.0
```

```
The document which is closest to the centroid is: 0
PS D:\SEM_4\Project\phase5\final_submission>
```

The code can be optimized to implement the required tasks more efficiently and with lesser time and space complexities. Tasks like the construction of the similarity matrix and the algorithm of hierarchical clustering can be computed in parallel to improve the efficiency of the code. Similarity matrices can also be represented as sparse matrices to optimize memory utilization and the overall system's performance, as well as to handle scalability efficiently.