PROGRAMMING ASSIGNMENT PHASE 5: HIERARCHICAL AGGLOMERATIVE CLUSTERING

Rishabh Sachdeva (rishabs1@umbc.edu, UI73138)

INSTALLATION AND PROGRAM EXECUTION:

Programming Language: Java 1.8

Prerequisite Installations:

Java 1.8,

Execution:

Program can be executed either via executable jar or manually creating class file and run.

Commands:

To compile: javac informationRetrieval.assignment1.QueryExecution.java

To run: java informationRetrieval.assignment1.DocumentClustering

Output: Top 10 documents with corresponding scores are displayed on console.

APPROACH:

The approach is based on **Simple Hierarchical Agglomerative Clustering** strategy**. I have not used any external library for creating clusters but written my own code**. I referred Simple HAC algorithm from Manning's book on Information Retrieval (Chapter 17). First step is to create Similarity Matrix. Similarity scores between two documents are built using **cosine similarity**. BM25 scores evaluated in previous assignment, are utilized to build similarity scores. Each token is compared among all possible pair of documents, and they contribute towards final cosine similarity of each possible document pair.

Let d1 and d2 be two documents, then:

Cosine_similiarity(d1, d2) = d1.d2/(|d1| |d2|)

Each token and its score is picked up from output of previous assignment (BM25 scores). For example, d1 represents all the tokens and their corresponding bm25 scores occurred in the document 1.html.

Once the similarity matrix is built, clustering algorithms starts and continues till the stopping condition is reached (no pair has similarity >0.4).

Simple HAC Algorithm:

1. Build the Similarity Matrix of the whole corpus. (See below for detailed explanation)
    a. Iterate over all possible pair of documents in pair.
    b. Evaluate cosine similarity of each pair and store them.
2. Run Clustering Algorithm
    a. Initiate with all documents as individual clusters.

   b. Find out the most similar clusters, i.e. pair with highest similarity score (say, cluster_x and cluster_y).
   c. Merge the clusters – cluster_x and cluster_y, into a new cluster- cluster_xy (**According to Single-Linkage Strategy)**
   d. Remove cluster – cluster_x and cluster_y from the stored collection.
   e. Add a new cluster cluster_xy into the collection.
   f. Store the newly formed cluster in a data structure (ArrayList).
   g. Run the clustering algorithm (steps b to f) tills stopping is reached.

3. Display all the clusters created (stored in array list) order-wise.

<u>IMPORTANT DATA STRUCTURES USED:</u>

Similarity Matrix:

Similarity Matrix is maintained in the form of **HashMap**. Key of this Map is name of cluster (String), and its value is a **Priority Queue** ordered in descending order of the similarity scores. An inner class is created to store document id and its similarity score corresponding to the cluster (key of map). A custom Comparator is built to maintain the required order of scores in a priority queue.

To keep it simple, a cluster is named as per documents in the cluster. It's a comma separated string with the corresponding documents. Like, cluster containing documents 102, 105 and 110 would be:

102.html, 105.html, 110.html.

Code Snippet:

```
HashMap<String, PriorityQueue<DocAndScore>> pqsimilarityMatrix = new HashMap<>();


static class DocAndScore{

        String docId;

        double score;


        public DocAndScore(String docId, double score) {

                this.docId = docId;

                this.score=score;

        }
```

Building Similarity Matrix:

Two for loops are used for building similarity matrix, to find out each pair and compute similarity. One document is picked (say 1.html). Its similarity is evaluated with all the other documents in corpus. Their scores are stored in a priority queue ordered by decreasing sequence of corresponding score. A HashMap described above is used as data structure.

So, in this case key of the map would be 1.html. And the value would be priority queue containing all other documents and corresponding similarity scores.

Let's say 10.html is most similar to 1.html, 15.html is second most similar, and 30.html is least similar. Then Map would look like this.

Key = 1.html

Value = (10.html, 0.93); (15.html,0.89) ……………………………………………………………….(30.html, 0.02)

Code Snippet:

```
for(int i=0;i<propListing.length;i++) {

        List<DocAndScore> list = new ArrayList<>();

        PriorityQueue<DocAndScore> pq = new PriorityQueue<DocAndScore>(502, new
ScoreComparator());

        for(int j=i+1;j<propListing.length;j++) {

                double score = calculateSimilarity(propListing[i],propListing[j]);

                pq.add(new DocAndScore(propListing[j].getName(),score));

        }

        pqsimilarityMatrix.put(propListing[i].getName(), pq);

}
```

PROGRAMMING ASSIGNMENT PHASE 5: HIERARCHICAL AGGLOMERATIVE CLUSTERING

Rishabh Sachdeva (rishabs1@umbc.edu, UI73138)

Finding Most Similar and Dissimilar Documents:

Initially each document forms an individual cluster. First two documents that merge and form a cluster would be the most similar ones. As the output suggests, **417.html and 420.html** are the most similar ones. It turns out that these documents are actually same and contain same terms.

To find most dissimilar documents, the cosine similarity of the two should be minimum across corpus. Following approach is used to find them:

1. Iterate over key set of Priority Queue Similarity Matrix. Maintain the global minimum score and documents.
2. Find the Priority Queue corresponding to the current key.
3. Now find the cluster which has minimum similarity in this queue. As Priority Queue is ordered according to similarity scores, the last element would be the one with lowest score. So, in simple words, get the last element of the current queue.
4. Compare the similarity queue with global minimum, if found lesser, update the global minimum documents.
5. Repeat steps 2 to 4 for each key in step 1.
6. Global minimum html docs would be the most dissimilar.

In the corpus, the most dissimilar documents are**: 484.html and 99.html**.There could be (in fact there are) more such pairs which are dissimilar whose similarity scores are equal with the mentioned pair (0 score). This are just one of these dissimilar pairs.

Output Format:

The program prints the clusters formed in an order line-by-line. The name of new clusters formed will be comma separated string of document names present in the cluster. Output(100 line) is present in different file in the zip.

Example: Following are the first few lines of my output. See comments in line 16 of below output for clarification

1. 1st cluster: 417.html; 2nd cluster: 420.html // first cluster formed
2. 1st cluster: 102.html; 2nd cluster: 130.html // second cluster formed
3. 1st cluster: 422.html; 2nd cluster: 424.html
4. 1st cluster: 421.html; 2nd cluster: 423.html
5. 1st cluster: 416.html; 2nd cluster: 418.html
6. 1st cluster: 412.html; 2nd cluster: 414.html
7. 1st cluster: 403.html; 2nd cluster: 405.html

8. 1st cluster: 404.html; 2nd cluster: 406.html
9. 1st cluster: 413.html; 2nd cluster: 415.html
10. 1st cluster: 399.html; 2nd cluster: 400.html
11. 1st cluster: 429.html; 2nd cluster: 431.html
12. 1st cluster: 407.html; 2nd cluster: 410.html
13. 1st cluster: 409.html; 2nd cluster: 411.html
14. 1st cluster: 438.html; 2nd cluster: 442.html
15. 1st cluster: 430.html; 2nd cluster: 432.html
16. 1st cluster: 413.html, 415.html; 2nd cluster: 417.html, 420.html ***// new cluster is formed joining //2 individual clusters which are (413,415 formed in line no. 9) and (417,420 formed in line 1) //and merging occurs via single linkage.***