Information Retrieval Project Phase 5

Mihir Kelkar

CMSC 676 University of Maryland, Baltimore County

Recap of the tasks performed in the first four phases

First Phase *:* In the first phase of the project, we used an HTML parser to parse the entire corpus. We were able to retrieve text from all of those documents. Stop word processing and a certain extent of stemming was performed on the retrieved text . The remaining terms were now called as index terms. The frequencies of index terms was calculated per document as well as throughout the corpus as a whole.

Second Phase : In the second phase, we used the term frequencies we calculated in the first phase to calculate the tf-idf weights of all index terms present in the corpus.

Third Phase : In the third phase of the project, we used the tf-idf weights to create a dictionary and postings file. The dictionary and postings file put together would be equivalent to an inverted index. The dictionary file contained the index term, the number of documents which contained the index term and the location of the first record of that term in the index file.

Fourth Phase: In the fourth phase of the assignment, we created a retrieval engine which could be used to query the inverted file index we created in the previous phase. The retrieval engine could take in multiple words as inputs , also accept corresponding weights assigned to input terms and then retrieve and rank relevant documents.

Task Performed in the Fifth Phase of the Project

In this phase of the project, we intend to cluster documents which are similar into one single entity and then compare it with the rest of the corpus. Initially each document is a cluster of its own. The document similarity score is calculated using cosine similarity as a metric. The two documents which are most similar based on this metric are then clustered together. The cluster's similarity score is then calculated as per the hierarchical agglomerative clustering formula. Individual entries for all clustered documents are deleted from the matrix and the entry for the cluster as a single entity is added.

This process is repeated several times, cosine similarity matrix is calculated again and most similar clusters are merged together to form new clusters. As a matter of choice, I repeat the entire process until there are no original documents remaining as individual clusters and then stop the process of clustering.

Brief Introduction to Hierarchical Agglomerative Clustering

Hierarchical Agglomerative Clustering treats each document as a singleton cluster at the outset and then successively merge (or *agglomerate*) pairs of clusters until all clusters have been merged into a single cluster that contains all documents. The algorithm is as follows:

```
SIMPLEHAC(d_1, ..., d_N)
 1  for n ← 1 to N
 2  do for i ← 1 to N
 3     do C[n][i] ← SIM(d_n, d_i)
 4     I[n] ← 1  (keeps track of active clusters)
 5  A ← []  (assembles clustering as a sequence of merges)
 6  for k ← 1 to N − 1
 7  do ⟨i, m⟩ ← arg max_{⟨i,m⟩: i≠m ∧ I[i]=1 ∧ I[m]=1} C[i][m]
 8     A.APPEND(⟨i, m⟩)  (store merge)
 9     for j ← 1 to N
10     do C[i][j] ← SIM(i, m, j)
11        C[j][i] ← SIM(i, m, j)
12     I[m] ← 0  (deactivate cluster)
13  return A
```

The variant of the HAC Algorithm you use depends upon the way you choose to calculate arg Max. In the single link variant of the HAC, the value selected is simply the higher of the two similarity scores. In complete link variant its the lesser of the two similarity scores which is selected. In the group average variant, its calculated by considering the number of total original documents and the documents in the cluster under consideration.

Implementation Details for Project Phase 5.

The first task was to calculate the cosine similarity matrix for the entire corpus. For this, I decided to read the dictionary and postings file into two separate hash tables. I used both of these hash tables to create vectors of index words for all the documents in the corpus. I stored all of these vectors in a separate hash table indexed by the document id and having index term vector as values. I used these vectors to calculate the similarity matrix, the matrix was stored as a list of list.

As the HAC Algorithm above illustrates, I iterate over the similarity matrix to gather the two most similar documents / clusters. I take into account the fact that the matrix is symmetric along the diagonal , so I optimize the iteration to make sure that it just traverses the lower triangular part of

the matrix. Thus this optimization reduces the traversal time complexity to O(n / 2) from O(n). This is a significant reduction since this is nested inside another loop.

Now that I have identified the two most similar documents (clusters), I iterate over each row in the matrix and replace every element's similarity score against either of the identified documents with the maximum of (sim (given doc, id_doc $_A$), sim (given doc, id_doc$_B$)) considering that id_doc$_A$ and id_doc$_B$ are the two document identified as the most similar clusters in this iteration. I calculate the average similarity score for the cluster. I delete entries for one of the identified documents and replace the other's entries with the values for the cluster. The new entered values are representative of similarity scores for both merged documents / clusters.

Considering the entire computation process, the time complexity of the algorithm is asymptotically $O(n^3)$. According to the calculations, the two documents which are merged in the first iteration are 434.html and 435.html.Both of these documents are Web Access Statistics and might actually have exactly identical contents once stop word processing is complete. The documents which are clustered at the end are 110.html and 357.html. So, essentially, these two documents might probably be the two most dissimilar documents. **The output only displays the first 100 lines of the clustering process.**

The cluster names included the names of the all the html documents which were merged so far. for eg. X.html and Y.html clustered together would give us X.html Y.html.