TF-IDF BASED RANKED RETRIEVAL

CS F469 ASSIGNMENT 1

* TARUN RAHEJA
* 2015A7PS01016H

Group members : Tarun Raheja (solo)

MAJOR DATA STRUCTURES USED

1. For indexing :

A dictionary called *inv\_index* is used which stores all words occurring in all documents as keys and a sorted list of DocIDs in which the tokens occur.

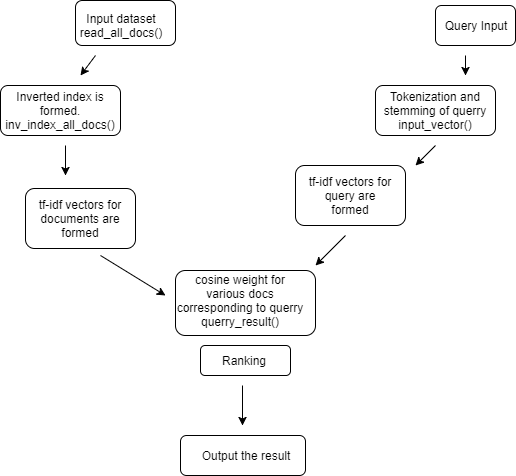
1. Vector space of all documents :

A list of dictionaries called *all\_doc\_vectors* is used and each element of this list is a dictionary, as stated, and contains tokenized, stemmed words as keys and TF-IDF values (finally) as values for the keys.

1. Document Frequency of all the words :

A dictionary called *doc\_freq* is used which simply has all the tokens as keys and document frequency of each word as values.

ARCHITECTURE DESIGN



FUNCTIONS IMPLEMENTED AND BRIEF NOTES ON CODE EXECUTION

1. First the inverted index for each word in the document is created in the form of a list of vectors. The following functions are used :
2. *read\_all\_docs*( ) : This function takes no arguments and adds vectors for each document to the *all\_doc\_vectors* variable.
3. *inv\_index\_all\_docs*( ) : This function takes no arguments and creates the inverted index for all the documents and stores it in the *inv\_index* variable.
4. *tf\_idf\_vectorize*( ) : This function takes no arguments and changes the frequency vectors in *all\_doc\_vectors* to TF-IDF vectors.
5. *stem\_and\_tokenize*( ) : This function takes no argument and returns a list of stemmed and tokenized words.
6. Next the query is taken as input from the user and the following functions are used, as evident from the code :
7. *input\_vector*( ) : This function takes the query as input and returns a term : frequency vector from the query in the form of a dictionary.
8. *tf\_idf\_query*( ) : This function takes the returned vector of the above function as an argument and returns the TF-IDF from of the vector, normalized by length.
9. *query\_result*( ) : This function takes the return value of the above function as input and returns the final documents list in sorted order in terms of their cosine similarity.

RUNNING TIME FOR PREPROCESSING

A corpus consisting of nearly 15,000 news articles taken from the Usenet newsgroups collection was used, and it only features the subject and the body of each message. The same has been uploaded in a zipped file called *Files.zip* on Google Drive at the following link :

<https://drive.google.com/file/d/0B5IBLP79305iMUs4OFpURkYyS00/view?usp=sharing>

On this dataset, it takes about **250 seconds** for the code to preprocess the entire dataset. For an average query, it takes around 0.1 - 1 seconds to return search results.

It is not possible to mention Precision and Recall for this search engine, since it employs the Vector Space Model for retrieval.