Exercise 1 – Basic Search System

Group09
Samira Mahdi Zadeh- 1329639
Thiago Carneiro-1028762

Index

The first part of the exercise is to generate the index. To create the bag of words index, first we need to access all the documents. We find files inside each folder recursively and then we read each line of the document and remove characters like underlines and numbers. The second step is to do case folding, removing the stop words and do the stemming based on user's preference. After normalizing the term, we use another function with the term and the docID where we will put then in a HashMap. The HashMap contains the inverted index which is the term as the map key, associated with a list of the docIDs in which the term appears in and the frequency that this term appears in the document. In the end we store the inverted index in a file, e.g. bow_InvertedIndex.txt, which has the following structure:
Term1 <DocID1,frequency><DocID2,frequency>….
Term2 <DocID3,frequency>…

We also keep the length of each document (=number of tokens each document has) in a file called docsLength.txt.

We will use these files to retrieve data in searching phase in a more efficient way. (since once the index file is created and stored on file system, it can be read from storage for searching phase and there is no need for recalculating indexes during searching)

To create the bi gram Index the first process is the same, to look for each document and to normalize the 2-word tokens. But there is no removing of the stop-words because if in a term there is a stop word and we remove it, there will be only one term. At the end of the process we will have the bg_InvertedIndex.txt which has the same structure as described above, but there will be two terms and a list with docID and the frequency that these two terms appear. And there will be also a docsLength.txt which contains the docID and the number of tokens in this document.

Vocabulary

The vocabulary is normalized accordingly by removing the stop words which we have in a list of strings i.e. "a", "able", "about", "across", "after", "all", "almost" ,"also", "am", "among", "an", "and", "any", "are", "as"," at", "be".
After we get the term from the document and based on user preference we put it in the lowercase and check if this document is not in this list, if it is not then we use the term. The stemming process is made after removing the stopword. (we

do stop word removing first, since our stop word list is not stemmed). For the stemming we use the popular Porter stemmer algorithm.
We also create a NormOptions.txt that contains the user's preference for normalization of vocabulary. It just contains true or false for each parameter: case folding, removing the stop words and stemming.

Search

The search works by choosing the index type user wants to use and typing the topicX the user wishes to search. The X is the number of the topic that can go from 1 to 20.
After the user selects the index, the search reads the inverted index file created by the Indexing phase, and creates the HashMap like in the index creating process.
The next step is to wait for the user to type the topic#. When he/she does it, we read the topic and tokenize it using the same processes like the index (We also read the file called NormOptions.txt which keeps the normalization option which were used for creating index. We then use these options to exactly perform the same normalization on the topic that we had used for our index creation phase.) then we create a file topicX.txt with the term and the frequency that this term appear in the document. That is the query that we use to create the score between the topic and all other documents.
For the document ranking we've used cosine score technique.(described in page 60 of indexing and scoring lecture)
We calculate the score by checking all the terms in the Topic and if this term exist in the document we calculate the total score of each document by summing up for each term in the query, the multiplication of frequency of this term in the document and the frequency of this term in the query divided by document length which we had calculated it and stored it in docsLength.txt . The result (DocID,score) is saved in a HashMap. In the end we sort this list in descending order based on the document scores and print the top 100 in the format specified in assignment to the user .

Run the Prototype

To run the prototype, you need to place the 20_newsgroups_subset and topics folder in the parent directory and then run the main.java class to make the index file and after that run search.java class to do the search.