IR Assignment 1 Report

Aryan GD Singh: 2019459
Shabeg Singh Gill: 2019388

Q1.

We first read the dataset and then start preprocessing

Preprocessing:

1. Text is converted to lowercase
2. Word tokenization is performed using NLTK word_tokenize
3. Stop words are ignored
4. Punctuation is removed, only alphanumeric tokens now remain
5. Stemming is performed using PorterStemmer
6. Blank space tokens are ignored/removed

Methodology:

- Unigram inverted index is built using python dictionary with key as the word tokens, and value being a list with 2 entries. The 1st entry is the document frequency, ie, the number of documents that contain the given word, and the 2nd entry is a list of document ids that contain the word.
- AND and OR functionality is implemented using while loops that compare the 2 lists taking individual values. Each iteration of the loop that compares the list values is counted as a comparison.
- NOT is implemented by creating a list of all document ids, and then removing the ones that are in the input list.
- OR NOT and AND NOT are handled using a combination of the above, for example, x AND NOT y is solved as x AND (NOT y).
- Input is taken as follows :-
    - Query is input as string
    - Operations are input separated by commas(,).

- Preprocessing is performed on query, and then the operations are performed left to right.
- Output is the number of documents matched, the number of comparisons performed, and the list of retrieved document names.

Assumptions:

1. 5 files were giving errors while reading, these are ignored as the total number of files is much greater(>1100).
2. Operations are performed in left to right order
3. Input is assumed to be in correct format

Q2.

We first read the dataset and then start preprocessing

Preprocessing:

Preprocessing is as specified in the assignment pdf, ie, same as Q1 without stemming.

1. Text is converted to lowercase
2. Word tokenization is performed using NLTK word_tokenize
3. Stop words are ignored
4. Punctuation is removed, only alphanumeric tokens now remain
5. Blank space tokens are ignored/removed

Methodology:

- Positional index is built using python dictionary with key as the word tokens, and value being a list with 2 entries. The 1st entry is the term frequency, ie, the number of times the given word appears in the dataset. The 2nd entry is a dictionary with keys being document ids and values being lists that contain the positions the word appears at in that document.
- Phrase queries are checked as follows:-
    - First we check if all the input query tokens are in the index.

    - For each token, we retrieve the list of documents that contain the term.
    - These lists are then intersected to get the list of documents that contain all the query tokens.
    - For each such document, we get the positions of the tokens in the documents, and then reduce their value by the index of the token. For example -

      Query: Good morning sir
      Tokens: ‘good’, ‘morning’, ‘sir’

      Each value in the position list for ‘good’ is reduced by 0.
      Each value in the position list for ‘morning’ is reduced by 1.
      Each value in the position list for ‘sir’ is reduced by 2.

    - Now if any document contained the tokens in the right order, ie, contained the phrase, then position lists for all the terms should have the same value for that occurrence of the phrase.
    - The position lists are now intersected, and if the list is not empty, that means the document contained the phrase. We add this document’s id to a list and then move on to the next document.

- Input is taken as a single string, ie, the given phrase.
- Preprocessing is performed on the query, and then the phrase is searched in the dataset.
- Output is the number of documents matched, and the list of retrieved document names.

Assumptions:

1. 5 files were giving errors while reading, these are ignored as the total number of files is much greater(>1100).
2. Input is assumed to be in correct format