Information Retrieval

Boolean Query and Inverted Index

Overview

In this assignment, you have to collect some online new pages to collect text data. You can directly collect sentences from the pages or collect the sentences from the source of pages by filtering out unnecessary texts. Prepare an input text file consisting of Doc IDs and sentences considering sentence as document (As given in bellow example). Based on this input text file, complete following tasks. Your implementation should be based only on Python3.

Example:

```
113257   COVID-19 and diabetes mellitus: implications for prognosis and clinical management
156757   Seroprevalence of Rodent Pathogens in Wild Rats from the Island of St. Kitts, West Indies
50439    Prevalence and genetic diversity analysis of human coronaviruses among cross-border children
```

Task 1: Build Your Own Inverted Index

Implement a pipeline that takes as input the given file, and returns an inverted index.

1. Extract the document ID and document from each line.
2. Perform a series of preprocessing on the document:
    a. Convert document text to lowercase
    b. Remove special characters. Only alphabets, numbers, and whitespaces should be present in the document.
    c. Remove excess whitespaces. There should be only 1 white space between tokens, and no whitespace at the starting or end of the document.
    d. Tokenize the document into terms using a white space tokenizer.
    e. Remove stop words from the document.
    f. Perform Porter's stemming on the tokens.
    g. Sample preprocessing output

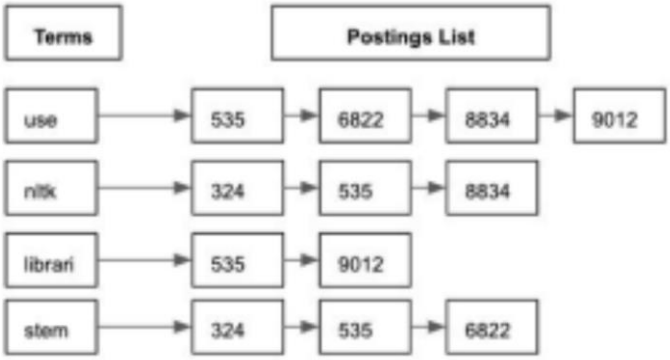
```
Input ->  535   You can  use NLTK(library)   for stemming   !

Doc id ->  535
Doc text ->  You can  use NLTK(library)   for stemming   !

Doc text post processing ->  you can use nltk library for stemming

Doc tokens post whitespace tokenizing ->  ['you', 'can', 'use', 'nltk', 'library', 'for', 'stemming']
Doc tokens post stopword removal ->  ['use', 'nltk', 'library', 'stemming']
Doc tokens post stemming ->  ['use', 'nltk', 'librari', 'stem']
```

3. For each token, create a postings list. Postings list must be stored as **linked lists**. Postings of each term should be ordered by increasing document ids.



4 . Add tf-idf scores to each element in the postings list. The tf-idf should be calculated using the following formulas:

Tf = (freq of token in a doc after pre-processing / total tokens in the doc after pre-processing)

Idf = (total num docs / length of postings list)

tf-idf = Tf*Idf

Task 2: Boolean Query Processing

You are required to implement the following methods and provide the results for a set of Boolean "AND" queries **on your own index**. Frame 5 queries relevant to the topic on which you have collected sentences.

Corpus

```
1       hello world
2       hello. hi world, world. hello jack
3       do i even know you?
4       is it a common known thing?
5       hello there. are you jack?
6       lets meet at the cafe
7       i am going to a swim meet this monday
8       swimming is good for health
9       hello. are you going for a swim?
10      just random text
11      more random text, and more text
12      random text randomly strikes back
```

Queries

```
hello world
hello swimming
swimming going
random swimming
```

1. Query Processing

Given a user query, the first step must be **preprocessing the query** using the same document preprocessing steps.

a. Convert query to lowercase

b. Remove special characters from query

c. Remove excess whitespaces. There should be only 1 white space between query tokens, and no whitespace at the starting or ending of the query.

d. Tokenize the query into terms using a white space tokenizer. e. Remove stop words from the query.

f. Perform Porter's stemming on the query tokens. g. Sample query processing:

```
Input query ->  Which library to use for stemming?
Query post processing ->  which library to use for stemming

Doc tokens post whitespace tokenizing ->  ['which', 'library', 'to', 'use', 'for', 'stemming']
Doc tokens post stopword removal ->  ['library', 'use', 'stemming']
Doc tokens post stemming ->  ['librari', 'use', 'stem']
```

2. Get postings lists

This method retrieves the postings lists for each of the given query terms. The input of this method will be a set of terms: term0, term1,..., termN. It should output the **document ID wise sorted** postings list for each term. Below is a sample format of the same:

```
'postingsList': {
    'go': [7, 9],
    'hello': [1, 2, 5, 9],
    'random': [10, 11, 12],
    'swim': [7, 8, 9],
    'world': [1, 2]
},
```

3. Get postings lists with skip pointers (OPTIONAL)

This method retrieves the reachable documents using skip pointers for each of the given query terms. The input of this method will be a set of terms: term0, term1,..., termN. It should output the **document ID wise sorted** postings list, which can be accessed by using the skip pointers. Return empty list in case of no skip pointer for the postings list. Below is a sample format of the same:

```
'postingsListSkip': {
    'go': [],
    'hello': [1, 5],
    'random': [10, 12],
    'swim': [7, 9],
    'world': []
}
```

Output

As output of above tasks, show followings,

1. Present the inverted index.
2. Present the query index
3. Present the query matching posting list of the documents. The number of input queries will be 3, but the query terms can be varied.

Reference link: *https://github.com/heetc27/Information-Retrieval-Projects/blob/master/IR%20Project%202/CSE%20435_535%20Information%20Retrieval%20Project%202.pdf*