Com S 435X/535X Programming Assignment 4
400 Points
Due: May 1, 11:59PM
Late Submission Due: May 2, 11:59PM(25% Penalty)

This programming assignment is on information retrieval. Given a collection of documents you will use vector space model to build an inverted index and when a query arrives, you will retrieve top k documents that are relevant to the query. In this assignment you will also use bloom filters.

Note that the description of a programming assignment is not a linear narrative and may require multiple readings before things start to click. You are encouraged to consult instructor/Teaching Assistant for any questions/clarifications regarding the assignment.

For this assignment, you may work in groups of two.

1 Vector Space Model

Recall that in the vector space model, every document is represented as a vector.

Given a collection of documents $D_1, D_2, \cdots, D_N$, first preprocess the documents to extract all term. For this assignment a words is a term. You should do following pre-processing to form terms: Convert all words into lower case, remove following punctuation symbols: Period, Comman, Colon, Semi Colon, apostrophe. Remove all STOP words, any word with length less than three is a STOP word and "the" is also a STOP word. Let $T = \{t_1, \cdots, t_M\}$ be the collection of all terms in the collection.

Recall that we defined the notion of "weight of a term in a document" as follows:

$$w(t_i, d_j) = \log(1 + TF_{ij}) \times \log(N/df_{t_i})$$

where TF_{ij} is the frequency of t_i in d_j and df_{t_i} is the number of documents in which t_i appears.

Now, every document d_j corresponds to the following vector:

$$v_i = \langle w(t_1, d_j), w(t_2, d_j), \cdots, w(t_M, d_j) \rangle$$

Given a query q, we can view it as a (very short) document represent it as a vector v_q. Now, the similar of d_i and q is the cosine similarity of the vectors v_i and v_q.

To enable fast query processing, you will build an inverted index. Recall that each member of an inverted index consist of two parts, dictionary part and postings part.

The dictionary part consists of the three tuple: a term t, df_t and a pointer to postings part. Postings part is a list of 2-tuples. Each tuple in this list is a document d in which t appears and TF_{td}.

1.1 IndexBuilder

This class should have following constructor and methods.

`IndexBuilder.` Gets the name of a folder containing document collection as parameter

`buildIndex()`. This method builds the inverted index.

`weight(String t, String d)` Returns the weight of term t in document d.

This class may have additional methods.

2 Biword Bloom filter

Recall that one deficiency of the vector space model is that it loses the proximity information of terms. I.e., once we build an inverted index we lose the information whether a term t_i appears next to term t_j in a a given document. A possible way to address this is via building a biword index or by building a positional index. For this assignment, we take a different route. We will use a bloom filter that stores all biwords of all documents. The notion of bi-word is best illustrated by a an example. Suppose we have a document whose contents are

`The theory of general relativity is postulated by Albert Einstein.`

Then the bi-words of the above document are:

`theory general, general relativity, relativity postulated, postulated albert, albert einsetin`

2.1 BiWordDocumentFilter

Design a class named `BiWordDocumentFilter`. This class is exactly same as the class `DocumentFilter` from PA1 except that it will contain all bi-words of a given document. This class will have following methods and constructors.

`BiWordDocumentFilter(int bitsPerBiWord, String fileName, String pathName)` Creates filter that can hold all bi-words that appear in the file `fileName` that resides in the folder `pathName`. The number of bits used per bi-word is `bitsPerBiWord`.

`addDocument()` Adds all bi-words that appear in the file `fileName` to the filter.
`appears(String s).` Returns true if string s appears in this filter; otherwise returns false. This should be case-insensitive

`getDocument()` Returns the name of the file whose contents are added to the filter.

If you wish you may make this class a subclass of `BloomFilterDet` or `BlommFilterRan` that you wrote for PA1.

3 QueryProcessor

This is the main program that puts together both the classes `BiWordDocumentFilter` and `IndexBuilder`. The program gets the name of a folder containing the document collection as input. It then builds an inverted index for the collection and creates an array of `BiWordDocumentFilters`(one BiWordDocumentFilter per each document in the folder). Use 8 bits per bi-word.

The program repeatedly prompts the user to enter a query q and an integer k and outputs (names of) top k documents that matches the query. For each document, also outputs the similarity score. The top k documents are determined as follows: Let $q = t_{i_1} t_{i_2} \cdots t_{i_r}$, i.e, the query has r terms. First retrieve a set S consisting of top $2k$ documents (along with cosine similarities with q) that matches the query using vector space model scoring. Let $B(q)$ be the set of all bi-words of q, i.e,

$$B(q) = \{t_{i_n} t_{i_{n+1}} \mid i < r\}$$

For every document $d \in S$, let $s(d)$ be the number of bi-words from $B(q)$ that appear in d. Compute $s(d)$ using `BiWordDocumentFilter` (Note that using BiWordDocumentFilter for this

computation may not give an exact value of $s(d)$ due to false positives, thats OK). Now, the final ranking of documents is based on the following criteria. Let d_1 and d_2 be two documents from S

- If $s(d_1) > s(d_2)$, then rank of d_1 is higher than rank of d_2
- If $s(d_1)$ equals $s(d_2)$ then compare $CosineSim(d_1, q)$ with $CosineSim(d_2, q)$. The document with higher similarity gets higher rank.

The program outputs (names of) top k documents along with cosine similarities with q according to the above order.

4 Report

This section will be added later

5 Data

This section will be added later.

6 What to Submit

- IndexBuilder.java
- BiWordDocumetFilter.java
- QueryProcessor.java
- Source files of additional classes that you used.

Only one submission per group please.