CS172: Information Retrieval

UC Riverside

Prof. Mariam Salloum Fall 2019

Problem Set 1

1 Description

In this assignment, you are asked to implement a *partial* retrieval system based on the Vector Space Model which was discussed in Lecture 2. You will extend this to a full retrieval system in future assignments. Given a *term*, the program should print the TF-IDF weight of that term for each document that contains the term.

The assignment is divided into two parts, Part A - document ingestion and index creation and Part B - query lookup.

Part A (document ingestion and index creation) takes a collection of documents to create the dictionary index and posting-list.

Part B (term lookup), takes a term and uses the index created in Part A to print the TF-IDF weights for that term in documents that contain the term. We shall provide instructions on both parts in the following sections. Please be sure to implement the retrieval system as specified. Some information that you will be asked to include in the index may not be relevant for this assignment, but will be useful for the next assignment.

1.1 Part A - Inverted Index Creation

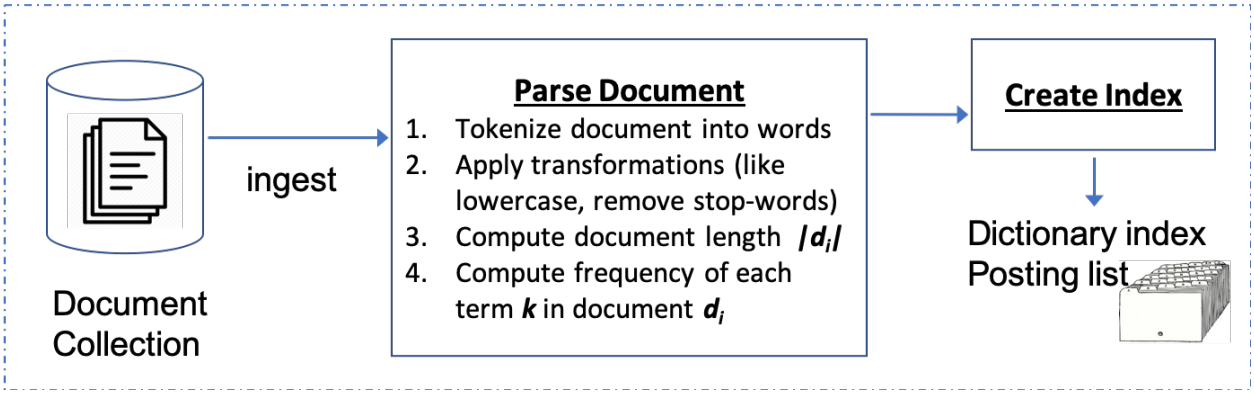

Figure 1: Part A - document ingestion and index creation

- You will be given a collection of documents (folder of documents).
- Write code to parse/read each document from the directory.

- For each document, tokenize documents into words, remove stop-words, lowercase each token (perform stemming and lemmatization for extra credit).
- Build two indices as described below.

Write a function called ***createIndex*** that receives the folder path to document collections, reads the provided input data and creates two indices, as defined below:

(a) Word / Posting Index: An inverted index (that excludes stop words from the file stoplist.txt) of terms/words and the corresponding frequency and documents in which the term is found. The index data structure should have some form such that you have random look-up of terms and allows new terms to be added without rebuilding it completely. The index should contain the document frequency of each term (i.e. the number of documents that contain the term).

(b) Document index: An inverted index that contains the number of terms in each document.

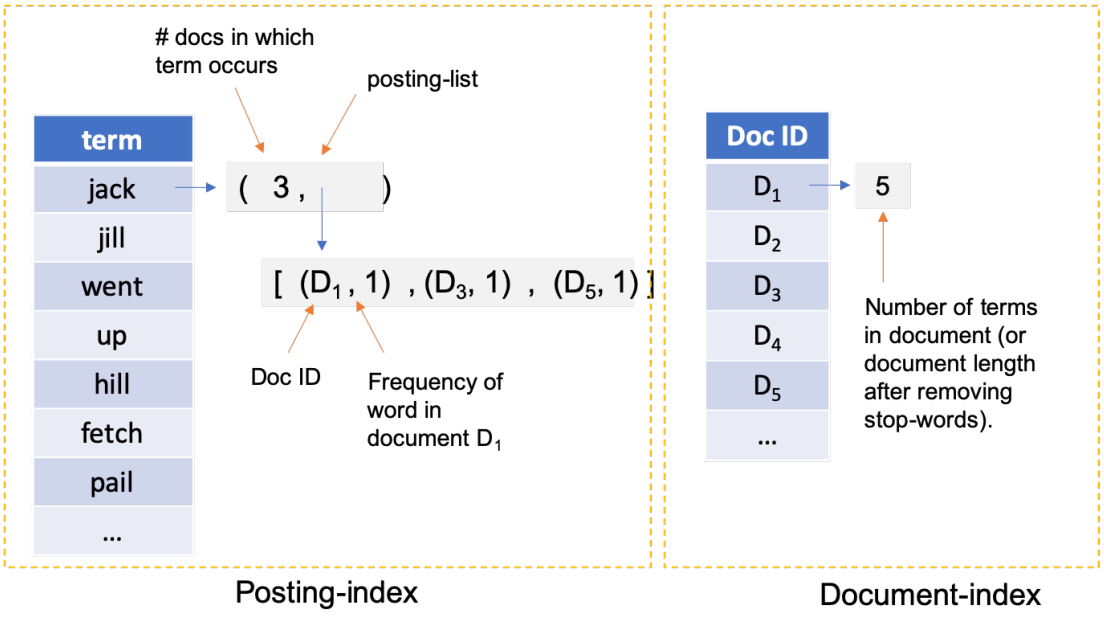

Figure 2: Posting-index and Document-index

An example is shown in Figure 1.1. A set of 8 document is shown, and a partial snapshot of the two indices are shown.

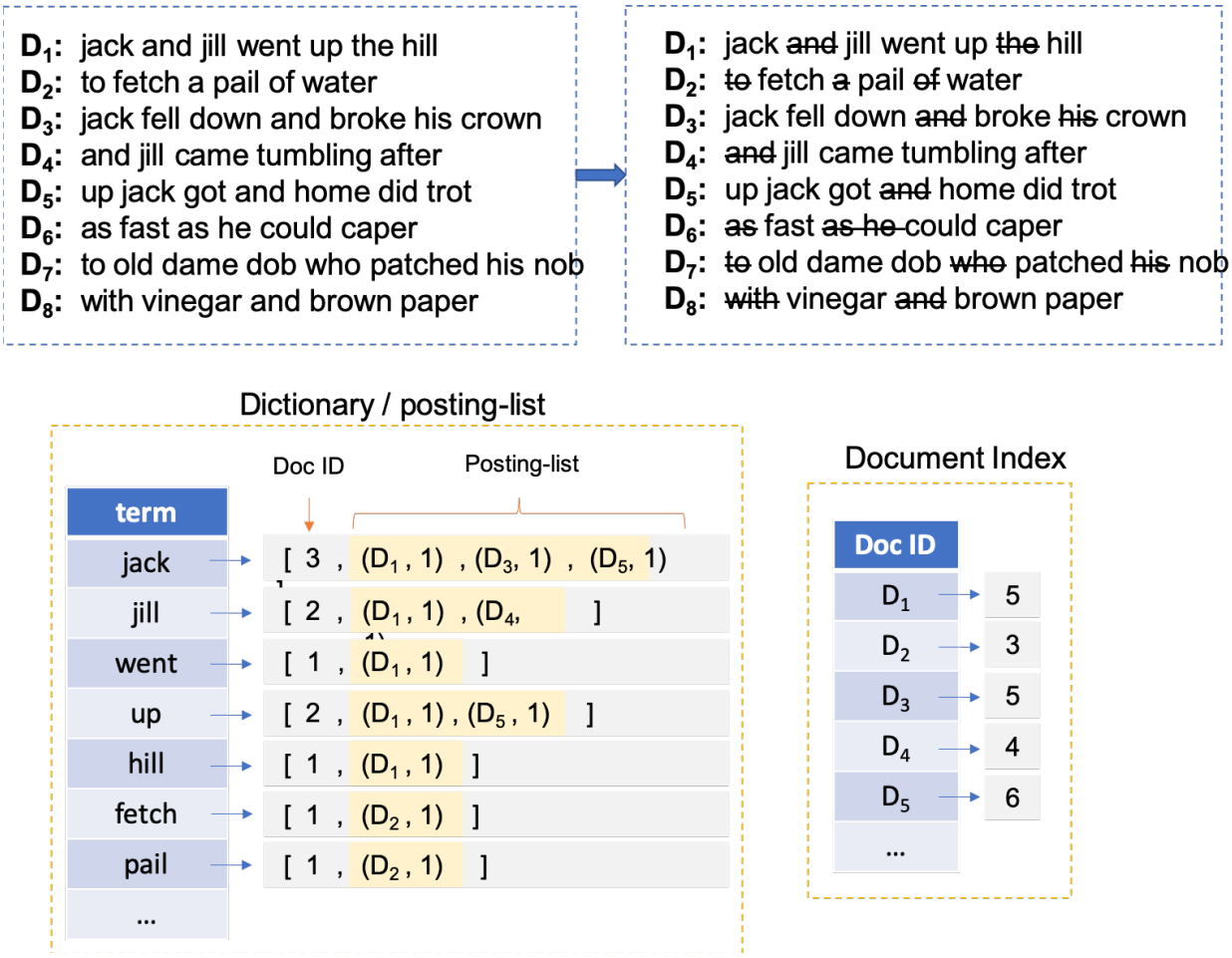

Figure 3: Example of 8 documents D_1-D_8 with a partial snapshot of the two indices.

1.2 Part B - Term Lookup

Write a function termLookup that accepts a user term/query (assume a single term), and prints the TF-IDF weights of that term for each document that contains the term. If the term is not found in any of the documents, then just print 'No Match'

- Prompt the user for a term(assume its just 1 term; in future assignments we will expand this to multi-term query).
- Process the term using the same transformations you applied to the documents (lowercase, remove stop-words, stemming, etc.).
- Print out the TF-IDF of that term in each of the document matches, if any.
  - Look-up term k in posting-list index.
  - If term k is not in the index, printout 'No Match'

– If term k is in the index, then for each document D_i in the posting-list compute the TF-IDF of the term k in document D_i and print to console.

How to compute TF-IDF for a term k and document D_i ? Use the following formulas discussed in class.

$$tf_{ik} = \frac{f_{ik}}{\sum_{j=1}^{t} f_{ij}}$$

$$idf_k = 1 + \log \frac{N}{n_k + 1}$$

$$\text{TF-IDF}_{ik} = \text{tf}_{ik} * \text{idf}_k$$

Figure 4: TF-IDF formula.

Below is an example for the query-term "jill".

Assume the user typed "jill" as the query term, then your job is to print the TF-IDF weight of the term "jill" for D_1 and D_4 because those are the two matches in the posting-list

For doc $D_i = D_4$ & term k = "jill" , then

$\text{TF-IDF}_{ik} = \text{tf}_{ik} * \text{idf}_k$

$= 1/4 * (1 + \log (8/(2+1)))$

For doc $D_i = D_1$ & term k = "jill" , then

$\text{TF-IDF}_{ik} = \text{tf}_{ik} * \text{idf}_k$

$= 1/5 * (1 + \log (8/(2+1)))$

You will notice that the IDF value (idf_k) is computed once per term. All information can be extracted from the indices you created.

Figure 5: Example TF-IDF computation for term "jill" in the example document set provided in Figure 3

1.3 README

Include a readme file that describes the design of your code and detailed instructions for running your program. We will not grade programs that don't compile, and won't attempt to run your program if the instructions are not provided.

2 FAQ

- Can I use libraries to perform stop-word removal? No, write this code yourself and use the provided stop-words.txt file to filter out the stop-words.
- Can I use libraries to perform stemming and lemmatization? Yes, you are welcome to use libraries to perform these two tasks (you don't have to write this from scratch). However, be sure to identify the packages used in the README file. The nltk package in Python can perform both of these operations. Note, stemming and lemmatization is not required, this is extra credit.
- Can I use libraries or code online to perform the retrieval or TF-IDF? No, this portion should be implemented from scratch. You will build upon this program in future assignments, so be sure you are following the instructions outlined here.

3 Submission

You will be submitting your assignment via Github. Please use the link provided to the github assignment. Your submission should include all source files and a README file. The README file should include a short description of what is included in the submission and how to run your program.

NOTE: its your responsibility to verify that your code was checked-in to the Github repo correctly and on-time.