Lab – 12

Week of Apr. 15, 2018

1 Instructions

1. This week's assignment will be a mini-project and has two parts. **Part 1** is your usual lab assignment (deadline: April 21). **Part 2** will help you earn up to 5 bonus marks and you will be given an additional *5 days* to complete (deadline: April 26).
2. You are free to use any of the inbuilt Ocaml modules.

2 Resources

Formula for calculating term frequency tf and inverse document frequency idf:

1. tf(t,d) = (Number of times term t appears in a document d) / (Total number of terms in the document d)
2. idf(t) = $\log_{10}$(Total number of documents / Number of documents with term t in it)
3. tf.idf score(t)= tf(t)* idf(t)
4. Useful link : http://www.tfidf.com/

3 Building a search engine

You will build a simple search engine for text documents which has the following functionality.

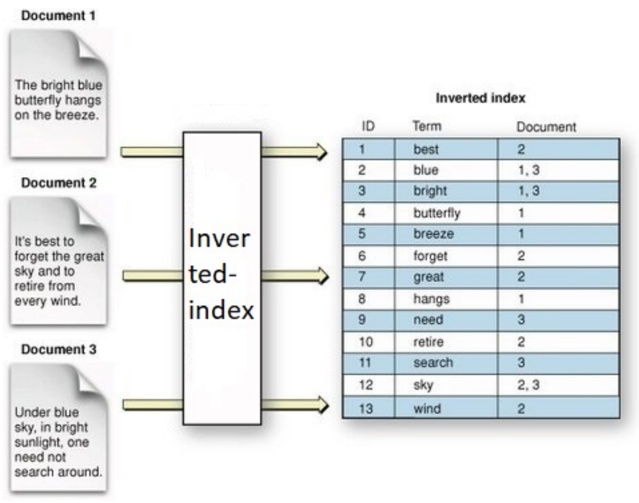
1. Supports keyword queries
2. Supports phrase queries
3. Ranks results using the $tf \cdot idf$ scoring model

There are two modules that you will implement. In the first module, you will build the *inverted index* from a set of text documents. In the second module, you will provide the search functionality and result ranking.

3.1 Part 1: Building the Inverted Index

1. Parse the text files (numbered by their id) (code to do this will be provided as part of the assignment) and it will return a list of strings where each string will represent a document. Use list index as document id.
2. Do not include characters like , and . in the inverted index. So, preprocess data to replace , and . with " " (1 space) in each string.
3. Extract the words from the documents by separating each string by the space character in the list.
4. Calculate Term frequency of each (term, document) pair.
5. Calculate Inverse document frequency of each term.
6. Compute Position information of each (term, document) pair.

7. Build the inverted index and output to file.



Document 1: The bright blue butterfly hangs on the breeze.

Document 2: It's best to forget the great sky and to retire from every wind.

Document 3: Under blue sky, in bright sunlight, one need not search around.

ID	Term	Document
1	best	2
2	blue	1, 3
3	bright	1, 3
4	butterfly	1
5	breeze	1
6	forget	2
7	great	2
8	hangs	1
9	need	3
10	retire	2
11	search	3
12	sky	2, 3
13	wind	2

NOTE: The above diagram just gives an idea about the inverted index. In addition, you need to store information including inverse document frequency (idf) of each term, term frequency (tf) and positional information of each (term, document) pair in your inverted index data structure.
For each term **w**, store a record which contains **w**, **idf(w)**, and a list of triplets. Each triplet in the list stores **docid**(docid in which word **w** appears), **tf(w,docid)**, and **P(w,docid)**, the set of positions where word **w** occurs in document **docid**.
Document ids and position index in each document starts from 0.

3.1.1 Format of files

1. **Format of text files**: You may assume that the text files consist only of digits 0 to 9, *lowercase* letters, and the following special characters: space, comma, full-stop, and the newline characters.

2. **Format of the inverted index file**: Return inverted index information for each word (in a new line) in the corpus and each line should follow the format specified below.

   word`<space>`idf(w)`<space>`[(doc1id`<colon>`tf(word,doc1id)`<colon>`[postion1`<semicolon>`position2`<semicolon>`...])`<comma>`
   (doc2id`<colon>`tf(word,doc2id)`<colon>`[position1`<semicolon>`position2`<semicolon>`...])`<comma>`.....]

   `<comma>` means , is used as separator.
   `<colon>` means : is used as separator.
   `<space>` means   is used as a separator.
   `<semicolon>` means ; is used as separator.

   tf(w,doc1id) = term_frequency of word w in the document doc1id/total number of words in document doc1id.
   idf(w) = log base 10 (Total number of documents / Number of documents with word w in it)
   NOTE: Here, doc ids and position list should be printed in sorted order.
   Truncate tf and idf values to 5 decimal places.
   Check sample input and output in Sections 3.1.3 and 3.1.4 for more clarification.

3.1.2 How your program will be run

Your source file will be compiled into an executable as follows: `ocamlc -o buildindex buildindex.ml` .The executable will run using the command `./buildindex infile outfile` where infile is input text file name and outfile is output text file. The program will print the inverted index in the format specified above in output file `outfile`.

`NOTE:` For reading file into list of strings, write the following code in your submission file:
open Input
read_filedoc(infile) (*this function will return list of strings*)

3.1.3 Sample Input file for Section 3.1

Give the file name as input to given function `read_filedoc` (filename) and this returns a list of strings as specified below:

["The bright blue butterfly hangs on the breeze."; "Its best to forget the great sky and to retire from every wind.";"Under blue sky in bright sunlight,one need to search around."]

3.1.4 Sample Output file for Section 3.1

best 0.47712 [(1:0.07692:[1])]
blue 0.17609 [(0:0.125:[2]),(2:0.09090:[1])]
and so on.

3.2 Part 2: Searching and Ranking

You will be using input file "inverted.txt" which will contain inverted index in the same format as specified in Section 3.1.1 part 2.

3.2.1 Boolean retrieval

1. Given a set of keywords, retrieve the set of documents containing *all of them*.
2. Sort the documents by id and return the ids.

3.2.2 Ranked retrieval

1. Given a set of keywords, retrieve the set of documents containing *all of them*.
2. Compute the $tf \cdot idf$ score for each document.
   (For more explanation of tf·idf score , check out the Resources section)
3. Sort the documents by score (highest score is at rank 1) and return document ids of the top-k (k is specified) documents.

NOTE: Given query q, score of document d :
Score(q,d) = Sum of tf*idf value of each word of query q in document d.

3.2.3 Phrase queries

The processing of phrase queries involves checking if the keywords in the given phrase occur in contiguous positions in the retrieved documents. Once the documents are filtered, scoring the documents proceeds as before.
For eg. if keywords in query are `<t1> <t2>`

1. Find document ids containing both `<t1>` and `<t2>`
2. In each document, if position of `<t2>` = position of `<t1>` +1 ,then add that docid to output list.

3.3 Format of outputs

1. For 3.2.1 `<-b>` : Print list of space separated top k document ids sorted by id.
2. For 3.2.2 and 3.2.3 `<-r>` and `<-p>`: Print list of space separated top k documents ids sorted by tf·idf score.

3.4 How your program will be run

Your source file will be compiled into an executable as follows: `ocamlc search.ml -o search`. The program will be run with the following parameters: `./search <-b/-r/-p> <k> <n> <t1> <t2> ... <tn> inverted.txt`.

1. The switch `-b` indicates boolean query, `-r` indicates ranked retrieval and `-p` indicates phrase query.
2. Parameter `k` indicates the *maximum* number of results to return. Note that this parameter is to be ignored if the first argument is not `-r`.
3. Parameter `n` indicates the number of keywords in the query.
4. Parameters `<t1>...<tn>` are the words constituting the query. Whether the keywords should be treated as a phrase will be decided by the switch `-p`.
5. inverted.txt is the input file containing inverted index.

3.5 Sample Input parameters to execute command for Section 3.4

Input file is same as mentioned in Section 3.1.3.

1. search -b 2 blue bright inverted.txt
2. search -r 1 blue breeze inverted.txt
3. search -p 1 great sky inverted.txt

3.6 Sample Output format for Section 3.4

1. 0 2
2. 0
3. 1