CS433-CS533 Project Part 3
(Due: May 3, 2020)

The goal of this project is to develop a small IR system and to practice implementing different components of an IR system. This project will be carried out in multiple parts during this semester. This is the Part 3 of this project.

The goals of this part are three-fold: (1) implement another similarity function (the Okapi function) to compute the similarity between a document and a query; (2) re-run your indexing component and search related data structures you developed during the first two parts of the project using a larger dataset; and (3) compare the effectiveness of the two similarity functions you implemented (i.e., Cosine function and Okapi function) based on a number of test queries.

Again, the project is a team project with two members of your own choice (one team has three members because we have an odd number of students).

Your project for this part needs to accomplish the following tasks:

1. Re-run your indexing component and search related data structures based on a new document collection called 200_content.txt. Each **paragraph** in 200_content.txt represents one document. In order to support the Okapi function (see below), which uses raw *tf*, you need to store the raw *tf*s in the posting lists (like in Part 1 of the project). Since the Cosine function uses the *tf*-weights, not the raw *tf*s, you can expand the posting format to include both raw *tf* and *tf*-weight, i.e., make it to the format (docID, tf, tfw), where tfw is computed in the same way as in Part 2 of the project. For each document D, compute two document lengths, one is the sum of the tfs of the terms in D and the other is the ||D|| using the tf-weights (same as in Part 2 of the project). The lengths are computed in advance and stored in one array or two arrays. Furthermore, the average length of all documents based on the first type of length (i.e., using tfs) is computed in advance and stored in a variable. This will be used by the Okapi function.

2. Implement the Okapi function. Use the following parameter settings for your implementation: $k_1 = k_3 = 1.2$, $b = 0.75$. Use a Document-At-A-Time-like technique to compute the similarity for one document at a time.

3. For each of the two similarity functions (i.e., Cosine and Okapi), retrieve the top 10 results for the following 4 queries: (1) europe; (2) stock rally; (3) debt crisis; (4) stock future higher. If a query yields less than 10 results for a similarity function, just show all the results that are retrieved.
    a. For Okapi function, create a separate text file for each query with the following information: for each of the top-10 documents, first show (t_i, w_i, dt_i, qt_i) for each query term t_i on a separate line and then show (docID, similarity) as the final line for the document; repeat the above for all documents. The documents should be shown in descending similarity values. The file name for the first query results should be okaquery1result.txt. Similar for other query results.

b. For Cosine function, create a separate text file for each query with the following information: for each of the top-10 documents, first show (t_i, $w_{t,q}$, $w_{t,d}$, 1/||D||) for each query term t_i on a separate line and then show (docID, similarity) as the final line for the document; repeat the above for all documents. The documents should be shown in descending similarity values. The file name for the first query results should be cosquery1result.txt. Similar for other query results.

4. For each retrieved result, manually check if the result is relevant to the query. Relevance judgment is somewhat subjective so some discrepancy will be allowed. You should try to use the following guideline to determine whether a document should be considered as relevant for each query:

    a. For query "europe", a document should be considered as relevant only if Europe is the focus in at least part (several sentences) of the document.
    b. For query "stock rally", a document should be considered as relevant only if stock rally is truly the subject in at least part (several sentences) of the document.
    c. For query "debt crisis", a document should be considered as relevant only if debt crisis related subject is discussed in at least part (several sentences) of the document.
    d. For query "stock future higher", a document should be considered as relevant only if at least part (several sentences) of the document talks about higher future of stocks.

    For each of the two similarity functions, report the relevant documents (based on your judgment) in a text file (cosrelresults.txt for Cosine function and okarelresults.txt for Okapi function). Each line in these two files has the following format: query, docID, …, docID. Here "query" is one of the four test queries and each docID is the docID of a relevant document.

5. Based on the relevance information above, for each similarity function, compute its MAP@5 and MAP@10. Compare the MAP@5 and MAP@10 for the two similarity functions in your evaluation report (see "What to submit" below for more details).

Programming language requirement

You may write the code using any programming language but it is required that your program must compile on harveyv.binghamton.edu. No exceptions. It should be purely a command line program. NO GUI will be accepted. This enables the TAs to run your code using test scripts. DO NOT assume that since your program compiled and ran correctly on your laptop it will also compile and run correctly on harveyv.binghamton.edu

What to submit?

Please send a [file_name].*tar.gz* file. The file name should contain the last name(s) of the team member(s). When the file is unzipped it should contain a directory with the same name as the zip file. The directory should contain the following files:

1. The source code of your implementation plus possibly a make file. The code should be reasonably commented.

2. A readme.txt file on how to run your code.
3. A status and evaluation report of your project. It should report what programming language you use, how complete your implementation is (especially for students who could not fully complete the project), whether correct results are obtained and the number of terms in your dictionary. For the evaluation part, for each similarity function and for each query, report the MAP@5 and MAP@10 results and your comparison analysis.
4. The new dictionary.txt.
5. The new postings.txt.
6. The result files cosquery1.result.txt, cosquery2.result.txt, cosquery3.result.txt, cosquery4.result.txt, okaquery1.result.txt, okaquery2.result.txt, okaquery3.result.txt, okaquery4.result.txt, cosrelresults.txt, and okarelresults.txt.

Where to submit?

Submit to the Project Part 3 submission folder in MyCourses.

Demonstration

The instructor/TA may ask teams to demo their projects as needed.

Plagiarism Check

All your code will be subject to check for similarity with other submissions using Moss. So you are advised not to look at each other's code.