Vartan Benohanian
ID – 27492049
November 13, 2018

# COMP 479 – Project 2

## Okapi BM25

## Description

A lot of the code comes from Project 1. Everything written in the Project 1 report (also included in the submission) is assumed here.

I make use of the index generated by the SPIMI in Project 1. From that index, I get relevant information such as which term is in which documents (document IDs), how many times a term is in a certain document, etc.

Using those values, I am able to implement the Okapi BM25 algorithm.

## Running

Make sure to use Python 3 and install the required packages listed in the *README.md* file of the submission.

To run the program, type in the command line: **python3 main.py [arguments]**. The arguments are the following (they all relate to Project 1):

1. -d or --docs: number of documents per block. Default is 500.
2. -r or --reuters: number of Reuters files to parse, choice from 1 to 22. Default is 22.
3. -rs or --remove-stopwords: stopwords in index are removed. Default is false.
4. -s or --stem: terms in index are stemmed. Default is false.
5. -c or --case-folding: tems in index are converted to lowercase. Default is false.
6. -rn or --remove-numbers: remove terms in index that are numbers. Default is false.
7. -a or --all: sets arguments 3 to 6 true. Default is false.

To modify the runing parameters of the Okapi BM25 ranking algorithm, you must go directly in the *main.py* file and add them in its initialization, on line 112. The parameters *b* and *k1* are both set to 0.5 by default.

## Challenge Queries

The test queries and their results are in the *test-queries.txt* file. The results are in descending order, by score.

## Conclusion

While completing this project, I learned that document relevance according to the user's search query is very important. When playing around and testing query results with my implementation of Okapi BM25, I noticed the documents that are the last in the list definitely seemed to be of the least relevance.

To make my program better, I removed stopwords from queries, which I noticed made it so that fewer results are returned. The difference was significant, as not only did it return a proportionally larger set of relevant documents, but the program was faster too, as there were fewer documents to work with for the algorithm to be computed.

One thing that my program doesn't do, is that it doesn't take into account phrase queries, i.e. multiple words that, when strung together, have a more specific meaning. For example, searching for "George Bush" might possible yield results with one of the terms in the document, with not relation to the actual George Bush.