COMS4995 002 Project Proposal

Team Name: Okapi

Ian Pan (ip2399) & Han-Ju Tsai (ht2572)

Background & Introduction

We are interested in the mechanics of ranking functions used by search engines that estimate documents' relevance from a given query. In natural language processing and information retrieval, we have learned about tf-idf, which is a popular weighting scheme for ranking and scoring documents used by search engines. In this project, we explore the intricacies of an even stronger ranking function, Okapi BM25, and apply it to a parallel setting.

Deliverables

In Okapi BM25, we can roughly divide its work into two stages. The first (preprocessing) stage consists of building an index for the documents at hand, in which we record the occurrences of words both inter-document and intra-document. The second stage copes with the queries of some given set of keywords. We apply the ranking function to each document, and output the top matched documents as our result.
We will count the number of occurrences of each word in the documents (Term-Frequency) and how many documents include that word (Document-Frequency). With the term-frequency and document-frequency calculated, the BM25 score of a document D can be determined.
The following formula produces the score of a document D for a given query Q.

$$\text{score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

, where $f(q_i, D)$ is the term frequency of the word q_i in document D. In addition, k_1 and b are empirically chosen hyperparameters.

After obtaining the score of each document, we can build a size-k min-heap if we want to retrieve the top k documents for the given query. This reduces our runtime from a normal sorting's $O(n \log n)$ down to $O(n \log k)$, where n is the number of documents. If we are only interested in retrieving the best matched document, we can obtain it in $O(n)$ time.

We plan to implement Okapi BM25 in Haskell, and apply parallelism to several steps to enhance performance. We also plan to make our functions reusable APIs and package the project into a user-friendly Haskell library.
In the preprocessing stage, the term-frequency and document-frequency can be computed and stored in a file. The Okapi BM25 classifier can be initialized with either a preprocessed file or a raw dataset. In the case of manually building the index, we can split each document into chunks and calculate the term frequencies for a particular document in a parallelized manner. We could also retrieve the document frequency of a given word parallelly by utilizing different sparks.
In the second stage, we can parallelize the calculation of different documents' scores with respect to the query and store the scores in min-heap for the final retrieval.

Resources & References

TF-IDF from scratch in python on a real-world dataset. | by William Scott | Towards Data Science
https://stackoverflow.com/questions/61877065/implementation-of-okapi-bm25-in-python