**RESEARCH PAPERS SEARCH ENGINE**

**PHASE 2 – RANKING AND QUERYING**

**ABSTRACT – 16PW24, 16PW33**

**Project Abstract:**

The aim is to develop a search engine which would search for papers related to a paper of interest to the user. It displays the user a list of relevant papers to the query. All the documents shown contain at least one of the query words and in the results shown, all query words are bold marked. The entire project is implemented in python.

In the second phase we have concentrated on the querying, ranking docs and also built a simple UI.

**Querying:**

At the end of the Phase 1, we have got the indexed terms which consists of the unique terms and the doc\_frequency. The index file was stored as a pkl object.

When a query term is given the first step is to preprocess the query – the same process (case folding, punctuation removal, stop words removal, lemmatizing) as we preprocessed the documents. The indexed file is used in getting the top k docs. After getting the top k docs by ranking function the query terms are checked in the doc and marked as bold.

**Ranking:**

BM25 is a ranking function used by search engines to rank matching documents according to their relevance to a given search query. It is based on the probabilistic retrieval framework.

BM25+ is an extension that was developed to address one deficiency of the standard BM25 in which the component of term frequency normalization by document length is not properly lower-bounded. As a result of this deficiency, long documents which do match the query term can often be scored unfairly by BM25 as having a similar relevancy to shorter documents that do not contain the query term at all.

* To compute the doc scores using BM25+, the idf of the word is to be found. Adding 1 to the value, before taking the log, which makes it impossible to compute a negative value.

Idf = log((no.of docs+1)/doc\_frequency)

* Computing tf follows the formula:

((k+1)\*tf)/(k+tf))

For BM25+, k is often set to 1.2. Changing k can be a useful tuning approach to modify the impact the TF. A higher k causes TF to take longer to reach saturation. By stretching out the point of saturation, you stretch out the relevance difference between higher and lower term frequency docs!

* The TF score above is further influenced by whether the document is above or below the average length of a document in the corpus. Hence introducing two variables: a constant b and a length value L.

((k + 1) \* tf) / (k \* (1.0 - b + b \* L) + tf)

L - how long a document is relative to the average document length.

Hence L is defined as the document length divided by the average document length.

The constant b will allow us to finely tune how much influence our L value has on scoring.

* The scoring function for each word in query w.r.t a doc is:

**IDF \* ((k + 1) \* tf) / (k \* (1.0 - b + b \* (|d|/avgDl)) + tf)**

* Based on the cumulative scores of docs w.r.t all the words present in the query, the top k docs are determined and are provided as results to the user.

**UI:**

The UI is built using Flask. The query can be typed into a search box. The results are displayed according to the relevance of query and the query terms present in the doc are highlighted. The link can be chosen to navigate to the particular research paper. Some docs are loaded initially. You can navigate to the next page for more queries.

**References:**

The following link helped us in choosing this ranking function rather than using tf\*idf weighting schema and cosine similarity to rank the docs w.r.t query.

https://opensourceconnections.com/blog/2015/10/16/bm25-the-next-generation-of-lucene-relevation/