Information Retrieval Project 2 Report:

1. **Problem Statement**

The project requires construction of five variations of a simple search engine.
Following tools were provided:

a. An inverted index is provided through a web interfaced which is used to lookup documents for terms in the query of a search engine.
b. Corpus Statistics
c. Document Information
d. File listing stem classes
e. File listing stop words
f. File mapping internal and external document ids
g. trec_eval script to evaluate retrieval model runs
h. Two files containing relevance feedback: NIST_qrel_file and IS4200/CS6200_qrel_file

The following five variations of a retrieval system are to be implemented:

- Vector Space Model, Okapi-tf
- Vector Space Model, Okapi-tf-idf
- Language modeling, maximum likelihood, Laplace Smoothing
- Language modeling, maximum likelihood, Jelinek-Mercer
- BM25

25 queries have been provided and they have to be run through all the implemented systems above. For each query, top 1000 ranked documents should be returned and placed in a file. This file should be then evaluated with the help of the trec_eval file provided. All the five runs should be evaluated using both provided qrel files for a total of 10 evaluations.

2. **System used for development:**

System Type: 64-bit Operating System
Operating System: Windows 8
Processor: Intel® Core™ i7-3612QM CPU @ 2.10 GHz
RAM: 8GB

3. **Implementation:**

Language used for implementation: **Python**

The implementation of the project consists of:

a. Processing the queries (__init__.py):

- Read each line of the given query file
- Lower case all the characters in the query string
- Transform punctuations like replacing double quotes by empty string, ‘,’ by empty string, ‘-‘ by space, single quote by space, opening and closing brackets by empty string.
- Remove all the stop words from the query string. (for database =3 and database =2)
- Create a query entry in a map to store the query id and the transformed query string

We “pickle” the query map object to objects/queryObjFile, so that we can use the stored object for multiple runs. Thus, we can save processing time required to read the query file.

We will see later how some further tuning of the query strings can increase our retrieval model results.

b. Implementing retrieval models (models.py):

The common algorithm used is:

- For each query term, obtain the document frequency, collection frequency and inverted list of documents containing the term.
- Then for each document containing the term, calculate a score depending on a particular retrieval model.
- Lookup in the 'docScore' dictionary for an entry of score for that document.
- If a score is already present for the particular document id in the dictionary, then update that score by adding/multiplying the score for the term. Else add a new entry in the 'docScore' dictionary for that document id.
- For all models except the Language models, print out the scores for each document in descending order of the scores within increasing order of the query number. Also, rank the documents in increasing order. So, effectively, documents with highest score will have a rank 1.

For language models with Laplace Smoothing or Jelinek Mercer smoothing we require special processing for terms not contained in the document.

So, for each term not in the document find the left over probability and add/multiply it to the score for the document.

After, the leftover probabilities have been accounted for we can rank the documents as done in the last step of our common algorithm.

We "pickle" objects/queryTermDocStats to store the query term to docid, doclen and term frequency and objects/collectionTermStats to store the terms to collection term frequency and document frequency of the term. This done so that we do not hit the server for each run. However, deleting the pickled objects will create the term statistics objects again after requesting data from the server.

SystemVariables.py is used to store constant variables for the system. Currently, it is used only to configure the database for the inverted index.

c. Evaluation

The results are stored in files named in the format <retrieval_model>-result-<date_stamp>.
So, in total we have five results files, one for each retrieval model.
The file entries are in the following order: **query-number Q0 document-id rank-score Exp**
Each, of this files is called a 'run'. We, evaluate each run against our relevance feedback files-NIST_qrel_file and IS4200/CS6200 qrel file)

4. Values used for the project;

a. Smoothing factor in Jelinek-Mercer language model : 0.2
b. BM25:
   $k_1 = 1.2$
   $k_2 = 100$
   b= 0.75

5. Results:

5.1. Analysis of retrieval models for index used for d3 database

Retrieval Model	MAP (Mean Average Precision)		R- Precision		P@10		P@30	
	NIST	IR Class	NIST	IR Class	NIST	IR Class	NIST	IR Class
Okapi-tf	0.1411	0.1997	0.1720	0.2103	0.2640	0.2240	0.2120	0.1600
Okapi- tf*idf	0.2382	0.2857	0.2642	0.2771	0.3520	0.2880	0.3027	0.2240
LM Laplace	0.1384	0.2134	0.1814	0.2074	0.2960	0.2720	0.2453	0.1987
LM Jelinek Mercer	0.1969	0.2120	0.2255	0.1992	0.2760	0.2240	0.2560	0.1787
BM25	0.2235	0.2561	0.2517	0.2332	0.3320	0.2920	0.2813	0.2200

Table: 1

Database Properties for d=3 (STOP STEMM)
Number of documents: 84678
Number of terms: 24401877
Number of unique terms: 166054
Average document length: 288

Analysis:

Okapi- tf*idf gives the highest Mean Average Precision (MAP), the second best MAP is provided by the BM25 retrieval model and the Language Model with Jelinek-Mercer smoothing comes in 3rd on the list.

However, we can improve the precision values by doing some further query processing. Please see the next 2 result tables, where we have done some further query processing.

5.2. Analysis of queries run over d3 Database (With Stopping and Stemming):

After removing the term “document” at the beginning of each query.

Retrieval Model	MAP (Mean Average Precision)		R- Precision		P@10		P@30	
	NIST	IR Class	NIST	IR Class	NIST	IR Class	NIST	IR Class
Okapi-tf	0.1632	0.2261	0.1968	0.2295	0.3120	0.2720	0.2387	0.1720
Okapi- tf*idf	0.2553	0.3107	0.2816	0.3087	0.3880	0.3200	0.3187	0.2440
LM Laplace	0.1677	0.2216	0.2139	0.2295	0.3560	0.3200	0.2773	0.2200
LM Jelinek Mercer	0.2102	0.2411	0.2376	0.2510	0.3080	0.2560	0.2640	0.1827
BM25	0.2453	0.3059	0.2695	0.3089	0.3760	0.3320	0.3067	0.2333

Table: 2

5.3. Analysis of queries run over d3 Database (With Stopping and Stemming):

The following query processing was done for the below results:

- Terms like ‘document’, ‘will’ were removed
- ‘U.S.’ was changed to ‘america’
- “d’etat” was removed because “detat” is not indexed and ‘d etat’ given other results except the term meaning too.

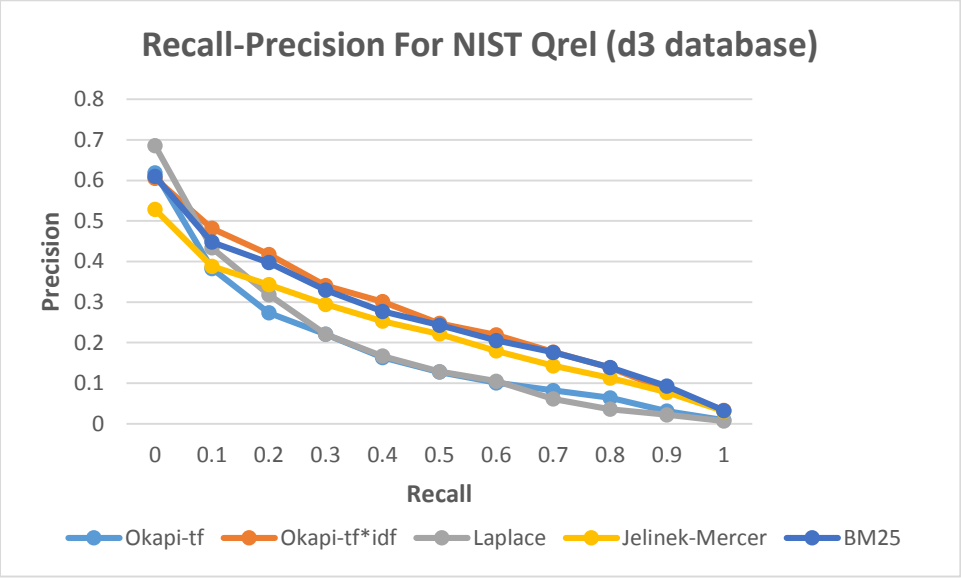
Retrieval Model	MAP (Mean Average Precision)		R- Precision		P@10		P@30	
	NIST	IR Class	NIST	IR Class	NIST	IR Class	NIST	IR Class
Okapi-tf	0.1644	0.2200	0.1989	0.2238	0.3240	0.2600	0.2427	0.1667
Okapi- tf*idf	0.2602	0.3083	0.2917	0.3170	0.4200	0.3240	0.3320	0.2427
LM Laplace	0.1708	0.2196	0.2153	0.2240	0.3720	0.3080	0.2800	0.2213
LM Jelinek Mercer	0.2210	0.2454	0.2495	0.2579	0.3320	0.2440	0.2840	0.1893
BM25	0.2513	0.3035	0.2786	0.3105	0.3880	0.3240	0.3133	0.2360

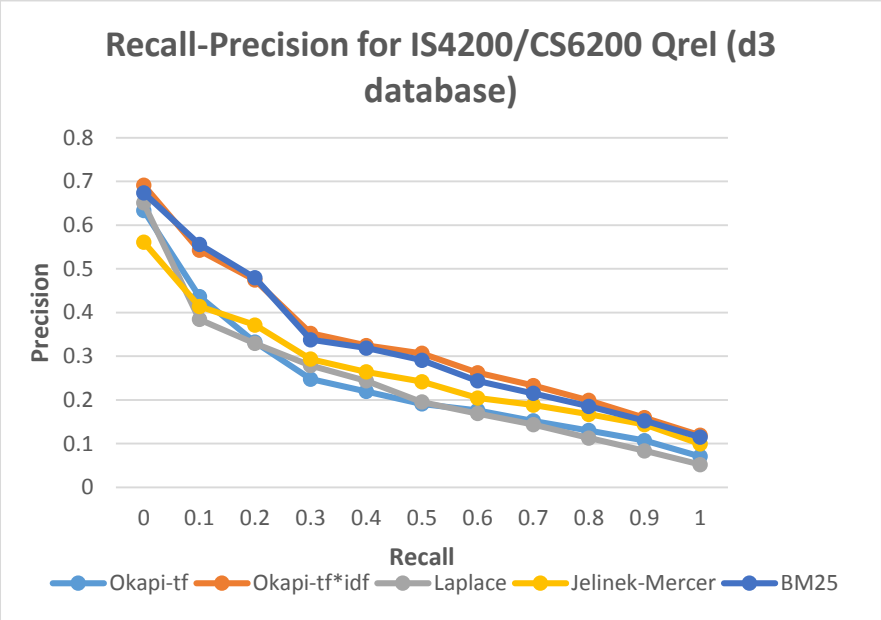
Table: 3

So, as we can see, the precision values improve by doing some further query processing like getting rid of common terms, transforming some terms to its corresponding indexed term.

Now, let’s see the Precision-Recall for the configurations mentioned in section **5.3**

Precision-Recall graph for database 3 after doing processing mentioned in section 5.3





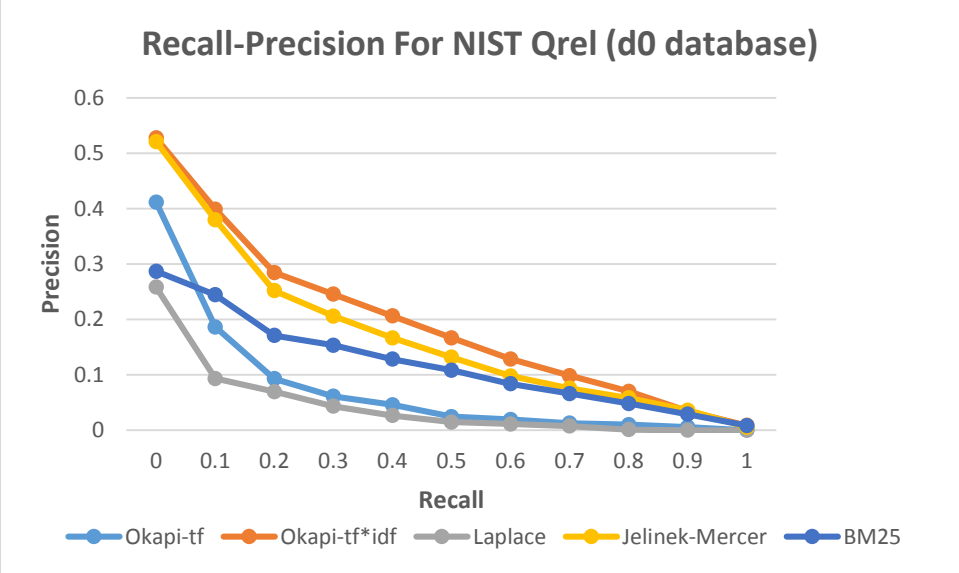
Analysis

a. As we can see from the above graphs, both Okapi tf * idf and BM25 maintains good precision through the recall values from 0 to 1.
b. But, if you consider the total number of document retrieved only, then the Okapi tf * idf performs well. It gets 1255 documents on the trec qrel as compared to 1206 documents fetched by BM25.
c. Now, let's compare our Language Models with Laplace smoothing and the other with Jelinek-Mercer smoothing. As you can see from the above precision-recall graphs, the Laplace smoothing model maintains a good precision in earlier recall values, but the precision drops a lot on later recall values. So, the slope for Laplace smoothing models is higher as compared to the model with Jelinek-Mercer smoothing which maintains descent precision values through recall 0 to 1.

5.4. Analysis of Retrieval Models for database d=0 (NOSTOP NOSTEMM)

Retrieval Model	MAP (Mean Average Precision)		R- Precision		P@10		P@30	
	NIST	IR Class	NIST	IR Class	NIST	IR Class	NIST	IR Class
Okapi-tf	0.0602	0.0935	0.0878	0.1117	0.1720	0.1640	0.1093	0.0973
Okapi- tf*idf	0.1786	0.1921	0.2071	0.1639	0.2920	0.1880	0.2520	0.1693
LM Laplace	0.0370	0.0507	0.0722	0.0755	0.1120	0.0960	0.0933	0.0760
LM Jelinek Mercer	0.1549	0.1429	0.1833	0.1554	0.2720	0.1840	0.2267	0.1413
BM25	0.1108	0.0797	0.1202	0.0849	0.1680	0.0880	0.1427	0.0787

Database Properties for d=0 (NOSTOP NOSTEMM)
Number of documents: 84678
Number of terms: 41802513
Number of unique terms: 207615
Average document length: 493



Analysis:

The above precision and recall graph indicates that without stopping and stemming, the precision levels will go down as compared to situations when stopping and stemming is done.
Also, one more interesting thing of note is that the precision of BM25 starts of low, but maintains a steady precision (smaller slope) compared to Okapi tf * idf and Jelinek-Mercer model.
The reason why BM25 has such low precision values at the beginning recall level 0 is because of the way the IDF weight is calculated for BM25 compared to Okapi tf*idf.
In BM25, the terms appearing in more than half of the corpus will provide negative contributions to the final document score.
For example: let's say we have 5 documents out of which 4 documents contain the term "of". So, the IDF weight for this term in case of BM25 will be log((5-4+0.5)/(4+0.5)) = -1.098. But, the IDF weight in case of the Okapi tf*idf model is log(5/4) = 0.223. Reference: http://en.wikipedia.org/wiki/Okapi_BM25

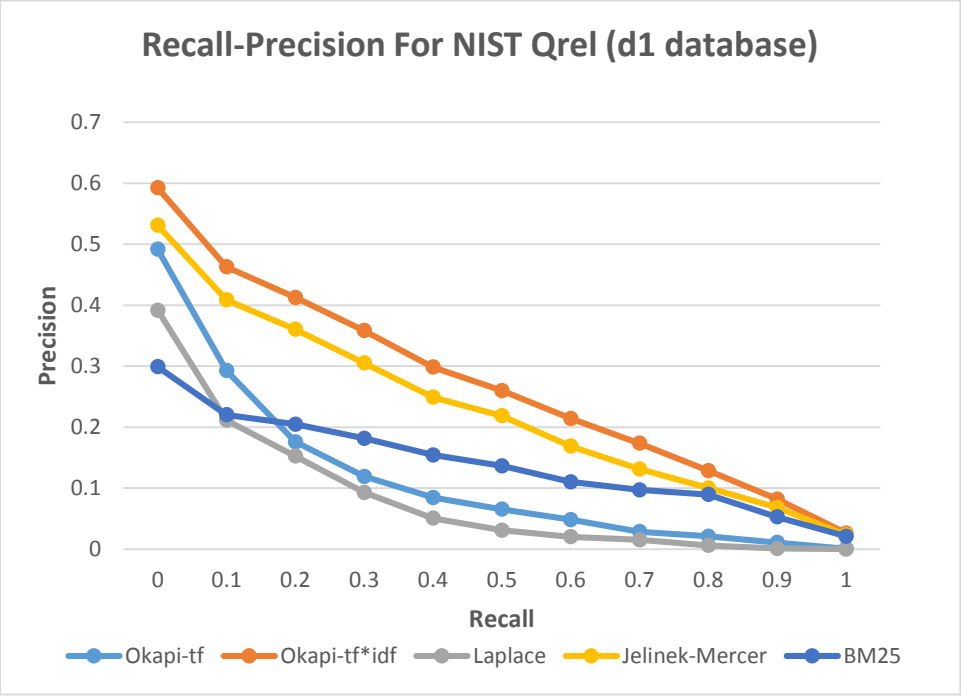
Also, based on conversation with Prof. David Smith, this is because scoring terms in the BM25 model assumes that the stop words are already removed and thus, the precision levels for BM25 are not great in this scenario when, stopping is not applied.

5.5. Analysis of Retrieval Models for database d=1 (NOSTOP STEMM)

Retrieval Model	MAP (Mean Average Precision)		R- Precision		P@10		P@30	
	NIST	IR Class	NIST	IR Class	NIST	IR Class	NIST	IR Class
Okapi-tf	0.1021	0.1455	0.1412	0.1656	0.2560	0.2400	0.1813	0.1427
Okapi- tf*idf	0.2578	0.3118	0.3010	0.3276	0.4080	0.3240	0.3307	0.2400
LM Laplace	0.0713	0.0817	0.1114	0.1126	0.2000	0.1480	0.1413	0.1133
LM Jelinek Mercer	0.2182	0.2387	0.2529	0.2635	0.3360	0.2440	0.2893	0.1880
BM25	0.1325	0.1140	0.1434	0.1092	0.1800	0.1120	0.1533	0.0947

Database Properties for d=1 (NOSTOP STEMM)
Number of documents: 84678
Number of terms: 41802513
Number of unique terms: 166242
Average document length: 493

And, here is the precision-recall graph:



Analysis:
There is similarity between the results for database 1 and database 0. The precision for BM25 model is low at recall 0 compared to the other models. However, it's precision remains stable for other recall levels (smaller slope) compared to the Okapi-tf and Laplace models, whose precision is good for recall 0, but then deteriorates drastically for further recall levels(the slope is large).

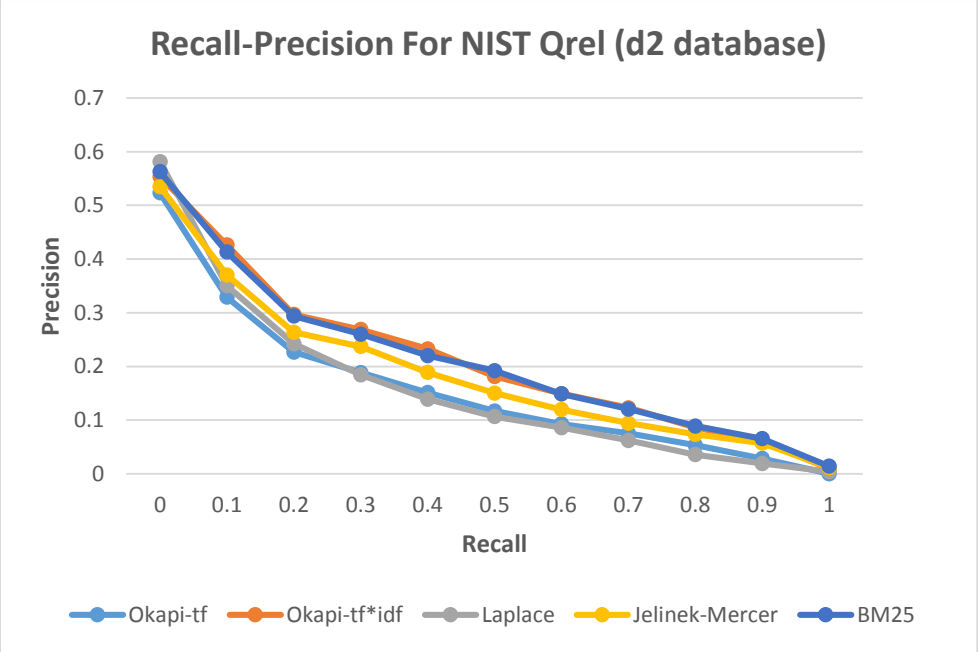
5.6. Analysis of Retrieval Models for database d=2 (STOP NOSTEMM)

Also stopping terms like 'document' and 'will'

Retrieval Model	MAP (Mean Average Precision)		R- Precision		P@10		P@30	
	NIST	IR Class	NIST	IR Class	NIST	IR Class	NIST	IR Class
Okapi-tf	0.1437	0.1813	0.1705	0.1982	0.2720	0.1920	0.2160	0.1360
Okapi- tf*idf	0.1996	0.2201	0.2209	0.2299	0.3080	0.2240	0.2653	0.1720
LM Laplace	0.1414	0.1686	0.1762	0.1759	0.2640	0.2400	0.2347	0.1747
LM Jelinek Mercer	0.1752	0.1840	0.1973	0.2010	0.2800	0.1680	0.2387	0.1413
BM25	0.2006	0.2229	0.2205	0.2328	0.2920	0.2240	0.2587	0.1653

Database Properties for d=2 (STOP NOSTEMM)
Number of documents: 84678
Number of terms: 24401877
Number of unique terms: 207224
Average document length: 288

And, here is the precision-recall graph:



Analysis:

The above precision-recall graph is much similar to the one we saw for database 3.

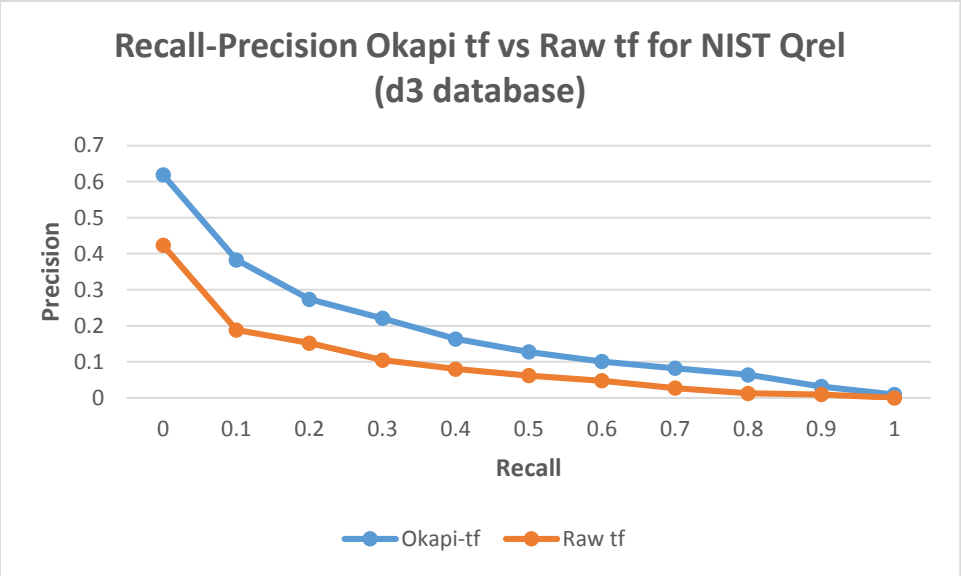
However, a difference that was noticed in BM25 and Okapi tf*idf in d=2 as compared to d=3 is that when stemming is not used, the document frequency for certain terms would be less as compared to document frequency for the terms which would be stemmed in d=2. So, this has affected the score calculations for BM25 and Okapi tf*idf as the calculations of IDF weights these two models depend on the number of documents a particular term occurs in.

Thus, the precision levels for those models have reduced when no stemming was applied.

5.7. Okapi tf vs Raw tf for d3 database with query processing mentioned in section 5.3.

Retrieval Model	MAP (Mean Average Precision)		R- Precision		P@10		P@30	
	NIST	IR Class	NIST	IR Class	NIST	IR Class	NIST	IR Class
Okapi-tf	0.1644	0.2200	0.1989	0.2238	0.3240	0.2600	0.2427	0.1667
Raw tf	0.0836	0.1230	0.1219	0.1233	0.2160	0.1600	0.1627	0.1453

Recall-Precision graph



Analysis

As we can see from the above graph, Robinson tf (Okapi tf) performs better than Raw tf. The reason for this is the verbosity factor, document length, which is employed while calculating the score for Okapi tf. The verbosity factor while calculating the scores in Okapi tf is used to normalize the effect of a high term frequency in a very large document.