Information Retrieval Project 3:

1. **Problem Statement:**
   To create and index of the CACM collection, together with code replication functionality of the Lemur interface used in Project 2.

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**
   a. Index Creation (indexmain.py):
      The main function of this python module will output 3 files:
      DocMapFile: This file contains the document relevant information: docid (internal document id), docname (external document id that is used for evaluations), doclen (length of the document)

      So, the index creation steps are as follows:

      - We parse every document till **the line before the line CAXXXXXX <time stamp>** and ignore the column of numbers below it as well. The line with the CAXXXXXX is ignored because it does not provide any useful information regarding the document.
      - For every document, we will ignore all the stop word and stem the rest of the words using the porter stemmer. (note that we will use the same stop word list and stemming algorithm for our query processing as well)
      - We will use data structures to store the total words, unique words and the document length of a document. We also use a map structures to remember the document term frequency statistics.
      - After all the documents have been parsed then calculate the corpus statistics like number of documents, total number of terms, the number of unique terms and the average document length. We will store these values in a list object and pickle it (serialize it), so that it can be used by our retrieval models.
      - So, after the documents have been parsed, we will have an inverted index like structure created using a map whose key is a term and value is the list of documents in which the term occurred with the respective term frequencies.
      - Using the map structure created in the above step, we then organize results into 3 files:
        DocMap: to store the document relevant information like the internal document id, the external document id or the document name and the length of the document.
        TermFile: to store the term id, term, the document frequency of the term, the length of the term and the corpus frequency of the term.
        IndexFile: to store the mapping of the term id and document id along with the term frequency.

b. Running the cacm queries on the 5 retrieval models implemented in project 2:
   - Query Processing (Parsing the cacm.query file):
     Read each and every query id and query string and process the query string in a similar manner in which we processed the documents while building the index.
     Also, as we mentioned earlier use the same stop word list to remove common words from the query string. Then, apply the Porter stemmer to stem a query term to its respective stem class.
     Then, we pickle (serialize) the query map object which has the query id to query list of query term to query term frequency mapping so that we don't have to read the queries again and again.

   - Running queries on our retrieval models (models.py):
     For running the queries on our retrieval models, we build the query term document statistics mapping information and the corpus or collection information for the terms in the query.
     This mapping information is built from the indexing files we built in part a.
     This information is stored in the 'queryTermDocStats' and 'collectionTermStats' objects which are also serialized so that we don't have to read the index files again and again on each evaluation run. However, deleting the object file will cause the program to read the new index files from the current file directory. This computing method is similar to the one adopted in Project 2.

     Then, we run the query mapping information on our retrieval models. The algorithm to rank the document for a particular retrieval model is the same as in Project 2.

c. Evaluation:

The results are stored in files named in the format <retrieval_model>-results-<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- cacm.rel

4. **Retrieval Model Parameters:**
   a. Smoothing factor in Jelinek-Mercer language model: 0.2
   b. BM25: k1 = 1.2, k2 = 100 b = 0.75

5. Results:

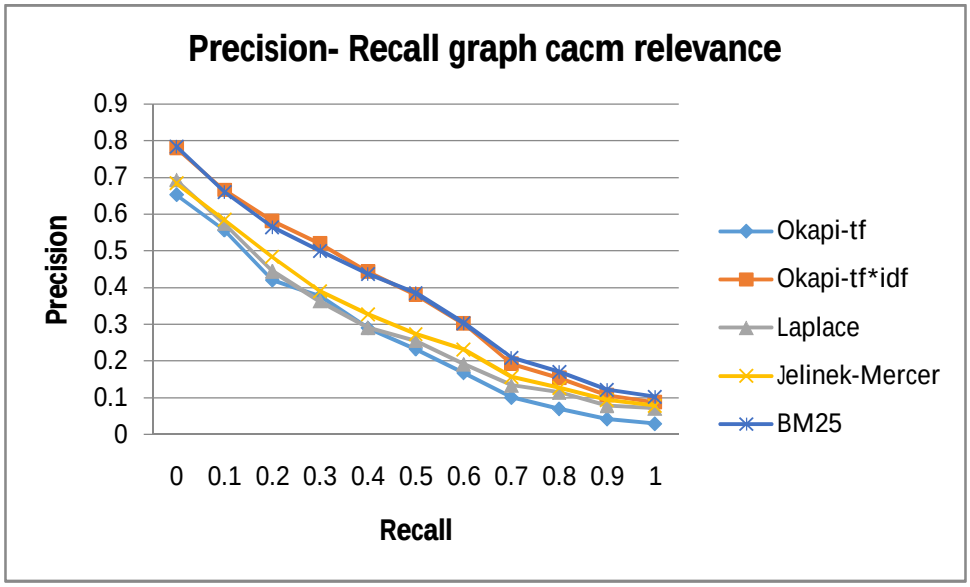
5.1. Using the porter stemmer and the stop word list from Project 2.

Retrieval Model	MAP (Mean Average Precision)	R- Precision	P@10	P@30
Okapi-tf	0.2553	0.2622	0.3019	0.1737
Okapi- tf*idf	0.3631	0.3792	0.3596	0.2160
LM Laplace	0.2535	0.2610	0.2865	0.1551
LM Jelinek Mercer	0.3322	0.3277	0.3231	0.2058
BM25	0.3626	0.3637	0.3519	0.2154

Corpus statistics:

Total number of documents: 3204
Total number of terms: 139358
Unique number of terms: 8868
Average document length: 43.50
Size of IndexFile after removing the stop words from the documents 1,278,004 bytes
Time taken to create the index: 11997 milliseconds
Average query length: 14.3125
Execution time for all models: 8969 milliseconds

Precision and Recall Graph:



Analysis: As the above Precision- Recall graph shows, the okapitf-idf and the BM25 retrieval models perform very well, which is quite similar to results we obtained for these retrieval models for the lemur corpus.

Also, the Okapi tf and the Laplace smoothing model performed strongly initially and had sharp drops in the precision levels for the lemur corpus, but for the cacm corpus we do not see any sharp drops in the precision levels for these two models and the slope is comparatively not big.

5.2. Removing stop words from the query, but not removing them from the document:

Retrieval Model	MAP (Mean Average Precision)	R- Precision	P@10	P@30
Okapi-tf	0.2506	0.2652	0.3000	0.1750
Okapi- tf*idf	0.3519	0.3568	0.3558	0.2160
LM Laplace	0.2547	0.2599	0.2865	0.1564
LM Jelinek Mercer	0.3220	0.3221	0.3269	0.2051
BM25	0.3562	0.3597	0.3481	0.2141

The size of the index is 1,734,339 bytes and the time taken to build the index was 10943 milliseconds.

Corpus statistics:

Total number of documents: 3204
Total number of terms: 215201
Unique number of terms: 9088
Average document length: 67.167

Average query length: 14.3125
time taken for executions of all models: 9222 milliseconds

Analysis:

Compared to the statistics we obtained in the scenario in which we perform stopping on the index as well, in this scenario our execution time is a little higher as the size of the index has increased.

Also, the precision levels of the models is a little less since the corpus level statistics include the stopping words as well. However, it is not as bad as the below scenario in which we do not do stopping on the index as well as the queries.

Compared to the scenario in which we do stopping and stemming both, the size of the index is large. This because now we index additional terms too. The execution time in creating the index is marginally less since we are not checking the each term against the stopping list.

Thus, this scenario can be viewed as a tradeoff between the time to create the index and the space availability for the index. Also, another parameter into consideration can be the precision level of the retrieval model.

5.3. Now, let's try not stopping the query as well as the index

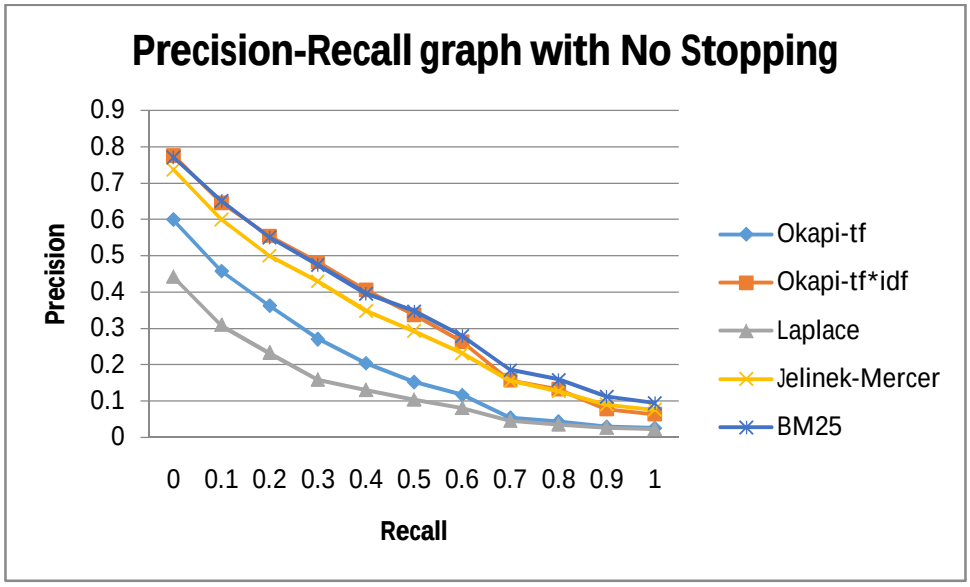
The size of the index is 1,734,339 bytes and the time taken to build the index was 10943 milliseconds. This is the same index as used in the above scenario

Corpus statistics:

Total number of documents: 3204
Total number of terms: 215201
Unique number of terms: 9088
Average document length: 67.167

Average query length: 21.61
Execution time for all models: 17807 milliseconds

Retrieval Model	MAP (Mean Average Precision)	R- Precision	P@10	P@30
Okapi-tf	0.1944	0.2247	0.2519	0.1391
Okapi- tf*idf	0.3386	0.3458	0.3558	0.2071
LM Laplace	0.1274	0.1448	0.1577	0.1013
LM Jelinek Mercer	0.3062	0.3079	0.3192	0.1955
BM25	0.3454	0.3588	0.3404	0.2064



Analysis:

- As we remove stopping, the number amount of space required by the index increases.
- We also do not apply stopping to our query strings to check the precision levels for our retrieval models and the amount of time required to compute the results. Since, the stop words were not removed from the query strings, the average query length of the query string has increased. Also the processing time has increased to 17 seconds compared to 8 seconds for stopped query strings.
- As we can see from the above table, the precision levels have reduced due to lack of stopping.
- Comparing with the results with those in Project 2, we can see that the performance of the Laplace model is consistent, i.e., its precision value dropped considerably when stopping was removed and the precision-recall graph shows a similar curve for the Laplace smoothing model.
- The Okapi tf*idf model and the Jelinek-Mercer model performs in a similar fashion for both project 2 and project 3 in case we do not use stopping in indexing and in query processing.
- However, one interesting point that we can deduce from the results is that, in project 2 when we did not incorporate stopping , the precision levels of the BM25 model plummeted. But, in project 3, without stopping the BM25 model performs fairly well. The reason for this good performance of the BM25 model is the size of the corpus in project 3 compared to that in project 2.

  The Inverse Document Frequency computation in case of project 2 that a huge impact because of the massive amount of documents of the trec collection. So, having a common word in the query would bring down the precision levels considerably. On, the other hand in project 3, since the number of documents is less, so the impact of a common word in the query string on the precision levels is relatively less.
