INFORMATION RETREIVAL

Final Project

Search Engine

Ankit Gadoya

12/10/2013

Contents

How the system works:

The project is divided into two phases – parsing the CACM collection and creating the index, and calculation of data based on the models.

Working of CACM parser:

- Read all html files and create a single text file which will only have all the text as it is displayed on the browser. We add a counter to separate each of the contents of the html files. By placing this counter before writing each of the html file into text file helps us in further steps while reading the text file
- We provide the output obtained from step-1 as input to this step. Here we focus on removing the numbers that are present at the end of each of the files as it has been mentioned in the problem statement that we can ignore the column of numbers that are present at the end of each file
- For performing stemming operations we used Porter Stemmer algorithm .The stem() method available in the python implementation
- The required data is read from each file in the CACM collection, stopped, stemmed, and inserted into a dictionary with the documentID as the key.
- Every unique term in this dictionary is assigned a unique ID and a temporary file is created with the termID as the name. This file contains the documentID and the frequency of that term in that document. These files are stored in the folder “terms”.
- A file “documnet_information.txt” is created containing document id, the number of unique terms present in that document and the total number of terms in that document.
- A file “term_information.txt” is created which contains a term, its ID, its frequency in the corpus, the number of documents containing it and its offset value.
- Each of the temporary files is read, the data in them is collected and put in the file “inverted_index.txt”. This file contains the termID, and for each termID, the documentID and the frequency of that term in that document.
- “Corpus_inofrmation.txt” is created which contains the number of unique terms in the collection, the total number of terms in the collection, total number of documents in the collection, and the average document length.

Model computation:

- The file cacm.query is read and data is stored in a dictionary with the query number as the key.
- An empty list is created and for each term in the cacm.query (stopped and stemmed), we get the ID corresponding to that term (from file “term_information.txt”) and store its

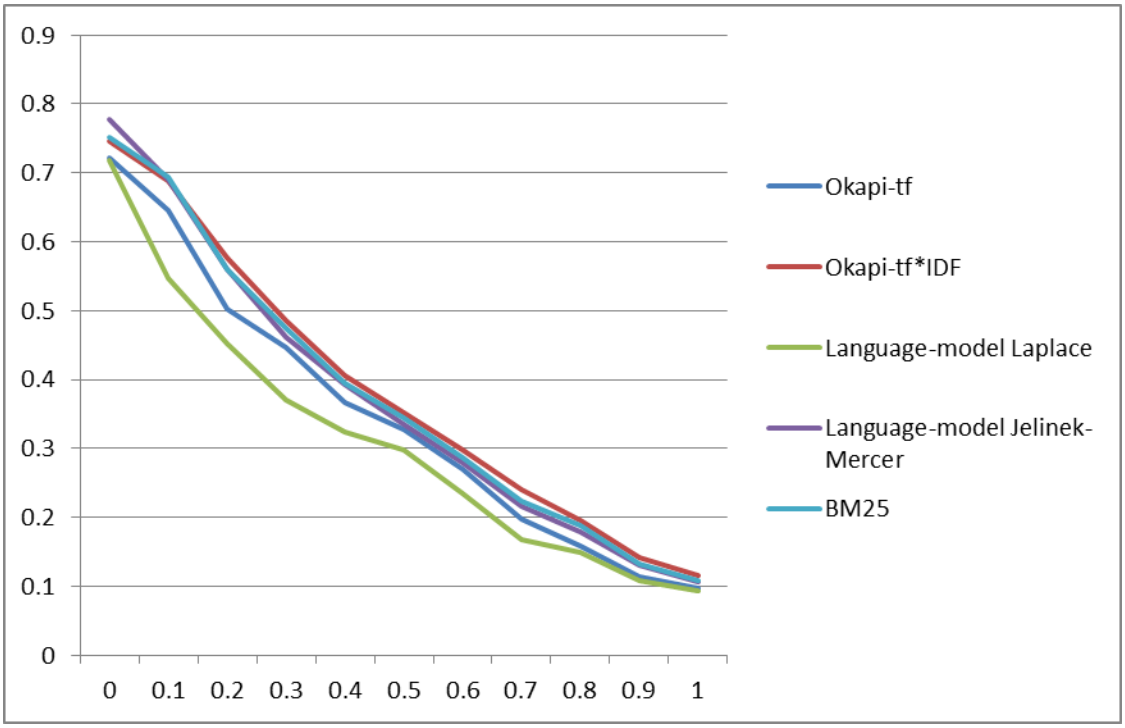
ctf and df. Using the offset value, we read data from “inverted_index.txt” and store the documentID, document length (from “document_info.txt”) and the frequency of that term in that document (tf). The list thus is like this [ctf, df, documentID, document length, tf]. This is similar to the data returned from lemur database when queried to in Project2.

- We compute the scores for documents using the five models used in Project2.
  The models are:
  - Vector Space Model using Okapi tf
  - Vector Space Model using Oakpi tf times IDF
  - Language Model with Laplace Smoothing
  - Language Model with Jelinek-Mercer Smoothing
  - BM25 Model
- The top 1000 documents are computed based on the scores, stored in a file in the required format and given to the cacm.rel file for evaluation.

Results:

	Mean Average Precision	Precision at 10 docs	Precision at 30 docs
Vector Space Model - Okapi tf only	0.3326	0.3538	0.1962
Vector Space Model - Okapi tf times IDF	0.3708	0.3731	0.2160
Language Model with Laplace Smoothing	0.2908	0.3000	0.1667
Language Model with Jelinek-Mercer Smoothing	0.3527	0.3500	0.2141
BM25 Model	0.3620	0.3827	0.2103

Graph of Interpolated Recall - Precision Averages comparing all the five models:



From the above table, it can be seen that BM25 and Vector Space Model-Okapi tf time IDF gives very much similar performance.

BM25 is a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of the inter-relationship between the query terms within a document (e.g., their relative proximity)

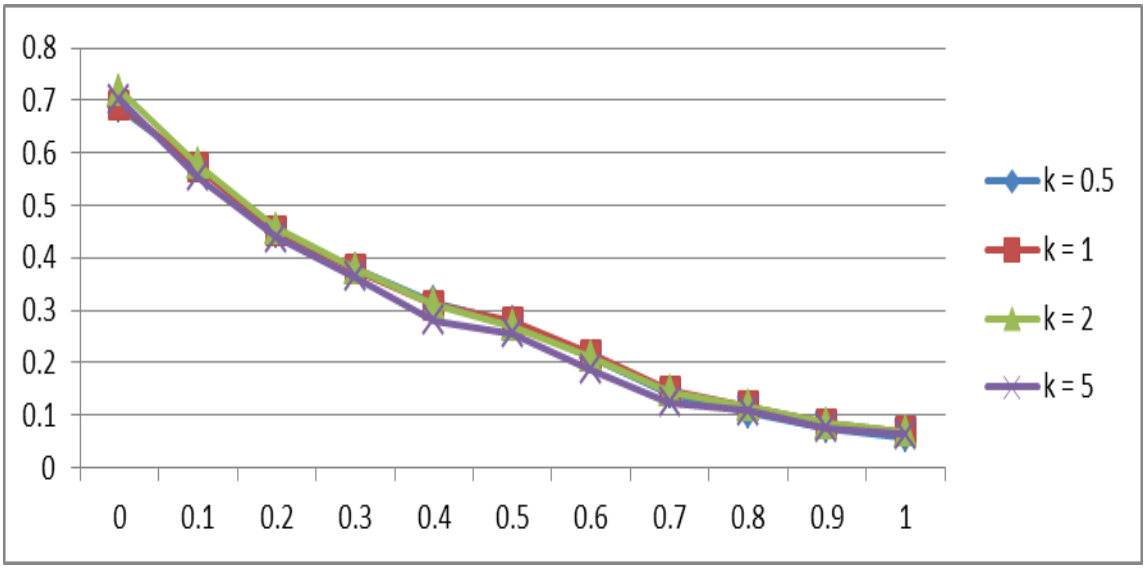
Vector Space Model- Okapi tf times IDF performs better than just Vector Space Model – Okapi.

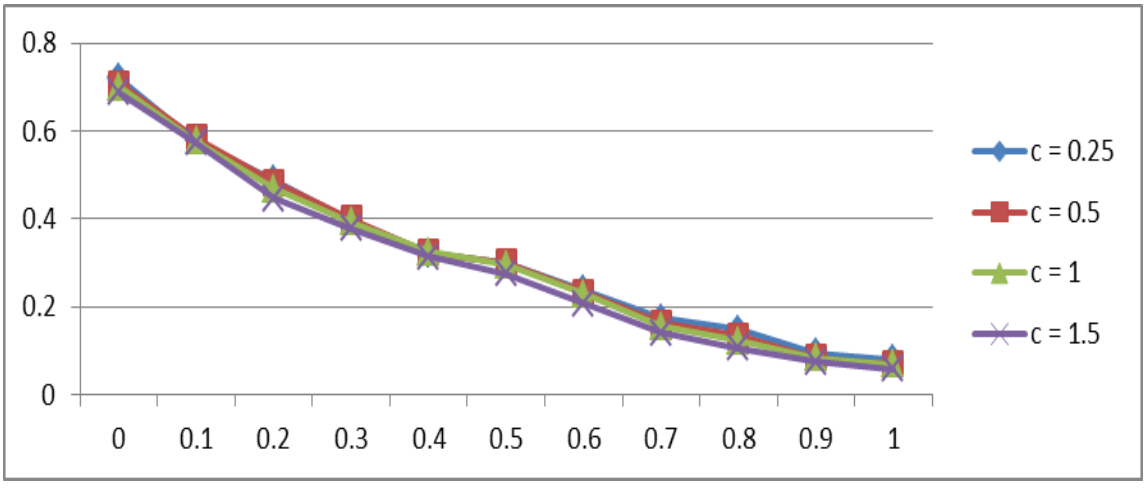
Since stemming is performed, the number of relevant documents is more and better precision is obtained.

Vector Space Model Okapi tf only:

Value of k at c = 1.5	Mean Average Precision
0.5	0.3326
1	0.3331
2	0.3352
5	0.3314

Value of c at k = 0.5	Mean Average Precision
0.25	0.3053
0.5	0.3019
0.1	0.2928
1.5	0.2791

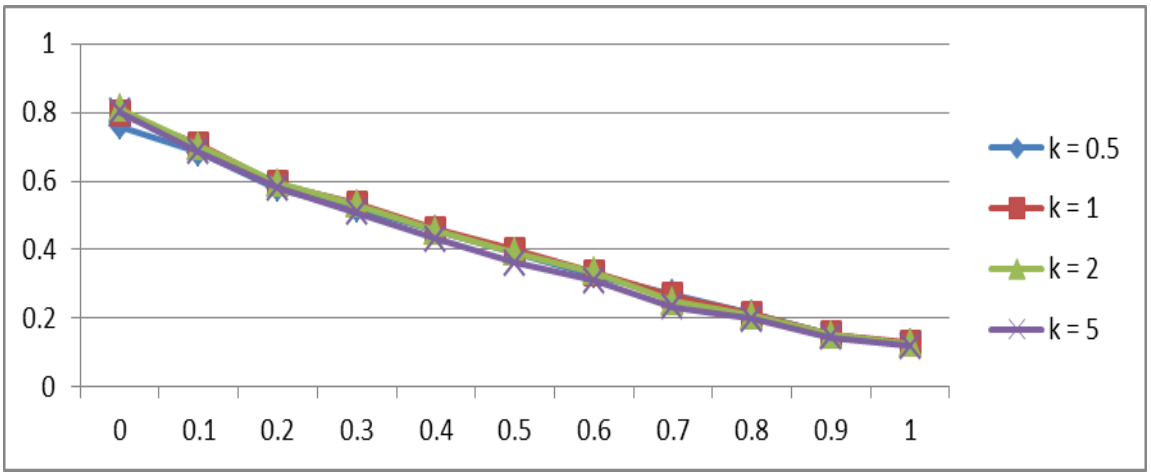


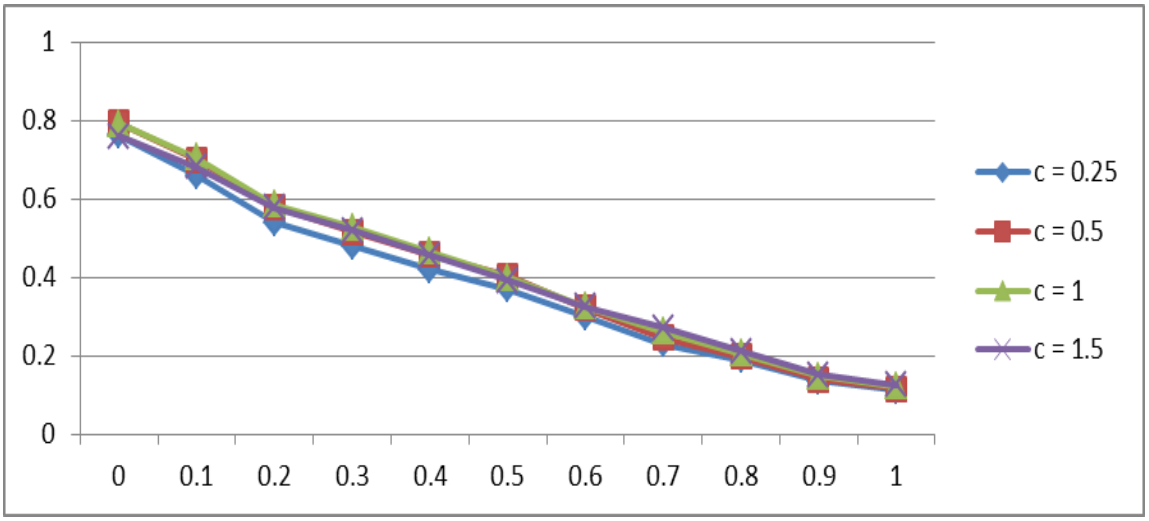


Vector Space Model Okapi tf times IDF

Value of k at c = 1.5	Mean Average Precision
0.5	0.3702
1	0.3812
2	0.3846
5	0.3788

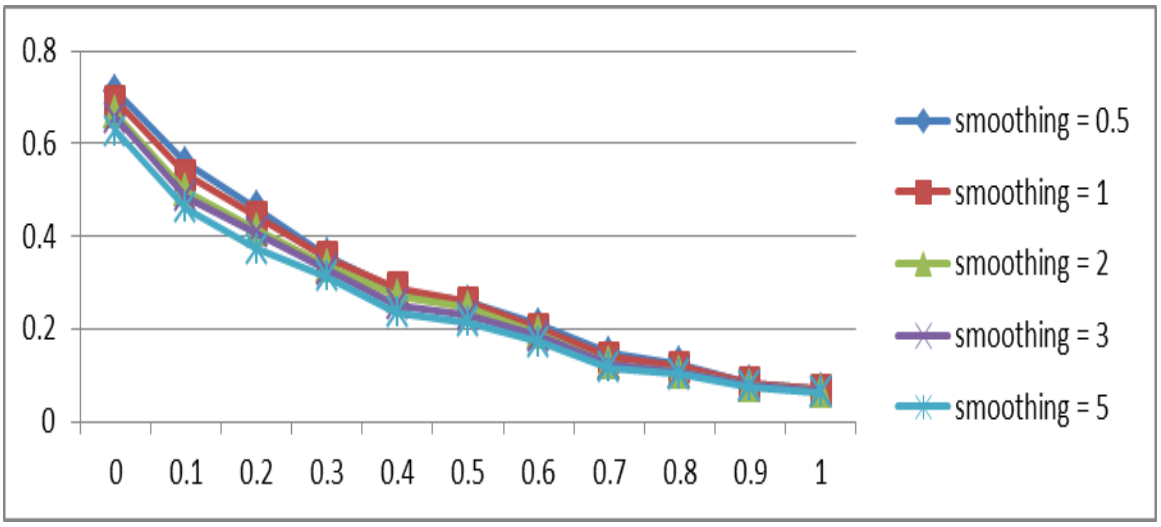
Value of c at k = 0.5	Mean Average Precision
0.25	0.3624
0.5	0.3848
0.1	0.3934
1.5	0.3872





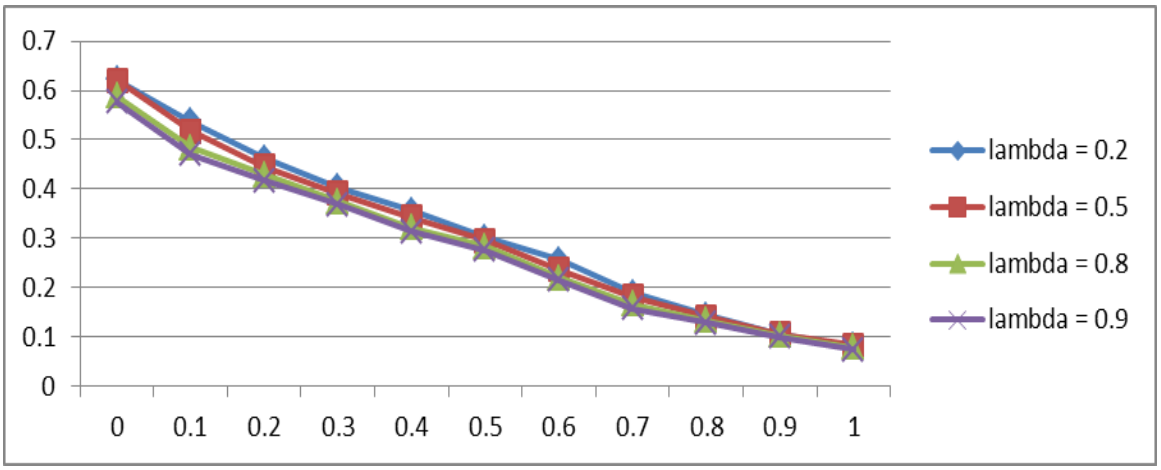
Language Model with Laplace smoothing

Value of smoothing	Mean Average Precision
0.5	0.2908
1	0.2804
2	0.2666
3	0.2529
5	0.2471



Language model with Jelinek Mercer smoothing

Lambda	Mean Average Precision
0.2	0.3527
0.5	0.3238
0.8	0.2908
0.9	0.2816



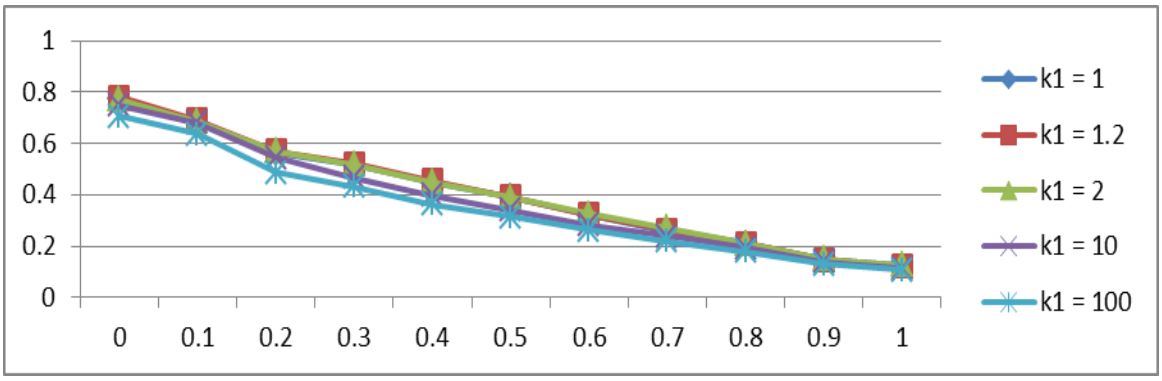
As in project 2, Jelinek-Mercer with lambda = 0.2 gives high precision and the precision decreases as lambda increases.

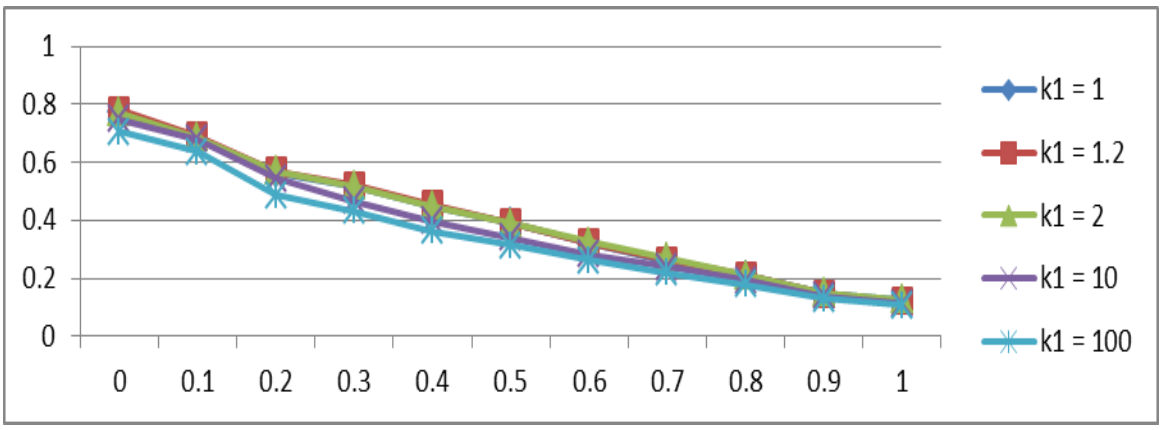
BM25:

Value of k1 for b = 0.75	Mean Average Precision
1	0.3620
1.2	0.3685
2	0.3690
10	0.3568
100	0.3268

Value of b for k1 = 1.2	Mean Average Precision
0.0	0.3446
0.25	0.3704
0.50	0.3838
0.75	0.3821
0.90	0.3732
1.0	0.3606

Changing the value of b does not affect precision much, but lower values of k yield better precision values.





Extra Credit:

- Inverted index file by removing the stop words.

Analysis

1. Time taken to create index: 30 seconds
2. Number of unique terms: 8455
3. Total number of terms: 114896
4. Total number of documents: 3204
5. Average document length: 35

- Inverted index file by keeping stop words.

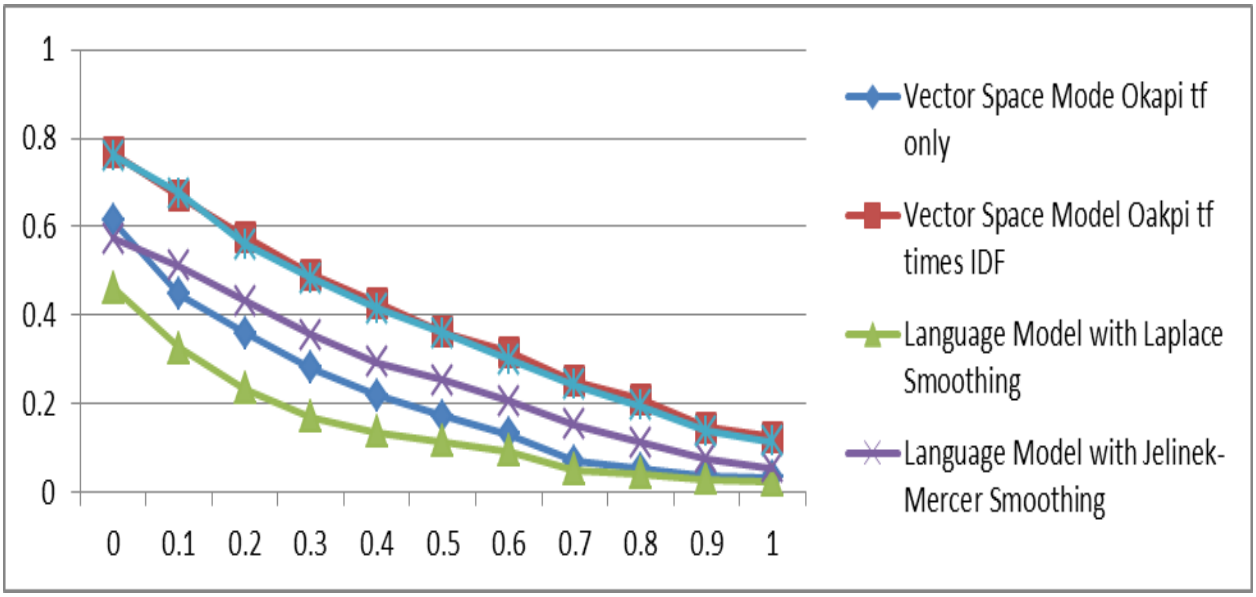
Analysis:

1. Time required to create index: 65 seconds
2. Number of unique terms: 8884
3. Total number of terms: 212468
4. Total number of documents: 3204
5. Average document length: 66

Result without removing stop words.

	Mean Average Precision	Precision at 10 docs	Precision at 30 docs
Vector Space Model - Okapi tf only	0.2203	0.2653	0.1424
Vector Space Model - Okapi tf times IDF	0.3357	0.3766	0.2137
Language Model with Laplace Smoothing	0.1563	0.1629	0.1046
Language Model with Jelinek-Mercer Smoothing	0.2260	0.2885	0.1722
BM25 Model	0.3667	0.3591	0.2250

Graph of Interpolated Recall - Precision Averages comparing all the five models:



Analysis:

a. BM25 and Vector Space Model 2 give the best precision values.

b. Language model with Laplace Smoothing gives the worst precision value.

c. Better precision values are obtained when stop words are removed from the query compared to when stop words are not removed from the query.

d. Building an index by removing or not removing the stop words does not affect the precision, but removing the stop words from the query has a lot of effect.

Some additional tests:

- Query processing time on larger index when stop words are removed from the query: 9 seconds
- Query processing time on larger index when stop words are not removed from the query: 17 seconds
- Query processing time on smaller index when stop words are removed from the query: 7 seconds
- Query processing time on smaller index when stop words are not removed from the query: 8 seconds

Observations:

a. Double the time is required when stop words are not removed while building the index and while processing the queries.

When stop words are removed from the query, size of index does not matter. This is because the offset is used to read the index file instead of performing a linear search on the index file