SEARCH ENGINE DESIGN

CS6200: Information Retrieval

Northeastern University

Fall 2018, Prof. Nada Naji, Prof. Rukmani

Project Members:

Aayushi Maheshwari

Aditya Sharma

Romil Rathi

1. Introduction

1.1 Overview

The goal of the project is to design and build our own information retrieval system, evaluate and compare their performance levels in terms of retrieval effectiveness. This project designs search engines using four distinct retrieval models as mentioned below:

- BM25
- Tf-idf
- JM Smoothed Query Likelihood Model
- Lucene

Along with these four baseline runs, we have three additional runs:

- BM25 retrieval model along with Pseudo relevance feedback query expansion technique
- BM25 with/without stopping, and with/without stemming
- tf-idf with/without stopping, and with/without stemming
- JM Smoothed Query Likelihood Model with/without stopping, and with/without stemming

We have also implemented the Snippet Generation and Query Term Highlighting on the results of BM25. We have also evaluated our work by using these retrieval effectiveness measures:

- Mean Average Precision (MAP)
- Mean Reciprocal Rank (MRR)
- P@K measure where K=5 and K=20
- Precision & Recall
- Precision - Recall Curve for all Queries

1.2 Contribution of Team Members

The detailed contribution of each individual member is elucidated below:

- **Aayushi Maheshwari**: Aayushi was responsible for implementing baseline runs of models BM25 and Lucene. She was also responsible for implementing the query enrichment by implementing the query expansion technique using Pseudo Relevance Feedback. She was responsible for the design choices, experimenting and parameter settings and documenting the outlooks. She was also responsible for writing the Report and Readme files for the project.

- **Aditya Sharma**: Aditya was responsible for implementing baseline runs of model tf-idf. He was also responsible for developing Snippet Generation, Query Term Highlighting, Extra-Credit problem and documenting his design choices.

- **Romil Rathi**: Romil was responsible for pre-processing of query and implementation of baseline runs of model JM Smooth Query Likelihood Model. He designed the search engines which perform stopping and stemming tasks. He was also responsible for implementing the various evaluation measures like MAP, MRR, P@K, Precision & Recall. He analyzed the evaluation results and generated MAP, MRR charts for comparison of the various models, and did query level analysis by generating Precision, Recall graphs for certain queries.

2. Literature and Resources

- **Pseudo Relevance Feedback:** The techniques used for pseudo relevance feedback is based on the textbook.
    - Croft, W.Bruce; Metzler, Donald; Strohman, Trevor Search Engines: Information Retrieval in Practice. Pearson Education 2015
    - Manning, Christopher D; Raghavan, Prabhakar; Schutze Hinrich An Introduction to Information Retrieval. Cambridge England: Cambridge University Press 2009
    - http://nlp.stanford.edu/IR-book/pdf/09expand.pdf

  The above sources were used to decide some of the parameters involved in Pseudo Relevance Feedback.
    - **Base Model:** We used BM25 as baseline run as the formula has terms which account for relevant documents and term frequency in relevant documents.
    - **N:** Most of the search engines and experiments suggested using 10 as a value for top N documents.
    - **E:** The range of E lies between 10 to 20. We took the median value 15 to expand the query words. E denotes the number of words added to query.
    - **alpha, beta and gamma:** Set of these values also differ largely but we used 1, 0.75 and 0.15 which is one of the most common settings.

- **Snippet Generation:** The techniques used for snippet generation is based out of 2 research papers.
    - Hans Peter Luhn. 1958. The automatic creation of literature abstracts. IBM Journal of research and development 2, 2 (1958), 159–165.
    - Qing Li, K. Selcuk Candan, Yan Qi. January 2007. Extracting Relevant Snippets from Web Documents through Language Model based Text Segmentation

  Luhn's paper [1] was used as the base algorithm for snippet generation. Some ideas from [2] was used to expand the query to find more words that can be considered significant word based on a relevance language model. [2] finds the region in the document that is most relevant to the topic and extracts sentences from that region to create snippet. However, because most CACM documents are small, we assume that the entire document is relevant to the topic and pick significant words from the entire document. The algorithm implemented can be viewed as Luhn's approach with query refinement using relevance language model.

3. Implementation and Discussion

3.1 Implementation

- **Indexing and Pre-Processing**:
  - Files used - pre_processing.py, inverted_index.py and query_processing.py
  - For parsing each HTML file, we used BeautifulSoup to extract the text for corpus.
  - Pre-Processing includes lower casing the documents, removal of punctuations and white spaces by default.
  - Cleaned Data was written to separate file for each document.
  - Inverted Index was generated by finding unigrams in each file and saved to a JSON file.
  - Query file was also pre-processed by using case-folding, removal of punctuations and were parsed into a file 'cacm.query.parsed.txt'

- **Retrieval Models**:
  - We implemented TF-IDF model, BM-25 Model and Smoothed Query Likelihood Model by using the information and formulae provided in the textbook.
  - Lucene was implemented by using StandardAnalyzer like we did in assignment.
  - We found the top-100 ranked documents and stored them in a file sorted by their relevance score from respective models

- **Pseudo Relevance Feedback**

  For performing pseudo relevance feedback using Rocchio's Algorithm (in BM25_PRF.py), perform the following steps:

  - Normal retrieval is performed with the initial query using BM25 as baseline Model.
  - The top k documents (in our case k = 50) from the results are included in the relevant set of documents and the rest of the documents are non-relevant set.
  - Rocchio's algorithm is used to generate the new query.
    - Initial query vector with the tf scores and aligned with the inverted index terms is generated.
    - Relevant set of document vector is generated.
    - Non-relevant set of documents vector is generated.
    - Then modified query is generated by the Rocchio's formula.
  - The top 15 terms with the highest weight and which are not present in the query are appended to the original query.
  - Normal retrieval with BM25 and relevance information is performed with the new query and the search results are generated.

- **Snippet Generation**

  The snippetGen.py file generate snippets of the search results, it shows the name of the file, then in following line it gives the small text summary of where the content appeared and highlights the query words.

It generates the results on the command line and a html file, on command line it highlights text with blue color and in html file it shows the same results but here it makes the matching query terms bold. This is one of the most common techniques used to how the result snippets.

- **Reading Lucene top doc for each query:** In previous steps we generated top documents for each query using Lucene and now we read those results files where top 100 documents are ranked.
- **Generating snippets for each query:** We read each query and generate snippets by finding locations of where the term words appear. Then we get the section of text with specific length and highlight the query terms which appear in the text for that specific html file from cacm corpus.
- **Computing the location of snippet:** Now we traverse through inverted index to find files where query terms appear, then we get the nearby text and print the query terms along with the text and highlight query words.
- **Showing results:** In this step we save the results in a html file and print it on terminal line as well, on terminal query terms are highlighted by blue color and in html they are set to bold.

- **Evaluation**:
  The evaluation.py file computes the various effectiveness measures like Recall and Precision for all queries at each document occurrence in the result list.

  - **Gold_relevance_file function** was used to get gold results,
  - **Model_relevance_file** was used to get model's results, evaluate function was used get Precision-Recall, P@5, P@20, MAP and MRR metrics.
  - **Write_precision_recall_table** is used to write these values to a file and generate interpolated values of Precision for Recall levels.
  - Average Precision values against standard recall level were used to plot Precision-Recall curve for all 8 models.

- **Extra Credit: Proximity search**

The proximity.py file BM25 scores while considering proximity of terms in the query with the terms in documents. Documents where terms appear in similar order and are closer to each other are given a higher a higher score.

It generates a file BM25_proximity.txt where all queries are listed followed by top 100 documents in order of relevance of term proximity and BM25 scores. User has to select where they want to do exact match or Best match or proximity match with N (default = 5).

- **Calculate BM25 score:** For each query we calculate BM25 score for all documents.
- **Calculate proximity score:** For each query and all docs in the inverted index that related to it, we calculate a proximity score and add it with the BM25 score with a smoothing factor. The equation used for this is: (1-lambda)*BM25 + lambda*QueryScore.
- **Computing proximity score:** For each query term, we find the associated documents and give it a score based on how close the query-terms are to each other in the document. If

terms are far away score is lower as compared to a high scoring documents where terms are closer.

- **Showing results:** In the end we write the results to a file, where we list each query followed by the top 100 documents listed by score of BM25 and proximity.

3.2 Discussion

- **Query by Query analysis**

We performed query-by-query analysis for the BM25 results with and without stemming. This is because stemming improved BM25's results significantly than it did for any other models.

Query 1: "**portabl oper system**" vs "**portable operating systems**"
Here we found that, one relevant document (as per cacm.rel.txt), CACM-2358.html, was ranked significantly higher for non-stemmed version compared to its stemmed version (42 vs 90). On analysis, again we found, as in query 1, that query term frequencies in the documents, between stemmed and non-stemmed versions, were similar and that stemming has diminished query term importance for this query also.

Corpus frequency in stemmed corpus	Corpus frequency in non-stemmed corpus
"oper" - 572 times	"operating" - 205 times
"system" - 1946 times	"systems" - 701 times

Query 2: "**distribut comput structur and algorithm**" vs "**distributed computing structures and algorithms**"
Here we found that, 2 particular relevant documents (as per cacm.rel.txt), CACM-3043.html was ranked significantly higher for non-stemmed version compared to its stemmed version (5 vs 46) and CACM-3128.html, was not ranked in the top 100 for the stemmed version while the document was ranked 3rd in the non-stemmed version.
On analyzing the term frequencies of the stemmed and non-stemmed queries, in the stemmed and non-stemmed corpus files, we found that they were similar. Hence, we analyzed the corpus frequencies for the query terms and found that the corpus frequency of the stemmed query terms was significantly higher. As we know BM25 does not favor corpus-wide highly frequent terms, these stemmed query terms, were scored relatively lower than their non-stemmed counter parts.

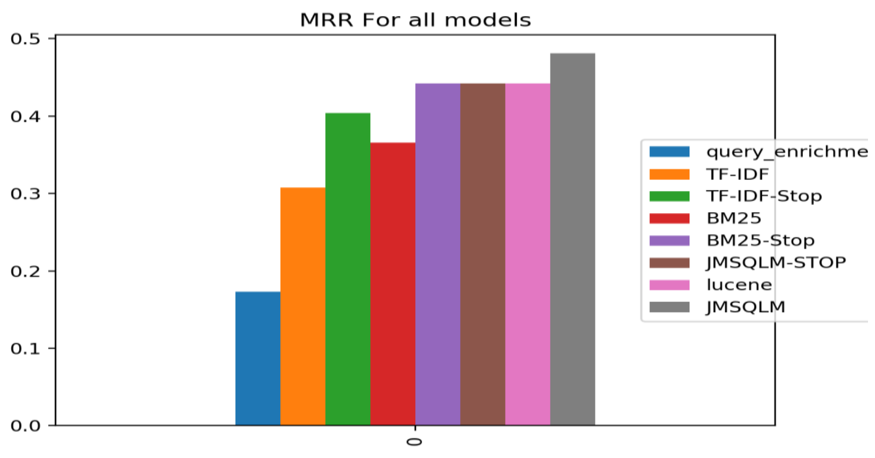
Corpus frequency in stemmed corpus	Corpus frequency in non-stemmed corpus
"distribut" - 240 times	"distributed"- 52 times
"comput" - 1945 times	"computing"- 195 times
"algorithm" - 2015 times	"algorithms" - 375 times

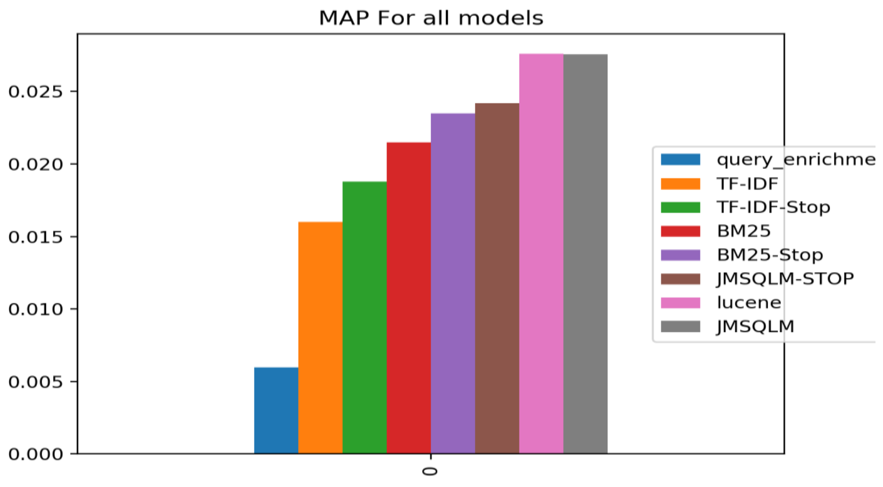
Query 3: “**perform evalu and model of comput system**” vs “**performance evaluation modelling computer systems**”
Unlike the previous query, for this query stemming improves the ranking for relevant documents significantly. For example, relevant documents, CACM-2862.html, CACM-1526.html, CACM-1533.html, CACM1572.html, CACM-2434.html were ranked 13th, 42nd, 49th, 52nd and 92nd respectively for the stemmed version but they did not appear in the top 100 ranked documents for the non-stemmed version. This is because, unlike query 2, the term frequencies of the stemmed query in the stemmed corpus is relatively higher than the term frequencies of the non-stemmed queries in the non-stemmed corpus.

Corpus frequency in stemmed corpus	Corpus frequency in non-stemmed corpus
“model” - 8 times	“modelling” - 0 times
“comput” - 1 time	“computer” - 0 times
“evalu” - 1 time	“evaluation” - 0 times

4. Results

4.1 Global Measures Analysis and P@K Comparison

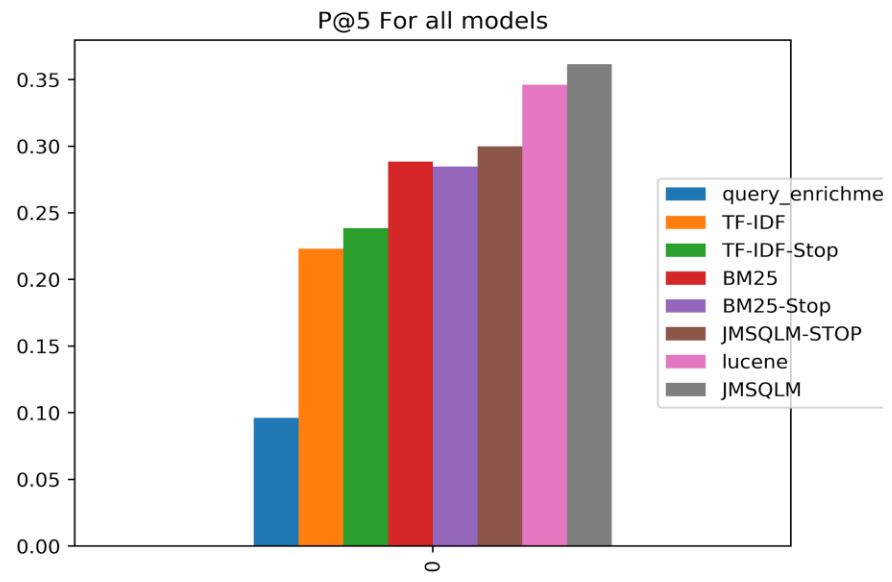


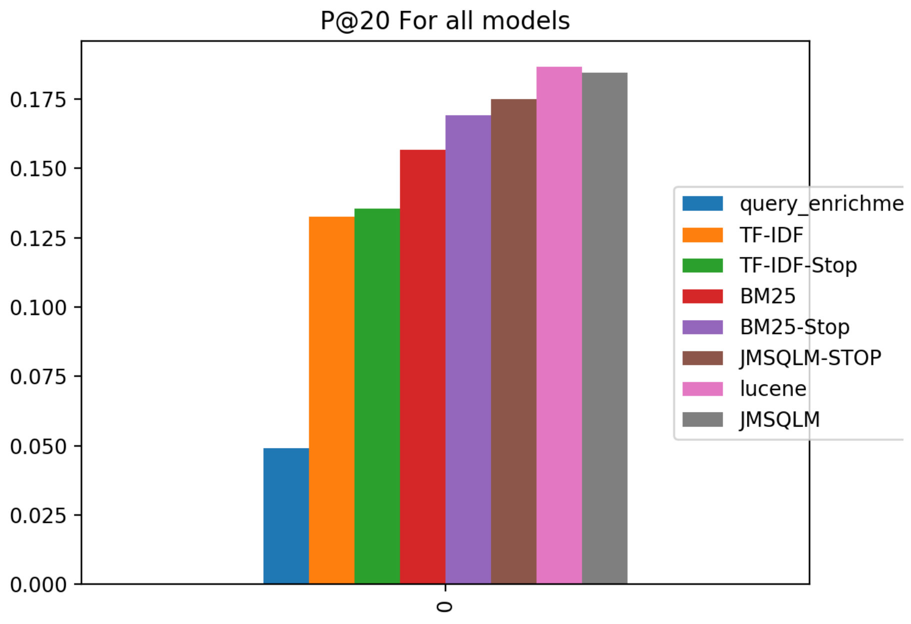


- Here, the metric used to evaluate the models are Mean Average Precision (MAP) and Mean Reciprocal Rank(MRR) for all the runs.

The results can be interpreted as follows:

Measure	Top Models	Worst Models
MAP	JM-SQLM, Lucene, BM-25 Stop	Query_Enrichment, TF-IDF
MRR	JM-SQLM, Lucene	Query_Enrichment, TF-IDF





- Here, Metric used is P@k i.e. Precision at rank K to compare the performance of all models

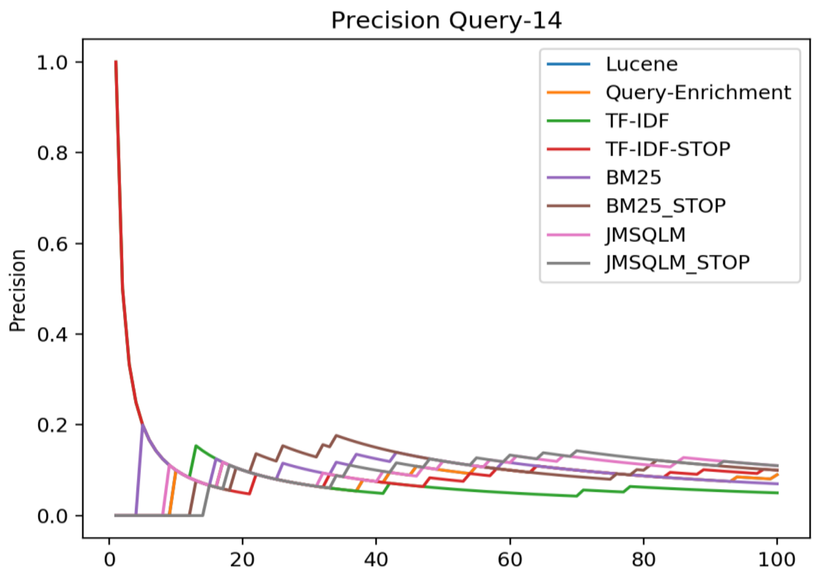
Measure	Top Models	Worst Models
P@5	JM-SQLM, Lucene	Query_Enrichment, TF-IDF
P@20	Lucene, JM-SQLM	Query_Enrichment, TF-IDF

4.2 Query Level Analysis:

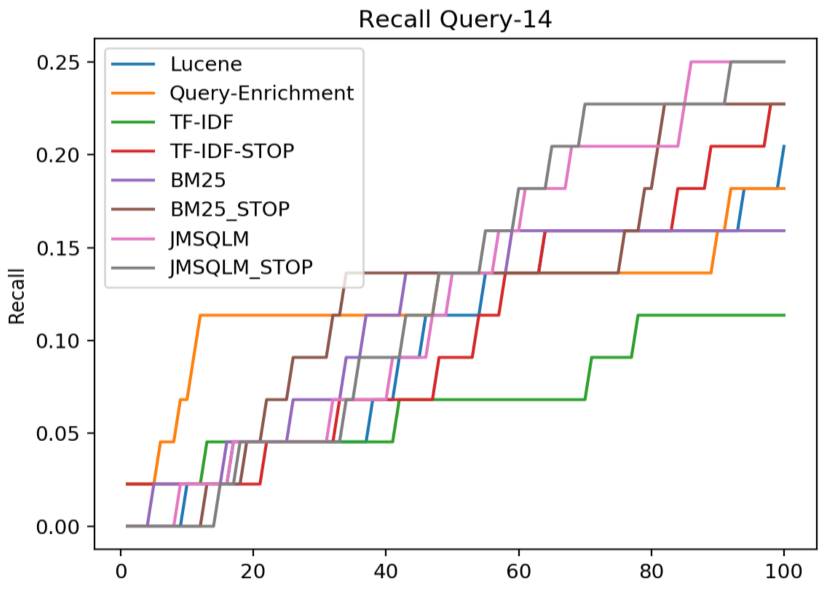
For Query-Level Analysis, we selected 4 queries and used their precision-recall values to evaluate the results by visualizing them.

Graph for Query-14:

- **"find discussions optimal implementations sort algorithms database management applications"**

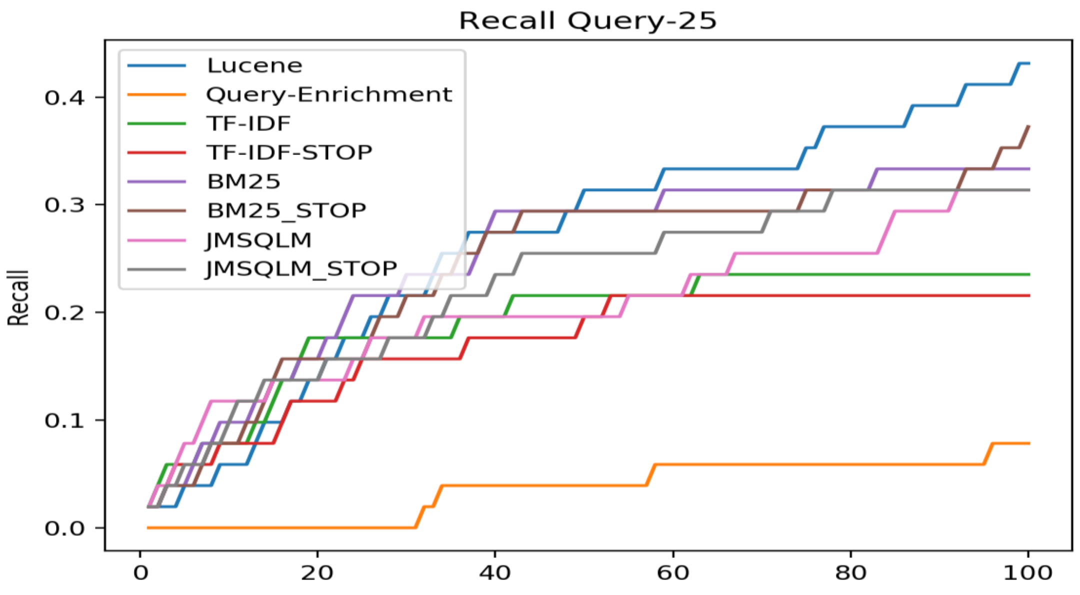


➢ Here it can be seen that TF-IDF has very high precision initially and even in recall it's better than other models but as the rank goes up, though it is not at top but it performs well. JM Smoothing Query Likelihood Model is first to get the highest recall level.

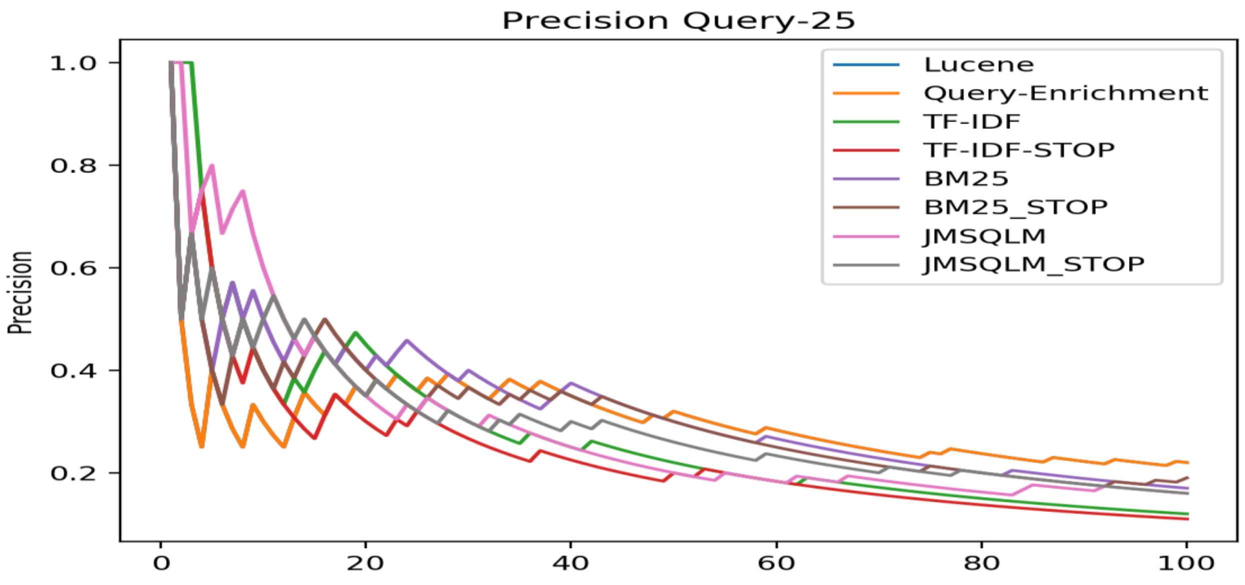


Graph for Query-25:

- “performance evaluation modelling computer systems”

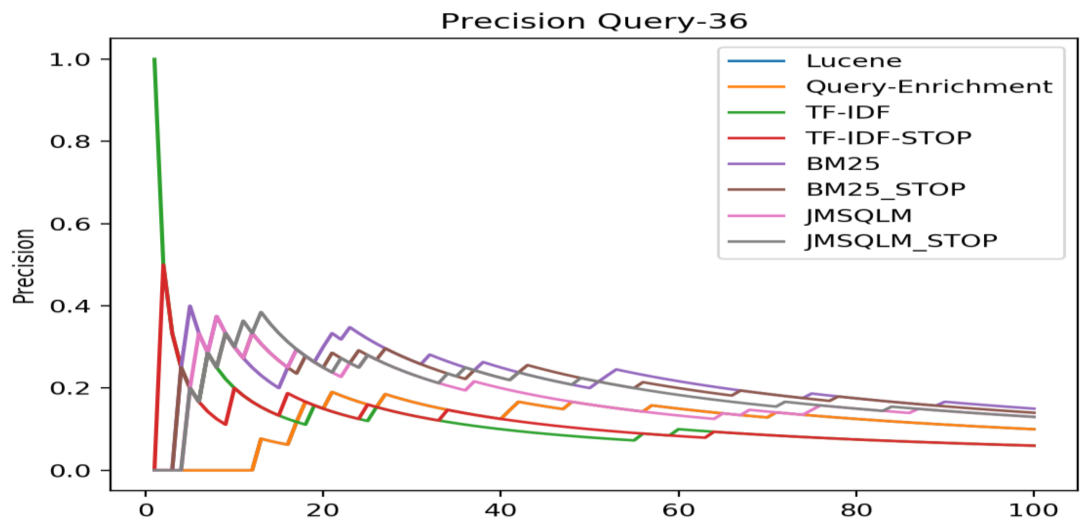


➢ It can be seen that, Lucene, JM SQLM and JM SQLM with stopping have high precision initially. **Lucene** gives the highest recall and outperforms other models while **Query-Enrichment** gives the worst performance followed by **TF-IDF**.

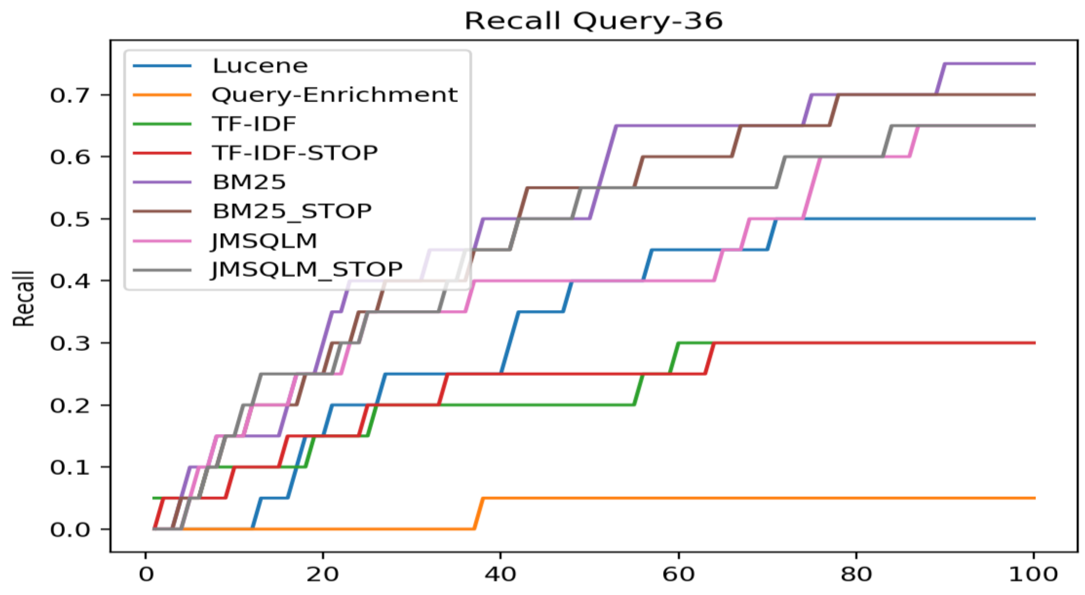


Graph for Query-36:

- "fast algorithm for context-free language recognition or parsing"

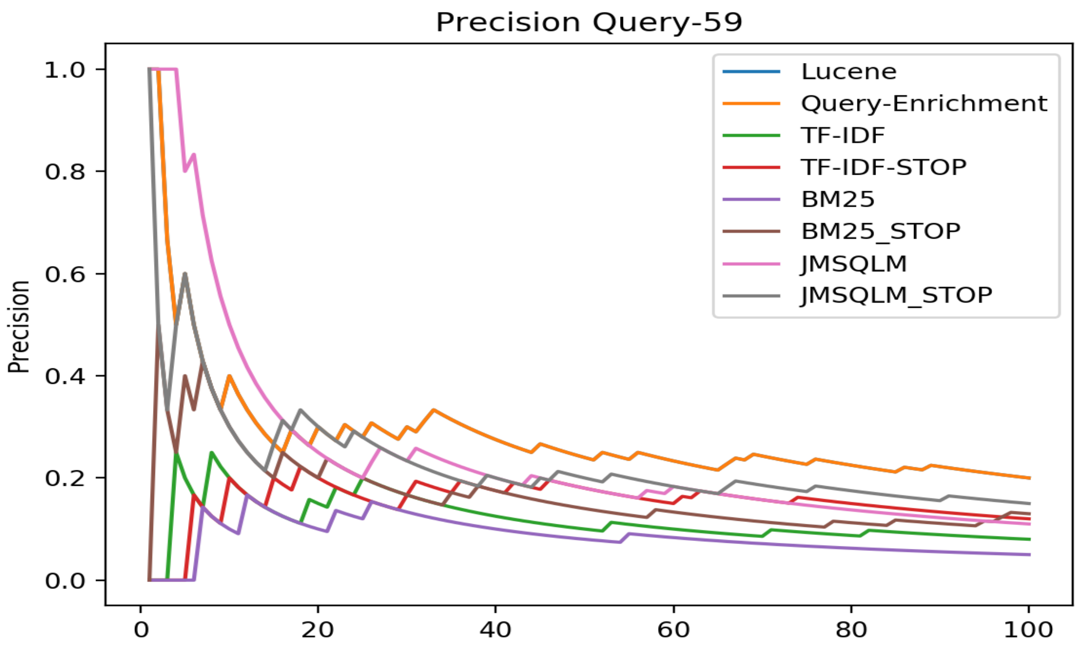


- There's not much difference in precision towards the end for all models and it is very low thus it cannot be used much for comparison while the recall curve shows that **BM25** gets the very **high recall** and can be our best model with BM25 with stopping. **Query Enrichment** is the **worst** performer according to its **recall score** followed by **TF-IDF**.

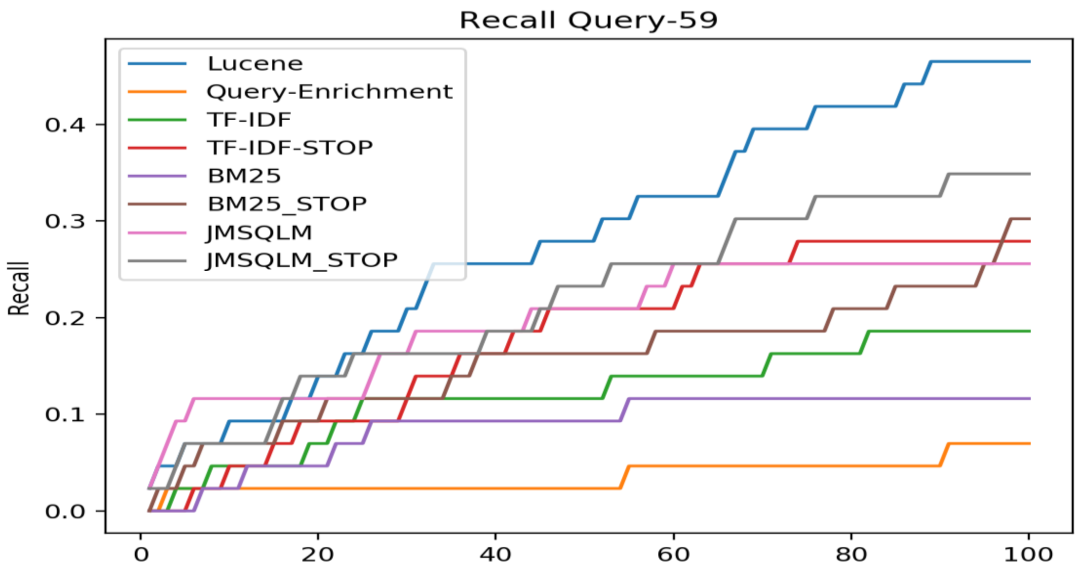


Graph for Query-59:

- “dictionary construction accessing methods fast retrieval words lexical items morphologically related information hashing indexing methods applied English spelling natural language problems”



➢ It can be observed that **Lucene** gives the **best** recall as well as low precision as rank increases among all the models **followed by JM-SQLM** model with stopping and **Query Enrichment** performs the **worst** in all models.



5. Conclusions and Outlook

5.1 Conclusion

- From Global Measure Analysis, we can conclude that Lucene and JM-SQLM models performs the best while Query Enrichment and TF-IDF are worst performers.
- From Query-Level Analysis, we can conclude that Lucene, JM-SQLM and JM-SQLM with stopping models performs very well than other models and Query Enrichment performs the worst here along with TF-IDF model and it's variant.
- Query Enrichment measures have produced the least values for all the evaluation techniques. This may happen due to some unrelated terms in new query.
- Lucene provided best result overall it may be because it is using Boolean model and vector space model with some query normalization.

5.2 Outlook

The search engine we designed meets all the basic requirements for Information Retrieval system and it can be improved by using various measures:

- We can use a different Query Enrichment Technique to see how it performs than the one we used.
- We can make a better system by incorporating other features such as Spell Correction
- We can generate better snippets by styling them so as to improve user experience.

6. Bibliography

6.1 Books

- Croft, W.Bruce; Metzler, Donald; Strohman, Trevor Search Engines: Information Retrieval in Practice. Pearson Education 2015
- Manning, Christopher D; Raghavan, Prabhakar; Schutze Hinrich An Introduction to Information Retrieval. Cambridge England: Cambridge University Press 2009

6.2 Scholarly Articles

- Hans Peter Luhn. 1958. The automatic creation of literature abstracts. IBM Journal of research and development 2, 2 (1958), 159–165. http://courses.ischool.berkeley.edu/i256/f06/papers/luhn58.pdf
- Qing Li, K. Selcuk Candan, Yan Qi. January 2007. Extracting Relevant Snippets from Web Documents through Language Model based Text Segmentation. https://pdfs.semanticscholar.org/7771/0bd1e8aa754e897a53093abd98cc572fd e11.pdf
- http://nlp.stanford.edu/IR-book/pdf/09expand.pdf