# DESIGN AND IMPLEMENTATION OF INFORMATION RETRIEVAL SYSTEMS

*Information Retrieval CS6200 – Spring 2018*

**Prof. Nada Naji**

**Hitesh Verma**

**Pradeepa Gopinath**

**Russia Aiyappa Benjanda Pemmaiah**

# INTRODUCTION

The purpose of this project is to reproduce our understanding of the core information retrieval concepts and processes learnt throughout the course of the semester by implementing and using our very own retrieval systems. The main goal of this project is to design and build our information retrieval systems, evaluate and compare their performance levels in terms of retrieval effectiveness.

We implemented BM25, QLM, tf-idf, Lucene retrieval models along with its variations. The results were then evaluated to analyze and compare their effectiveness.

**Team members' contributions:**

1. Hitesh Verma: tf-idf, query enrichment (pseudo relevance feedback), analysis of the results, extra credit part 1.
2. Pradeepa Gopinath: Tokenizing and indexing, BM25, snippet generation and query term highlighting, extra credit part 2.
3. Russia Aiyappa Benjanda Pemmaiah: Smoothed Query Likelihood Model, Lucene and complete task 3 in phase 1, evaluation reporter (phase 3).

All the team members contributed towards the documentation of the report.

# LITERATURE AND RESOURCES

**Pseudo relevance feedback (PRF)**:

This is a technique where the user is not prompted to identify the relevant documents; the system simply assumes that the top ranked documents are relevant. High frequency words from those are used to expand the query to fetch better results. The expansion terms generated by pseudo-relevance feedback will depend on the whole query, since they are extracted from documents ranked highly for that query, but the quality of the expansion will be determined by how many of the top-ranked documents in the initial ranking are in fact relevant.

**Reasons for choosing PRF as our query enrichment technique:**

- It doesn't require user intervention.
- The words that occur more frequently in the relevant documents are used to expand the query which will result in getting better results.

**Algorithm/Approach used:**

- Pseudo relevance model was run on the BM25 scores for the queries as it had higher Mean Average Precision (MAP) value compared to other baselines.
- Initially a query Q was considered and the top K documents were obtained.
- The most commonly occurring words in those documents were considered after removing the stop words provided in the common_words file.
- Different number of top k documents and top n frequently occurring combinations were considered and evaluated.
- These words were added to the query and this revised query is used to obtain the results.
- The steps involved can be represented as follows:

According to 'Magdy W. and G. J. F. Jones. A Study on Query Expansion Methods for Patent Retrieval PAIR 2011 - CIKM 2011' paper. It was stated that the best evaluations scores were found when 5 most frequently words were considered from top 5 documents

**Trial 1:**

Criteria: Three most frequently occurring words in top 5 documents were considered (K=3, n=5).

- MAP: 0.358
- MRR: 0.572

## Trial 2:

Criteria: Five most frequently occurring words in top 5 documents were considered (K=5, n=5).

- MAP: 0.421
- MRR: 0.672

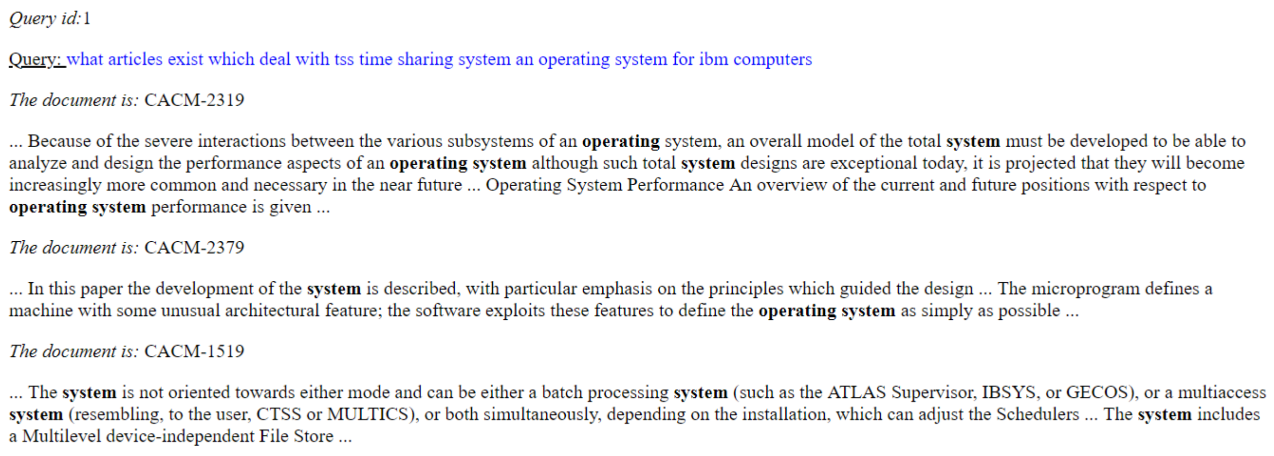
The evaluation clearly backed the claim made in the paper and hence we considered the second trial's approach in the project.

## Snippet Generation:

Successful interaction with the user depends on user's understanding of the results.

- Top ranked documents from BM-25 were considered for snippet generation.
- We obtained valid sentences from the raw documents.
- We have used the Luhn's approach to rank each sentence in the document using a significant factor.
- The formula used to calculate the significant factor score for a sentence is: *Significance score = (number of significant words in the sentence)$^2$/ total number of words*

- Here the significant words are emphasized using bold in the html output files.
- The top two sentences in the document with the highest significant value score is taken for document summary.



*Query id:* 1

Query: what articles exist which deal with tss time sharing system an operating system for ibm computers

*The document is:* CACM-2319

... Because of the severe interactions between the various subsystems of an **operating** system, an overall model of the total **system** must be developed to be able to analyze and design the performance aspects of an **operating system** although such total **system** designs are exceptional today, it is projected that they will become increasingly more common and necessary in the near future ... Operating System Performance An overview of the current and future positions with respect to **operating system** performance is given ...

*The document is:* CACM-2379

... In this paper the development of the **system** is described, with particular emphasis on the principles which guided the design ... The microprogram defines a machine with some unusual architectural feature; the software exploits these features to define the **operating system** as simply as possible ...

*The document is:* CACM-1519

... The **system** is not oriented towards either mode and can be either a batch processing **system** (such as the ATLAS Supervisor, IBSYS, or GECOS), or a multiaccess **system** (resembling, to the user, CTSS or MULTICS), or both simultaneously, depending on the installation, which can adjust the Schedulers ... The **system** includes a Multilevel device-independent File Store ...

Figure 1. Snapshot of results for snippet generation.

# IMPLEMENTATION AND DISCUSSION

This section contains detailed description of the chosen models and design choices made during the implementation of these models. This section also includes query-by-query analysis of the run performed in the task3 of phase1.

## Task1- Phase1:

### ➢ BM25 Model:

It's an effective ranking algorithm based on binary independence model which also includes document and query term weights.

$$\sum_{i \in Q} \log \frac{(r_i+0.5)/(R-r_i+0.5)}{(n_i-r_i+0.5)/(N-n_i-R+r_i+0.5)} \cdot \frac{(k_1+1)f_i}{K+f_i} \cdot \frac{(k_2+1)qf_i}{k_2+qf_i}$$

We have taken K1 and K2 values as 1.2 and 100 respectively.

$$K = k_1((1-b) + b \cdot \frac{dl}{avdl})$$

-dl = document length.

-b = 0.75, avdl= average document length

-$r_i$ = number of relevant documents containing the term i. We have taken $r_i$=0

-R = number of relevant documents for the query. We have taken R=0

-$n_i$ = number of documents containing the term i.

-N = total number of documents in the collection.

-fi = frequency of the term i in the document

-$qf_i$ = frequency of the term i in the query

**Implementation:**

• The list of documents is taken from the inverted index for each term in the query.

• The scores are stored in the dictionary and sorted in the descending order.

## ➢ Tf-idf Model:

This is the simplest model which only considers term frequency and document frequency.

$$Score = (f_i/dl) * (1+ \log (N/(n_i+1)))$$

-dl = document length

-$f_i$ = frequency of the term i in the document

-N = total number of documents in the corpus

-$n_i$ = number of documents containing the term i

**Implementation:**

• The list of documents is taken from the inverted index for each term in the query.

• A table containing unigram tokens with the name of the document is used to get document length for each of the document

• Score for each document for each term is calculated using the above-mentioned formula.

• The document and its respective score is stored as a dictionary and sorted based on the score in descending order.

## ➢ Smoothed Query Likelihood model:

In this model documents are ranked by the probability that the query text could be generated by the language model. Query generation is the measure of how likely it is that a document is about the same topic as the query. Smoothing is included in this to avoid various estimation problem and to overcome data sparsity.

$$\log P(Q|D) = \sum_{i=1}^{n} \log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})$$

-$f_{q_i}$, D = number of times the term i occurs in the document D

-|D| = length of the document

-lambda = 0.35 as given in the problem statement

-$Cq_i$ = number of times the term i occurs in the whole collection of documents

-|C| = total length of all the documents present in the corpus.

**Implementation:**

• The list of documents for each term of the query is taken from the inverted index.

• fqi is calculated for each term with the help of inverted index.

• Document length is found out using the unigram tokens generated

• The score for each document is calculated using the above formula and stored in descending order in a dictionary.

➢ **Lucene Model (using version 4.7.2):**

Lucene is widely used in both academic and commercial search engine applications.

Source: https://lucene.apache.org

Three jar library files were imported:

- Lucene-core-VERSION.jar
- Lucene-queryparser-VERSION.jar
- Lucene-analyzers-common-VERSION.jar

**Implementation:**

• Using “simpleAnalyzer”, we indexed the raw documents.

• The top 100 results for each query is obtained and analyzed.

## <u>Task2- Phase1</u>:

➢ **Pseudo Relevance Feedback Model:**

We have implemented the Pseudo Relevance Feedback model for Query Enrichment using BM25 ranked results.

**Implementation:**

• Select top (k=5) five ranked documents from BM25_ranking result for each query

• We have removed the stopwords in those documents and then fetched top five (n=5) high frequency terms from each document.

• For all those terms not present in individual query, we have expanded the query using those five high frequency terms from each document.

• BM25 is run again on this new enriched query to rank documents to get better results.

## <u>Query-by-Query Analysis</u>:

➢ Analysis for Task 3 using three queries to compare the effect of stemming on ranking:

a) Query: code optimization for space efficiency

Query with stemming: code optim for space effici

Consider below example:

(DocID : Rank)

CACM-1947 :1 (unstemmed version) | CACM-1947 :3 (stemmed version)

CACM-2748 :3 (unstemmed version) | CACM-2748 :1 (stemmed version)

CACM – 1947: Initially CACM-1947 would only compare with files that contain the term “optimization” now after stemming it has to compare with all documents which contain word optimize, optimizations, optimal which are now reduced to stem class ‘optim’ and hence the rank falls for document CACM - 1947.

CACM-2748: Before stemming CACM-2748 had no occurrence of ‘efficiency’ but after stemming the word ‘efficient’ has been stemmed to word effici creating more hits with query term hence its ranking rises.

b) Query: Parallel processors in information retrieval

Query after stemming: parallel processor in inform retriev

Consider below example:

CACM-2714 :1 (unstemmed version) | CACM-2714 :3 (stemmed version)

CACM-1811 :6 (unstemmed version) | CACM-1811 :1(stemmed version)

In the query and in the index all the occurrences of ‘processors’ have been stemmed to ‘processor’ so BM25 with stemming model picks up the documents that have the term ‘processor’, whereas BM25 model without stemming picks up only those documents containing the word ‘processors’. Document 2714 contains 3 mentions of the word ‘processors’ and Document CACM-1811 has 2 mentions ‘parallel-processor’ and ‘parallel-processors’ after stemming, CACM-1811 will have more hits than CACM-2714. So, its ranking goes up after stemming the corpus.

c) Query: Parallel algorithms

Query with stemming: parallel algorithm

Both the query seem to retrieve the same set of documents in terms of the ranking of documents along with some variations in ranking of specific documents. That is because both the stemmed and the unstemmed version of the query is the same, and the only variation is in the query term “algorithms” which is stemmed to “algorithm”.

Consider below example for the variation:

CACM-2714 :9 (unstemmed version) | CACM-2714 :1 (stemmed version)

CACM-2714 had many occurrences of query term ‘parallel’,and has 5 mentions of word ‘algorithm’.

The un-stemmed BM25 would not match ‘algorithm’ with ‘algorithms’ hence the rank of CACM-2714 drops in the unstemmed version.

## <u>Analysis for Extra credit task</u>:

- For the synthetic spelling-error generator- First we are sorting the query terms based on their length and then we choose the top 40% to add errors to it. We are doing 2-3 random swaps for the term by preserving the first and last characters of the term.

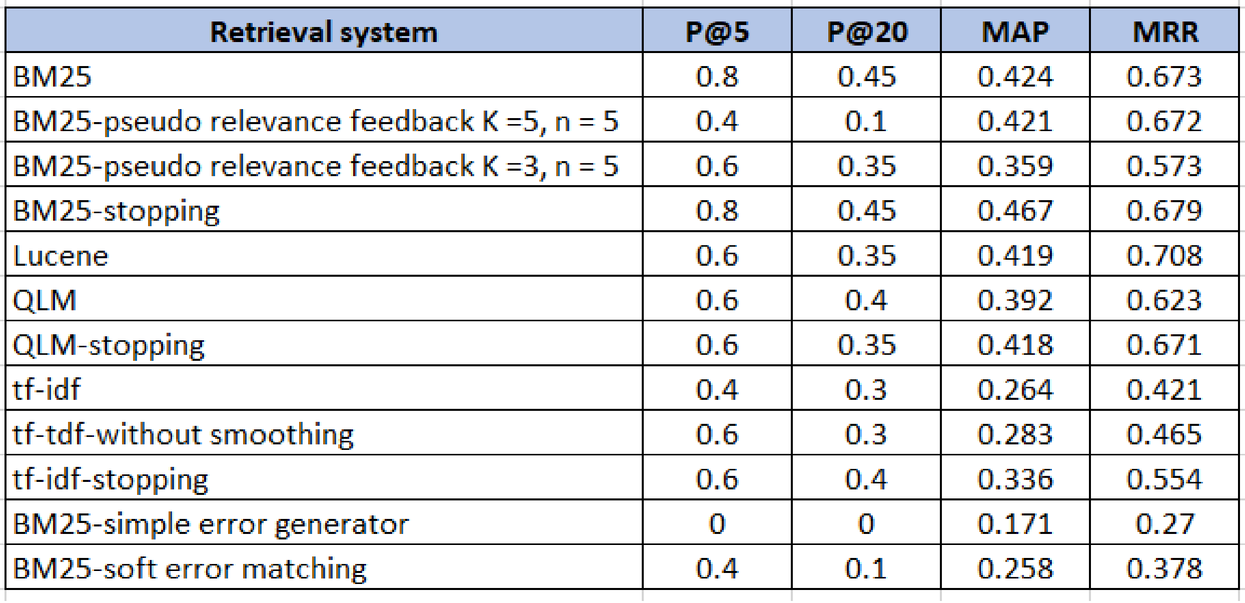
- The modified noisy query from the previous step is taken as the input for the Soft-matching query handler. If any query term is not present in the corpus we get the closest matching term from the corpus. We check the correlation between the word and its previous word using the bigrams indexing. We also assumed that the contextual words that have high frequency in the corpus may be the correct form of the query term. Using these steps, light correction was done to minimize noisiness in the query.

## RESULTS

The below result table is constructed for the various retrieval systems for query “performance evaluation and modelling of computer systems” (query_ID: 25).

K: top K$^{th}$ documents that are considered for PRF

n: number of high frequency terms taken from each document



| Retrieval system | P@5 | P@20 | MAP | MRR |
|---|---|---|---|---|
| BM25 | 0.8 | 0.45 | 0.424 | 0.673 |
| BM25-pseudo relevance feedback K =5, n = 5 | 0.4 | 0.1 | 0.421 | 0.672 |
| BM25-pseudo relevance feedback K =3, n = 5 | 0.6 | 0.35 | 0.359 | 0.573 |
| BM25-stopping | 0.8 | 0.45 | 0.467 | 0.679 |
| Lucene | 0.6 | 0.35 | 0.419 | 0.708 |
| QLM | 0.6 | 0.4 | 0.392 | 0.623 |
| QLM-stopping | 0.6 | 0.35 | 0.418 | 0.671 |
| tf-idf | 0.4 | 0.3 | 0.264 | 0.421 |
| tf-tdf-without smoothing | 0.6 | 0.3 | 0.283 | 0.465 |
| tf-idf-stopping | 0.6 | 0.4 | 0.336 | 0.554 |
| BM25-simple error generator | 0 | 0 | 0.171 | 0.27 |
| BM25-soft error matching | 0.4 | 0.1 | 0.258 | 0.378 |

Table 1: Table comparing the retrieval systems

# CONCLUSIONS AND OUTLOOK

**Conclusions:**

- From our evaluation, BM25 with stopping and Lucene was found to be the most effective retrieval system for the given corpus and queries. BM25 with stopping has a highest MAP value of 4.67 and Lucene has the highest MRR value of 0.708.
- Lucene scoring uses a combination of the Vector Space Model and the Boolean model to determine how relevant a given Document is to a User's query
- Higher MAP, MRR values for PRF was found when k=5, n=5. Considering too many documents will broaden the search and may not fetch relevant documents.
- BM25 is found to be more effective than the QLM as BM25 considers the query term weights along with the document term weights.
- Tf-idf was observed to have a lower MRR and MAP values as it just considers the term frequency and it favors large documents. If the term is appearing in all documents, it doesn't account much for the score
- Simple error generator using BM25 model has the lowest values of MAP and MRR as few query terms were changed and it could not be mapped to the correct index.
- To mitigate the effects of error models, we devised a soft-matching query handler which tried to improve the relevance of the retrieved documents and was found to be effective.

**Outlook:**

- PRF doesn't do well against short queries. This can be improved by implementing semantic query enrichment method. Further, high frequency words like 'cacm', 'jb', '1978' can be included as a stop words for the corpus and not include them in the query expansion for better results.
- Along with term frequency, we can keep track of the positions of terms in the documents that will help us to know the proximity between two terms.
- Text summary in the snippet generation can be made more concise which would justify the ranking of the documents
- Soft-matching query handler can be improved with the help of noisy channel model.

## BIBLIOGRAPHY

- W. Bruce Croft, Donald Metzler, Trevor Strohman (2015). **Search Engines, Information Retrieval in Practice**
- Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze (2009). **An Introduction to Information Retrieval**
- https://lucene.apache.org/
- https://archive.apache.org/dist/lucene/java/4.7.2/
- PRF: http://doras.dcu.ie/16517/1/A_Study_on_Query_Expansion_Methods_for_Patent_Retrieval.pdf
- Magdy W. and G. J. F. Jones. A Study on Query Expansion Methods for Patent Retrieval PAIR 2011 - CIKM 2011' paper.
- Extra credit:
  - get_close_matches from Lib/difflib.py
  - autocorrect feature from the library