CS6200 - INFORMATION RETRIEVAL FALL 2018
PROJECT REPORT

SUBMITTED BY:

PRACHI VED, COMPUTER SCIENCE, SEM 3
PREYANK JAIN, COMPUTER SCIENCE, SEM 3
RISHAB MALIK, COMPUTER SCIENCE, SEM 3

INSTRUCTOR:
NADA NAJI

INTRODUCTION:

As a part of this project, we built different Retrieval systems and evaluated their performance. The project was built in three phases:

PHASE 1:

Task 1:

In Phase 1, we first parsed and indexed the corpus. No stopping and stemming was done initially, and a Unigram index was built in the following format:
[TERM]: [(docId1:freq) (docId2:freq)…….]
Then, we performed four baseline runs using the following models to rank the documents for all the given queries:

1. BM25
2. Tf-idf
3. JM Smoothed Query Likelihood Model
4. Lucene's default retrieval model

Task 2:

We then implemented query enrichment using the BM25 run as a baseline. We assumed the top 10 ranked documents of our BM25 run to be relevant and then expanded the given query with highly weighted terms from those documents. The weights for the terms were calculated using Rocchio's Algorithm which is discussed in detail later.

Task 3:

For task 3, we picked the following three baseline runs:

1. BM25
2. Tf-idf
3. JM Smoothed Query Likelihood Model

We performed stopping first without stemming and created a new index and then ranked the documents for all the queries using each of the three techniques mentioned above.
Then, we indexed the stemmed version of the corpus cacm_stem.txt and retrieved the results for the queries in cacm_stem.query.

PHASE 2:

Phase 2 involved displaying the results with snippets. The snippets were generated using H.P. Luhn's approach that is discussed later in detail. The query terms were displayed in bold.

PHASE 3:

In Phase 3, evaluation of our retrieval systems was performed, the following evaluation measures were calculated to determine the performance of the different information retrieval models,

a) Mean Average Precision

b) Mean Reciprocal Rank
c) P@K (K=5 and K=20)
d) Precision
e) Recall

DESCRIPTION OF EACH MEMBER'S CONTRIBUTION:

Prachi Ved:

1. Implemented tf-idf
2. Implemented Lucene
3. Evaluation
4. Documented the part she implemented

Preyank Jain:

1. Implemented BM25
2. Implemented query enrichment using pseudo relevance feedback
3. Implemented Snippet generation
4. Evaluation
5. Documented the part he implemented

Rishab Malik:

1. Implemented JM Smoothed Query Likelihood Model
2. Implemented task 2, i.e., performing baseline runs after stopping and stemming
3. Implemented the extra credit retrieval systems
4. Documented the part he implemented

LITERATURE AND RESOURCES

1) Query enrichment Technique:

We implemented Query enrichment using pseudo relevance feedback. Pseudo relevance feedback is a query expansion technique in which, the system assumes that the top ranked documents are relevant to the user and then the words that occur frequently in these documents are used to expand the initial query. To weigh the terms that appeared in the top ranked documents, we used the Rocchio algorithm. Given that only limited relevance information is typically available, the most common form of Rocchio algorithm modifies the query vector Q to produce a new query vector Q' according to:

$$q'_j = a.q_j + \beta.\frac{1}{|REL|}\sum_{D_i \in REL} d_{ij} - \gamma.\frac{1}{|NonRel|}.\sum_{D_i \in NonRel} d_{ij}$$

Where, q'_j is the new weight of the query term *j* and q_j is the old weight, *REL* is the set of identified relevant document, which, in our case, were the top 10 documents from the BM25

baseline run for each query. *NonRel* is the set of non-relevant documents, d_{ij} is the weight of the j^{th} term in the document i, and a, β, γ are parameters that control the effect of each component. In our implementation, we focused only on expanding the query with relevant terms and hence we kept the value of γ equal to zero, and we kept the value of a equal to 0.8, and the value of β equal to 1, to get the new weights of query terms. Those terms that appeared in the top ranked documents and didn't appear in the query were initially given a weight of zero, and then that weight was increased using the formula above. After calculating the new weights for all the terms, we expanded the query with 10 terms that had the highest weights and were not initially present in the query.

2) Snippet Generation:

For Snippet Generation, we used H.P. Luhn's approach to rank each sentence in a document by a *significance factor* and then select the top sentence for the summary. The *significance factor* for a sentence is calculated based on the occurrences of significant words. Those words that also occur in the query are Significant words for our implementation. First, text spans are found, text spans are nothing but a series of words that contain both significant and non-significant terms. There is a limit set, on how many non-significant term can appear consecutively in a text span, in our case, it is 4. Then, the significance factor for these text spans is calculated by the following formula:

$$\text{significance factor} : \frac{no.\ of\ significant\ terms}{total\ number\ of\ terms}$$

The above-mentioned formula is applied on all the text spans in a document and then the top scoring text span is selected as the snippet. This technique has been studied from our text book "Information Retrieval in Practice" by W. Bruce Croft, Donald Metzler, and Trevor Strohman.

IMPLEMENTATION AND DISCUSSION:

TF-IDF Algorithm:

Tf-idf algorithm is a ranking algorithm which gives score to documents based on the frequency of the query words in the given document.

tf - Term frequency is the number of times the term is present in the given document.
idf - Is the number of documents that contains the given query term. It is calculated by adding 1 to the log of total number of documents in the collection divided by 1 + number of documents in which term appears. 1 is added to prevent it from becoming zero.

tf-idf(t,d,D) = tf(t,d) * idf(t,d,D)

Meaning of scores for the document:

- A document has a high score when it contains the term in high frequency and the term has low frequency across the collection.

- A document has a moderate score when the term frequency is moderate and higher frequency across the collection.
- A document has a low score when the term frequency is low and the document frequency is high.

The top 100 documents are displayed as result.

BM25 Algorithm:

The BM25 algorithm is a ranking algorithm that takes into consideration the relevance information and determines the rank for every document that contains at least one term of the query. It is an extended version of the binary independence model.

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

Where,

ri - number of relevant documents containing the term i
ni - number of documents containing the term i
N - total number of documents in the corpus
R - total number of relevant documents for the query
fi - frequency of the term i in the document
qfi - frequency of the term i in the query
k1 - determines how the tf component of the term weight changes as fi increases
k2 - determines the same thing for the query
b - constant, normalizes the length of the document.

The top 100 documents are displayed as result.

JM Smoothed Query Likelihood Model:

In Query Likelihood retrieval model, we rank documents by the probability that the query text could be generated by the document language model. We calculate the probability of pulling the query terms out of the *bucket* of words in the document. The implementation in this project ranks all documents based on the probability that the query text could be generated by the document language model. We use Bayes Rule and calculate P(D|Q) as,

$$p(D|Q) = P(Q|D)P(D)$$

where,

$$P(Q|D) = \prod_{i=1}^{n} P(q_i|D)$$

If there are no query term present in the document then the score for that document will be zero, so by using Jelinek-Mercer smoothing, we get $P(q_i|D)$ as,

$$p(q_i|D) = (1 - \lambda) \frac{f_{q_i,D}}{|D|} + \lambda \frac{c_{q_i}}{|C|}$$

And thus,

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

Where,
$f_{qi,D}$ = number of times word qi occurs on document D
|D| = number of words in D
C_{qi} = number of times word qi occurs in the collection of documents
|C| = total number of word occurrences in the collection
λ= constant, here its value is 0.35 (as per TREC standard)

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

$$= \sum_{i:f_{q_i,D}>0} \log((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|}) + \sum_{i:f_{q_i,D}=0} \log(\lambda\frac{c_{q_i}}{|C|})$$

$$= \sum_{i:f_{q_i,D}>0} \log \frac{((1-\lambda)\frac{f_{q_i,D}}{|D|} + \lambda\frac{c_{q_i}}{|C|})}{\lambda\frac{c_{q_i}}{|C|}} + \sum_{i=1}^{n} \log(\lambda\frac{c_{q_i}}{|C|})$$

$$\overset{rank}{=} \sum_{i:f_{q_i,D}>0} \log\left(\frac{((1-\lambda)\frac{f_{q_i,D}}{|D|}}{\lambda\frac{c_{q_i}}{|C|}} + 1\right)$$

The top 100 documents are displayed as result

EXTRA CREDIT:

The extra credit is done by implementing BM25 algorithm.

There are three types of searches that our retrieval model provides:

1) Exact Match: All the documents in the collection that have the exact matching query present in them are retrieved.

2) Best Match: All the documents that have an exact match of the query are ranked using BM25 algorithm.

3) Ordered best match within proximity N search: Same as above, but the order matters and any two query terms should be separated by no more than N other tokens.

The code consists of two files:

- **Query_matcher.py -** All the documents that qualify based on the type of search that the user has selected are retrieved.
- **Query_runner.py -** BM25 algorithm is run on all the documents that have the qualify based on the type of search that the user has selected are retrieved (the output documents of query_matcher.py). It returns the documents in the order of their ranks.

EVALUATION:

Evaluation is instrumental in the process of measuring the efficiency and effectiveness of an Information retrieval system.

In this project, we have used the following measures to accurately determine the efficiency of our Information Retrieval System.

1) MAP:
2) MRR:
3) P@K (K= 5 and K=20)
4) Precision vs Recall Graph

PRECISION:
Precision is an evaluation measure that measures the exactness of the retrieval process (among documents retrieved, how many documents are relevant).

$$Precision = \frac{|Relevant \cap Retrieved|}{|Retrieved|}$$

RECALL:

Recall is an evaluation measure that measures the completeness of the information retrieval process (among documents retrieved, how much of the relevant set of documents are retrieved).

$$Recall = \frac{|Relevant \cap Retrieved|}{|Relevant|}$$

MAP:

Mean Average Precision is the mean of all average precision values of all the queries.

Average Precision is the sum of precision values when a relevant document is found to the total number of relevant documents for that particular query.

MRR:

Mean Reciprocal Rank is the mean of all reciprocal ranks for all the queries.

Reciprocal Rank is the reciprocal of the rank at which the first relevant document is found.

P@K (K=5 and K=20):

P@K is a measure of the performance of our Retrieval System, we measure the precision after k results. Here, K=5 and K=20, and therefore, we measure the performance of our system by evaluating the precision of the 5th document and the 20th document.

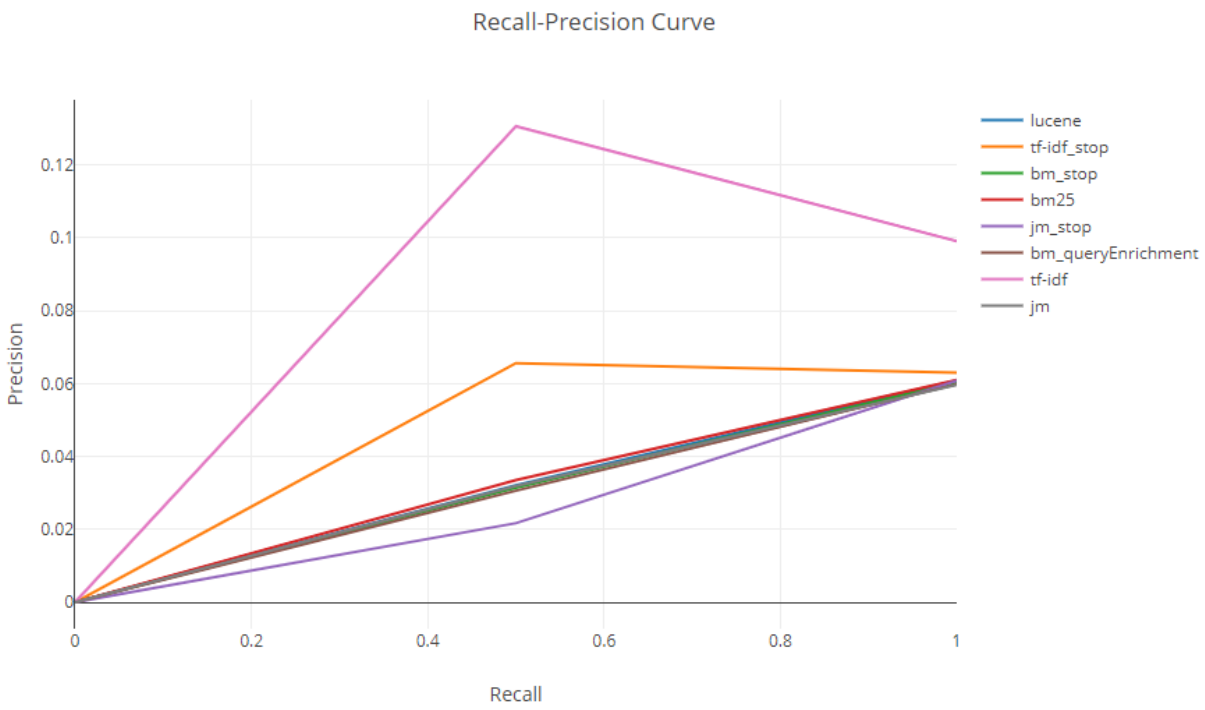
Higher the precision values at a smaller rank, better the system.

Recall-Precision Curve:
We also plotted a Recall-Precision curve that was created using the following approach:
For each run, we calculated the average precision values over all the queries at each of the following Recall values:

(1) Recall = 0
(2) Recall = 0.5
(3) Recall = 1

We obtained the following results:

Run\Recall	0	0.5	1
lucene	0	0.032109	0.060889
tfidf_stop	0	0.065622	0.063085
bm_stop	0	0.031406	0.060434
bm25	0	0.033538	0.061065
jm_stop	0	0.021798	0.060884
bm_queryEnrichment	0	0.030623	0.059854
tfidf	0	0.1307	0.099106
jm	0	0.031939	0.059692



Query-by-query Analysis:

First Query:
queryId 1: Stemmed query: “portabl oper system”

BM25 Top 3 documents:
1 Q0 cacm-3127 1 11.560385880868964 BM25Stemming
1 Q0 cacm-2593 2 7.080929066203819 BM25Stemming
1 Q0 cacm-2246 3 6.998791417128138 BM25Stemming

TF-IDF Top 3 documents:
1 Q0 cacm-2246 1 0.82919082175 TF-IDF_Stemming
1 Q0 cacm-3127 2 0.701201872495 TF-IDF_Stemming
1 Q0 cacm-1924 3 0.618988743051 TF-IDF_Stemming

Among the documents displayed, documents 3127, and 2246 are highly relevant topically. Both TF-IDF and BM25 produce almost similar results for this query. All these documents have high frequencies of the terms “portabl”, “oper”, and “system”.

Second query:
queryId 3: “parallel algorithm”
BM25 Top 3 documents:

3 Q0 cacm-2266 1 4.983255750657896 BM25Stemming
3 Q0 cacm-2973 2 4.983255750657895 BM25Stemming
3 Q0 cacm-2714 3 4.829238591352105 BM25Stemming

TF-IDF Top 3 documents:
3 Q0 cacm-2896 1 0.803943344356 TF-IDF_Stemming
3 Q0 cacm-2342 2 0.602957508267 TF-IDF_Stemming
3 Q0 cacm-2223 3 0.387771870658 TF-IDF_Stemming

All these documents have high frequencies of the terms “parallel” and “algorithm”, BM25 produces results that are more topically relevant whereas for TF-IDF document 2342 doesn’t seem to be topically relevant.

Third Query:
queryId 5: “appli stochast process”
BM25 Top 3 documents:
5 Q0 cacm-2535 1 7.723326103407494 BM25Stemming
5 Q0 cacm-1194 2 7.703339395026951 BM25Stemming
5 Q0 cacm-1410 3 7.4776312850706255 BM25Stemming

TF-IDF Top 3 documents:
5 Q0 cacm-3043 1 0.632814474004 TF-IDF_Stemming
5 Q0 cacm-1926 2 0.452010338574 TF-IDF_Stemming
5 Q0 cacm-2342 3 0.395509046252 TF-IDF_Stemming

This query is interesting because TF-IDF and BM25 produce highly varying results for them.
2535 seems very relevant topically
3034: high frequency of the term process
1926: seems topically relevant
2342: very high frequency of term “process”

RESULTS:

The results for all the metrics can be found in our project submission. To find the various metric tables:

1) Unzip our submission.
2) Go inside the folder named Phase3
3) The following folder contain the results for each metric for each of the query:
    a) For BmWithQueryEnrichment : BM_QueryEnrichmentEvaluationOutput
    b) For JM with stopping : JM_Evaluation_Stopping_Output
    c) For JM : JM_EvaluationOutput
    d) For Lucene : Lucene_EvaluationOutput
    e) For TF-IDF with stopping : TF-IDF_Evaluation_Stopping_Output
    f) For TF-IDF : TF-IDF_EvaluationOutput
    g) For BM with stopping : BM_Evaluation_Stopping_Output
    h) For BM : BM_EvaluationOutput
4) All the query-level results i.e., MAP, MRR can be found at the bottom of query-64 results, they are also shown in the table below:

	BM with enrichment	JM With Stopping	Lucene	TF-IDF with stopping	TF-IDF	BM with stopping	BM	JM
MAP	5.187743	13.7392	1.726855	0.5655	0.2105	2.472471	1.07886	2.363213
MRR	16.49683	41.2417	5.998921	1.2689	0.25	8.248473	2.749338	6.598571

CONCLUSIONS AND OUTLOOK:

CONCLUSION:

1) On average, TF-IDF returned the most number relevant documents among the top 100 ranks.
2) BM25 returned more relevant documents near the top of the search results, for example, if we consider query 2, BM25 returns all the three relevant documents with in the top 3 ranks, while TF-IDF ranked the relevant documents at lower ranks.
3) For query enrichment, we chose pseudo relevance feedback and as expected, we observed that when the top documents in the original ranking were not relevant, they lead to even worse rankings with the enriched query, for example: ranking for query 4 contains only 2 relevant documents among top 10, and the results of the enriched query produce a ranking which contains no relevant document among top 10. Also, if the top ranked documents in the first run are relevant to the user then the rankings produced after query enrichment are better or equally good, but not worse.

4) Overall, if the user is interested in only the top ranked documents i.e. the user is only going to view the top ranked documents, then BM25 is the best retrieval system, but if the user is interested in getting just the relevant documents and he/she is willing to go to lower ranks to search for the relevant documents, then tf-idf would be better, based on the results we observed.

OUTLOOK:

1) Incorporating Relevance feedback into the system to ensure that the user gets relevant results. We can ask the user to provide feedback and then produce a better ranking based on that.
2) We can perform clustering to present all the documents that appear to be on the same topic in one cluster, rather than presenting all the documents separately.
3) We can also incorporate searching in multiple languages where the user enters the queries in languages other than English.

BIBLIOGRAPHY:

- https://nlp.stanford.edu/IR-book/html/htmledition/pseudo-relevance-feedback-1.html
- https://nlp.stanford.edu/IR-book/html/htmledition/using-query-likelihood-language-models-in-ir-1.html
- https://en.wikipedia.org/wiki/Okapi_BM25
- https://www.crummy.com/software/BeautifulSoup/
- https://en.wikipedia.org/wiki/Rocchio_algorithm