Information Retrieval Course Project
Retrieval Systems Construction, Optimization and Evaluation

Shihao Wang,
Kartik Dave,
Chirag Shivnani

December 10, 2017

Semester:	2017 Fall
Instructor:	Professor Nada Naji
Institution:	Northeastern University

1 Introduction

1.1 Brief Description

In this project, we realize two retrieval systems based on Java and Python. For Java version, the classic retrieval package Luence is called and the default retrieval model is used. For Python version, we develop three models: BM25, tf-idf and Smoothing Query Likelihood as candidate ones and support stemming, stopping and query refinement in the system in the form of parameters of constructor function. Moreover, we also design snippet generation for better user experience. Finally, several evaluation experiments are conducted to verify and compare the performance of different retrieval models and different searching settings. This report will discuss the details in different sections.

1.2 Contribution

As for the programming, we divide this workload according to phases. Shihao Wang is in charge of developing phase 1: retrieval system building and query refinement. Kartik Dave designs phase 2: Snippet Generation. Chirag Shivnani conducts phase 3: evaluation and results display.
As for the document, Shihao Wang writes the section 1, 2.1, 3.1, 3.2 and 3.3. Kartik is responsible for section 2.2 and 3.3. Chirag completes the section 4.

2 Literature and Resources

2.1 Query Refinement Approach

As an effective query expansion and refinement approach, Pseudo Relevance Feedback (PRF) stands out because it not only can expand query terms number in arbitrary scale but also can decrease work for human annotation (White et al. (2007)). In this project, we realize PRF as a class with a data structure *extend_query*. The constructor template of PRF is as follows:

$$PRF(query, top\_doc\_num, top\_term\_num, term\_association, system\_name)$$

where query is the original query. top_doc_num represents how many result documents as relevant set. top_term_num means expansion term number. term _association decides which association measure is used (So far we support both dice coefficient and mutual information). system_name is the system you want to conduct query refinement on, which is a string type.
We use the thought of PRF in chapter 6 of Croft et al. (2010), calculating the sum of term association weights of a term to all query terms as the score of this term. Then rank these score in descending order and pick certain number of terms as our expansion terms. We use stopping and duplicate check to prevent common useless and repeated words from being added.

2.2 Snippet Generation Approach

An Experimental Comparison Of Time Sharing And Batch Processing
CACM-1605
The effectiveness for program development of the MIT Compatible **Time-Sharing System** (CTSS) was compared with that of the IBM IBSYS batch-processing **system** by means of a statistically designed experiment.

Interarrival Statistics For Time Sharing Systems
CACM-1410
The optimization of **time**-shared **system** performance requires the description of the stochastic processes governing the user inputs and the program activity.

Sharer, A Time Sharing System For The Cdc 6600
CACM-1523
A **time sharing system** embedded within the standard batch processing **system** for the CDC 6600 is described.

The Design Of The Venus Operating System
CACM-2379
The Venus **Operating System** is an experimental multiprogramming **system** which supports five or six concurrent users on a small computer.

Tenex, A Paged Time Sharing System For The Pdp-10
CACM-2380
TENEX is a new **time sharing system** implemented on DEC PDP-10 augmented by special paging hardware developed at BBN.

George 3-A General Purpose Time Sharing And Operating System
CACM-1519
An **Operating System** is described which will run on a wide variety of configurations of the I.

Operating System Performance
CACM-2319
computer **system**, **operating system**, performance evaluation, performance measurement, measurement, techniques, modularity, layering, structured programming, paging, virtual memory, input/output, disk storage facility, drum storage facility, sector queueing

Figure 1: BM25_Unigram_Case-folded Snippet for Query No.1

A good snippet will help the user to reflect the understanding of the document in a sentence or two which will help them fulfill their information need.
We implemented a simple and fast algorithm for generating snippet which is query dependent and will find the most significant sentences from the text based on the score how related it is with the query passed. The different features of the algorithm and implementation are:

a. The query sentence relationship score is established with the help of the frequency of important terms in query (not considering the stop words)

which are there in the sentence. The algorithm is based on Bast and Celikik (2009).

b. The text is broken into sentences at the time of indexing which will reduce the load at the time of query processing resulting in optimization of the system.

c. The derived and inflected word of the query words are also considered while scoring the sentence which leads to an increased effectiveness of the output result.

d. The title that is the first line of the text in the corpus is displayed for every document as it is the best summary of the page given by the writer. The key word in the most significant sentence for a document is highlighted in bold.

3 Implementation

3.1 Parser

In this project, parser is classified as query parser or corpus parser. The main difference of both parsers is query parser controls whether stopping is used and corpus parser controls stemming. For the corpus parser, there are two different format for corpus file, which are HTML(no-stemming) and TXT(stemming). So we design two sub-parsers for both file formats. Each will have the processes case folding, remove punctuation, and a special process - remove extra digits. Our strategy to remove these extra digits is combining digital lines searching and key word matching. The digital lines are the lines with only digits and spaces. Moreover, we found a regularity that those digits are all coming after a timestamp of the document. So it is possible to use the key word "pm" or "am" to remove extra digits. This combining strategy will boost the accuracy of removing useless information.
The query parsers also handle two separate formats (TREC and simple text format). Since we found that the no-stemming queries are offered in TREC document style in Croft et al. (2010), we use a boolean parameter TREC_format to switch between text formats we need to handle. For stopping strategy, we simply conduct a word matching to remove all common words for queries. Here are the constructor templates of both parsers:

$$CP(stemming = False, case\_folding = True, punctuation = True)$$

$$QP(filename, TREC\_format, stopping, case\_folding = True, punctuation = True)$$

3.2 Retrieval Models

The design of three retrieval models in the Python retrieval system are pretty straightforward. The only difference in BM25 model compared to HW4 is con-

sideration of relevance information (R and r_i will no be zero). We use Jelinek-Mercer Smoothing to optimize Query Likelihood Model. All three models have uniform usage so that we can change them easily in our retrieval entrance program by changing the parameter 'retrieval_model'. Another optimization is that the inverted index will be pre-loaded in memory rather than read from files when it is used, which saves significant searching time after comparison.

$$BM25(query, stemming\_corpus = False)$$

$$SmoothingQLM(query, stemming\_corpus = False)$$

$$TfIdf(query, stemming\_corpus = False)$$

3.3 Query-by-query Analysis

The first interesting finding comes from query No.1. Table 1 shows the top three results of different retrieval models of query No.1. The results of those models and their deformations are pretty similar (doc 3127 and 3068 are the most common top results). Further more, the document 3127 not only ranks top one among three models but leave a pretty large gap in ranking score with the following top results. From the table, we could see that the differences between No.1 and No.2 result are more than 10 times of the differences between No.2 and No.3 for all three models, which means doc 3127 is the best answer of query No.1.

Table 1: Top three ranks of different models for query No.1.

Retrieval model	Doc ID	Ranking score
SQLM_Stemming	3127	−11.5760
	2593	−17.4939
	3068	−17.6599
BM25_Stemming	3127	14.2812
	3068	9.1730
	2319	9.1037
tf-idf_Stemming	3127	0.3132
	3068	0.1793
	2796	0.1657

The second interesting finding comes from query No.4 which is very different from our previous finding. We can see in table 2 the result of SQLM and BM25 retrieval model are having two documents in common but there is no similarity between tf-idf as compared to the other two. This difference reflects the completely different working of tf-idf. As in tf-idf we do not consider the weight of query term in calculating the score only the term and document frequency is considered the result is highly affected and very different from the other two models where the query terms are also given importance.

Table 2: Top five ranks of different models for query No.4.

Retrieval model	Doc ID	Ranking score
SQLM_Stemming	2454	−22.5237
	2276	−22.9092
	2406	−23.3788
	2032	−23.4352
	2004	−23.5928
BM25_Stemming	2032	8.3366
	2649	7.3679
	2926	7.3389
	2406	7.0610
	3043	6.9825
tf-idf_Stemming	1778	0.3841
	1733	0.3585
	1941	0.3585
	2549	0.3512
	1944	0.2988

The third interesting finding comes from query No.6 which is very similar to query 1 where we are having the first three top documents same in SQLM and BM25 and two in tf-idf. The similarity reflects the presence of the term "system" in both the query which is a common as well as significant term in the entire corpus. Moreover, the decrease in the score is not very high as the query length is more with many important words in it.

Table 3: Top three ranks of different models for query No.6.

Retrieval model	Doc ID	Ranking score
SQLM_Stemming	2318	−23.5813
	3070	−27.6377
	3048	−27.6391
BM25_Stemming	2032	8.3366
	2649	7.3679
	2926	7.3389
tf-idf_Stemming	2318	0.7066
	2984	0.3863
	3070	0.3741

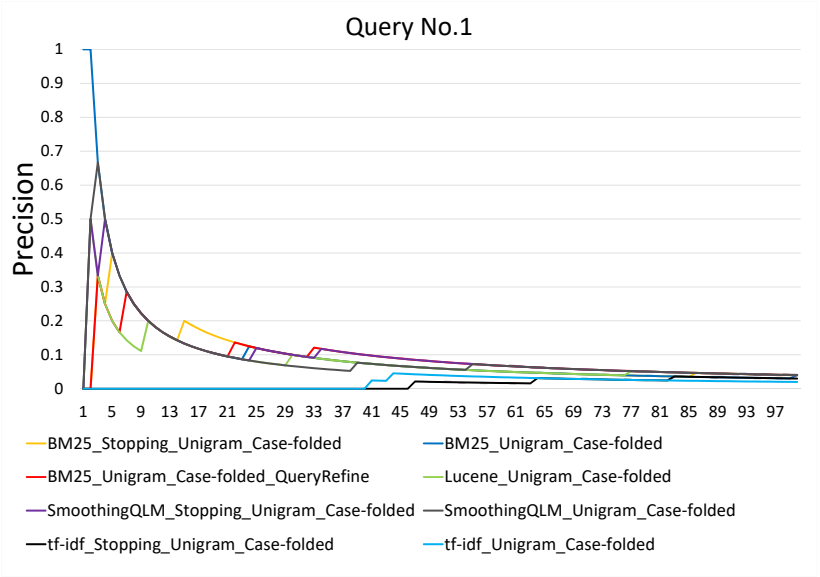

Figure 2: Eight baselines evaluation with Precision for Query No.1

4 Evaluation

4.1 Precision and Recall

To evaluate the performance of the retrieval models, we used precision and recall data of two queries as our sample data.
From Figure 2 and 3, we can see that for BM25_Unigram_Case-folded model Document 1 has a high precision value and a low recall rate meaning that less documents have been retrieved but they are accurate and occur at the beginning of the result. All other retrieval models have precision and recall rate as 0 for Document 1. Referring the same figures, we can see that tf-idf models have the worst performance compared to other models. Both its variants have a low precision and low recall value meaning most of the documents that are returned are incorrect and the initial 40 documents have value of 0 for precision and recall. As we progress through the list of our 100 documents, we see that the value of precision is decreasing with most of the documents that are being encountered, indicating that all the models output a high number of false positives. We can also see that BM25_Unigram_Case-folded performs the best among all the retrieval models. At the end of our run for Query 1, it has a precision of 0.04 which is equal or greater to all other runs and a recall value of 0.8 which is equal to or higher than all other runs.
Unlike Query 1, in Query 2, BM2_Unigram_Case-folded model has a precision and recall rate of 0 for five initial documents as evident from Figure 4 and 5.

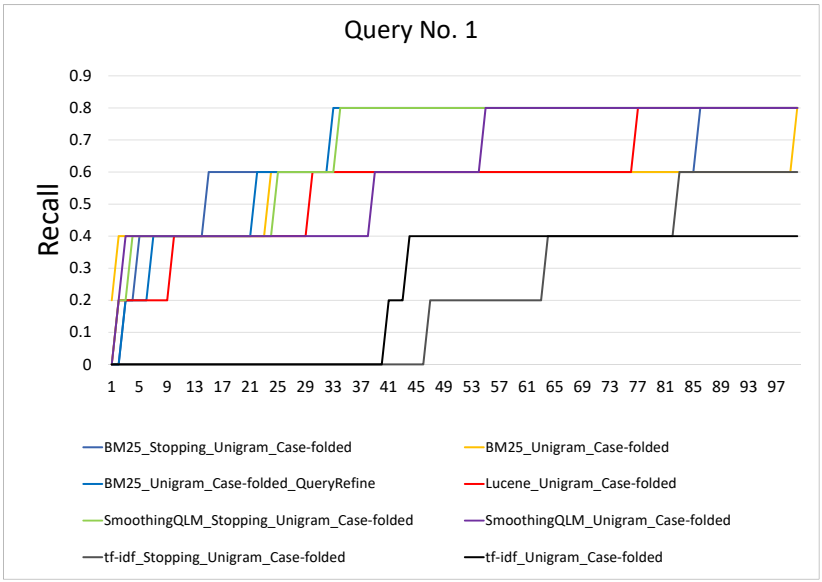

Figure 3: Eight baselines evaluation with Recall for Query No.1

We can see that the aforementioned model for Document 1 has a high precision value (1) and a low recall rate (0.16) meaning that few documents have been retrieved but they are accurate. All other retrieval models have precision and recall rate as 0 for Document 1. After the first 44 documents, the performance of all BM25 models is same. The tf-idf models have the lowest precision and recall rates. Both the models have a precision and recall value of 0 for the initial 47 documents

4.2 P@K

From the Table 4, we can see that the consistent good result is shown by BM25_Unigram_Case-folded_QueryRefine. For Query 1 it has a lower precision value compared to other models but it doesn't have a precision of 0 for any of the queries. One of the finding can be that the model helps to get more words which are of same context, the relevant documents comes at top ranks giving a good precision at rank 5.
From the Table 5, we can see that BM25_Stopping_Unigram_Case-folded shows best performance, reaching highest P@K value for three queries. Besides, All BM25-based models perform well compared to other baselines.

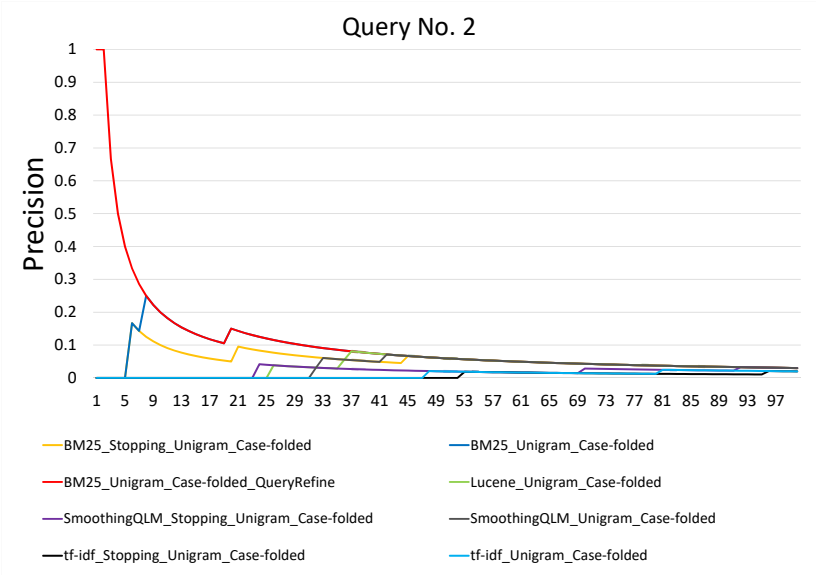

Figure 4: Eight baselines evaluation with Precision for Query No.2

Table 4: Comparison of eight baselines on P@5 using four queries.

Retrieval model	Query 1	Query 2	Query 3	Query 4
BM25_Stopping_Unigram_Case-folded	0.4	0.6	0	0.4
BM25_Unigram_Case-folded	0.4	0.6	0	0.4
BM25_Unigram_Case-folded_QueryRefine	0.2	0.6	0.4	0.4
Lucene_Unigram_Case-folded	0.2	0.6	0	0.2
SmoothingQLM_Stopping_Unigram_Case-folded	0.4	0.6	0	0.4
SmoothingQLM_Unigram_Case-folded	0.4	0.2	0	0.4
tf-idf_Stopping_Unigram_Case-folded	0	0	0	0.2
tf-idf_Unigram_Case-folded	0	0	0	0.4

4.3 MAP

From Figure 6, we can see that BM25 (all three variants) is the only model that scores above 0.5, which implies that it returns less number of false positive documents. There is a huge gap between the BM25 and Lucene model which comes in at second. The worst performance displayed is by tf-idf variants that score below 0.3.

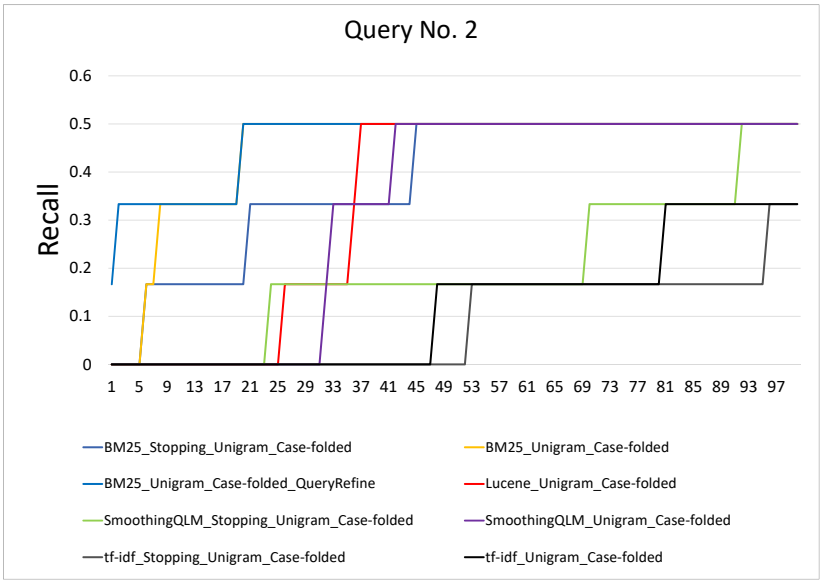

Figure 5: Eight baselines evaluation with Recall for Query No.2

Table 5: Comparison of eight baselines on P@20 using four queries.

Retrieval model	Query 1	Query 2	Query 3	Query 4
BM25_Stopping_Unigram_Case-folded	0.15	0.15	0.05	0.2
BM25_Unigram_Case-folded	0.1	0.15	0.15	0.15
BM25_Unigram_Case-folded_QueryRefine	0.1	0.15	0.15	0.15
Lucene_Unigram_Case-folded	0.1	0.15	0	0.1
SmoothingQLM_Stopping_Unigram_Case-folded	0.1	0.15	0	0.1
SmoothingQLM_Unigram_Case-folded	0.1	0.1	0	0.1
tf-idf_Stopping_Unigram_Case-folded	0	0	0	0.1
tf-idf_Unigram_Case-folded	0	0	0	0.2

4.4 MMR

Similar to MAP, we see that BM25 models return the highest MRR value reflecting that the relevant documents were found at a higher position than they were in other retrieval models as evident in Figure 7. Among this model's variants, the BM25_Unigram_Case-folded model has the highest MRR value which indicates that a relevant document was encountered the earliest for most of the 52 queries. The BM25_Unigram_Case-folded_QueryRefine showing the similar performance comes second.

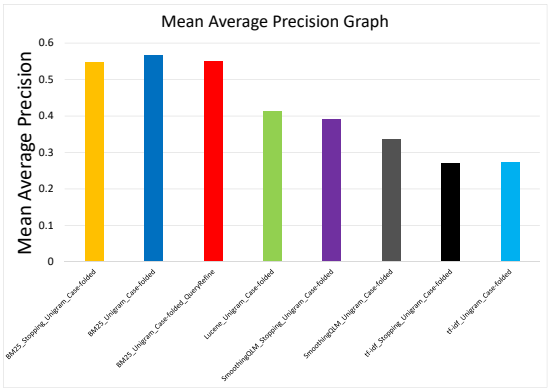

Figure 6: Eight baselines evaluation with Mean Average Precision (MAP)

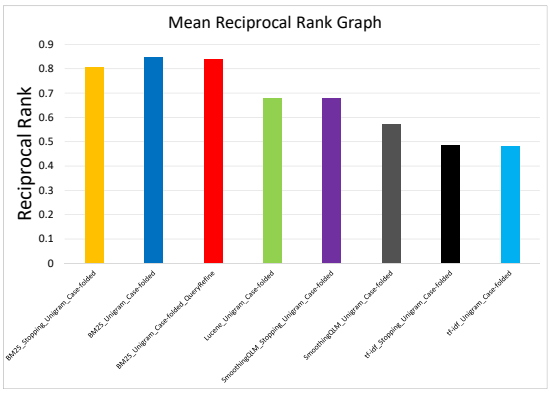

Figure 7: Eight baselines evaluation with Mean Reciprocal Rank (MRR)

4.5 Proximity Retrieval Model Analysis

From Query 4, we can see that there are pretty much difference between stopping version and non-stopping version. The top one result of non-stopping is CACM-1890, which contains no important searching words of Query 4. However, it still reaches such a high rank due to some same proximity information of common words such as "on the" and "in a". For the stopping version, the results are pretty similar with BM25, indicating that stopping is a key step for Proximity Retrieval Model.

From the Figure 8 we can observe that PRM variant "PRM_Stopping_Unigram _Case-folded" performs better when compared with non-stopping PRM variant which reflects the understanding that stopping will result in removal of common words and will lead to consider position of only the important words. This will

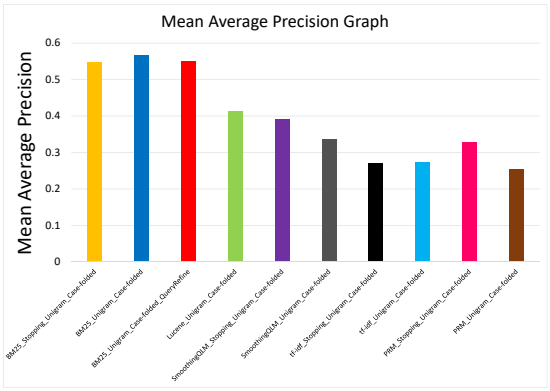

Figure 8: Comparison of two Proximity Retrieval runs with other baselines on Mean Average Precision (MAP)

result in output of more relevant documents. It can also be observed that is better than tf-idf model. The worst performance is shown by "PRM_Unigram_case-folded" variant of PRM as it considers both the negative aspect of not stopping and constrains with position of terms without considering frequency.
Implementing proximity enabled search led to some new findings. It was seen that this method with an attached restriction will result in very less documents with a high score. As the complexity of English language makes it possible to represent the information in many forms, it is possible that the important terms in the text can be away from each other still reflecting important information. As the frequency of the terms in the text is completely ignored and only togetherness is given importance the model will output inaccurate results. Such kind of strategy can be useful in information extraction where we have a very large corpus and we need to do a filtering to get the relevant documents and then process it further. The best plan can be combining the strength of this model with some other retrieval model to achieve better performance.

5 Conclusion

Two retrieval systems have been used in our work, where Python retrieval system supports snippet generation, query refinement, stopping and stemming and Java retrieval system generates a baseline for the evaluation. We also conduct a brief analysis on using of proximity information. From the evaluations of eight baselines, we know that BM25 and its deformations performs the best comprehensive performance.

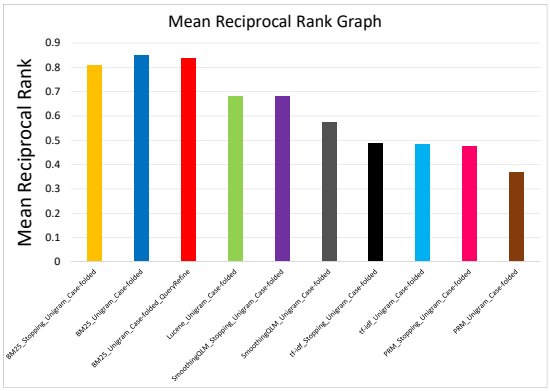

Figure 9: Comparison of two Proximity Retrieval runs with other baselines on Mean Reciprocal Rank (MRR)

References

Bast, H. and Celikik, M. (2009). Efficient index-based snippet generation. *Acm Transactions on Information Systems*, 32(2):431–447.

Croft, W. B., Metzler, D., and Strohman, T. (2010). *Search engines: Information retrieval in practice*, volume 283. Addison-Wesley Reading.

White, R. W., Clarke, C. L. A., and Cucerzan, S. (2007). Comparing query logs and pseudo-relevance feedbackfor web-search query refinement. In *International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 831–832.