IFT6255 - TP1

Caroline Dakouré, Jinfang Luo

March 20, 2018

Abstract

Search engines which are often the interface to access information must be effective. The problem of finding information is to find relevant documents to an information need among a large set of documents. This study was primarily a way for us to learn about the information retrieval but also to develop research skills. That's why we used Indri which is a search engine used in many companies. We will rely on rigorous tests and different approaches to provide a study that we hope complete.

1 Introduction

Through this assignment, we would know the principle of how a search engine works. From building index for corpus AP88-90, then querying the index file with different information needs, and then using measures to analyst and evaluate the results. Furthermore, we performed local and global query expansion with relevant model. Further understand the impact of expanding queries on the results.

We provide our analysis based on the results. Including the results for indexing with and without stemming, with and without stop words list, while querying with different retrieval models, with different smoothing values in language model. Also we use two kinds of expanding method to improve the results. Based on these results, we use several measures to evaluate it, including precision, recall, f1 score, MAP, P@10, P@5.

2 Tools

2.1 What is Indri

« Indri is a search engine that provides text search and a rich structured query language for text collections of up to 50 million documents (single machine) or 500 million documents (distributed search) [1]». It's an open source tool, we wrote the parameter XML file and then it helped us to build an index with the parameter requirements. We could also use this tool to execute our query by setting the query parameter file.

2.2 Related commands

- IndriBuildIndex

  We use below combination setting in our parameter file to build our Index files for the TREC Text format documents (AP88-90). Then we get following index files.

  1. No stemming also No stop list
  2. No stemming with stop list
  3. Porter stemmer
  4. Krovetz stemmer
  5. Porter stemmer and stop list
  6. Krovetz stemmer and stop list

Here, we noticed the time for building each index is about 3 to 4 minutes. This time can be decreased if you have more RAM, when we increased our RAM to 4GB it took 2 to 3 minutes. We will see below the parameters that can affect the indexing time. The space requirement for each index file, should be more than 1G.

- IndriRunQuery

  We use this command to perform our information retrieval with different models for three topics(1-50, 51-100, 101-150).

  1. Vector space model with tf-idf
  2. Vector space model with BM25
  3. Language Model with Dirichlet smoothing

- Treceval

  trec_eval is a standard tool used by the TREC community for evaluating an ad hoc retrieval run, given the results file and a standard set of judged results.

  It allowed us to obtain measurements and evaluate the performance of different retrieval results.

  We set our query parameter file with <trecFormat>true</trecFormat>, it means resulting output will be in standard TREC format:

  1 Q0 AP880502-0134 1 -6.62621 indri

  1 Q0 AP890320-0105 2 -6.80593 indri

  1 Q0 AP901220-0249 3 -6.8707 indri

2.3 Related discussion

- We faced a problem during executing query, we wanted to modify our query parameter file to execute the information retrieval. But with baseline methods, we could not use Indri query language such as: #combine(Dumping Charges). So we deleted the command '#combine' between <text> and </text> then we found another problem. Using or not using "" for the words would get different results. We supposed that's because double quotation marks "" is interpreted as logical 'AND'. Without double quotation marks, it is interpreted as logical relationship 'OR'.

- There was another problem, when we were scripting the query parameter XML file, if there were punctuations and some other special characters, we found that Indri couldn't handle it correctly. What we did was deleting the punctuation period. As for the alphabetic letter and numeral, we replaced them by space. For example, we changed 'Hostage-Taking' to 'Hostage Taking', 'U.S.' to 'US'.

  After we looked for the information in lemur project website, we found that the default setting for 'normalize' is true while performing indexing, it means while performing indexing, 'U.S.' was treated as 'US' too. For such situation, it was fine. But if the query contains ' /' or '&', for example, 'Alternative/renewable Energy Plant & Equipment Installation', with our solution, it would be changed to 'Alternative renewable Energy Plant Equipment Installation'. It changes the original query information needed somehow. We didn't find a solution for this issue while scripting our parameter file. But we think we should pay attention on this problem, try to keep as much as the information needs of the original query.

- The third difficulty we faced was the index XML parameter file. There is one element called *field* which is used to specify the fields to index as data. We tried specifying TEXT (meaning that only the words between TEXT tags would be indexed) and we also tried specifying the others such as HEAD, DATELINE to index as data. We found that the number of documents

retrieved over all queries(num_ret) will be increased with more parameters. However, the increase is not obvious.

For instance, if we don't set TITLE as a *field*, the files we'll get will contain less information because we did not choose to index words which locate between TITLE tags.

- File information of index AP with Porter Stemmer and specifying *field* setting to TEXT:

total 477788

-rw-r–r– 1 root root 182091420 Mar 4 2018 directFile

-rw-r–r– 1 root root 971672 Mar 4 2018 documentLengths

-rw-r–r– 1 root root 5830032 Mar 4 2018 documentStatistics

-rw-r–r– 1 root root 1385721 Mar 4 2018 fieldsFile

. . . . . .

- File information of Index AP with Porter Stemmer but without specifying *field*:

total 472956

-rw-r–r– 1 root root 180162287 Mar 10 14:59 directFile

-rw-r–r– 1 root root 971672 Mar 10 14:59 documentLengths

-rw-r–r– 1 root root 5830032 Mar 10 14:59 documentStatistics

-rw-r–r– 1 root root 0 Mar 10 14:59 fieldsFile

. . . . . .

2.4 Running environment

For this assignment, we installed this tool on Linux(ubuntu) environment: Ubuntu-16.04.03 + indri-5.12 + treceval-8.1

3 Compare and analyst the results

3.1 Indexing file

Most indexing important files are located under /index/0 path (or other number if you rebuild your index files). Pay attention, while we rerun IndriBuildIndex command without changing the location path, we found that if the newest indexing file is smaller than the original file, it would keep the original one, meaning that it will keep the larger one.

Figure 1: Index file sizes comparison

Index file	File Size
No stemmer and No stop list	1.07GB
Stemmer with Krovetz	1.05GB
Stemmer with porter	1.04GB
No stemmer with stop list	965.8MB
Stemmer with Krovetz and stop list	946.4MB
Stemmer with porter and stop list	941MB

We compared the size of the different indexes files which were created by different methods. The largest index file is without stemming and without stop word setting, it's 1.07GB. The smallest index file is Porter Stemming with stop word, it's 941MB, compared to 1.04G without the stop word. As for comparing with Krovetz stemming and stop word, the sizes are close.

3.2 Time consumed

Using a stop list and choosing a stemming method does affect the time (Figure 2) for building each index folder. We found it's interesting to compare the size of the index and the time to build this index.

Figure 2: Average Indexing Time

Average Indexing Time (mins)			
Stop list \Stemming	No stemming	Porter stemming	Krovetz stemming
No stop list	3:06	2:52	3:03
Stop list	2:46	2:44	2:49

The computer would have different indexation time depending on processes that are being executed in the background. That's why we've decided to took the average time for each setting. Although we didn't push the experience so far (10 runs), our results are in line with the previously observed index sizes. The index is the heaviest when using no stemming algorithm and no stop list, and it corresponds to the longest time for building the index. Generally speaking, using a stop list will decrease the indexing time because it will significantly reduces the number of words that the system will store. It would take more testing to have more accurate and indicative results.

We even pushed the experiment further by measuring the time to execute a query until a result is obtained (Figure 3). We wanted to see how different ways of indexing could impact the retrieval.

Figure 3: Average Retrieval Time based on 10 runs

Average Query execution Time (seconds) - Topics 1-50			
Index \IR models	tf-idf	bm25	LM: Dirichlet(u=2000)
No stemmer	3:93	3:92	4:12
No stemmer with stop list	1:52	1:70	1:80
Porter Stemmer	4:12	4.12:	4:50
Krovetz Stemmer	4:13	4:18	4:26
Porter Stemmer and stop list	1:97	1:93	2:40
Krovetz Stemmer and stop list	1:84	1:86	2:30

The measured time may vary slightly from run to run, deciseconds are not a good measure. We consider some uncertainty, when there is no difference of at least 1 second, the result is not interpretable. With such tights results and a number of runs so low, we can not compare in a safe way the different retrieval models. However, we can see that indexing plays an important role for the execution time of the query.

For now, what we can say for sure is that the use of a stop list considerably reduces the execution time of the request. This is quite normal since the number of indexed words is smaller. It would also seem that the execution of the query is longer with the use of the language model with Dirichlet smoothing than with the Porter and Krovetz stemming.

3.3 Evaluations and analysis based on measure results

We would use some measures in trec_eval evaluation file. Here we would explain the meaning of them. Based on these results, we will perform our analysis.

- Precision: It measures the fraction of relevant instances among the retrieved instance. Basically, it answers the following question : How many documents are relevant among the retrieved documents. Precision = $\frac{\#tp}{\#tp+\#fp}$

Figure 4: Topics 1-50 Precision

Topics 1-50 Precision = #num_rel_ret/#num_ret				
Index \IR models	LM: PonteCroft1998	tf-idf	bm25	LM: Dirichlet(u=2000)
No stemmer and No stop list	2.83%	2.79%	2.69%	2.83%
No stemmer with stop list	2.81%	2.78%	2.92%	2.79%
Stemmer with porter	3.2%	3.03%	2.88%	3.2%
Stemmer with Krovetz	3.08%	2.97%	2.83%	3.08%
Stemmer with porter and stop list	3.18%	3.04%	3.19%	3.17%
Stemmer with Krovetz and stop list	3.05%	2.97%	3.09%	3.04%

Precisions were low on topic.1-50, but it's not surprising because we displayed a large list of results when there were only few documents relevant to the queries. We also calculated precisions for topics 51-100 and 101-150 but we'll introduce them later.

- Recall: It measures « the fraction of relevant instances that have been retrieved over the total amount of relevant instances » [2]. Recall = $\frac{\#tp}{\#tp+\#fn}$

Figure 5: Topics 1-50 Recall

Topics 1-50 Recall = #num_rel_ret/#num_rel				
Index \IR models	LM: PonteCroft1998	tf-idf	bm25	LM: Dirichlet(u=2000)
No stemmer and No stop list	36.51%	35.97%	34.70%	36.56%
No stemmer with stop list	36.29%	35.94%	37.77%	36.08%
Stemmer with porter	41.40%	39.19%	37.15%	41.34%
Stemmer with Krovetz	39.87%	38.52%	36.72%	39.87%
Stemmer with porter and stop list	41.02%	39.25%	41.18%	40.99%
Stemmer with Krovetz and stop list	39.52%	38.55%	40.00%	39.35%

- F-measure: It is a weighted harmonic mean of Precision and Recall.

  $F_\beta = \frac{(1+\beta^2)recall \times precision}{recall+\beta^2 \times precision}$

  It takes harmonic mean of Precision and Recall instead of arithmetic mean, because of considering the small value of relevant documents.

  while β=1 which means the weight of Precision and Recall equals the same weight, then we call F_1 score.

  F_1 score = $\frac{2 \times recall \times precision}{recall+precision}$

  « These three measures(precision, recall and f measure) are set-based measures [3], which means they evaluate the quality of an unordered set of documents ». « The F_1 score is often used in the field of information retrieval for measuring search, document classification, and query classification performance » [4]

After performing the retrieval, we will get a set of ranked retrieval documents. Based on these ranked retrieval results, we used the MAP and P@K to evaluate the system.

- MAP: « Mean Average Precision, which provides a single-figure measure of quality across recall levels » [3]. This value is calculated by averaging precision values from the rank positions where a relevant document was retrieved.

- P@k: Pooling technique used in TREC, top k results from the rankings are merged into a pool. For most of the users, especially for Ad-hoc retrieval, the users care more on how many good results have been retrieved in top 10 results. But this value is not quite suitable for evaluating the performance of system, but it affects user's feedback.

Once we get the Precision and the Recall, we could get the F_1 score.

Figure 6: Topics 1-50 F1-score

Topics 1-50 F1-score = 2(P*R)/(P+R)				
Index \IR models	LM: PonteCroft1998	tf-idf	bm25	LM: Dirichlet(u=2000)
No stemmer and No stop list	0.05247	0.0517	0.04989	0.05255
No stemmer with stop list	0.05217	0.05166	0.05429	0.05186
Stemmer with porter	0.05948	0.05631	0.05338	0.05940
Stemmer with Krovetz	0.05712	0.05519	0.05261	0.05712
Stemmer with porter and stop list	0.05894	0.05639	0.05917	0.05890
Stemmer with Krovetz and stop list	0.05661	0.05523	0.05731	0.05638

Figure 7: Topics 51-100 F1-score

Topics 51-100 F1-score = 2(P*R)/(P+R)				
Index \IR models	LM: PonteCroft1998	tf-idf	bm25	LM: Dirichlet(u=2000)
No stemmer and No stop list	0.15540	0.15386	0.15212	0.15482
No stemmer with stop list	0.15523	0.15373	0.15858	0.15417
Stemmer with porter	0.18442	0.18290	0.17518	0.18395
Stemmer with Krovetz	0.18357	0.18282	0.17777	0.18313
Stemmer with porter and stop list	0.18615	0.18385	0.18429	0.18554
Stemmer with Krovetz and stop list	0.18323	0.18244	0.18391	0.18251

Figure 8: Topics 101-150 F1-score

Topics 101-150 F1-score = 2(P*R)/(P+R)				
Index \IR models	LM: PonteCroft1998	tf-idf	bm25	LM: Dirichlet(u=2000)
No stemmer and No stop list	0.14093	0.13956	0.12697	0.14170
No stemmer with stop list	0.14086	0.13943	0.14644	0.14093
Stemmer with porter	0.16631	0.16190	0.14334	0.16608
Stemmer with Krovetz	0.16605	0.16221	0.16776	0.16625
Stemmer with porter and stop list	0.16131	0.15810	0.14254	0.16181
Stemmer with Krovetz and stop list	0.16067	0.15747	0.16284	0.16158

When we compared the F_1 score of our 3 sets of queries (0-50, 51-100, 101-150), we found that for topics 51-100, the F_1 score is highest than the two others. The score of queries 1-50 was very low, so we would say that in these three queries sets, topic.51-100 and topic.101-150 were more closed to the TREC AP88-90.

3.3.1 Comparison between using (or not) stemming

The goal of stemming is to reduce different grammatical forms / word forms to a common base form. It truncates the prefixes and suffixes of a word to have a word form or a stem form. Grouping together several words will improve the indexation. The effectiveness of the stemming depends on the language of the texts of the corpus.

Based on an article published for the International **Journal of Computer Technology and Applications** [5], we'll present briefly two stemming algorithms called Porter and Krovetz.

- Porter Stemmer

  « It is based on the idea that the suffixes in the English language (approximately 1200) are mostly made up of a combination of smaller and simpler suffixes. It has five steps, and within each step, rules are applied until one of them passes the conditions. If a rule is accepted, the suffix is removed accordingly, and the next step is performed. The resultant stem at the end of the fifth step is returned.

The rule looks like the following: <condition> <suffix> -> <new suffix>

For example, a rule (m>0) EED -> EE means "if the word has at least one vowel and consonant plus EED ending, change the ending to EE". So "agreed" becomes "agree" while "feed" remains unchanged. This algorithm has about 60 rules and is very easy to comprehend »

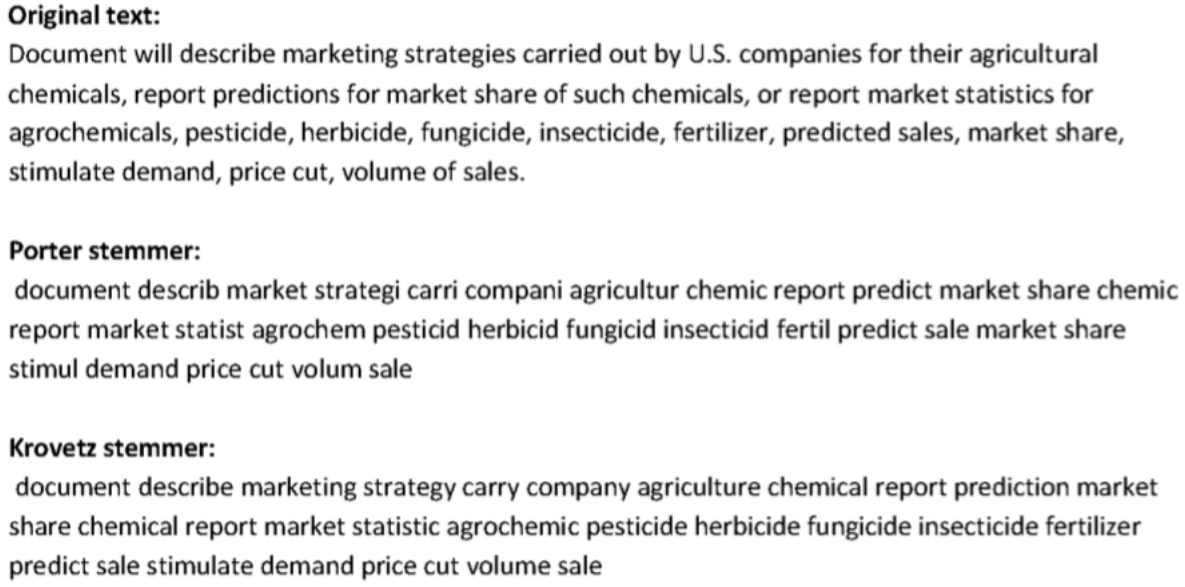
- Krovetz Stemmer

  « Compared to Porter [...] this is a very light stemmer. The Krovetz stemmer attempts to increase accuracy and robustness by treating spelling errors and meaningless stems.

  If the input document size is large this stemmer becomes weak and does not perform very effectively. The major and obvious flaw in dictionary-based algorithms is their inability to cope with words, which are not in the lexicon. Also, a lexicon must be manually created in advance, which requires significant efforts. This stemmer does not consistently produce a good recall and precision performance »

In summary, Porter Stemmer « consists of a series of rules designed to strip off the longest possible suffix at each step » [6]. It produces stems not words while Krovetz Stemmer uses a dictionary and produces words. We will see below a more concrete example.

Figure 9: Stemmer Comparaison



Picture from Cornell University's Course [6].

Now let's compare the result between using stemming or not. We used F_1 score with Language model with Dirichlet smoothing to do our analysis. Figure below (Figure 10) shows us that using stemming improves precision and recall. Porter stemmer gets higher F1 scores, it performs better than Krovetz stemmer on TREC AP88-90.

Figure 10: F1 score LM: Dirichlet and μ=2000

	topic1-50	topic51-100	topic101-150
No stemming	0.05255	0.15482	0.14169
Porter stemming	0.05940	0.18395	0.16608
Krovetz stemming	0.05712	0.18313	0.16181

Figure 11: Precision(topics.51-100)

	tf-idf	bm25	LM
No stemming	0.0967	0.0956	0.0973
Porter stemming	0.1146	0.1098	0.1153

Figure 12: Recall(topics.51-100)

	tf-idf	bm25	LM
No stemming	0.3764	0.3721	0.3787
Porter stemming	0.4520	0.4329	0.4545

We have seen that the use of stemming allowed to slightly improve the f1-score. It is interesting to see now if this improves the MAP (the way the relevant results are ranked across multiple queries). Let's go back to the previous case using the language model but this time to get the MAP and not the F1-score.

Figure 13: MAP (LM Dirichlet Smoothing $\mu = 2000$)

	topic1-50	topic51-100	topic101-150
No stemmer and no stop list	0.1192	0.1892	0.1841
No stemmer with stop list	0.1216	0.1889	0.1848
Porter stemming	0.1554	0.2404	0.2026
Porter stemming and stop list	0.1577	0.2413	0.2037
Krovetz stemming	0.1564	0.2228	0.2035
Krovetz setmming and stop list	0.158	0.2214	0.2038

The last four lines of the table show very clearly that the use of stemming greatly improves the MAP. The use of stemming combined with the use of a stop list allows to slightly improve the results, except for the Porter Stemming. Using stemming does improve ranked results, the relevant documents will be better ranked among the first results.

3.3.2 Comparison between using (or not) stop words

We use stop words to reduce the size of index, and we also use them to reduce the amount of useless terms that may bring noise.

But after comparing the performance with stop words and without stop words, we found out that using stop words will also bring some problems which will effect the performance. Bring in stop words may discard useful terms. For example, among the list of stop words we found the word 'computer'. If it's a key information related to some queries, using stop words will make us miss related information.

The results below are based on the language model with Dirichlet smoothing value $\mu = 2000$. Figure 14 shows the F1 score for the three topics by combining several situations. If we look at the Porter stemming, using stop words improved results for topics 51-150. On the other hand, for the Krovetz stemming, the use of stop words decreases the f1-score.

Figure 14: f1 score

	topic1-50	topic51-100	topic101-150
No stemmer and no stop list	0.05255	0.15482	0.14170
No stemmer with stop list	0.05186	0.15417	0.14093
Porter stemming	0.05940	0.18395	0.16608
Porter stemming and stop list	0.05890	0.18554	0.16625
Krovetz stemming	0.05712	0.18313	0.16181
Krovetz setmming and stop list	0.05638	0.18251	0.16157

We get confused, why did we get higher results with Porter stemming and stop words for topics.51-150 ? After we analyzed, we found out that was because we used another stop list which was provided by Indri, it contained 418 words, instead of using demo stop list with 319 words which is provided by the professor. That's why our results are different from the others groups.

The Figure 15 shows the differences between the two stop word lists. We tested on topics 51-100 with Porter Stemming. When we used Porter stemming with the demo stop words, the recall rate dropped. But with Indri's stop word list, the callback rate increased.

We can hypothesize that removing stop words will generally improve IR performance with TREC AP88-90. But there would be exceptions if we use an "appropriate" stop word list.

Figure 15: Difference between two stop word lists (Porter stemming)

	num_ret	num_rel	num_rel_ret
no stoplist	47092	11946	5430
Indri stoplist(418 words)	47092	11946	5477
demo stoplist(319 words)	47092	11946	5299

3.3.3 Evaluations and analysis based on ranked retrieval results

The results below are based on index file with Porter stemming and with Indri standard stop list. We used language model with Dirichlet smoothing.

- Comparing results between different models for each topic

  In the first situation, we used the same retrieval model for different topics. For instance, with tf-idf model, there is a difference of 0.1085 between topics 1-50 and 51-100 and there is a difference of 0.0806 between topics 1-50 and 101-150.

  In the second situation, we used different models with the same topics of queries. Here, for topics 51-100, the difference between the different retrieval models is about 0.0038 (bm25 and LM) to 0.0085(LM and tf-idf).

  So with the same set of queries, between different topics, the gap of the MAP shows more difference. This means that to compare our system, we should choose quite diverse topics.

  From Figure 16, we can say that the diversity between topics 51-100 and topics 101-150 is small. So we used topics 1-50 and topics 51-100 for comparing our retrieval system with different set of queries. So with different IR models while using language model with $\mu = 2000$ the performance is best.

Figure 16: MAP of different IR models

	topic1-50	topic51-100	topic101-150
tf-idf	0.1243	0.2328	0.2049
bm25	0.1513	0.2375	0.2053
LM, u=2000	0.1577	0.2413	0.2037

  Figure 17 shows that Users are more interested in top 10 ranked results so it will affect theirs happiness. Obviously, query with language model get the best result.

Figure 17: P@10 of different IR models

	topic.1-50	topic.51-100	topic.101-150
tf-idf	0.146	0.432	0.39
bm25	0.17	0.432	0.39
LM, u=2000	0.188	0.456	0.402

- Comparison of the results with the Language Model and the Dirichlet smoothing with 5 different values of μ

« Instead of overtly modeling the probability P(R = 1|q, d) of relevance of a document d to a query q, as in the traditional probabilistic approach, the basic language modeling approach instead builds a probabilistic language model Md from each document d, and ranks documents based on the probability of the model generating the query: P(q|Md) » [7]

Language model with Dirichlet smoothing considers the term frequency in the collection. The model is: $P_{Dir}(w_i|D)=\frac{tf(w_i,D)+\mu P_{ML}(w_i|C)}{(|D|+\mu)}$, and $tf(w_i, D) = |D| * P_{MLE}(w_i|D)$.

So we could transform the model to: $P_{Dir}(w_i|D)=\frac{|D|}{|D|+\mu} * P(w_i|D) + \frac{\mu}{|D|+\mu} * P(w_i|C)$.

The use of collection model is influenced by document length. If document length is small in order to strengthen the collection, μ values have to be larger. Else if the document length is large to weaken the collection, μ value should be set smaller.

From Figure 18, we found that for topic.1-50 and topic.51-100 with μ=2000, it produced the higher results for these two topics. For topic.101-150, μ=4000 get better performance. So if we want to get the best result, we need to adjust μ.

Figure 18: MAP of different u value

	u=1000	u=2000	u=2500	u=4000	u=6000
topic.1-50	0.146	0.1577	0.1569	0.1525	0.1493
topic.51-100	0.2392	0.2413	0.2397	0.2359	0.2310
topic.101-150	0.1888	0.1967	0.1980	0.1998	0.1995

4 Local methods for query reformulation with Relevant Model

We used the words from the titles in topics as our query keywords, they just contained limited information. For example, we took the word 'law', we could also find others words like 'act', 'principle' to represent the same meaning. We recognized them as synonyms. It will impact the recall rate. For improving our results, there are two methods, one is local method, another is global method. Local method adjust a query relatively to the documents that initially appear to match the query. Global method is a technique for expanding or reformulating query terms independent of the query and results returned from it [8].

The local approach is to expand the query from feedback document, we would use pseudo relevance feedback for expanding the query. We considered using 10 documents and extracting 10 terms, with given 0.75 weight on original query but also with 0.5 weight. Besides, we also did the same experiment by taking 5 documents instead of 10. Fig 19 shows the results for topics.51-100 based on using stemming Porter with Indri stop words list, with language model Dirichlet smoothing equals 2000.

With expansion of the query word, from MAP values we could see, the result is better than without using it. When we used 10 documents and 10 terms to expand the query, while we assigned 0.5 weight on expanding query, the MAP is higher than when we assigned 0.25 weight on expanding query. With the same weight for expanding query, using 5 documents and extracting 10 terms performed better than using 10 documents and extracting 10 terms. But for P@5 and P@10, using 10 documents and extracting 10 terms performed better. Extracting more terms get better results but from more documents, it's not for sure.

We compared P@5 and P@10 for the three topics, we can see that the result with expansion gets better. User doesn't have to enter more information manually by completing the query and it's an effective way to improve its happiness.

Figure 19: MAP with local method on topic.51-100

	without expansion	10d10t0.5w	5d10t0.5w	10d10t0.75w	5d10t0.75w	5d5t0.5w
MAP	0.2413	0.2837	0.2871	0.2684	0.2696	0.2758
P@10	0.456	0.488	0.478	0.474	0.464	0.466
P@5	0.460	0.5	0.5	0.48	0.488	0.476

5 Global methods with two methods for expanding query

As for global methods, we built a thesaurus automatically to expand a query. Here we used two ways to calculate the weight of the relationship between two terms. First one is based on the frequent of the co-occurrence terms. $P(t_j|t_i) = \frac{\#cooc(t_i,t_j)}{\sum_{t_k}(t_i,t_k)}$. Another one is using PMI(pointwise mutual information): $PMI(t_i,t_j) = \frac{P(t_i,t_j)}{P(t_i)P(t_j)}$.

Within distance to 5 in an internal window, we calculate the frequent of co-occurrence terms, we would take top 10 from these co-occurrence terms. From these two methods, we will get two different expansion lists because the calculating method is different. We used the expansion lists separately for querying on topic.51-100 with default language model in Indri, with stemming Porter and Indri stop words list. We used the new formula for building a new query model. $P(t_j|Q)$=$\lambda P_{ML}(t_j|Q) + (1-\lambda) \sum_{t_i \in Q} P(t_j|t_i)P_{ML}(t_i|Q)$

For Figure 20, we used λ=0.5, from the results, we found the performance ranking is: Local method > Global method (freq_cooc) > Global method (PMI). By using global method to improve the performance, the effect is not obvious. Local method works better than it.

Figure 20: Performance on topic.51-100

	freq_cooc	PMI	local method	no expansion
MAP	0.2494	0.248	0.2871	0.2397
P@10	0.468	0.476	0.478	0.458
P@5	0.484	0.496	0.5	0.46

With different relevance model, the expansion queries would be different. Also with different λ setting in our queries, it would get a different result. Here, we tried to extract relevant term by $P(t_j|t_i)$ model, we set the distance to 10 to find the co-occurrence term within an internal window. In Figure 21 we used λ=0.5 and we executed our query on topics.51-100, with default language model in Indri, with stemming Porter and Indri stop words list. After setting the distance from 5 to 10, the performance is better. With λ=0.75 assigned to original query, the performance raised a little but not obviously.

Figure 21: Performance with different setting

	0.5w_dist10_freq_cooc	0.75_dist10_freq_cooc	0.5_dist5_freq_cooc	no expansion
MAP	0.2514	0.2499	0.2494	0.2397
P@10	0.472	0.472	0.468	0.458

6 Related work

- Extract the title words from topics

  How to deal with the special characters in our title of the topics ? First, for the symbol '.', we replaced it by null character. Also, if the character is not a letter of an alphabet or a digit, we replaced by blank space.

```
result = re.match(r"<title>\s*Topic:\s*(.*)\n", line)
title_str = title_str.replace(".","")
title_str = ''.join([(x if x.isalpha() or x.isdigit() else ' ') for x in title_str])
```

- Compare the stop words with the words in title

  We would like to see how many stop words there are in our topics titles. So we scripted to compare these titles and stop words list. We found for topic.1-50, the word 'computer' was showed in title but also in stop words list. That reminded us the stop words list we used was getting from Indri. That's why we compared it with another stop words list in 3.3.2 Figure 15

```
rep = []
for s in stp:
    for t in titleword:
        if s == t:
            rep += [s]
```

- Extracting relationships between terms with two measures

  Here, we would like to mention that below implement code was scripted by Jiechen Wu, we used this script to get the relevant terms with different settings:

  1. Terms in distance set to 5
  2. Filter noise if frequency less than 10 times
  3. Select top 10 terms for relevant terms list

  The script was written by Python with NLTK(Natural Language Toolkit). It contains the methods to process tokenized and calculate the frequency for each item.

  – nltkConditionalFreqDist - Mainly used to calculate the conditional frequence distribution. Can be taken as dict which each value is the FreqDist for each key. Take tuple list as parameter.
  – nltkFreqDist - Derived from high performance Counter object in python. Calculate the frequency for each item.

  Using method freq_cooc, $P(t_j|t_i) = \frac{\#cooc(t_i,t_j)}{\sum_{t_k}(t_i,t_k)}$, "cfd[term_i][term_j]" returns the frequency for $\#cooc(t_i, t_j)$, FreqDist returns the frequency of term_i and other terms within internal windows in the whole file.

```
for term in wnd[-WND_SIZE:-1]:
   cfd[term][new_term] += 1
   cfd[new_term][term] += 1

p_term_j_in_term_i = cfd[term_i][term_j] / (list_freq[term_i]*TERM_DISTANCE*2)
```

  Another method PMI(pointwise mutual information): $PMI(t_i, t_j) = \frac{P(t_i,t_j)}{P(t_i)P(t_j)}$,

```
pmi = math.log10(cfd[term_i][term_j]*N /
    (list_freq[term_i]*list_freq[term_j]*(TERM_DISTANCE*2)))
```

  Because the different query item may get the same expansion term. We should sum the same expansion term scores for the different query items t_i, $\sum_{t_i \in Q} P(t_j|t_i)P_{ML}(t_i|Q)$

```
list_scores = {ex_term: cfd[query_term][ex_term] for ex_term in expan_candidats}
#Here we do the sum part
expans = expans + nltk.FreqDist(list_scores)
expans_weight_query = "#weight( " + ''.join(['{:.3f} {} '.format(s, w) for w, s in
    expans_list.items()])+')'
```

7 Conclusion

Doing this study, we learned how to use Indri to build an index and to use a query model to perform the information search. We also learned how to implement a search pattern for query expansion.

We saw that the indexing process could be altered with the use of a stemming method or a stop list. While using stemming and stop lists, the index file size may be reduced. With a stop list, the index time and query time will be decreased.

It's not only a question of size or time, stemming does improves precision and recall. On TREC AP88-90, Porter stemming performs better. Using a too large stop list may give worse results. In that case removing stop words will generally improve IR performance, but there would be exceptions if we use an "appropriate" stop word list.

We learned by analyzing Porter stemming with language model, that the combination of a good stemming method and a good research model, allows to return documents which are more related to user's requirement. About the Language model with Dirichlet smoothing, we have seen that we needed to adjust μ for getting the expected good result.

We also met some difficulties like building a thesaurus automatically according to different models or understanding the formula of new query model.

The local method (the pseudo relevance feedback) allowed us to improve our results even more, as we have seen on the topics.51-100. We have seen the importance of choosing the weight because assigning 0.5 weight on feedback terms, got better performance than assigning 0.25 weight.

The number of documents to use also has an impact. With the same weight, using 5 documents and extracting 10 terms got an higher MAP than with 10 documents and 10 terms. As for P@5 and P@10, using 10 documents and 10 terms performed better. These choices are important because it's an effective way to improve user's happiness.

The global method gave better results than without expansion for the topics.51-100 but these results are worse than those of the local method.

Comparing local method and global method, we found that using local method performed better than global method.

References

[1] Center for Intelligent Information Retrieval. *A Comparative Study of Stemming Algorithms*. July 2016. URL: https://www.lemurproject.org/components.php.

[2] *Precision and recall*. 2018. URL: https://en.wikipedia.org/wiki/Precision_and_recall.

[3] *Evaluation of ranked retrieval results*. 2009. URL: https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-ranked-retrieval-results-1.html.

[4] *F1 score*. 2016. URL: https://en.wikipedia.org/wiki/F1_score?oldid=747315814.

[5] Anjali Ganesh Jivani et al. *A Comparative Study of Stemming Algorithms*. 2011. URL: https://pdfs.semanticscholar.org/1c0c/0fa35d4ff8a2f925eb955e48d655494bd167.pdf.

[6] *Stemming*. 2013. URL: http://www.cs.cornell.edu/courses/cs4300/2013fa/lectures/processing-text-2-4pp.pdf.

[7] *Language models for information retrieval*. 2008. URL: http://www.dsc.ufcg.edu.br/~sampaio/Livros/IntroductionToInformationRetrieval/CapituloACapitulo/12lmodel.pdf.

[8] Manning et al. *Introduction to Information Retrieval*. Cambridge University Press, 2008.