Cover Page

Yao Huang

yh569@georgetown.edu

COSC-488-01

Information Retrieval Search Engine Project - Part 3

Status of Assignment: complete

Time spent on project: 50 hours

Things I wish I had been told prior to being given the assignment:

I wish I could have been told earlier about a more efficient way of testing the effects of different methodologies. I am trying different parameter values referring to the slides in class, since there are a lot of combinations. Also, I wish I was told earlier about the ‘metric’ meaning in the project requirements. But I emailed the TAs to ask about it.

For details, I have put all code on my github repo. Check:

https://github.com/youko70s/IR-SearchEngine

Design Document

Relevance Feedback

In this experiment, I implemented both query expansion and reduction techniques. I implemented pseudo relevance feedback (for query expansion), rocchio relevance feedback and index pruning (in the below, referring as 'common') for query reduction. I also implement a way to use both expansion and reduction.

To better compare those methods and how parameters could affect the performance, we set our baseline models as:

- Expansion: use title in queryfile.txt. title.txt
- Reduction : use narrative in queryfile.txt. narrative.txt
- Expansion + Reduction: use narrative in queryfile.txt. narrative.txt

I have already put baseline results in baseline.

Considering performances using different index and metric done before, the baseline results were retrieved based on single term index using BM25 as metric.

To see the implementations of Expander and Reducer, check: relevance_feedback.py

Query Expansion

In this experiment, I implemented pseudo relevance feedback to do query expansion.

$ python query_expansion.py [raw-results-path] [expanded-results-path] [n] [t] [threshold]

- [raw-results-path] path of the retrieved results using raw queries. (Here: ./baseline/title.txt)
- [expanded-results-path] path to store the retrieved results using expanded query
- [n] number of top documents (i.e. number of documents in relevant set)
- [t] number of top terms in relevant set
- [threshold] minimum document frequency for the terms to be added

example command:

$ python query_expansion.py ./baseline/title.txt ./relevance_feedback/expansion.txt 10 10 2

In pseudo feedback system, there are several issues we want to consider. How many top documents are we going to use? How many top terms are we going to pick? Does the added terms make sense?

To measure these parameters and how they affect the quality of the engine, I tried different n, t, df_threshold. When measuring one parameter, I fixed the other two variables.

- n: [50,20,5,1]
- t: [20,10,5]
- df_threshold: [10,5,2]

To make it easier, I created a demo file for trying all these parameters. To run it:

$ python expansion_demo.py [results_dir]

- results_dir: specify the directory you want to store the expansion demo results for various parameters.

Query Reduction

In this experiment, I implemented two methods for reducing a query

$ python query_reduction.py [raw-results-path] [reduced-results-path] [reduce-method] [n] [threshold]

- [raw-results-path] path of the retrieved results using raw queries. (Here: ./baseline/narrative.txt)
- [expanded-results-path] path to store the retrieved results using expanded query
- [reduce-method] should be one the the following: rocchio, common
- [n] number of top documents used for rocchio reduction (i.e. number of documents in relevant set)
- [threshold] reduced size proportion of the raw query. Should be a value between 0-1.

An example command would be:

$ python query_reduction.py ./baseline/narrative.txt ./relevance_feedback/reduction.txt rocchio 10 0.5

About the two methodologies:

- index pruning with query threshold (common): this is using the similar idea as index pruning. Based on the threshold (0~1), it will select the terms with largest tf*idf and use them as the reduced query.
- rocchio: this is based on rocchio algorithm.

To measure the performance of the two methodologies, and the effect of threshold, I let the threshold to be 0.2,0.5,0.7 for both two methodologies. (notice that I fixed n to be 5). To make it easier, I also created a demo file for you to try out. To run it:

$ python reduction_demo.py [results_dir]

- results_dir: specify the directory you want to store the reduced demo results for various parameters.

Query Expansion + Reduction

This is the implementation for using both expansion and reduction.

$ python [raw-results-path] [new-results-path] [reduce-method] [n] [t] [df_threshold] [threshold]

The parameters here are the same as mentioned above.

example command:

$ python ./baseline/narrative.txt ./relevance_feedback/both_implementation.txt common 5 10 2 0.5

I have chosen the optimal parameters in this case and created a demo file for you to run directly. To run it:

$ python both_demo.py [results_dir]

Report & Analysis

Query Expansion

For different n values, remember we discussed in class. If the n is too large, which means our relevant set size is large, then it might lead to adding redundant words to the raw query when doing expansion. However, if the n is too small, it might lead to missing some important documents for the query expansion.

Here, I tried several different values for n (see table 1). When fixing the other variables (t, df_threshold), we can see that when n is very large (50), the MAP and R-precision are both very low. When we are using exact top 1 document, it achieves highest MAP and R-precision. However, none of these exceeded the baseline performance.

	n=50	**n=20**	**n=5**	**n=1**	**Baseline**
Runtime	182.71	152.17	99.41	82.79	14.3
MAP	0.2012	0.1868	0.2722	**0.2951**	0.3739
R-Precision	0.1368	0.1831	0.2653	**0.2764**	0.3475

Table 1: Query Expansion Based on Top n Documents Using Different n (t=10, df_threshold = 2)

It makes sense, as in our case, we have a relatively small document collection, so although we were basing our expansion on the top documents we retrieved, many of them were still irrelevant to the query. That is why using a lot of documents would actually adding much disturbing information, thus leading to adding 'bad' terms to the original query. When we are only using the top document retrieved, the performance was much better. However, it was still not good enough, because if the top document was actually irrelevant, then it would still add bad terms to the raw query.

Next, we are trying out different t values. We fixed t to be 5 and df_threshold to be 2 in this case. Similar to using different n values, if t is too large, it might lead to adding irrelevant terms to the original query; if t is too small, then it would probably miss many useful documents. The results are put in table 2 below.

	t=20	**t=10**	**t=5**	**Baseline**
Runtime	87.2	93.76	104.87	14.3
MAP	0.2690	0.2722	**0.2972**	0.3739
R-Precision	0.2612	0.2653	**0.2734**	0.3475

Table 2: Query Expansion Based on Top t Terms of Top 5 Documents Using Different t (df_threshold = 2)

In this case, a little surprisingly, the performances do not differ a lot, though again, they all perform worse than the baseline model. However, when t = 5, the performance was slightly better than using other t values.

While working on query expansion, I noticed that, since I was using n*idf as measure, the idf for some special tokens/digits, for example, '$1200', are very high, thus leading to its n*idf very high. In this case, the top terms might include these terms, which seems that adding them as

expanding the query won't make sense. Considering this, I put a df threshold (the minimum document frequency) to work around the issue with very infrequent terms like numbers. For instance, if a term only appears in 1 document, it may not be a good expansion term to add. To see how it will affect the results, I fixed n to be 5, t to be 10, and experimented. If the df_threshold is very high, it tends to exclude those nonsense terms; while the df_threshold is small, it tends to include more nonsense terms.
The results are provided below in table 3.

	df_threshold =10	**df_threshold=5**	**df_threshold=2**	**Baseline**
Runtime	112.14	108.37	104.9	14.3
MAP	0.2671	0.2679	**0.2722**	0.3739
R-Precision	0.2596	0.2451	**0.2653**	0.3475

Table 3: Query Expansion Based on Top 10 Terms of Top 5 Documents Using Different df_threshold

We can see that, again, the performances do not seem to differ a lot. There might be possible that the values we tried were at the same level compared to the size of our collection; It's also possible that df_threshold doesn't affect the performance much.
In general, the n parameter affects the performance most according to our analysis.

Query Reduction

In the query reduction, I tried two methodologies. The common method is of similar idea to index pruning. We also want to see how the reduced size of query would probably affect the performance of our engine. The results are provided below.

	Common			Rocchio			Baseline
	thre=0.7	**0.5**	**0.2**	**0.7**	**0.5**	**0.2**	
Runtime	150.56	142.5	3.76	25.89	5.57	0.88	371
MAP	0.2283	**0.2602**	0.2119	0.3723	**0.4096**	0.2196	0.1588
R-Precision	0.1680	**0.1942**	0.1862	0.2665	**0.2830**	0.2156	0.1000

Table 4: Query Reduction Using Common and Rocchio With Different Thresholds

Surprisingly, in this case, the performances differ a lot. We have the following observations:

- In general, Rocchio algorithm, which uses irrelevant document set to exclude noise terms, performs much better than simply using common method, which is based on processing only terms with high tf*idf. This makes sense, since while we consider both irrelevant set and relevant set, we are more likely to exclude terms that do not really contribute to the search.
- While threshold does affect the performance, both too long and too short are bad. While using a threshold of 0.5, it got the best performance in both retrievals. While the query is reduced to very short, it will probably miss important terms; in this case, however, the baseline model is actually the worst.
- Rocchio runs much faster than common method.

Expansion + Reduction

Considering the above results, for implementing both techniques for our feedback system, I am choosing the optimal parameters.

For query expansion: n = 5, t = 10, df_threshold = 2

For query reduction: use rocchio method, with threshold = 0.5.

The performance is shown below.

Both	**Feedback**	**Baseline**
Runtime	111.97	371
MAP	0.3619	0.1558
R-Precision	0.2288	0.1000

Table 5: Implementing Both Expansion and Reduction on Query With Optimal Parameters

In general, we could see while implementing both reduction and expansion techniques, it got a relatively high MAP compared to using expansion alone.