GEORGETOWN UNIVERSITY

Georgetwon University
Computer Science Department

Cosc-488

IR - P3 Document

- State: *Complete*
- Time: *25 hrs*
- Things to be told: *N/A*

Contents

1 Overview

In P1, we implemented the indexing part of the search engine. Four different indices, that were created, included:

- Single term index
- Stem index
- Positional (aka proximity) index
- Phrase index

Afterwards, in P2, we processed a given query and then retrieved ranked list of documents considering retrieval models. The main part of P2 was devoted to implementing retrieval models including:

- Vector Space Model ($\oplus$ Cosine Similarity)
- BM25
- Language Model ($\oplus$ Dirichlet Smoothing)

Currently, in P3, we aim to use some IR techniques to improve the retrieved documents. The techniques include:

- Query Expansion
- Query Reduction

The details are explained in associated sections.

2 Design Document

2.1 Query Expansion

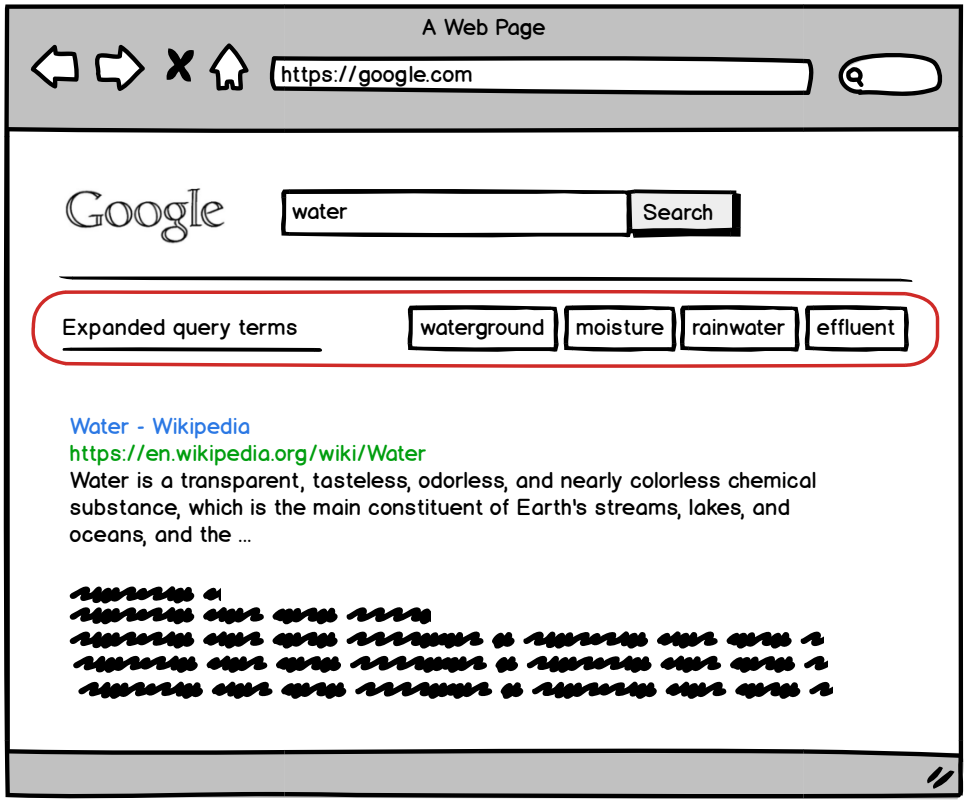

Figure 1: Query expansion schema

Roughly speaking, query expansion approaches are expected to increase the search engine performance in Recall and Precision metric because they help bringing up more documents from the relevant set by exploring new terms (i.e., expanded terms). To test the effect on search engine performance, we employed Pseudo Relevance Feedback (PRF) which is amongst the most popular query expansion approaches. In PRF, we tend to first retrieve root documents –by passing original query, and then extract expanded terms from these documents.

2.1.1 Retrieving First Generation Documents

As mentioned earlier, we retrieve the first generation documents (root documents) by passing original queries to query processor module. The query processor then

will make some pre-processing modifications to received queries and then send them out to the retrieval models. What we will receive finally is a set of documents (we assume them to be relevant) that has been ranked by the retrieval model. This is exactly the task that we had in P2. *It has also to be noted that since language model was the best-performing retrieval model in previous phase, we merely use it as our retrieval model in P3.*

2.1.2 Identifying Expanded Terms

After retrieving the relevant documents, we need a similarity measure that captures the similarity between query terms and retrieved document tokens. Having a look at literature, different similarity measures including: Jaccard Coefficient, Dice coefficient, tf-idf, and etc. can be utilized for calculating similarity. There are also other methods proposed in the past to enhance query expansion. Among them, we use Query Language Modeling (QLM) for retrieving context terms and augmenting the given query. The QLM is calculated as follows:

$$p(t|M_{rel}) = \sum_{\theta_d \in R} [p(\theta_d) p(t|\theta_d) \prod_{i=1}^{n} p(q_i|\theta_d)] \quad (1)$$

where M_{rel} denotes the set of relevant documents which are obtained as the first generation of documents by using the main unmodified queries, θ_d and θ_q are the document and query language models, $p(\theta_d)$ indicates the document selection probability and set to be uniform in our experiments.[1] q_i signifies the ith term in the given query. Term documents are ranked based on their estimated value for $p(t|M_{rel}$ and top n terms are selected as expanded terms.

It has to be noted that in expansion approach, after finding context words, we augment (append) the obtained terms to the original query set.

2.1.3 Retrieving Next Generation Documents

After augmenting expanded terms to the main queries, we send the modified query set to the retrieval model. This model will take care of retrieving the next generation of documents with regard to the modified query set.

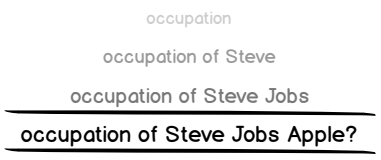

Figure 2: Query expansion schema

2.2 Query Reduction (Thresholding):

In Query Reduction approach, our main goal is to "*improve query run-time by partial result set retrieval*" [2] which finally will mostly yield a big improvement in CPU performance in terms of new retrieved docs per cycle.

There are approaches to set constrain on the given query including:

- **Position thresholding**: In this approach, we remove query terms considering the given threshold from the last query term toward the first.
- **Goodness thresholding**: In this approach, we remove query terms with regard to their specific features such as: tf, idf, qtf, tf-idf and etc.

Important note: by setting the threshold as $x\%$, the program will assume reducing $x\%$ of the query. That is, $(100 - x)\%$ of the query will remain and be sent to the query processor. We also round threshold to obtain an integer number for reduction boundary in case a float number for reduction comes up. An example may enlighten this:

Example: Take "domestic violence child" (3 query terms) as an example. Suppose that we tend to apply a 40% threshold on this query. By our definition of threshold, we know that we should hold 60% of the query which equals to $3 * (60/100) = 1.8$. Obviously, we need an integer number (upper bound) to set our decision boundary. So, by rounding this amount to the nearest integer which equals to 2, we maintain 2 first query terms i.e., "domestic violence".

It should be noted that in this phase, we tend to use idf as "*goodness*" metric and remove query terms based off of their idf (i.e., higher idf's are kept). We will be experimenting with both approaches to see their results.

[1] It can be estimated using some methods such as Topic Modeling, though
[2] quoted from Class Lectures

3 Results and Analysis

In this section, we aim to present experimental results, along with an analysis elaborating those results. The experimental plan that we follow in this section is investigating the effect of different model parameters.

3.1 Query Expansion Results *(Report 1)*

Table. (1) summarizes our experimental results for query expansion runs in terms of P@k, R@k in some ranking points, and MAP. As shown, it seems that generally expanding the given queries would result in enhancing Precision and Recall metrics at different ranking points. More specifically, expanded terms approaches outperform the baseline in 5 (out of 7) metrics. When comparing expanded approaches, we see that expansion with only one term achieves the highest results in $P@5$, $P@10$, $P@20$. This improvement indicates that not only can expansion with single term improve the overall precision (at different points) compared to the other expansion approaches, but it can also beat the baseline in these metrics. This also signifies the fact that by appending a good term (expansion term) to the main query, we will be able to retrieve more relevant number of documents in better rankings (boost their rankings).

While comparing achieved $R@k$ between the baseline and expansion approaches, we notice that expansion approaches generally outperform the baseline slightly (except for $R@10$ which difference is much higher). All together, it turns out by expansion, we can enhance the recall in overall as expected. Since by expansion, we generally retrieve more relevant documents (on the basis that we have expanded the query with salient expansion terms). For instance, there might be some relevant documents where in which none of the baseline's query terms has occurred, but an expansion candidate has. It is obvious that these kinds of documents can be captured by using expansion terms.

As we increase the number of expanded terms in our experiments, it stands that there is a drop in precision and recall metrics (except for $R@5$). This might due to the fact that the other expanded terms (terms #2, #3) are not really good estimated candidates for expansion. That is to say, they might also appear in irrelevant documents as much as they do in relevant documents (or eve more). This can also be improved by some methods such as: 1) estimating document prior probability ($p(\theta_d)$ in Eq. (1), instead of assuming it to be uniform across the collection. 2) using Expectation-Maximization (EM) algorithm which discriminates the first generation documents (relevant documents) from the entire collection. In fact, these algorithms

#Expanded terms	P@5	R@5	P@10	R@10	P@20	R@20	MAP
0 (baseline)	0.2100	0.2547	0.1800	**0.4000**	0.1175	0.4332	**0.3196**
1	**0.2700**	0.2513	**0.1900**	0.3307	**0.1375**	**0.4430**	0.3155
2	0.2300	**0.2729**	0.1650	0.3008	0.1175	0.4340	0.2672
3	0.1200	0.0946	0.1000	0.2189	0.0800	0.2327	0.1524

Table 1: Query expansion results considering number of expanded terms

will look for those features (terms) in relevant documents set which distinguish them from the whole set of collection.

Although expansion approaches has performed well in $P@k$ and $R@k$ metrics at first rankings, they under-perform the baseline in terms of MAP (which is Mean Average Precision). This also tells us the fact that our expansion approaches mainly affect the $P@k$ metrics for lower k's, but not for all values of k. This can be justified if we take this observation into account: expansion approaches can effectively retrieve and boost relevant documents to better ranks, but they also retrieve irrelevant documents at lower ranks which results in softening the overall average precision. This problem can efficiently be addressed by the aforementioned approach (Approach (2): EM algorithm).

In summary, we learn that the key to query expansion is to select the most proper term candidates for the expansion, instead of just blindly choose terms (In P3, we also used *Query Language Model (QLM)* to find the expansion terms).

Threshold (reduced) %	#retrieved	#retrieved relevant / #total relevant	MAP
0 (w/o reduction)	23491	**140 / 146**	**0.2236**
25	21239	138 / 146	0.1836
50	18439	135 / 146	0.1591
75	12162	130 / 146	0.1512

Table 2: Query reduction, position threshold

3.2 Query Reduction Results *(Report 2)*

As mentioned in previous sections, we have implemented query reduction approach in two ways including: 1) Position threshold, and 2) Goodness threshold *(See Sect. 2.2)*. Table. (2) shows the results of position threshold, whereas Table. (3) demonstrates the goodness threshold.

Threshold (removed) %	#retrieved	#retrieved relevant / #total relevant	MAP
0 (w/o reduction)	23491	**140 / 146**	0.2236
25	12952	136 / 146	0.2259
50	5472	128 / 146	**0.2940**
75	1063	57 / 146	0.1736

Table 3: Query reduction based on goodness order (query term idf)

When comparing two settings with each other, as expected, goodness threshold which is based on reduction of more common terms (lower idf) almost significantly outperforms the position threshold. To be more specific, it achieves MAP value of 0.2259 when 25% of the query is reduced, while position threshold attains MAP value of 0.1836 in the same situation. It is also interesting that best-performing settings of query reduction approach (i.e., 25% reduction) have different retrieved document numbers (Table. (2) is almost twice Table. (3). This also signifies the fact that removing common terms, which are noisy signals (they occur in relevant and irrelevant documents) would help the model capture relevant documents, while ignore retrieving irrelevant documents. Another observation between two tables is that when reducing query for 25% based on term positions, the system can retrieve 138 relevant documents (out of 146 whole relevant), while goodness threshold retrieves 136 relevant documents in the same situation. Although position threshold can retrieve 2 more relevant documents, compared to goodness threshold, it achieves lower MAP since it contains twice retrieved documents of goodness threshold, of which most are irrelevant. So, this has significantly dampened the effect of those 2 more retrieved documents.

Comparing intra-table relations, we see that when we use position threshold (Table. 2), as we increase the percentage of reduction, the MAP value decreases in Table. (2) and increase in Table. (3) until some threshold 50%. This observation indicates that blindly removing terms based on their position would hurt the overall performance. However, that is almost not the case in goodness threshold (Table. 3) as we see the best-performing setting is when we reduce the query for 50%. This observation would lead us to the fact that removing common terms meaningfully helps the retrieval to some fixed points (should be obtained empirically). It is also worthwhile that the best-performing setting in goodness threshold is when we reduce the query for 50% (MAP: 0.2940) and could remarkably beat the baseline which achieved MAP value of 0.2236.

Table. (4) shows statistics of average query length before and after doing reductions. It has to be noted that for query reduction approach, we have utilized the long queries

(narrative).

As the final analysis, Table. (5) shows the results on the following query:

Title: *"food/drug laws"*

Narrative: *"A relevant document will contain specific information on the laws dealing with such matters as quality control in processing, the use of additives and preservatives, the avoidance of impurities and poisonous substances, spoilage prevention, nutritional enrichment, and/or the grading of meat and vegetables. Relevant information includes, but is not limited to, federal regulations targeting three major areas of label abuse: deceptive definitions, misleading health claims, and untrue serving sizes and proposed standard definitions for such terms as high fiber and low fat."*

Having looked at this the results, it stands that finding a good boundary for query reduction would help the retrieval in terms of F1 (overall combination of P and R). It is also interesting to see that 25% and 50% reductions would yield MRR 1. It means that in these settings, the first retrieved document is considered relevant. While 0% and 75% retrieve the first relevant document in ranking 8. Although thresholds 25% and 50% have higher MRRs, they lose to threshold 75% in terms of F1. This might be due to the fact that these thresholds work best on the first initial rankings, but not for the further rankings (they can boost relevant documents to better ranks, but their result set also contains irrelevant documents in lower ranks). All together, this observation suggests that removing common terms from the query would improve the overall precision of retrieval, while decreasing recall. As shown, F1 score always has an upward trend indicating that query reduction is generally boosting the overall retrieval performance on this sample query.

Threshold (reduced) %	**Average query term length**
0 (w/o reduction)	23.15
25	17.59
50	11.87
75	5.93

Table 4: Average query term length

4 Libraries

In this implementation, as well as P1 and P2, the built-in libraries of python have been used:

- **NLTK:** for stemming purposes.

Threshold (%)	Precision	Recall	F1	MRR
0	0.0376	**0.9846**	0.0724	0.1250
25	0.0518	0.9692	0.0983	**1.00**
50	0.1250	0.8615	0.2183	**1.00**
75	**0.3953**	0.2615	**0.3148**	0.1250

Table 5: Query reduction (goodness reduction) results on a sample query

- **time:** for capturing time.
- **math:** for computing math formulas related to each model.
- **operator:** for sorting the documents based on their scores after the scores are assigned by the model.
- **json:** for writing and loading json files into the memory.
- **os:** for handling file directories.
- **sys:** for getting arguments from command-line.
- **pickle:** for storing/loading document tokens into disk/memory.
- **argparse:** for defining command line arguments. *(See Sect. 5)*

5 Appendix

5.1 How to run program?

For running the program, you will need to first create indeces. For experiments, we will use single-term index. So, use the following command in terminal to create it:

```
$ python build.py /tmp/Mini-Trec/BigSample/ single /tmp/my-indexes/
```

While implementing, I did not separate the logic of P2 and P3 since I think P3 is an extension of P2 in some way. Therefore, we will be using the same command sets of P2 with some added parameters. The structure and commands are as follows:

```
$ python query.py [--index_dir] tmp/my-indexes
                  [--query_dir] tmp/data/queryfile.txt
                  [--retrieval_model] lm
                  [--index_type] single
                  [--result_dir] /result/results_exp.txt
                  [--expansion] [--threshold]
                  [--threshold_value] 10
                  [--position_threshold][--goodness_threshold]
                  [--exp_term_threshold] 2
```

- `--exp_term_threshold`: number of top terms selected for query expansion. (default: 1)
- `--threshold_value`: percentage of mask used for query reduction. (default: 10)

For convenience, just ***copy*** the following and paste them into your terminal:

1. Expansion:

```
$ python query.py --index_dir tmp/my-indexes --query_dir tmp/data/
   queryfile.txt --retrieval_model lm --index_type single --
   result_dir /result/results_exp.txt --expansion
```

2. Reduction:

```
$ python query.py --index_dir tmp/my-indexes --query_dir tmp/data/
   queryfile.txt --retrieval_model lm --index_type single --
   result_dir /result/results_red.txt --threshold --threshold_value
    20 --goodness_threshold
```