Christopher Zawora
Information Retrieval
11/13/15

Project 2 - Query Processing

Cover Page

Project: Complete

Hours worked - 40 hours

Things I would like to have known:

What optimal results might look like.

This project used the revised qrels.txt file

Design Document

The general idea here is to first process the query texts using the preprocessing methods from project 1 and then using the query text and inverted index to retrieval and rank documents.

Initially what must be available are all four indices from the previous project. This includes the inverted index and the lexicon. What also must be pre-generated is another index that will be useful when normalizing query rankings. This a document-term index. It will consist of a document ID and a list of term ID's for every term that it contains.

For obtaining and preprocessing the query texts, each query text is associated with a query number. Then the query text is put through the normalization process. This case folds all the letters to lowercase, replaces all the useful &**** strings with their unicode equivalents. Then all the dates are found and normalized into the mm/dd/yyyy format. This will still allow for partial date lookups since all other '/' characters will be removed. So a month from a date can be searched with 'mm/' and a year can be searched with '/yyyy'.

After dates have been normalized then the query will be tokenized. For all but the phrase index the tokenization will first capture IP addresses, URL's, emails, standardize numerals, and capture monetary values. It will then deal with abbreviations and split the text on all forbidden characters. Finally, it will deal with the hyphenated words. However, for phrases the goal is to keep words together that are not separated, so all special tokens and characters will be replaced with a "<" char. That way one final split can be done by '<' and this will leave you most of the phrases. Then all that is left to check for phrases is the stop words and length of the resulting splits. For the positional index, the term location in the document will also be recorded.

Each query text will then be processed into four different formats: single term, single term positional, stem, and phrase.

Next, is the process of searching and ranking. For the single term, stem, and phrase indices the search will return all the documents which contain the query term's term ID. However, for the positional index the additional constraint of making sure terms appear in positional order will also be applied when retrieving documents. Since the queries are short and I don't want to miss any documents, I will set the positional-phrase threshold for query terms checked against the positional index to a sequence of only two terms.

While each query will ultimately be evaluated to only one of the indices, some checks are necessary to determine which index is most suited for each query. The index to be used will depend on both a phrase threshold and a positional threshold. These thresholds will be experimented with and their results analyzed. First the phrase index will be checked, and if the number of documents retrieved pass the phrase threshold then the phrase index will be used for processing. If it fails, then the same will be attempted for the positional index. If the positional index also fails, then either the single term index or the stem index will be used, depending on the user's input to choose a default index.

After the index is determined, the query will be processed with either cosine similarity, BM25, or KL-divergence. The choice of ranking method will also be determined by the user's input.

Finally, once each of the query texts and their retrieved documents is evaluated and scored, they will be sorted, written out to a file, and can be evaluated with treveval.

All of this will be handled in Query_Processor, Index, and Query classes. The Index and Query classes are for keeping the indices, lexicons, and doc-term indices and queries organized, respectively. The Query_Processing class is a wrapper class for all the methods and logic used in processing the queries and writing out the search results. The main function will only take in the command line arguments regarding default index and ranking method as well as create an instance of the Query_Processor.

I plan on using Python again for this project. One main reason for this is due to its advantages when it comes to text and string processing. The other reason is that I foresee its dictionary data structure being useful in quickening lexicon insertions and searches as well for creating the inverted index. Its map function combined with lambda functions might also be useful.

Flow diagram:

QueryProcessor object is made with all the indices, query texts, default index setting, and

ranking method -> all queries are preprocessed (normalized and tokenized) -> documents are

then retrieved for each query and an index is chosen for evaluation based on threshold settings

-> the retrieved documents for each query are then scored and ranked -> resulting documents

are then written to an output file

Analysis

For the following analyses KL-divergence used average document length for a tuning parameter. Also, BM25 used 1.2, 500. 0.75 as tuning parameters K1, K2, and B respectively. These choices were based on the slides seen in class.

Report 1

	single term index		**stem index**	
	MAP	time (seconds)	MAP	time (seconds)
Cosine similarity	0.38	156	0.34	186
BM25	0.44	72	0.45	66
KL-divergence	0.33	72	0.37	84

There are a few things to note in Report 1. The first is that the single term index performed worse than the stem index. This is likely due to stemming's ability to retrieve more documents. The second is that BM25 provided best MAP score of the three ranking methods. This held true for both single term index and stem index. I speculate it was the fact that BM25 used odds which allowed it to return more relevant results. BM25 also took the least amount of time for both indices. On the other hand, Cosine Similarity performed better than KL-divergence with the single term index, however, KL-divergence had a higher MAP than Cosine Similarity with the stem index. The reason for this is that since the stem index returns more documents per term, normalization, as done in the denominator of Cosine Similarity, will produce a lower score.

The other glaring result is that Cosine Similarity took more than twice the time the other rank methods did under both indices. This is definitely due to the fact it and the normalized tf-idf that it utilized both involved denominators that required summation of several values. The function call overhead and simple number of the values needed to normalize for each document, while small, could easily have added a second or two to each query. In fact, this explanation makes more sense when observing that the stem index Cosine Similarity took longer than with single term index. The stem index provided a greater amount of documents for each term and therefore more summations.

Report 2

The variety of data that is presented here came from my curiosity to see how changing the positional threshold and the phrase threshold would change the outcomes of the rankings. In going about this I experimented with several combinations of the two thresholds. One general finding that influenced the data I presented in the report is that as the positional index got too low (below 20) there was a severe drop in MAP scores, therefore it was clearer a far from optimal setting.

Some other information that will inform the following charts:

The phrase threshold was reached by 1 query when set to 20, 3 queries when set to 10, and 4 queries when set to 5.

The distribution of number of documents returned when only the positional index was used: 0,0,0,0,0,0,2,2,3,3,4,5,6,10,11,27,31,68,113,121

The overlap between the queries passing phrase thresholds and those passing positional thresholds was only ever 1 or 2 queries.

The other intention was to take measurements using both the single term index and stem index as default indices, therefore two tables were produced.

default index: single term

	Positional Threshold	**100**	**100**	**100**	**50**	**20**
	Phrase Threshold	5	10	20	20	20
cosine similarity	Retrieved	1534	1625	1807	1775	1706
	Relevant Retrieved	104	105	108	90	77
	MAP/ time(seconds)	0.3/127	0.35/127	0.35/127	0.33/127	0.33/127
BM25	Retrieved	1534	1628	1807	1775	1706
	Relevant Retrieved	109	110	119	101	82
	MAP/ time(seconds)	0.39/58	0.41/58	0.41/58	0.39/58	0.38/58
KL-divergence	Retrieved	1534	1623	1807	1775	1706
	Relevant Retrieved	103	104	113	100	82
	MAP/ time(seconds)	0.29/63	0.3/63	0.31/63	0.29/63	0.29/63

There are several things to note about these results. The first is that MAP scores and times for processing both remained consistent with the relationship between rank method, MAP, and time that Report 1 highlighted. The more interesting things to note pertain to the threshold values. As the phrase threshold increased, there was both an increase in the documents retrieved and the number of relevant documents retrieved as well as an increase in the, sometimes very slightly, in the MAP. This makes sense since there are more likely to be higher df for a term than a

phrase. Additionally, sending a query to the phrase index with too few documents would effectively ignore many of the terms in the query that did not form phrases. So the phrase index, when used, was most helpful when only 1 document was sent to the phase index. Thus it became clear that 20 was the ideal phrase threshold. That value remained consistent as I experimented with positional thresholds.
As I lowered the positional threshold from 100, which only sent at most 2 queries to the positional index, I was effectively increasing the number of queries possibly being sent to the positional index. At 50 there were at most 3 possible queries sent to the positional index and at 20 there were at most 5. MAP scores were barely effected by the lowering of the positional threshold when it was above 20 (lower than 20 saw a steep drop-off in the MAP scores since close to half the queries were passing), however number of documents retrieved and number of relevant documents retrieved both declined. The reason for this was likely due to presence of stop words in the positional index and the consequently lowering effect they had on term weights. So since the a positional threshold of 100 retrieved more documents and more relevant documents, it was found to be optimal, when the positional index was used.

default index: stem

	Positional Threshold	**100**	**100**	**100**	**50**	**20**
	Phrase Threshold	5	10	20	20	20
cosine similarity	Retrieved	1638	1732	1911	1879	1810
	Relevant Retrieved	107	108	114	103	84
	MAP/ time(seconds)	0.3/146	0.32/146	0.32/146	0.32/146	0.31/146
BM25	Retrieved	1638	1732	1911	1879	1810
	Relevant Retrieved	115	116	126	112	88
	MAP/ time(seconds)	0.39/55	0.4/55	0.41/55	0.4/55	0.39/55
KL-divergence	Retrieved	1638	1732	1911	1879	1810
	Relevant Retrieved	103	104	112	101	84
	MAP/ time(seconds)	0.33/57	0.33/57	0.34/57	0.33/57	0.33/57

The results from using the stem index support the observations from Report 1, the stem index provides some advantage. While the trends with regard to phrase and positional thresholds remained the same as with the single term index, the number of documents retrieved and

number of relevant documents retrieved were higher in each instance when the stem index was used. BM25 and KL-divergence were a bit quicker with the stem index than the single term index, but Cosine Similarity took longer to compute with the stem index. These results also reflect the results of Report 1. The MAP scores are also reflect the observations of Report 1.

One final observation is that, when both reports are compared, it becomes apparent that the best MAP scores are produced when no phrases of any kind are considered. This only bolsters the claim that Bag of Words representations of documents work very well. One caveat may exist if the inclusion of phrases perhaps results in more relevant documents being retrieved. However, I did not collect this data for Report 1, and therefore did not include it in this writeup. This is a question for further consideration.