University of Iowa at TREC 2008 Legal and Relevance Feedback Tracks

Brian Almquist,[1] Yelena Mejova,[2] Viet Ha-Thuc,[2] and Padmini Srinivasan[1,2,3]
[1]Department of Management Sciences
[2]Computer Science Department
[3]School of Library and Information Science
The University of Iowa
{brian-almquist, yelena-mejova, hathuc-viet, padmini-srinivasan}@uiowa.edu

The University of Iowa Team, coordinated by Padmini Srinivasan, participated in the legal discovery and relevance feedback tracks of TREC-2008. This is our second year participating in the legal track and the first year in the relevance feedback track.

LEGAL TRACK

1 Introduction

This is the second year that our research group has participated in the TREC Legal Track. Our ad hoc retrieval system has been modified to extract the additional Boolean query fields added to the 2008 topics, and to privilege documents found by the Boolean reference run when conducting our queries. We have also submitted runs that fuse the results from existing runs.

For the relevance feedback task, our system uses ranking information of relevant and non-relevant documents from previously submitted runs to the TREC Legal Track to train a classifier. The classifier is applied to the remaining unjudged documents to create a new ranked list. This approach is applied to sets of input runs, including a hybrid run where a classifier trained on one set of runs is applied to the unjudged documents from another set of runs.

2 Ad Hoc Track

Our index remains unchanged from our submission to TREC Legal in 2007, and new changes to our retrieval system are accomplished through the selection of topic fields for input and post-retrieval processing. A brief review of our system follows.

2.1 The System

The Lucene set of indexing and search tools is extendable and scalable. [1] For our index, we merged the OCR text and document title into a single searchable field, which we indexed using the Lucene StandardAnalyzer. Our submitted runs to the ad hoc task are conducted by constructing a query vector of evenly weighted terms, which is used to query the database. For each topic, we retrieve 1.5 times the maximum submission size for the task (100,000 for 2008), using Lucene's similarity scoring. The documents are reranked according to post-processing steps, and the top 100,000 documents are submitted to the track organizers.

The post-processing step applied to all of our runs consists of calculating a new document similarity score using the Okapi-BM25 formulation. Our implementation of Okapi-BM25 used parameters k_1 = 1.2, and B = 0.75. Each repeated instance of a term in our query vector was treated as a unique term, which is consistent with setting the k_3 parameter to ∞. These new scores are used for the purpose of reranking the set of documents returned by the query.

2.2 Retrieval Strategies

2.2.1 Input Fields

The RequestText field contributes to the queries for each of our runs. For all runs but the requested reference run, our system incorporated the three Boolean queries: the “FinalQuery”, the “ProposalByDefendant”, and the “RejoinderByPlaintiff”. The queries are converted into term vectors by stripping the Boolean operators, with the exception of wildcard characters, from the query. For one run, we included the entire negotiation history in the query vector.

The RequestText field contains language that is common when expressing an information need in the context of a legal discovery request. This content includes terms that are consistent with the form of the request without directly connecting to specific needs of the topic. For certain runs, our system analyzes the RequestText field across all topics to remove words that appear in the field without directly referring to the specific need of the topic.

2.2.2 Query Expansion

Our system uses two techniques to expand the number of terms in our queries. Handling the wildcard operators included in the set of Boolean queries provided a challenge as the system would replace the wildcard term in the query vector with all possible wildcard expansions in the corpus, including all qualifying terms generated by OCR error. We found that limiting this expansion to the two candidate terms with the greatest document frequency provided an improvement in performance while limiting the capacity for the corpus’ OCR error to break our system’s functionality.

Pseudo-relevance feedback, as implemented in our system, begins with a preliminary run to retrieve an initial set of top-ranked documents. From this batch, we extract terms for query expansion from a subset of the top ranked documents. The system re-executes the queries using the exact configuration as the preliminary run, except that candidate query expansion terms have been added to the query term vector.

Based on our earlier research, we use an expansion of five terms extracted from three documents. To qualify for inclusion in the query term vector, a term must consist entirely of letter characters, it must not already exist in the query vector, it must meet a minimum document frequency threshold, and it must not consist entirely of consonants. Terms are also passed over for inclusion if they are not recognized as words by WordNet or if they are included in the SMART list of stopwords, a much more extensive list than the one used by the Lucene StandardAnalyzer. [2, 3]

2.2.3 Reference run boosting

In previous iterations of the ad hoc task, the reference run conducted using the Boolean “FinalQuery” consistently retrieves more relevant documents than other ranked runs, especially those generated using our own system. Such documents, if found in our own runs, are more likely to be relevant, and ought to be assigned a higher ranking. For certain runs, the Okapi-BM25 score for these documents is multiplied by a "boost" factor before the ranking of our result set. The amount of the boost was found through testing against a three-fold cross-validation using ad hoc results from 2007. For runs using the RequestText query reduction filter, the boost was set to 1.5, and for other runs it was set to 1.8.

2.2.4 Merged Results

Combining results from high performing runs has proven to be a successful technique for this task. Our merged runs use a weighted CombSum method for generating the merged set, where the sum of the inverted ranks is used to calculate a ranking score for each document. The weights are based on estimated F1@R scores of all contributing runs using strategies that could be trained against 2007 topics. If that is not possible, the merge weights are arbitrarily assigned.

2.2.5 Estimating K

We found during training using 2007 ad hoc topics, that setting K to B, the number of documents found by the Boolean reference run, generated estimated F1@K scores that exceeded any other of the formulations that we devised. As a consequence, we set K to B for each of our submitted runs, and, estimating that a third of all relevant documents are highly relevant, set K_h to $B/3$.

2.3 Submissions

- **Reference run (IowaSL08Ref):** This run is a requested standard run. ***Query Source***: Fields – Request Text.
- **Query Expansion (IowaSL0804):** ***Query Source***: Fields – Request Text, Final Query, Defendant's Proposal, Plaintiff's Rejoinder. ***Preprocessing***: Pseudo-relevance feedback: 5 terms from 3 documents.
- **Query Expansion + Boost (IowaSL0804b):** ***Query Source***: Fields – Request Text, Final Query, Defendant's Proposal, Plaintiff's Rejoinder. ***Preprocessing***: See **IowaSL0804**. ***Additional Postprocessing***: Okapi-BM25 scores for documents also found by the Boolean Reference run are multiplied by a "boost" factor of 1.8, and the set is then reranked.
- **QE + Request Text Filtering (IowaSL0805):** ***Query Source***: Fields – Request Text, Final Query, Defendant's Proposal, Plaintiff's Rejoinder. ***Preprocessing***: Request Text field is automatically filtered to remove excess verbiage. Query Expansion handled as in **IowaSL0804**.
- **QE + Request Text Filtering + boost (IowaSL0805b):** ***Query Source***: Same as **IowaSL0805**. ***Preprocessing***: Same as **IowaSL0805**. ***Additional Postprocessing***: Okapi-BM25 scores for documents also found by the Boolean Reference run are multiplied by a "boost" factor of 1.5, and the set is then reranked.
- **QE + AllBooleans+boost (IowaSL0808b):** ***Query Source***: Fields – Request Text, Final Query, Defendant's Proposal, Plaintiff's Rejoinder, Defendant2, Plaintiff2, Defendant3, Consensus1. ***Preprocessing***: Same as **IowaSL0804**. ***Postprocessing***: Same as **IowaSL0804b**.
- **Merge2 (IowaSL08m2):** This run is generated by combining the results from **IowaSL0804b** and **IowaSL0805b**. The results are be fused using a weighted CombSum (combine sum) where the weight assigned to each run is based on the estimated $F_1@R$ measure for the contributing run's strategy when applied to the 2007 topics.
- **Merge3 (IowaSL08m3):** This run is generated by combining the results from **IowaSL0804b**, **IowaSL0805b**, and **IowaSL0808b**. The merging strategy is a weighted CombSum (combine sum), applying arbitrary weights of 2, 1, and 3 respectively, such that the queries with the most information are given the most weight.

2.4 Results

The results from TREC for the reference run "boost" proved largely successful for the F1-based measures. Runs incorporating the boost improved performance over the baseline non-boosted runs on average of 1.1% (est_F1@K) and 5.1% (est_F1@R) for relevant documents, and 13.7% (est_F1@K) and 29.4% (est_F1@R) for highly relevant documents. Incorporating the Boolean query negotiation history resulted in a decline in performance for both relevant (-5.9%) and highly relevant (-2.3%) documents in the est_F1@R measure. Our two-run merge failed to improve on the average of its contributing runs in est_F1@R (-1.0%) for relevant documents, while improving, slightly, for the highly relevant documents (0.8%). Our three-run merge also failed to improve on the average of its contributing runs in est_F1@R for both relevant (-1.2%) and highly relevant documents (-2.8%). Finally, we note that our runs using the filtered Request Text matched the highest estimated recall@100000 for relevant documents, and our three-run merge did likewise for highly relevant documents.

3 Relevance Feedback Task

The design of our system for the TREC Legal relevance feedback task is much different than the system used for the TREC Relevance Feedback Track. Inspired by the “relstrings” included in the 2007 ad hoc task results, our system finds relevant documents amongst the unjudged documents in runs previously submitted to TREC, using the position of the judged documents in those runs as the basis for developing features that can be used by our system.

3.1 Feedback Processing System

3.1.1 Inputs

The input to our system consists of TREC Legal ad hoc track submissions and the existing qrels. We use this information to construct a database that allows us to find for any submitted document for any topic:

- Which submitted runs included the document
- The ranking of the document in each of the submitted runs
- The relevance judgment, if it exists, for that document.

Using this database, we can rapidly implement variants of this system that use all existing runs or just a subset of them. Example subsets might include all of the runs from a single TREC Legal participant, or the top *n* runs of all participants ranked using a specific metric, or a mixture of the best and worse runs for the metric. Because our research group did not participate in the 2006 ad hoc task, we used results from our training conducted against 2006 topics as unofficial contributing sets.

3.1.2 Split Pools

Once the set of contributing runs have been selected, two pools of documents are identified for each topic. One pool consists of the judged documents from the contributing runs, the other pool is build from the unjudged documents. The judged documents are used to train our system, while the unjudged documents are then ranked for output. For the relevance feedback task, all the pool of unjudged documents for topics based on the 2006 ad hoc task are insufficiently large enough to generate a complete 101,000 document output. In those cases, the entire unjudged document pool is included in the output. For the purposes of creating the split pools, “gray” documents are treated as judged.

3.1.3 Features

For each document, a set of features is calculated. The features are based on the position of the document over the contributing runs, or on the relevance judgments for document in the proximity of the document. These features are calculated as means across the runs, or as weighted means favoring those runs with better performance. For the purposes of calculating features, “gray” documents are treated as unjudged.

3.1.4 The Classifier

The classifier is built using the WEKA API. [4] A brief exploration of the available classifiers suggested that the WEKA SMO implementation provides reasonably accurate predictions while able to build classifiers on our training data within seconds. The classifier uses Platt’s sequential minimal optimizing algorithm to train a support vector machine, in this case employing a polynomial kernel for the SVM. [5]

3.1.5 Calculating Output

The features calculated for the pool of judged documents are used to train a model for a particular topic. The model is then applied to each of the documents in the unjudged pool. A logistic regression is fit to the outputs of the SVM to generate the relevance probability estimates for the document. These estimates are then used as the ranking score when compiling the unjudged documents for submission to TREC.

3.2 Experiments

Inspired by the “relstrings,” we initially limited our pools to the top 100 ranked documents from contributing runs. In order to evaluate our system, we use only judged documents, though unjudged documents are used for calculating document features. For each topic, our system iterates through the judged documents. The current document is switched to “unjudged” in our database, the set of features are calculated, the classifier retrained, and the new model is applied to the temporarily “unjudged” document. The relevance probability estimate is retained, the document is restored to its original judgment in the database, and the system moves to the next judged document. After all documents have been processed the system uses the relevance probability estimates to rank the judged documents.

We compared our results against two baseline systems. The first baseline is a ranking of the judged documents created using the combined sum (CombSum) of their inverted rankings from all contributing runs. Our second baseline is derived from a weighted CombSum, similar to the first baseline, except that the inverted rankings are weighted by a run’s Precision@100 score for a topic. For each of our experiments, we compared our system’s precision and recall at increments of five documents up to position 25 in the ranking.

We tested fifteen different sets of contributing runs using this design. The most successful set used all available contributing runs, which garnered a significant improvement over both baselines for both measures, at all depths. A run using the top and bottom four runs in Precision@100 resulted in significant improvements at 70% of our depths over both measures, including all depths against the simple CombSum baseline. Utilizing only the eight runs with the best performance in precision or recall failed to produce significant improvements against the baselines.

We constructed ten sets of contributing runs solely from specific participating research groups. The performance of our system using these sets varied widely, from significant improvements at 80% of our depths over both measures (Iowa-Srinivasan and CMU) to no improvements and a significant decline at depth 25 in Recall against the weighted CombSum (OpenText).

3.3 TREC 2008 Submissions

For the 2008 TREC Legal Relevance Feedback track, we hoped to assess the scalability of our system. Using the pooling system described above, we created the following runs.

- **MergeAllCombMNZ (IowaSL08RF1B):** A baseline for runs merging the unjudged documents from all participant runs. ***Merge Source***: Each submitted run for each topic is used. ***Merge Processing***: Each document is given a score that reflects a summation of its ranking across each of the runs. This score is then multiplied by the number of runs it appears in. This final score is used to rank the documents in the pool, with the top 101,000 (when the pool is large enough) submitted.
- **MergeAllClassifier (IowaSL08RF1A):** ***Merge Source***: Same as **IowaSL08RF1B.** ***Merge Processing***: The run is compiled as described in section 3.1.
- **MergeIACombMNZ (IowaSL08RF2B):** A baseline for runs merging the unjudged documents from ad hoc retrieval runs generated by our research group. ***Merge Source***: The seven runs submitted by the IowaS group for the TREC Legal 2007 Ad Hoc task will be used for the subset of 2007 topics. Results generated post-TREC 2007 by applying 2007 strategies against the 2006 topics will be used as sources for the 2006 topics. ***Merge Processing***: This will be conducted as in **IowaSL08RF1B**.
- **MergeIAClassifier (IowaSL08RF2A):** ***Merge Source***: Same as **IowaSL08RF2B**. ***Merge Processing***: The run is compiled as described in section 3.1.
- **MergeMixCombSum (IowaSL08RF3B):** This run uses provides a baseline for runs generated using a mix of top and bottom performing runs. ***Merge Source***: For 2006 topics, the four runs with leading performance on R-Prec across all topics were selected, and the four runs with lowest performance were selected. A similar technique was used to identify eight contributing runs for the 2007 topics, using the Est_Recall@B measure. ***Merge Processing***: This will be conducted as in **IowaSL08RF1B**.
- **MergeMixClassifier (IowaSL08RF3A):** ***Merge Source***: Same as **IowaSL08RF3B**. ***Merge Processing***: This will be conducted as in **IowaSL08RF1A**.
- **ClassifierCrossover (IowaSL08RF2C):** The classifier models built using all participating runs are applied to a subset of the pool of unjudged documents, in this case, those from **IowaSL08RF2B**. ***Merge Source***: Unjudged documents are from the same pool as **IowaSL08RF2B**. ***Merge Processing***: This will be conducted as in **IowaSL08RF1A**, though the classifiers are trained from the judged documents from all participating runs instead of the limited collection of runs providing the unjudged documents.
- **ClassifierCrossover (IowaSL08RFTr):** Our last submitted run is constructed using more traditional techniques. This run modifies our pseudo-relevance feedback run submitted to the ad hoc task, **IowaSL0804b.** ***Query Source***: Fields – Request Text, Final Query, Defendant's Proposal, Plaintiff's Rejoinder. ***Preprocessing***: Pseudo-relevance feedback: 10 terms from the 12 top ranking relevant documents. ***Additional Postprocessing***: Okapi-BM25 scores for documents also found by the Boolean Reference run are multiplied by a "boost" factor of 1.8, and the set is then reranked.

3.3.1 Results

In runs constructed using the classifier output to rank unjudged documents, using all available contributing runs provided our best result for finding relevant documents. Using a classifier built on a mix of the best and worst runs, however, proved more adept at finding highly relevant documents. This mix of runs also provided the best basis for our baseline runs, for both relevant and highly relevant documents. The classifier built from all contributing runs was better than the one built using only runs from our research group when applied to the unjudged documents from our own runs. This result occurs for both relevant (7.7%) and highly relevant (68.5%) documents, using est_F1@R. Interestingly, our baseline runs merging the unjudged documents from all contributing runs matched the highest estimated recall@100000 for both relevant and highly relevant documents.

3.3.2 Baseline vs Feedback

For all three of our run selection configurations, our baseline system outperformed the system that was trained using relevance feedback data. This average difference was far greater for highly relevant documents (47.8%) than it was for merely relevant documents (19.8%) when using est_F1@R.

4 Conclusion

The use of the Reference Run boost appears to be a useful strategy for the ad hoc task. Our strategy to train classifiers for the purpose of ranking unjudged documents for the relevance feedback task appears to have been unsuccessful at the larger scale of full TREC Legal submissions. Our success using smaller pools of documents suggests that value in this strategy can be found in smaller scales, possibly in an iterative setting, where the user is presented with new documents to evaluate after providing feedback. Further research in this area could also entail alternative means of interpreting classifier output for ranking and implementation of other learning methods.

RELEVANCE FEEDBACK TRACK

1. Introduction

The Relevance Feedback Track at TREC 2008 is a new task designed to explore the different factors in relevance feedback (RF). Specifically, the aim was to compare both statistical and NLP techniques in RF. All participating groups received the same relevance judgments. These came in variants to establish a good baseline: no documents, 1 relevant document, 3 relevant documents, 3 relevant and 3 nonrelevant documents, 10 documents, and finally up to 800 documents.

This year our work in Relevance Feedback Track focused on the establishment of text retrieval and relevance feedback system that exploits query expansion techniques. The text retrieval module of the system used out-of-the-box Lucene indexing system with modified Okapi re-ranking [6]. Multiple pre- and post-retrieval query expansion techniques were used. The query was represented as a set of phrases, quoted text, individual terms, and terms in certain proximity of each other; combination of these techniques produced the best retrieval results. Furthermore, the provided relevance feedback judgments along with pseudo-feedback were used to extract terms for further expansion of the query. The terms were extracted via a statistical examination of the most relevant terms in the feedback documents. Multiple term discount techniques, such as linear, exponential and query length-sensitive discounts were tested. A systematic evaluation of all of these and other techniques is described in our paper.

2. Document Retrieval

2.1. Indexing

The GOV2 data comes in a regular file structure of 272 segments each containing 100 files. Each file contains information on about 950 documents. This segmented structure was utilized during the indexing process by first creating indexes for each of the 272 segments and then merging them. The data is in XML format, with Document ID, Document HTTP Header, and Document Contents for each document. Most of the documents are HTML files, so an HTML parser was needed to extract the actual text from the document. The regularity of HTML allowed us to extract additional information about the documents, such as title, keywords, and anchors to other documents. The anchor information was recorded in a separate file for later use.

Several parameters had to be set during the indexing, so as to improve the retrieval performance of the index. The length normalization was set to "Sweet Spot Similarity", available in Lucene's contrib. package [6]. In it, the length normalization (lengthNorm) is held constant for all document lengths in the [*min*, *max*] range. Below the *min* and above the *max*, the lengthNorm drops off in a square root function. We used *min* = 250 and *max* = 10000 since these values represent the range of average document lengths [7]. These parameters were just heuristic guesses, and could be improved in later stages when the data is further analyzed. Thus only the documents with unusually long or short lengths are "punished". Only the contents field was processed with Sweet Spot Similarity, since it was the only field with a wide range of lengths. The other fields, such as title and URL, had min = max = 1 and steepness of 0.5, degenerating the above equation to 1/sqrt(L).

Furthermore, we used Lucene's StandardAnalyzer, a tool to pre-process the text before it gets inserted into the index, with an expanded stoplist from SMART system mentioned earlier. StandardAnalyzer performs a series of "cleaning" functions, such as removing "'s" in words and periods in acronyms, and converts everything to lower case.

The content field was indexed along with the information about the word offsets. This way we are able to search the index using phrase quotes and proximity measures. The URL and document IDs, on the other hand were stored "as is".

2.2. Training Data

The training data was assembled so as to be the closest approximation of the testing query set. First, the odd Million Query Track queries with no less than 40 judgments were collected from the 2007 query set. Then the odd Terabyte Track queries were collected from the three previous years such that each query had no fewer than 100 judgments. The topics were combined in the same proportion as the testing set (1 TB query to 3 MQ queries).

2.3. Query Expansion

The first task was to optimize our retrieval system. Much of the results' quality depends on the queries. The simplest strategy is just to ask the index to return all documents that have the terms in the query. This would be a simple Boolean query. There are many ways of changing a query to make sure we get what we want. Intuitively, it is better to retrieve a document with all of the terms in a single phrase, then them far apart in the text. Thus, first option is to quote the query or make sure the words are found within a certain number of terms. For this purpose a special function in Lucene can be used: spanNear. It allows the query to specify the number of other terms allowed between the query terms in a text. We set the number of terms allowed between documents to four heuristically.

Another way to modify the query is to find its phrases. Phrases can be found by splitting the query at special characters or stop words (such as "and" and "or"). By using phrases we can make sure we zero in on concepts by grouping words that are related to each other. For example, "printable maps of nc counties" can be separated into "printable maps" and "nc counties".

Yet another way to improve a query would be to search multiple fields. Each document in our index has a *contents* field, which has the terms extracted from the body of the document, but also *title*, *keywords* and *description*. It may be the case that keywords and title of the document may have terms in them that are more representative of the document's main theme than most words in the text body.

We also may want to apply different weights to different fields in the query. Lucene lets us to do so by allowing assignment of a weight (or a “boost”) to a term or a set of terms at query time.

A final way to improve the query would be to remove the most popular terms in the collection. Besides the widely-used stop words, other terms may be just as useless at identifying a document in a collection. For example, since our collection is of governmental web pages, the term “US” doesn’t distinguish the documents well.

To find out which of these modifications would actually work, each expansion technique was tested individually. To keep this exploration orderly, all but one dimension were held constant while finding the optimum value for the active dimension. Our findings are as follows (see tables in Results section):

- Querying multiple document fields instead of just the *contents* has proven ineffective. Probably because many documents either did not provide this information or the data was not as high quality as was hoped.
- Combining both quotes and phrases produced the best results. By weighting (emphasizing importance of) the quoted segments we made sure that they are favored over phrases and single words.
- Removing the collection’s most popular terms from queries produced a mixed result - some queries were improved while others weren’t. It was decided that the overall effect, even though insignificant, was negative, and so the technique was discarded.

2.4. Post-processing

Another way to improve our retrieval system was to use a different document ranking algorithm. Okapi is a well-established and highly effective document ranking algorithm which we used to re-rank the results received from Lucene. The results were indeed improved dramatically. Assuming we are looking for a ranked list of X documents, we first retrieve using the Lucene strategy more than X documents. We then re-rank them using the Okapi formulation applying a cutoff of X documents. The Okapi formula used is shown below where the weight of a term in given document is

$$w_{td} = \frac{qtf \cdot tf \cdot (k_1 + 1)}{k_1\left(1 - B + B(dl/avdl)\right) + tf}$$

where *qtf* is the term tf score in the query and *tf* is the term tf score in the document, *dl* and *avdl* are document length and average document length, respectively. Values for constants $k_1 = 1.2$ and $B = 0.75$ were taken from Okapi at TREC-3 paper [8]. The formula attempts to both discount long documents and promote terms that appear both in the query and the document (thus have high *qtf* and *tf* scores).

3. Relevance Feedback

3.1. Training Data

Since only relevant documents were used in our relevance feedback calculations, a training set was constructed separating the relevance judgments into training and relevance feedback sets. Only the Terabyte Track queries were used, since the qrels files for it was easier to extract the relevance judgments from the standard TREC format than from the custom Million Query Track prels format. The relevance feedback set contained 6 relevant documents for each query.

3.2. RF Term Weighting Schemes

The provided relevance feedback documents were used to extract the most popular (assuming important too) terms in them using term TFIDF weights. These terms were then appended to the queries using several discounting techniques. We tried appending from 1 up to 40 top terms to the query, and experiments showed that only a small number of top terms benefit the query. Only five terms were chosen for the runs submitted.

The querying process was modified in several ways in attempts to improve retrieval performance. The new terms which come from the relevance feedback documents need to be "downgraded" so that the original query got more importance. It was done in a linear fashion (all terms got a "boost" of 0.4 whereas the query got a "boost" of 5), and in a exponential fashion (where "boost" was the largest for the top term and decreased exponentially for the rest). Since longer queries may be more specific, we took more liberty with the RF terms, and discounted the RF terms for the longer queries less than for shorter ones (where the effect of RF terms may be more drastic).

3.3. Pseudo-feedback

Finally, we modified the RF term retrieval process by not only looking at the documents provided as relevance feedback but also at the top documents originally retrieved. These top documents are probably highly relevant. By using them we make sure the RF terms don't lead us too far astray from the original topic. Experimental data shows that the use of pseudo-feedback relieves the effect of bad RF terms and makes the results more robust.

3.4. Decoupling Retrieval and Re-ranking

The ranking process was also modified. The process was divided into first collecting the documents using a Lucene search and then re-ranking them with Okapi algorithm. These two stages could use different forms of the query. The best way was determined to be doing the retrieval with the modified query and re-ranking with the original one (without RF terms).

Method	**MQ MAP**	**TB MAP**
Terms AND-ed, no quotations	0.115	0.159
Quoting phrases	0.151	0.129
Term proximity of 4 terms	0.039	0.001
Term proximity of 8 terms	0.044	0.036
Quoting query, phrase prox. of 4	0.021	0.059
Quoting phrases, AND-ing phrases	**0.151**	**0.184**

Table 1. Varying quoting and proximity

Method	**MQ MAP**	**TB MAP**
Quoting phrases, AND-ing phrases	0.151	0.184
Same with Okapi @ 1000/1500	**0.266**	**0.232**
Same with Okapi @ 1000/2000	0.249	0.236

Table 2. Introducing Okapi re-ranking at various amounts of top documents

4. Results

The results of our training runs (using data described in 3.1) are shown in tables above. Table 1 shows the system's performance for various query expansion methods. There are two MAP scores, one for each track set, since different evaluators had to be used for each. Note the decreased performance when using term proximity. These runs also took much longer to execute than any others. The slowing of the system as well as drastic drop in MAP scores prompted us to drop the proximity expansion technique. On the other hand, phrase extraction combined with quoting of the original query proved to be the best expansion technique. It was used for the rest of the training.

The built-in Lucene document scoring system does not have an option for using Okapi. Thus a modified Okapi re-ranker was implemented. As seen in Table 2, it improves performance dramatically. Once the Lucene searcher returned the results, these were passed on to Okapi. To get 1000 results back, we made Okapi re-rank the top 1500 (or 2000) results retrieved by Lucene and took the top 1000 from the new ranking.

Finally, various relevance feedback query expansion techniques were tested. Each technique was tested for several numbers of added terms. These were ranged between 0 (no expansion terms added) to 40. Table 3 shows the performance of the methods. Note that the measure here is MAP at 1000. Recall that only Terabyte Track queries were used for this part. The last method - a combination of exponential query length sensitive term discount, pseudo-feedback, and Okapi re-ranking - was used for the submitted runs.

# RF terms	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
0	**0.222**	**0.222**	0.222	**0.222**	0.222	0.222	**0.222**	**0.222**	0.222
1	0.214	0.219	0.223	0.222	0.223	**0.234**	0.217	0.195	0.231
5	0.205	0.215	**0.229**	0.213	**0.226**	0.228	0.211	0.200	**0.232**
10	0.192	0.199	0.221	0.204	0.225	0.225	0.209	0.201	0.229
15	0.177	0.185	0.218	0.198	0.222	0.222	0.209	0.198	0.229
25	0.147	0.155	0.209	0.179	0.219	0.220	0.202	0.194	0.228
40	0.121	0.125	0.196	0.163	0.216	0.215	0.193	0.187	0.224

Table 3. Augmenting queries with top N popular terms from the relevance feedback documents

Table 3 Methods:

[1] Searching and reranking with expanded query
[2] Same but weighting new terms at 0.4
[3] Searching with expanded query (w/ 0.4), reranking with original
[4] Searching with original query, reranking with expanded (w/ 0.4)
[5] Searching with expanded query (w/ exp discount), reranking with original
[6] Searching with expanded query (w/ exp length 0.5 discount), reranking with original
[7] Searching with expanded query (w/ exp length 2 discount), reranking with original
[8] Searching with expanded query (w/ exp length 0.5 discount), reranking using Lucene and Okapi
[9] [6] + terms from top N=10 retrieved documents

It should be mentioned that the index used for testing was flawed. It was discovered at the time of submission that the index contained duplicates. The corpus was re-indexed, and new set of results was achieved. Table 4 shows the nine methods from Table 3 run with five RF terms added. It now seems that method 2 would have been the best if judging by MAP at 1000, which is the first measure in the table.

Measure	**[1]**	**[2]**	**[3]**	**[4]**	**[5]**	**[6]**	**[7]**	**[8]**	**[9]**
MAP	0.2241	**0.2327**	0.2064	0.2243	0.1995	0.2029	0.1847	0.1592	0.2020
R-prec	0.2871	0.2969	0.2736	0.2863	0.2735	0.2751	0.2537	0.2210	0.2683
Bpref	0.2788	0.2896	0.2462	0.2760	0.2435	0.2483	0.2270	0.2416	0.2416
P@5	0.6301	0.6329	0.5918	0.6110	0.6082	0.6000	0.5863	0.6082	0.5863
P@10	0.6164	0.6219	0.5479	0.6110	0.5589	0.5616	0.5548	0.5274	0.5466
P@100	0.3347	0.2407	0.3236	0.3316	0.3249	0.3267	0.3056	0.2533	0.3195
P@1000	0.0647	0.0688	0.0691	0.0696	0.0623	0.0646	0.0556	0.0571	0.0671

Table 4. Nine methods from Table 3 rerun on a new index with 5 RF terms added

Figures 1 a through b show the results of the submitted (testing) runs as follows: A – no relevance information provided, B – one, C – three, D – ten, E – 40 to 800 relevance feedback documents provided. The performance improves for top 10 results, but is more erratic for the whole result set. Also note that the performance actually drops for top 10 documents (run B) before it improves (run E).

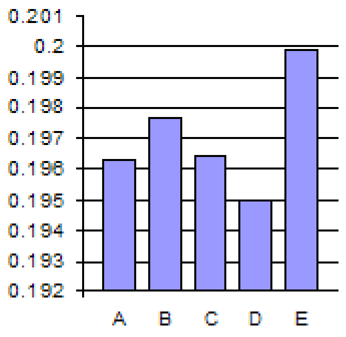

Figure 1a. statMAP

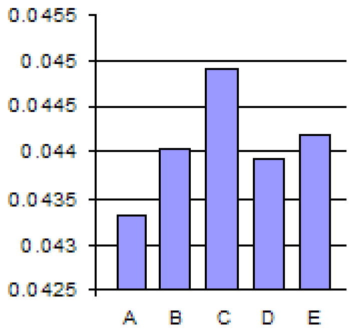

Figure 1b. eMAP

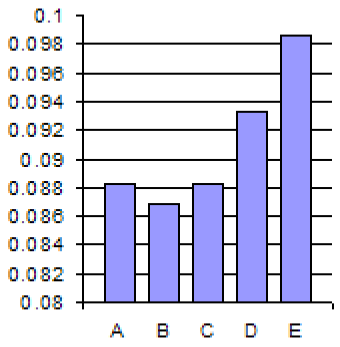

Figure 1c. Top 10 MAP

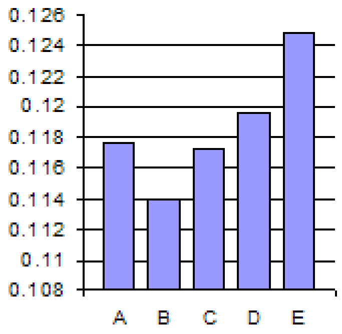

Figure 1d. Top 10 R-Prec

5. Conclusion

Although the relevance feedback information improves performance, only top 10 measures are improved consistently. It will be our task to make sure the measures for the whole result set (in our case, MAP@1000) can also be consistently improved. Using small amount of feedback (as in runs B and C) is a challenge, since it seems to hurt performance (possibly because the information is not reliable). Our system must be robust enough so that the inclusion of limited amounts of feedback information does not result in topic drift.

WORKS CITED:

[1] Welcome to Lucene! [Online] http://lucene.apache.org/.

[2] Fellbaum, C, [ed.]. Wordnet: An Electronic Lexical Database. Cambridge : MIT Press, 1998.

[3] Salton, G. (ed) (1971), *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice-Hall, Englewood Cliffs, NJ.

[4] Witten, I. H. and Frank, E. *Data Mining: Practical machine learning tools and techniques.* 2nd Edition. San Francisco : Morgan Kaufmann, 2005.

[5] Platt, J. Machines using Sequential Minimal Optimization. [ed.] B Schoelkopf, C Burges and A Smola. 1998.

[6] Amitay, E., Carmel, D., and Cohen, D. (2007). Lucene and Juru at Trec 2007: 1 Million Queries Track. in Proceedings of the 16th Text Retrieval Conference. NIST, 2007.

[7] McCurley, K. S. (2008). Observations about the GOV2 TREC data set. <http://www.mccurley.org/trec/>

[8] Robertson, S E, et al. 1995. Okapi at TREC-3. Proceedings of TREC-3.