Real-time Query Expansion in Relevance Models

Victor Lavrenko and James Allan
Center for Intellignemt Information Retrieval
Department of Computer Science
140 Governor's Drive
University of Massachusetts
Amherst, MA 01003-4610, USA
{lavrenko,allan}@cs.umass.edu

ABSTRACT

Automatic query expansion is well known as a technique that improves query effectiveness on average. Unfortunately, it is usually very slow, increasing the time to process a query by 20 times or more. In this study, we use relevance models to show how the process can be made almost as fast as running a non-expanded query.

1. INTRODUCTION

The information retrieval research community has widely adopted various forms of pseudo-relevance feedback since it was shown to be successful (on average) in TREC-3 [4, 9, 8]. By and large, these techniques work using two stages of retrieval. In the first stage, the query is used to create a normal ranked list of documents likely to be relevant. The top N documents from that list (with N varying depending on the particular technique) are then mined for words or phrases that occur frequently. Those new features are added to the query and the revised query is re-run. Because these techniques results in an expanded query, pseudo-relevance feedback is a type of automatic query expansion (AQE).

On average, the revised query is better. When the initial query was reasonably successful, the revised query may be substantially better at finding relevant documents. The caveat is that when the initial query did poorly, the feedback process could easily be misdirected and result in even worse results. A good deal of research has been done to understand when it makes sense to expand the query [3, 2], since empirically a system improves substantially if it always makes the right thoice. Unfortunately, predicting the value of expansion is unreliable at this time.

Nonetheless, automatic query expansion is very popular and is used by almost all high-performing research systems. Some examples of approaches are:

- Rocchio is used primarily in vector space IR systems. Here, the vectors for the top ranked documents are added to the query vector:

  $$Q' = \alpha Q + \beta \frac{1}{N} R_i - \gamma \frac{1}{M} L_i$$

  where R_i represents one of the N top-ranked "relevant" documents and L_i represents one of the M "not relevant" documents taken from very far down the ranked list (usually). The coefficients control the weight of the different components. Often $\alpha = \beta = 1$ and $\gamma = 0$ (i.e., there is no non-relevant automatic feedback). The set of terms added to the query is often limited to a small number. This approach and variations on it are very effective.

- LCA (Local Context Analysis) [10] is a modification of the strict Rocchio approach. It examines top-ranking *passages* of text and computes a weight for every word that occurs in those passages. The weight is a complex formula used that gives credit to terms occurring with multiple query words and reduces the weight for terms that occur too often in the corpus. After all terms have been weighted, the top 50-75 are added into the query and it is re-run. LCA was implemented in the Inquery system.

- Relevance models [5] is a version of Rocchio approach developed for the language-modeling framework. For every word in the vocabulary it computes the probability that this word would appear during random sampling from the same distribution that produced the query. Operationally, the main difference from Rocchio is that top-ranked documents are *weighted* such that documents further down the list have smaller and smaller influence on word probabilities.

Automatic query expansion of one form or another is widely used in research systems, but comes one major downside: it is slow. Because all approaches require two rounds of retrieval, they take at a minimum twice the time of direct retrieval. Furthermore, since document analysis takes some time and since the revised queries are much larger than the original queries, AQE runs are typically vastly slower. Anecdotal evidence suggests that queries may run 20 or more times slower for some systems. This slowdown prevents the adoption of AQE for real retrieval systems, but also puts a

Times				
	AP	FT	LA	WSJ
# qrys.	100	150	100	200
LM	0.2165	0.2329	0.2458	0.2511
time[s]	50.6	24.4	7.6	70.2
RM	0.2740	0.2525	0.2719	0.2978
time	1446.16	1639.22	611.43	1543.32

Table 1: Comparison of time to run queries using a language modeling technique that does not include AQE and the relevance model (RM).

serious crimp in research: only 5% as many experiments fit into a particular window of time.

This study shows how automatic query expansion can be done quickly enough that it is only marginally slower than the original query. The work is done using the type of AQE provided by the relevance model, but is applicable to many other AQE implementations.

We start in Section 2 by demonstrating empirically what the slowdown is for relevance modeling. In Section 3 We then describe a range of techniques that are commonly used to improve the speed of AQE, but that generally result in reduced effectiveness. Section 4 describes manipulation of the relevance model definition that shows how some preprocessing during index time can reduce the work needed at query time. In Section 5 we present the results of a set of experiments that measure the time savings. We discuss some aspects of the results in Section 6, with a focus on the applicability of this work to other AQE approaches, the expense of the pre-processing step, and some additional insights that the manipulation of Section 4 reveal.

2. SPEED OF QUERY EXPANSION

We start by demonstrating the impact of the multiple phases of query expansion. These numbers are for an implementation of relevance models similar to though independent of that available in the Lemur toolkit and its Indri retrieval system[1].

Table 1 shows the impact of incorporating relevance models on four different collections. For each collection, we run from 100 to 200 queries and show the total time (in seconds) it takes to process that set. The first row represents an unexpanded run and is quite fast—e.g., the AP queries take roughly a half second apiece on average. The second row shows the impact of relevance models: the AP queries now take about 15 seconds apiece, a factor of almost 30. The other collections show similar or worse results, with the LA collection being the worst at nearly 90 times the time.

We have not done similar comparisons for other AQE techniques, though we know anecdotally that the results are comparable.

3. SPEEDING EXPANSION UP

Most methods for speeding up AQE limit the amount of processing and/or expansion. For example, fewer documents might be considered when looking for candidate expansion terms, reducing the time to parse and process top ranking documents and their terms. Alternatively, a smaller number of additional terms might be added to decrease the time to run the second query. In general, those approaches reduce processing time to the detriment of effectiveness, because optimal results are obtained with more documents and/or more expansion terms.

Times				
	AP	FT	LA	WSJ
# qrys.	100	150	100	200
LM	0.2165	0.2329	0.2458	0.2511
time[s]	50.6	24.4	7.6	70.2
RM-10	0.2280	0.2079	0.2208	0.2466
time	234.75	336.67	144.07	296.54
RM-20	0.2516	0.2189	0.2315	0.2701
time	275.08	393.56	165.36	337.45
RM-50	0.2631	0.2401	0.2435	0.2819
time	364.6	491.7	208.43	441.95
RM-100	0.2697	0.2426	0.2494	0.2901
time	470.31	631.28	253.19	562.48
RM-200	0.2711	0.2477	0.2548	0.2948
time	677.25	821.97	320.25	771.48
RM-500	0.2728	0.2520	0.2612	0.2972
time	1079.54	1110.99	462.6	1142.03
RM-full	0.2740	0.2525	0.2719	0.2978
time	1446.16	1639.22	611.43	1543.32

Table 2: Impact of reducing the number of documents considered for expansion in relevance models. The row RM-N means that the top-ranking N documents were included in processing.

To illustrate the impact of these approaches, we carried out a series of experiments on the relevance model. Table 2 shows the impact of using from 10 to 500 documents in the expansion process. Note that the increase in time can be cut dramatically (though it is still 4 or more times slower), but at a substantial cost in terms of effectiveness. For example, the AP queries take just over 2 seconds each rather than almost 15, but the effectiveness gain is 5% rather than 27% (relative).

Similar results occur when the number of expansion terms is reduced, though careful parameter tuning can ameliorate that somewhat [6].

4. AN OPPORTUNITY

In this section we will take a very different approach to speeding up relevance-based language models. We are going to closely analyze the ranking formula use by [5], and derive an algebraic re-arrangement that will ultimately lead to a significant increase in computational efficiency of the approach. According to [5], good retrieval performance can be achieved if we rank the documents D in the collection by the cross-entropy of their language model with the estimated relevance model R:

$$H(R||D) = \sum_{w} P(w|R) \log P(w|D) \tag{1}$$

[1] http://www.lemurproject.org/

Here the summation goes over all vocabulary words w, and $P(w|D)$ represents a smoothed language model of document D, which is normally [11] estimated as follows:

$$P(w|D) = \lambda_D \frac{\#(w,D)}{|D|} + (1-\lambda_D)\frac{\#(w,C)}{|C|} \tag{2}$$

where $\#(w,D)$ and $\#(w,C)$ represent the number of times we observe the word in the document D and the entire collection C. λ_D is a document-specific smoothing parameter, which was set to 0.2 in all our experiments. The final component of relevance-based ranking (equation 1) is the relevance model R, which after a slight re-arrangement can be expressed in the following form:

$$P(w|R) \approx P(w|q_1...q_k) = \sum_M P(w|M)P(M|q_1...q_k) \tag{3}$$

Here $q_1...q_k$ represents the original query, the summation goes over a set of all document models M in our collection, and probabilities are estimates using equation (2). The posterior probability $P(M|q_1...q_k)$ is computed using the Bayes' rule under the assumption that $q_1...q_k$ are conditionally independent given M.

Now let us plug the relevance model estimate from equation (3) into the cross-entropy ranking formula:

$$\begin{aligned} H(R||D) &= \sum_w \log P(w|D) \times P(w|R) \\ &= \sum_w \log P(w|D) \times \left(\sum_M P(w|M)P(M|q_1...q_k)\right) \\ &= \sum_M \sum_w [P(w|M)\log P(w|D)] \times P(M|q_1...q_k) \\ &= \sum_M H(M||D) \times P(M|q_1...q_k) \qquad (4) \end{aligned}$$

The first step in our derivation is a direct result of substituting equation (3) into equation (1). The second step involves changing the order of summations $(\sum_w)$ and $(\sum_M)$, which can always be done because the summations are finite. The last step involves noticing that $P(M|q_1...q_k)$ does not depend on w, moving it outside of $(\sum_w)$, and recognizing the inner summation to be the cross-entropy $H(M||D)$.

Far from being a just a curious exercise, the derivation we carried out in equation (4) has an important probabilistic interpretation. The original relevance-based ranking can be thought of as a log-linear similarity (entropy) with a massively expanded query (relevance model). We demonstrated that it is equivalent to computing a set of affinities $H(M||D)$ of document D with every model M in our collection, and then constructing a query-weighted average of these affinities. When expressed in this manner, relevance-based ranking may be immediately recognized as a kernel-based approach, similar to spread-activation methods, self-organizing maps and regularization techniques.

4.1 Retrieval Algorithm

Aside from theoretical elegance, the result we derived in equation (4) has strong implications for the computational expense of relevance-based ranking. We observe that the *affinity* component $H(M||D)$ is independent of the user's query and can be pre-computed and stored at the time when the collection is indexed. These affinities can be thought of as another inverted index, associated with every document M in the collection. Given a new query $q_1...q_k$, we perform retrieval as follows:

1. use the language-modeling approach [7] to retrieve a set of top-ranked documents M from the collection.
2. convert language modeling scores $P(q_1...q_k|M)$ into Bayesian posteriors $P(M|q_1...q_k)$
3. for each of the top-ranked documents M:
   - fetch its affinity list – a vector containing $H(M||D)$ for documents D that are most similar to M
   - merge the affinity list into the final ranked list, weighing the components $H(M||D)$ by $P(M|q_1...q_k)$

The algorithm outlined above is very efficient because it avoids the main pitfall of relevance-based ranking – the need to evaluate a giant "query" potentially consisting of every word in the vocabulary.

Original relevance-based ranking algorithm:

1. use the language-modeling approach [7] to retrieve a set of top-ranked documents M from the collection.
2. convert language modeling scores $P(q_1...q_k|M)$ into Bayesian posteriors $P(M|q_1...q_k)$
3. for each of the top-ranked documents M:
   - fetch the vector of word counts for M and convert it into a document language model $P(w|M)$
   - merge the document language model into the overall relevance model, weighing word probabilities $P(w|M)$ by $P(M|q_1...q_k)$
4. use the estimated relevance model $P(w|R)$ as a giant query in the language modeling approach to come up with the final ranked list

The main computational expense of relevance-based ranking comes from step (4) in the algorithm presented above. Running a query consisting of thousands of words is very inefficient and requires a lot of computational resources.

5. RESULTS

5.1 Datasets and processing

We use four different datasets in our evaluation of adhoc retrieval effectiveness. Table 3 provides detailed information for each dataset. All four datasets contain news releases; the majority of them are print media. The datasets vary in size, time frame, and word statistics. All datasets are homogeneous, i.e. they contain documents from a single source. For each dataset there is an associated set of topics, along with human relevance judgments. The datasets contain *pooled* judgments, i.e. only top-ranked documents from a set of retrieval systems were judged with respect to each topic by annotators at NIST. TREC topics come in the form of queries, containing title, description and narrative portions. We used only the titles, resulting in queries which are 3-4 words in length.

Name	Sources	Years	#Docs	#Terms	dl	cf	Queries	ql
AP	Associated Press	89-90	242,918	315,539	273	210	51-150	4.32
FT	Financial Times	91-94	210,158	443,395	237	112	251-400	2.95
LA	Los Angeles Times	89-90	131,896	326,609	290	117	301-400	2.52
WSJ	Wall Street Journal	87-92	173,252	185,903	265	247	1-200	4.86

Table 3: Information for the corpora used in ad-hoc retrieval experiments. *dl* denotes average document length, *cf* stands for average collection frequency of a word, and *ql* represents average number of words per query.

Times				
	AP	FT	LA	WSJ
# qrys.	100	150	100	200
LM	0.2165	0.2329	0.2458	0.2511
time[s]	50.6	24.4	7.6	70.2
RM-full	0.2740	0.2525	0.2719	0.2978
time	1446.16	1639.22	611.43	1543.32
fRM	0.2671	0.2511	0.2653	0.2951
time	67.99	42.14	15.83	99.89
sims[h]	91.40	70.45	38.55	46.60
fRM-10	0.2500	0.2280	0.2298	0.2655
sims	16.68	13.52	4.86	8.90
fRM-20	0.2594	0.2400	0.2421	0.2817
sims	25.10	21.39	7.70	13.67
fRM-50	0.2649	0.2471	0.2526	0.2907
sims	45.53	39.37	14.29	24.39
fRM-100	—	0.2525	0.2540	0.2943
sims	—	55.08	20.34	32.87

Table 4: Compares the times (in seconds) to run queries without AQE, with RM, and then with fast relevance models. For all fast RM runs, the preprocessing time (in hours) needed to build the similarity matrix is also listed.

Prior to any experiments, each dataset was processed as follows. Both documents and queries were tokenized on whitespace and punctuation characters. Tokens with fewer than two characters were discarded. Tokens were then lowercased and reduced to their root form by applying the Krovetz stemmer used in the InQuery engine [1]. The stemmer combines morphological rules with a large dictionary of special cases and exceptions. After stemming, 418 stop-words from the standard InQuery [1] stop-list were removed. All of the remaining tokens were used for indexing, and no other form of processing was used on either the queries or the documents.

5.2 Timing runs

Table 4 shows the impact on query time for fast RMs in the third row. The query effectiveness is nearly identical to that of full relevance models. The differences are primarily the result of converting very low probability values to zero, but partially the result of rounding differences with the reordering of calculations. More importantly, the time to process the query has dropped precipitously. Where RMs took almost 15 seconds per query on AP, fast RMs require about 2/3 of a second, compared to the 1/2 second for unexpanded queries. The results are comparable on the other collections.

Of course, the downside of the high-speed query processing is the large amount of pre-processing that needs to happen at indexing time. The AP collection required over 90 hours of time to build the document similarity matrix. For a collection that will be indexed rarely and searched often, this is a fine tradeoff. In other cases, the indexing time may be unacceptable.

One way to reduce the amount of time needed is to reduce the number of documents whose similarities are included in the matrix. This is similar in spirit to using fewer documents for the expansion. Table 4 shows that the time to build the matrix can be cut to a large degree by doing that. The drop in effectiveness is less than that seen when fewer documents were used for RM expansion (see Table 2).

6. DISCUSSION

In the previous sections we presented a novel and efficient algorithm for constructing relevance-based rankings. However, we omitted one very important detail in our discussion – the computational expense of constructing a set of affinity lists $H(M||D)$ for every document M in the collection. A naive approach to this problem would involve a quadratic algorithm that iterates over every possible document pair M, D computing $H(M||D)$ along the way. The computational cost of this approach is prohibitive. A more intelligent implementation would take advantage of the inverted indices and operate as follows:

1. for each document M in the collection:
   - select representative words from M forming a query Q_M
   - use Q_M to retrieve a set of top-ranked documents $D_1 \ldots D_k$
   - for each top-ranked D_i, compute and store the affinity $H(M||D)$

This algorithm leverages the following observation: we do not need to know the affinity between every pair of documents in the collection, it is sufficient compute $H(M||D)$ for most similar documents. The hope is that the query Q_M will bring those documents to the top of the ranked list. Computational expense of this algorithm is determined by the size of the query Q_M – the fewer words it contains, the faster we will be able to compute the similarity matrix. However, if we use too few words from each document M, we will likely miss many documents D that are similar to M, degrading the quality of our similarity matrix.

We have conducted a set of experiments analyzing the impact of query size on the running time and final retrieval accuracy of our algorithm. The results are summarized in table 4. We have investigated several methods for selecting words to include in the query Q_M. The approach that provided the best tradeoff between running time and performance is to consider a set of k words with the highest frequencies in M. While this might not strike an IR researcher as a very sophisticated term selection formula, it does seem to bring to the top documents D with the highest $H(M||D)$. We constructed queries consisting of 10, 20, 50 and 100 most frequent words. We also did a run (*fRM*) where an entire document was used as a query, resulting in exact retrieval of documents D with the highest $H(M||D)$.

7. CONCLUSION

We have shown that by expending time during indexing, we can reduce the time needed for automatic query expansion to the point where it provides almost no delay at query time. Although we developed and demonstrated these results using the relevance model approach to expansion, the same ideas should apply readily to many other (though not all) expansion techniques.

The approach we have used depends upon substantial processing of the corpus at index time to build a document-document similarity matrix. We have shown that some simple ideas can reduce the cost of building that matrix, but even those approaches will not scale well to very large collections. We have sketched some ideas of how we could improve the speed using approximations, and are continuing to work in that direction.

We believe that fast relevance models—and more generally, fast query expansion—will be a tremendous boon to the research community. It should allow researchers to carry out more experiments to better understand the technique and its applicability. It will also allow interactive studies to use AQE techniques rather than less effective but more efficient reterieval methods.

8. ACKNOWLEDGMENTS

This work was supported in part by the Center for Intelligent Information Retrieval and in part by the Defense Advanced Research Projects Agency (DARPA). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the sponsor.

9. REFERENCES

[1] J. Allan, M. Connell, W. B. Croft, F. F. Feng, D. Fisher, and X. Li. INQUERY and TREC-9. In *Proceedings of the Ninth Text REtrieval Conference (TREC-9)*, pages 551–562, 2000.

[2] Steve Cronen-Townsend, Yun Zhou, and W. Bruce Croft. A framework for selective query expansion. In *Proceedings of CIKM*, pages 236–237, 2004.

[3] Donna Harman and Chris Buckley. The NRRC reliable information access (RIA) workshop. In *Proceedings of SIGIR*, pages 528–529, 2004.

[4] K. L. Kwok and L. Grunfeld. TREC-3 ad-hoc, routing retrieval and thresholding experiments using PIRCS. In *Proceedings of TREC-3*, 1995. In press.

[5] V. Lavrenko and W. B. Croft. Relevance-based language models. In *Proceedings of the Twenty-Fourth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 120–127, 2001.

[6] Donald Metzler, Trevor Strohman, Yun Zhou, and W.B. Croft. Indri at TREC 2005: Terabyte track. In *Proceedings of TREC 2005*, 2005. Fortcoming.

[7] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of the Twenty-First Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 275–281, 1998.

[8] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proceedings of TREC-3*, 1995. In press.

[9] Paul Thompson and Howard Turtle. TREC-3 ad hoc retrieval and routing experiments using the WIN system. In *Proceedings of TREC-3*, 1995. In press.

[10] Jinxi Xu and W. Bruce Croft. Improving the effectiveness of information retrieval with local con text analysis. *ACM Transactions on Information Systems*, 18(1):79–112, 2000.

[11] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the Twenty-Fourth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 334–342, 2001.