Fast Query Expansion Using Approximations of Relevance Models

Marc-Allen Cartright[1], James Allan[1], Victor Lavrenko[2], and Andrew McGregor[3]

[1]CIIR
Dept. of Computer Science
Univ. of Massachusetts
Amherst, MA 01003
irmarc,allan@cs.umass.edu

[2]School of Informatics
University of Edinburgh
Edinburgh, UK
vlavrenk@inf.ed.ac.uk

[3]Dept. of Computer Science
Univ. of Massachusetts
Amherst, MA 01003
mcgregor@cs.umass.edu

ABSTRACT

Pseudo-relevance feedback (PRF) improves search quality by expanding the query using terms from high-ranking documents from an initial retrieval. Although PRF can often result in large gains in effectiveness, running two queries is time consuming, limiting its applicability. We describe a PRF method that uses corpus pre-processing to achieve query-time speeds that are near those of the original queries. Specifically, Relevance Modeling, a language modeling based PRF method, can be recast to benefit substantially from finding pairwise document relationships in advance. Using the resulting Fast Relevance Model (fastRM), we substantially reduce the online retrieval time and still benefit from expansion. We further explore methods for reducing the preprocessing time and storage requirements of the approach, allowing us to achieve up to a 10% increase in MAP over unexpanded retrieval, while only requiring 1% of the time of standard expansion.

Categories and Subject Descriptors: H.3.3 Information Search and Retrieval: Relevance Feedback, Retrieval Models

General Terms: Algorithms, Performance

Keywords: relevance model, pseudo-relevance feedback, distributed computing

1. INTRODUCTION

Lavrenko and Croft's Relevance Model (RM) [6] is a pseudo-relevance feedback method developed for the language modeling (LM) framework. The standard formulation of this method involves submitting an original query (LM), using the resulting ranked list to perform weighted query expansion, and performing a second round of retrieval (RM). The second query can consist of hundreds of terms, resulting in a slow evaluation over the collection. Table 1 illustrates this problem.

Even for a small collection such as AP89, the original

Collection	# docs	terms	unique	Ratio (RM/LM)	
	(10^3's)	(10^6's)	(10^3's)	MAP	Ret. Time
AP89	84.6	42.1	211.5	1.21	191.37
WSJ	173.2	81.7	243.5	1.28	160.63
Robust05	1033.4	484.2	892.2	1.38	430.88

Table 1: Collection statistics. The last column presents *ratios* of RM to LM effectiveness and speed: RM is more effective but *much* slower than LM

Relevance Model (RM) is nearly 200 times slower than the Language Model (LM), while providing a 20% relative improvement in retrieval. This tradeoff is unacceptable for any realistic setting. In this work, we reformulate RM to perform much of the computation offline, allowing for improved retrieval times. We also investigate techniques for reducing the time and space requirements of the offline computation while avoiding a significant negative impact on retrieval performance. These improvements result in up to a 10% increase in MAP over LM while requiring only 1% of the retrieval time of the original RM. Although our experiments and discussion focus on the RM, the ideas generalize to most forms of PRF and are an important step towards overcoming the inefficiency that prevents their being adopted.

2. FASTRM IN MAPREDUCE

We begin with the derivation made by Lavrenko and Allan [5], essentially requiring the cross-entropy between document M (the model) and document D (the candidate for scoring). Calculating the cross-entropy between two documents lends itself nicely to the MapReduce framework, though some care must be taken to account for the smoothing factors involved in the second probability term. Starting from the $H(M\|D)$ term:

$$H(M||D) = \sum_{t \in M} P(t|M) \log P'(t|D) \qquad (1)$$

An inner product over a set of documents can be performed simply by incrementing a score accumulator using the posting lists of an indexed collection [3, 7]. However, because of smoothing the situation is not so simple. Consider these two example documents:

$$d_1 = \{ \text{ The cat crossed the street } \}$$
$$d_2 = \{ \text{ The dog crossed the river } \}$$

Substituting into Equation 1:

$$\begin{aligned} H(d_1\|d_2) &= P(\text{the}|d_1)\log P'(\text{the}|d_2) \\ &+ P(\text{cat}|d_1)\log P'(\text{cat}|d_2) \\ &+ P(\text{crossed}|d_1)\log P'(\text{crossed}|d_2) \\ &+ P(\text{street}|d_1)\log P'(\text{street}|d_2) \end{aligned} \tag{2}$$

Notice that neither posting list for the terms "cat" nor "street" will have d_2 in them. We will never be able to generate the second nor the fourth additive terms of the sum in Equation 2, yet they are nonzero values, as the second part of each product is a *smoothed* probability, and therefore never zero. We circumvent this issue by producing the background score for every document pair, and then incrementing from that point to produce the final scores. Let us begin with Equation 1 again:

$$H(M||D) = \sum_{t\in M} P(t|M)\log P'(t|D) \tag{3}$$

meaning if the term does not occur in D but does occur in M, we still need the term's contribution to the document pair. Early experiments indicated that Jelinek-Mercer smoothing is superior to Dirichlet smoothing for this method, therefore we expand our smoothed probability as follows:

$$P'(t|D) = \lambda P(t|D) + (1-\lambda)P(t|C) \tag{4}$$

Let $\beta(t)$ represent the background probability of t. This is the quantity $(1-\lambda)P(t|C)$ in Eq. 4:

$$\begin{aligned} \sum_{t\in M} P(t|M)\log P'(t|D) &= \sum_{t\in M} P(t|M)\log(\beta(t)) + \\ &\sum_{t\in M\cap D} P(t|M)\left[\log(P'(t|D)) - \log(\beta(t))\right] \end{aligned}$$

We now focus our attention on the second term to expose the smoothed probability:

$$\begin{aligned} &\sum_{t\in M\cap D} P(t|M)\left[\log(P'(t|D)) - \log(\beta(t))\right] = \\ &\quad\sum_{t\in M\cap D} P(t|M)\log\left[\frac{\lambda P(t|D)+\beta(t)}{\beta(t)}\right] = \\ &\quad\quad\sum_{t\in M\cap D} P(t|M)\log\left[\frac{\lambda P(t|D)}{\beta(t)}+1\right] \end{aligned}$$

Substituting back into Equation 1:

$$\begin{aligned} H(M\|D) = \sum_{t\in M} P(t|M)\log(\beta(t)) + \\ \sum_{t\in M\cap D} P(t|M)\log\left[\frac{\lambda P(t|D)}{\beta(t)}+1\right] \end{aligned} \tag{5}$$

This formulation has two crucial advantages over Equation 1: the second sum is now over the *intersection* of M and D, and all of the terms are raw probabilities. The cross-product of postings will in fact recover all of the terms in the second sum for every M,D pair. At the end of scanning a document, we emit the value of the first sum as a "null term", which now represents the background cross-entropy score of all documents with respect to M. All we need to do is carry the null term downstream to the final sums for all documents. This process only produces an extra $|C|$ increments to shuffle, a negligible increase in volume.

3. EXPERIMENTAL SETUP

We implemented the offline calculation using Hadoop MapReduce v0.20.1, and processing was performed on Yahoo! Inc.'s M45 cluster[1]. For retrieval, we modified a copy of Indri 2.10[2] to support merging the previously calculated scores into the ranked list. We conduct our experiments over the three collections shown in Table 1. We use the built-in Krovetz stemmer and the INQUERY stopword list during indexing. We use topics from the early ad-hoc tracks of TREC[3] as the query set for AP89 and WSJ. Robust05 represents the topics and documents from the TREC Robust 2005 track. We use only the title text of each topic. For all PRF experiments, we set the number of feedback documents ($fbDocs$) to 10, and the number of feedback terms ($fbTerms$) to 100. We report Mean Average Precision (MAP) and in order to gauge retrieval performance between different methods. The LM and RM methods act as the baselines for our experiments: we want LM speed with RM accuracy. We use the paired sample randomization t test as described by Smucker et al. [8] for significance testing. We use 10 million samples for each significance test. Often RM can be improved if the original query's likelihood is interpolated into the score [1]. All experiments here use that variation of RM.

4. MAKING FASTRM FASTER

Although Lavrenko and Allan have shown that fastRM can be slightly slower than LM while providing much of the gain of RM [5], there are two issues that appear as the collection size grows:

1. The time needed to calculate the matrix is reasonable for the AP89 collection, but will grow unacceptably with more documents – even using cloud-based approaches such as MapReduce.
2. The full matrix grows quadratically. The AP89 collection produces a matrix of over 53GB, which is manageable. However we estimate the size of the full matrix for Robust05 to be over 8TB. This growth factor will quickly impact storage costs and scan time at retrieval.

In the rest of this section we present methods for addressing the issues above via approximation of the matrix.

4.1 Measuring approximation

Let A represent a fully calculated matrix of cross-entropy scores, and let $\hat{A}$ be an approximation of A. The approximation methods used here may drop entries from A, or alter values in the cells, however the cells will never change row-rank order. We can use standard ranked list evaluation measures to inform us how $\hat{A}$ impacts retrieval. However, when comparing a A and $\hat{A}$, we would like a more direct

[1]http://research.yahoo.com/node/1884
[2]http://www.lemurproject.org/indri/
[3]http://trec.nist.gov

cols (c)	AP89	WSJ	Robust05
0	13.11	31.92	66.13
100	19.22	37.92	146.92
1000	25.66	44.84	157.76
10000	110.50	138.52	262.60
∞	800.80	2125.29	18131.89
RM	2506.90	5124.07	28494.73

Table 2: Query-processing times for different number of scanned columns (c). Time is in milliseconds. $c = 0$ is the Language Model run. $c = \infty$ is using the entirety of the provided matrix. Collection is AP89.

measure of how much of the original matrix is recovered by the approximation. We desire a method that assigns more importance to documents at higher ranks in a particular row. We would also like a measure that is bounded, since by definition the best performance we can hope for is recovering exactly the elements of the row we specify.

We draw inspiration from the NDCG measure [4] to fulfill this role, calling the resulting measure RowEval. Let A_i be row i in a fully calculated matrix of cross-entropy values. Let $\hat{A}_i$ be the corresponding row i in $\hat{A}$. Some documents do not appear in the approximate row, so if $|A_i|$ represents the number of non-empty entries in row A_i, then $|A_i| \geq |\hat{A}_i|$. NDCG is defined over two lists of the same length, so we need to treat $\hat{A}_i$ as if it has the same number of entries as A_i. To do this we simply inject a 'non-relevant' entry into $\hat{A}_i$ in place of every document not recovered, to create a new list $\hat{A}'_i$. For example, if:

$$\begin{aligned} A_i &= \{1,2,3,4,5,6,7,8,9,10\} \\ \hat{A}_i &= \{1,6,7,8,10\}, \text{ then} \\ \hat{A}'_i &= \{1,\times,\times,\times,\times,6,7,8,\times,10\} \end{aligned} \tag{6}$$

If we define $rel(id)$ to be 1 if document id is not a $\times$ symbol, and 0 otherwise, then RowEval@p is:

$$RowEval@p(\hat{A}_i, A_i) = \left(\sum_{j=1}^{p} \frac{rel(\hat{A}'_i[j])}{\log_2(j+1)}\right)\left(\sum_{j=1}^{p} \frac{rel(A_i[j])}{\log_2(j+1)}\right)^{-1}$$

Simply put, we use the non-relevant entries to make sure the actual entries in $\hat{A}_i$ are assigned the proper gain during computation.

4.2 Reducing Storage Requirements

Previous work [9, 2] has shown that binning retrieval scores can simultaneously increase efficiency and reduce space requirements while not significantly impacting retrieval performance. Binning the values would reduce the total number of cells required for each row scan; Table 2 shows retrieval times for various scan-lengths, indicating a substantial improvement. We study two binning techniques to determine the impact on retrieval performance.

ϵ-based binning. For a given retrieval run, we define the variable ϵ to be the amount two scores must differ by in order to create a new bin. Given two row-wise adjacent entries in a matrix, $A_{i,j}$ and $A_{i,j+1}$, if $|score(A_{i,j}) - score(A_{i,j+1})| < \epsilon$, then $A_{i,j+1}$ is placed in the same bin as $A_{i,j}$ and uses the same score that $A_{i,j}$ is using (which itself may be a surrogate score). Otherwise we create a new bin, using $score(A_{i,j+1})$ as the score for that bin. As $\epsilon \to 0$, # bins $\to |C|$. Inversely, as $\epsilon \to \infty$, # bins $\to 1$.

Stepwise binning. The step function uses two values, the binsize (b) and the number of bins (n). For example, a bin size of 100 with 10 bins means that the first 1000 documents are placed into 10 bins, where the first bin contains the 100 most highly scored documents, the second bin contains the 100 next documents, and so on. For each bin, the highest score in a that bin is used for all of the documents contained in the bin. If the number of documents in the row is greater than the number specified by $b \times n$, all of the remaining documents are placed in the last (i.e. rightmost) bin.

Table 3 shows results for several values of ϵ and stepwise binning. Both methods appear to show marked improvement over the run without binning (leftmost), suggesting that we could reorganize our matrix to only store the values that start a bin, and delete the other scores in each bin. Notice that the stepwise method shows more variability between values, indicating higher sensitivity to parameter changes.

4.3 Reducing Offline Computation Time

We can drop columns without dropping effectiveness [5], which implies that we can save computation time if we can only calculate the pairs we need. In order to achieve this effect, we use the following method. When processing a document, we order all of the unique terms by an impact function $\mathcal{I}$. We set some threshold τ, and only emit the first τ unique terms from each impact-ordered document. We then form a list of document pairs based on the intersections within the remaining term posting lists. Using this list we then generate the exact cross-entropy scores to form the approximate matrix. We set 'tf-idf' as the $\mathcal{I}$ function, while varying τ. The operation behaves like a high-pass filter on the document terms, therefore we refer to this method as Highpass.

Figure 1 shows results for different settings of τ compared to the Language Model (LM) and best fastRM (full) runs for the respective collections. All values have been scaled to reflect the increase from the lowest to the highest value for that particular axis and collection. Therefore a value of 0.5 indicates the actual value to be $low + 0.5(high - low)$. Figure 1 shows that for $\tau = 10$ and $\tau = 20$, the the build times are relatively short, but the relative increase in MAP is substantial. The runs where $\tau = 100$ took longer to construct than the original full calculation were due to how the cross-entropy scores are calculated from the resulting list of document pairs. It demonstrates that although high τ value does increase the build time as expected, it does not bring increased accuracy.

Table 4 provides some insight into why the Highpass method performs well with so few terms projected. At $\tau = 10$, the RowEval@1K is above 0.5 for all 3 collections. However one can easily see the diminishing returns in increasing the value of τ. This suggests that while the RowEval increases as expected, at higher values of τ the documents for a given row are on average of lower quality (i.e. less similar). This explains the phenomenon of the $\tau = 100$ runs actually performing worse than the runs with fewer terms used. This begs the question of how much the performance is tied to the number of terms projected, as opposed to just picking the *right* terms to project.

	WB	ϵ-binned					stepwise binned			
Collection		0.1	0.5	1	1.5	1.75	1-1000	10-100	100-10	1000-1
AP89	0.2159	$0.2171^{\dagger}$	0.2155	$0.2191^{\dagger}$	$0.2246^{\dagger}$	$0.2223^{\dagger}$	0.2159	$0.2216^{\dagger}$	$0.2376^{\dagger}$	$0.2211^{\dagger}$
WSJ	0.2099	$0.2298^{\dagger}$	$0.2278^{\dagger}$	$0.2363^{\dagger}$	$0.2347^{\dagger}$	$0.2312^{\dagger}$	0.2099	$0.2240^{\dagger}$	$0.2392^{\dagger}$	0.2085
Robust05	0.1264	0.1268	0.1252	$0.1324^{\dagger}$	0.1304	0.1273	0.1264	$0.1298^{\dagger}$	$0.1405^{\dagger}$	0.1285

Table 3: Retrieval results (MAP) for the binning techniques. The fastRM run with 1000 retained top docs for each row. The run without binning (WB) is in the leftmost column. † indicates statistical significance of $p < 0.05$ over the WB run.

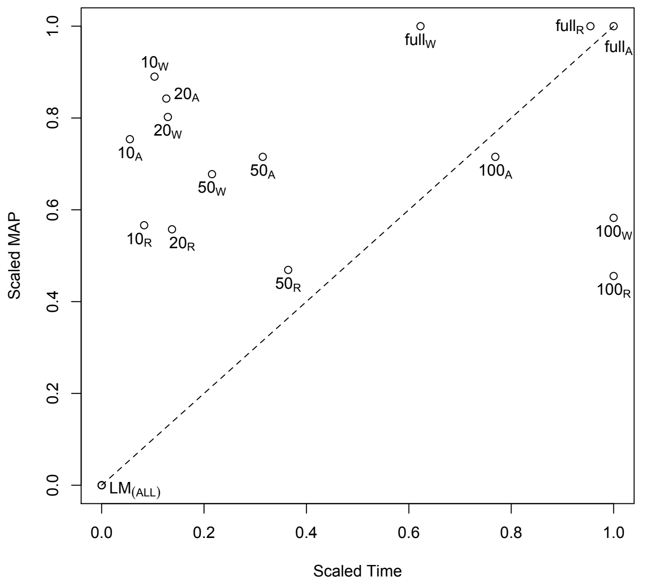

Figure 1: Graph comparing increase in retrieval performance versus time to build the matrix used. 'A','W', and 'R' indicate AP89, WSJ, and Robust05 respectively. Labels are the value of τ.

	Value of τ			
Collection	10	20	50	100
AP89	0.524	0.727	0.921	0.984
WSJ	0.567	0.786	0.948	0.984
Robust05	0.718	0.887	0.977	0.990

Table 4: RowEval@1K of Highpass for the different values of τ, across all collections.

5. CONCLUSIONS AND FUTURE WORK

We have shown that we can reformulate the traditional Relevance Model to allow for much of the computation to occur offline. Additionally, we can reduce the computational requirements of calculating the matrix by accurately predicting the high-quality document pairs and only producing values for those entries. Results indicate that at least one method, the Highpass algorithm, shows considerable promise.

Several avenues emerge to continue this work. We plan to investigate other approximation techniques that can further improve pre-processing time and storage requirements. We would also like to broaden the scope of our work to apply to other PRF methods, with the intention of discovering principles that hold true across all PRF methods.

We believe this work demonstrates the potential of our approach towards improving efficiency of theoretically sound query expansion. Furthermore, this work represents a significant step in bridging the gap between applying statistical PRF in a lab setting versus using it in the real world.

6. ACKNOWLEDGMENTS

This work was supported in part by the Center for Intelligent Information Retrieval, in part by NSF IIS-0910884, and in part by NSF CLUE IIS-0844226. We thank Yahoo! for the use of the M45 cluster. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.

REFERENCES

[1] N. Abdul-Jaleel, J. Allan, W. B. Croft, F. Diaz, L. Larkey, X. Li, M. D. Smucker, , and C. Wade. UMASS at TREC 2004 — novelty and HARD. In *Proceedings of TREC*, Gaithersburg, MD, USA, 2004. NIST.

[2] V. N. Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. In *Proceedings of SIGIR*, pages 372–379, 2006.

[3] T. Elsayed, J. Lin, and D. W. Oard. Pairwise document similarity in large collections with mapreduce. In *Proceedings of ACL/HLT*, pages 265–268, Morristown, NJ, USA, 2008. Association for Computational Linguistics.

[4] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[5] V. Lavrenko and J. Allan. Real-time query expansion in relevance models. IR 473, University of Massachusetts Amherst, 2006.

[6] V. Lavrenko and W. B. Croft. Relevance based language models. In *Proceedings of SIGIR*, pages 120–127, New York, NY, USA, 2001. ACM.

[7] J. Lin. Brute force and indexed approaches to pairwise document similarity comparisons with MapReduce. In *Proceedings of SIGIR*, pages 155–162, New York, NY, USA, 2009. ACM.

[8] M. D. Smucker, J. Allan, and B. Carterette. A comparison of statistical significance tests for information retrieval evaluation. In *Proceedings of CIKM*, pages 623–632, New York, NY, USA, 2007. ACM.

[9] T. Strohman. *Efficient Processing of Complex Features for Information Retrieval.* PhD thesis, University of Massachusetts Amherst, December 2007.