Using Query History to Prune Query Results

Daniel Waegel
Ursinus College
Department of Computer Science
dawaegel@gmail.com

April Kontostathis
Ursinus College
Department of Computer Science
akontostathis@ursinus.edu

Abstract

The most common retrieval systems run queries independently of one another — no data about the queries is retained from query to query. This paper describes an unsupervised learning algorithm that uses information about previous queries to prune new query results. The algorithm has two distinct phases. The first trains on a batch of queries; in doing so, it learns about the collection and the relationship between its documents. It stores this information in a document-by-document matrix. The second phase uses the accumulated knowledge to prune query results. Documents are removed from query results based on their learned relationship to documents at the top of the results list. The algorithm can be fine-tuned to be very aggressive or more conservative in its pruning. This algorithm produces increased relevancy of the results and significantly reduces the size of the results list.

Keywords: Query Pruning, Query History, Vector Space Retrieval

1 Introduction

In information retrieval systems which use traditional methods, the level of precision is very low. Much time and effort has been expended to develop methods to push the most relevant documents to the top of the query results. Many of these methods have met with high levels of success; however, the overall precision of all documents returned by a query is typically still very low. The theoretical optimum is to have each query return only the documents which are truly relevant to it; in practice, even the most advanced systems (such as the Google search engine) return huge percentages of the collection which are irrelevant to the query. We have developed a learning algorithm that can be applied to a collection to reduce the number of irrelevant documents returned.

In the next section we set up the terminology needed to discuss the learning algorithm. In section 3 we discuss related work and why this line of research is unique. In section 4 we present an overview of the algorithm and our preliminary results. In section 5 we offer our concluding comments and present our views on why this technique holds promise for future research.

2 Background

In this section we set up the terminology that will be used in the following sections.

2.1 Overview of Information Retrieval

In information retrieval, data are often categorized with two entities: terms and documents. Documents are the logical units of a collection — for example, a paragraph, a web page, an article, or even a whole book. An index term (or simply term) is a word (or group of words) that is used to represent the content of the documents in the collection.

2.1.1 Traditional Vector Space Retrieval

Traditional vector space retrieval is a method where documents and queries are represented as vectors in t-dimensional space, where t is the number of indexed terms in the collection. Generally, the document vectors are formed when the index for the collection is generated (these vectors form a matrix that is often referred to as the 'term-by-document matrix'), and the query vectors are formed when a search of the collection is initiated. In order to determine the relevance of a document to the query, the similarity between the query vector and each document vector is computed; the cosine similarity metric is often used [8]. This metric provides an ordering of the documents, with higher weight assigned to documents that are considered more relevant to the query. This ordering is used to assign a *rank* to each document, with the document whose weight is highest assigned $rank = 1$. Retrieval systems typically return documents to the user in rank order.

2.1.2 Term Weighting

Term-weighting schemes are commonly applied to the entries in both the query and the document vectors [8, 3, 5]. The purpose of weighting schemes is to reduce the relative importance of high frequency terms while giving words that distinguish the documents in a collection a higher weight.

Two term-weighting schemes are used in our experiments. The first is *log/entropy* and the second is *term frequency /inverse document frequency* (abbr. *TF/IDF*). Term frequency (F_{ij}) is the number of times that term i appears in the document j. This is the implicit default in vector space retrieval if term weighting is not explicitly used. Log $(1 + logF_{ij})$ helps minimize the impact of a term that appears frequently within a document.

Inverse document frequency is defined $log\left(\frac{N}{n_i}\right)$, where N is the number of documents in the corpus and and n_i is the number of documents containing term i.

Entropy awards higher weights to terms that appear fewer times in a small number of documents. It is defined as

$$1 + \sum_{j=1}^{N} \frac{\frac{F_{ij}}{F_i} log \frac{F_{ij}}{F_i}}{log\,(N)},$$

where F_i is the number of times term i appears in the collection [1].

2.1.3 Query Precision and Recall Query precision and recall are two metrics for measuring the performance of a query. *Precision at rank n* is number of relevant documents with rank less than n divided by n. *Recall at rank n* is the number of relevant documents with rank less than n divided by the total number of relevant documents in the corpus.

2.2 TextMOLE TextMOLE is an information retrieval tool that is designed to load collections in a variety of formats and run queries on the loaded collection using vector space retrieval [9]. It supports term weighting and integrates many commonly used term weighting schemes. It also has features to measure query precision and recall for collections that come with a query file and a results file. A query file contains a set of standard queries. The results file links each query in the query file to the truly relevant documents in the collection. In section 4.3 we use TextMOLE as the platform for our experiments.

3 Related Work

Most efforts for improving vector space retrieval concentrate on improving the query by adding terms using some manner of feedback (often implicit and blind) to improve precision and/or recall [6, 2]. Since the query becomes much more detailed, the results list has the documents which match more of the terms closer to the top and an improvement in precision often occurs. In one study, improvements in precision at rank 20 ranging from 77-92% were reported in an idyllic, context-rich scenario [6]. However, in many scenarios where context was not particularly helpful, the algorithms saw little or no improvement (and even hurt the results). Unlike our algorithm, these methods do not typically keep a record of query histories, nor do they directly modify the query results list. Our algorithm focuses on pruning the query results list by actually removing documents.

Methods that do keep track of query histories have done so for purposes of developing a context for future queries [7, 4]. They assume that queries will follow a logical progression and attempt to use that sequence to enhance the query's precision with a variety of methods. These approaches use the query history to narrow in on what a user is looking for rather than to assemble an overall perception of the collection. The algorithm we present in the next section is markedly different in that it treats queries as independent samples.

4 Algorithm Overview

Traditional vector space retrieval and other retrieval methods make no provisions for sharing results between queries. Each query is run independently of every other query. Over the process of running a batch of queries, the discarding of the query-results information results in the loss of data that correlates how documents are related to one another. We are developing an algorithm that captures and stores query results data so that the data may be applied to future queries in order to further enhance precision.

The algorithm uses learning techniques to determine which documents are closely related to one another. When two documents both appear at the top of a ranked list of query results, the odds of them being related to one another is very high. Conversely, if one document appears at the top of a ranked query list and another document does not appear at all, then the odds of them being unrelated are also very high. When recording these statistics over the span of many queries, the algorithm provides information about the relationships between pairs of documents.

The algorithm can apply this information to query results in order to remove documents. It does this by taking the top n-ranked documents in the query results list and comparing their learned relevancies to the other documents returned. If a document has low scores, according to the learned data, it is removed (it is assumed to be not relevant to the query).

The algorithm can be divided into two distinct phases — the portion which learns the relationship between documents, and the portion which removes documents from new query results based on these learned results.

4.1 Learning Phase The learning phase is the portion of the algorithm that observes which documents are returned by queries and then extrapolates and records their relationships.

4.1.1 The Learning Matrix Currently, the algorithm stores the data it learns from queries in a document-by-document matrix. For each pair of documents a and b in the collection, there are two entries in the matrix: (a, b) and (b, a). These two entries are distinct to the algorithm: (a, b) represents the learned similarity score of document b to document a; that is, the score when a appears higher than b in the sorted query results list, or when a appears in the list and b does not appear at all. The matrix is not symmetric — it is possible to have a very strong relation from document a to document b but not vice versa (for example, if the subject of document b is a specialized topic of a more generalized area discussed by document a).

Two attributes are recorded during the learning process — the positive associations and the negative associations. These are recorded separately so that certain situations, such as whether the documents never appear in any queries, or whether they are equally positive and negative, are not ambiguous. Along with the raw 'learned similarity score', each entry in the matrix also stores the number of queries that contributed to the score. This is necessary to put the raw score in perspective.

4.1.2 Assigning Matrix Values For any given query, the learning algorithm will assign related documents a score between 1 and 0 for positive and negative associations. For positive associations, if two documents have the two highest rankings for a given query, then they will have a number very close to 1 added to their raw score. Conversely, the entries that represents the highest ranked and the lowest ranked documents will have a number very close to 0 added to their score. Documents in between these two extremes will have a value somewhere between 1 and 0 added. In addition to their placement at the top or bottom of the rankings, their proximity to one another influences this score. The closer two documents are to each another, the higher their score will be. The formula for positive scoring is shown in equation 4.1. Where a and b are two documents in the query results list, r_a is the rank of a, and s is the overall size of the query results list.

4.1.3 Negative Weighting The formula for negative weighting is similar. If a document a is at the top of the results and a document b is not in the results, then the entry (a, b) is given a weight of 1. If a document a appears at the bottom of the query results, and document b does not appear at all, then the negative association between them would be a number close to 0. The formula for negative scoring is shown in 4.2. Where a is the document that appeared in the results and s is the total size of the results list. For each query, each of these values are accumulated in the matrix.

$$\frac{\left(1-\frac{|r_a-r_b|}{s}\right)+\left(1-(\frac{r_a+r_b}{2*s})^2\right)}{2} \tag{4.1}$$

$$1-\frac{r_a)}{s} \tag{4.2}$$

If neither document appears in the results list then their scores in the learning matrix are not altered. When the raw scores are used in the next section, they are divided by the number of queries that contributed to their scores, thereby producing the 'average' score for a given number of queries.

4.1.4 Discussion These formulas produce data results that should capture precisely the data we want — which documents are related to one another and to what degree. If the documents consistently appear at the top of the result lists during querying, then their averaged scores will be very close to 1. If one document appears and the other never does, then they will have a negative association falling somewhere between 0 and 1. In practice, it is very probable that most entries in the learning matrix will have a combination of positive and negative scores when training is done.

4.2 Document-Removal Phase Once sufficient training has been completed, a document-removal phase attempts to remove extraneous documents from a query's results list. This algorithm is used to post-process the result list produced by traditional vector space retrieval (described in section 2.1.1).

4.2.1 Removal Criteria The algorithm uses several different criteria to determine which documents should be removed from the results list. In an ideal situation, there would be a threshold at which all relevant documents had greater positive (or lower negative) scores, and all non-relevant documents fell below the threshold. Depending on the needs of the particular system, however, these parameters can be set to remove a larger number of documents (and potentially remove some relevant documents as well), or 'play it safe' and remove a smaller number of documents and probably not any truly relevant documents.

The algorithm uses the first n documents as a basis for comparison to the rest of the documents on the list, for it is likely that the majority of those documents are truly relevant to the query. The algorithm then compares each other document in the results list to each of those n documents. The 'threshold' that these other documents must pass in order to stay in the results list is twofold: the first test is to see if the average positive score between the first n documents and the others is greater than a threshold x, where x is generally 0.7 or greater. The second test is to compare the ratio of positive scores to negative scores — this number is very dependent on the size of the training set, but with a large amount of training data, ratios around 20:1 or 15:1 produce good results. This is also dependent on the general strength of the relationship between documents in a collection. If the collection pertains to a very specific topic, these ratios may be higher. The second test against the threshold x is necessary in order to assure that x is not an aberrant score gained by just one or two positive scores when the correlation between the documents has dozens of negative scores as well.

4.2.2 Discussion It is very important to note that the original precision of queries must be reasonably high for the first n documents in order for this algorithm to properly weed out irrelevant documents. If the precision of the original query is too low, then the algorithm will not function well and will remove relevant documents as readily as irrelevant ones. Vector space retrieval — without any term weighting on the query or the documents — provides very minimal benefits when coupled with this approach. This is because normal vector space retrieval is not accurate without weighting schemes (it essentially just ranks documents according to whichever has the most number of words from the query). However, when vector space retrieval uses term-weighting schemes (such as log/entropy or TF/IDF), then the results are much more promising when coupled with this technique.

4.3 Results Current results have had success, as shown in Tables 1 and 2. Using the methods described above, we have been able to successfully reduce the number of documents returned by a query; the number of irrelevant documents removed is typically much greater than the number of relevant documents removed. The ratio of irrelevant documents removed to relevant documents removed almost always exceeds the baseline ratio of irrelevant to relevant in the original results list (if the ratio was not greater, then one might as well pick documents to remove at random).

The data in Tables 1 and 2 showcase the algorithm's effectiveness when applied to the MED, CACM, and CISI collections. In each case, the algorithm removed on average 89.8% of the documents from each query results list when an aggressive threshold is used. Precision and recall are calculated using all documents with a similarity score greater than zero. When the algorithm is tested using aggressive parameters, on average the precision rises by 439% and the recall declines by 37%. When using conservative parameters, the precision rises by 58.7% and the recall declines by 5.7%.

For all three collections, the same term-weighting schemes were used throughout the querying process. The documents were weighted using the log/entropy scheme and the queries were weighted using the TF/IDF scheme. These schemes were selected because they are known to consistently produce accurate results.

For the CISI collection, 49 queries were used for training and 27 separate queries were used to collect the test data. For CACM, 21 queries were used for training and 21 different queries were used to collect the data. In the MED collection, the same batch of 30 queries was used for both training and testing due to the small size of the collection. The thresholds used when aggressively testing the collections: the first 15 documents were used for comparison, documents had a ratio of at least *10:1* to at least 2 of the first 15, and the documents had an average positive score of at least 0.65/1.00 to the first 15 documents. The thresholds when testing conservatively: the first 15 documents were used for comparison, documents had a ratio of at least *4:1* to at least 1 of the first 15, and the documents had an average positive score of at least 0.65/1.00 to the first 15 documents.

The drop in recall in these results occurs when erroneous or misleading information has been incorporated into the learning matrix. For example, if a document that is actually relevant to the current query has a negative association — or a lack of a strong positive association — with all or most (depending on the thresholds) of the documents at the top of the results list then it would be erroneously removed.

5 Conclusions

The results so far are promising. The capacity to reduce the size of a query results list from 1000 documents to 200 or 100 documents without losing any relevant documents would be a revolutionary achievement in information retrieval. When this technique is fully developed, it will provide a method for creating an extremely high-precision results list with a minimal loss of relevant data. This would have a positive impact for a wide variety of applications where high precision is a necessity, such as medical or law databases.

In addition to developing algorithms to minimize the loss of recall within the current algorithm, we will

Table 1: Test data across 3 collections (aggressive)

	Average Baseline Results			Average Post-processed Results		
Identifier	**Size**	**Precision**	**Recall**	**Size**	**Precision**	**Recall**
MED	440.4	9.4%	89.2%	44.5	41.8%	58.7%
CACM	1256.4	1.3%	90.0%	176.1	6.4%	70.1%
CISI	1313.3	1.7%	98.0%	87	11.6%	47.0%

Table 2: Test data across 3 collections (conservative)

	Average Baseline Results			Average Post-processed Results		
Identifier	**Size**	**Precision**	**Recall**	**Size**	**Precision**	**Recall**
MED	440.4	9.4%	89.2%	203.7	17.1%	82.5%
CACM	1256.4	1.3%	90.0%	879.7	2.0%	86.6%
CISI	1313.3	1.7%	98.0%	930.8	2.4%	92.3%

add another step of the algorithm that attempts to 'normalize' the similarity of the first n documents to each another during removal. Currently the algorithm assumes that the first n documents are all relevant to the query, even though in practice this is rarely true. By comparing these initial documents to one another (again using the learning matrix) we can throw out documents that do not fit certain criteria — i.e., we could throw out documents that are very dissimilar to the rest in the list.

It is also worth noting that the algorithm can be fine-tuned to an individual collection in order to greatly increase performance. We hypothesize that much of this fine-tuning can be automatically incorporated into the functionality of the algorithm by scanning the learning matrix prior to document removal and using statistical facts about the learning matrix to set or tweak the removal parameters. Currently we have manually optimized the removal algorithm to the CISI and MED collections, and it has resulted in an increase of precision and recall. These same parameters also provided improvement in CACM, even though they are not optimized for this collection. When optimization can be done automatically and dynamically, the effectiveness of the algorithm will increase dramatically.

References

[1] Kolda Chisholm. New term weighting formulas for the vector space method in information retrieval. Technical report, 1999.

[2] Cui, Wen, Nie, and Ma. Probabilistic query expansion using query logs. In *Proceedings of the 11th international conference on World Wide Web*, pages 325–332, Honolulu, Hawaii, USA, 2002.

[3] Donna Harman. An experimental study of factors important in document ranking. In *Proceedings of the Ninth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 186–193, Pisa, Italy, 1986.

[4] Hayes, Avesani, Baldo, and Cunningham. Re-using implicit knowledge in short-term information profiles for context-sensitive tasks. In *Proceedings of 6th International Conference on Case-Based Reasoning*, Chicago, Illinois, USA, 2005.

[5] Manning and Schutze. *Foundations of Statistical Natural Language Processing.* MIT Press, Cambridge, MA, USA, 1999.

[6] Mandar Mitra, Amit Singhal, and Chris Buckley. Improving automatic query expansion. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 206–214, Melbourne, Australia, 1998.

[7] Xuehua Shen, Bin Tan, and ChengXiang Zhai. Context-sensitive information retrieval using implicit feedback. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 43–50, Salvador, Brazil, 2005.

[8] C. J. van Rijsbergen. *Information Retrieval.* Butterworths, London, UK, 1979.

[9] Daniel Waegel and April Kontostathis. Textmole: Text mining operations library and environment. In *Proceedings of the 2006 Technical Symposium on Computer Science Education*, Houston, TX, USA, 2006.