Document Filtering With Inference Networks

Jamie Callan
Computer Science Department
University of Massachusetts
Amherst, MA 01003-4610, USA
callan@cs.umass.edu

Abstract

Although statistical retrieval models are now accepted widely, there has been little research on how to adapt them to the demands of high speed document filtering. The problems of document retrieval and document filtering are similar at an abstract level, but the architectures required, the optimizations that are possible, and the quality of the information available, are all different.

This paper describes a new statistical document filtering system called InRoute, the problems of filtering effectiveness and efficiency that arise with such a system, and experiments with various solutions.

1 Introduction

Retrieval of documents from an archival collection (*retrospective retrieval*) and filtering documents from an incoming stream of documents (*document filtering* or *selective dissemination of information*) have been described as two sides of the same coin [2]. Both tasks consist of determining quickly how well a document matches an information need. Many of the underlying issues are the same; for example, deciding how to represent each document, how to describe the information need in a query language, what words to ignore (*stop words*), whether or not to stem words, and how to interpret evidence of relevance.

Much of the recent research on document filtering is based on the assumption that effective document retrieval techniques are also effective document filtering techniques. The TREC conference is a good example. The Routing track is oriented towards learning effective queries from training data, and the Filtering track is oriented towards determining useful dissemination thresholds. Neither track requires that a filtering system be used. Indeed, many participants index the set of Routing documents and then search the index with a retrospective retrieval system [7].

When filtering research is conducted with a retrieval system, important issues can be overlooked. Different architectures are possible, and perhaps required, to rapidly compare persistent information needs to transient documents. A filtering algorithm must make decisions based upon incomplete information; it may know what has happened in the past, but it cannot know, nor wait to know, what documents will be seen in the near future. Traditional corpus statistics, such as inverse document frequency (*idf*), have different characteristics when documents are encountered one-at-a-time. These issues are important, because they determine how efficient and effective statistical document filtering systems will be in "real world" environments.

This paper describes some of the issues that arose while developing the InRoute document filtering system from parts of the INQUERY retrospective retrieval system [5]. The two systems are based on the inference network model of information retrieval [11, 12, 2], share a common query language, and started out sharing much of the code that matches information needs to documents. However, there are as many differences as similarities between the two systems. The differences are the subject of this paper.

The paper begins with this Introduction and a brief review of previous research on document filtering. The InRoute architecture is presented next. Section 4 discusses the problem of maintaining effectiveness (precision) with incomplete corpus statistics, and presents experiments with some possible solutions. Section 5 discusses the problem of filtering quickly with complex profiles (queries), and presents experiments with some possible solutions. Section 6 concludes.

2 Document Filtering

Document filtering, also known as selective dissemination of information (SDI), has a long history, most of it based on the unranked Boolean retrieval model [8]. A user's information need is expressed by a *query*, also called a *profile*, in a query language. (Sometimes a profile is actually a set of queries for one user; in this paper, query and profile are considered synonymous.) Queries are expressed with Boolean logic. A query either matches or does not match a document. There is no ability to partially satisfy a query, or to determine how well a document matches or satisfies a query. Instead, the emphasis is on speed, and on indexing methods that enable very fast processing of documents against profiles.

LMDS [15] is an example of this class of systems. Each Boolean profile is analyzed to identify the least frequent trigram (LFT) that *must* occur whenever the profile matches

a document (a necessary, but not sufficient, condition for matching). Documents are converted into a restricted alphabet, and represented as a sequence of trigrams. For each profile, a table lookup determines whether its LFT is present. If not, the profile can not possibly match the document. This first stage is designed to eliminate $> 95\%$ of the profiles in just a few instructions each. If a profile's LFT is present, a slower second stage determines whether the document actually satisfies the Boolean query.

It is generally accepted that statistical systems provide better precision and recall for document retrieval than do unranked Boolean systems. The growing power of computer hardware has made statistical systems increasingly practical for even large scale document filtering environments. A common approach has been to simulate document filtering with an existing vector-space or probabilistic document retrieval system on a collection of new or recent documents (e.g., Pasadena [13], LSI SDI [6], INQUERY [4], Okapi [9], most TREC systems [7]). This approach is simple, effective, and has the advantage of a corpus from which to gather statistics like *idf*. However, it is not well-suited to immediate dissemination of new information, and it adds index creation, storage, and maintenance to the cost of document filtering.

SIFT [14] is a document filtering system based on the well-known vector-space retrieval model [10]. SIFT queries are unstructured (i.e., no query operators), so they can be indexed with inverted lists. A document "retrieves" a set of inverted lists that indicate which profiles to evaluate for the document. The document's score for a profile is determined with a vector-space algorithm. If the score exceeds a dissemination threshold, it matches the profile and is routed to the user. SIFT also incorporates relevance feedback algorithms that enable a user to refine a profile based upon relevant and nonrelevant documents.

3 The InRoute Architecture

The InRoute document filtering system is based upon the inference network model of information retrieval and filtering [11, 2]. The major tasks performed by InRoute are creation of query networks, creation of the document network, and use of the networks to filter documents. The document network is created automatically by mapping documents onto content representation nodes, which are implemented with traditional inverted lists. Query networks are specified by a user in either natural language or a structured query language. Document filtering is performed by using recursive inference to propagate belief values through the inference net, discarding any documents whose belief is below a dissemination threshold. Figure 1 shows the major components of the InRoute system, and how information flows between them. The following sections discuss each component in more detail.

3.1 Parsing Profiles into Query Nets

As with INQUERY [5, 4], InRoute information needs may be specified in either a query language or natural language. InRoute shares INQUERY's rich query language, which includes probabilistic AND, OR, and NOT operators, proximity operators, probabilistic phrase and passage operators, and a weighted sum operator for user-specified weights.

The query network is a directed acyclic graph (DAG) in which the root is a query operator (e.g., #SUM), internal nodes correspond to nested query operators (e.g., AND, OR, phrase, proximity, etc), and leaves correspond to query terms. When parsing is complete, the DAG is optimized, for example by removing redundant query terms and operators, reordering arguments to Boolean operators, etc, in order to minimize the cost of evaluating the query.

Retrospective document retrieval systems may optimize a query by removing query fragments not found in the document collection. For example, if "retrieval" does not occur in the collection, the query "information AND retrieval" cannot match any document. This optimization does not apply to document filtering, because the system sees the query before seeing any documents. However, the same principle applies to *document parsing*. The document parser can discard any document term that does not occur in at least one profile. Therefore the query parser maintains a dictionary of terms that occur in profiles, for use by the document parser.

3.2 The Clipset

InRoute compares a single document at a time to a clipset. A *clipset* is a set of query networks, each representing a different profile, and a set of profile-specific dissemination thresholds between 0.0 and 1.0 (Section 3.4). Clipsets are persistent. When a user adds, deletes or modifies an information need, the corresponding query network is added to, deleted from, or modified in the clipset. When InRoute is filtering documents, the clipset resides entirely in memory.

The persistence of a clipset makes it possible to cache information that is transient in traditional document retrieval. For example, inverse document frequency and default belief values are calculated, used and then discarded by INQUERY during document retrieval. InRoute calculates them once, when the query net is added to the clipset, and then retains them in the nodes of the query net to improve the speed of document filtering. Each node in the query net is also associated with space from a buffer pool, so that the overhead of allocating and freeing dynamic memory is avoided during filtering.

The desire to cache information in the query nets is offset by the need to represent query nets efficiently. A system in a commercial environment might need to filter tens of thousands of profiles on a single processor. Each node in an InRoute query net requires about 100 bytes, enabling InRoute to store 100,000 profiles, with an average of 22 terms and operators each, in about 256 megabytes of memory. This enables InRoute to keep the profiles in main memory for most document filtering tasks.

Adding, deleting, and modifying profiles are accomplished quickly, due to the independence of profiles from one another and the unordered nature of the clipset. New profiles are appended to the end of the clipset. Modification requires parsing the modified profile into a query net, switching a clipset pointer from one query net to another, and freeing the old query net. Deletion requires freeing a query net, and moving the last profile in the clipset up to fill the hole created by deletion.

3.3 Document Parsing

Much of what is known about document parsing and indexing for retrospective document retrieval also applies to document filtering. For example, recognizing document structure (begin, end, title, author, date, etc), removal of frequent words (*stopwords*), and removal of word suffixes (*stemming*) are all important in both retrieval and filtering. However, in a document filtering environment, a document enters the system, is parsed and indexed, is filtered or routed to the

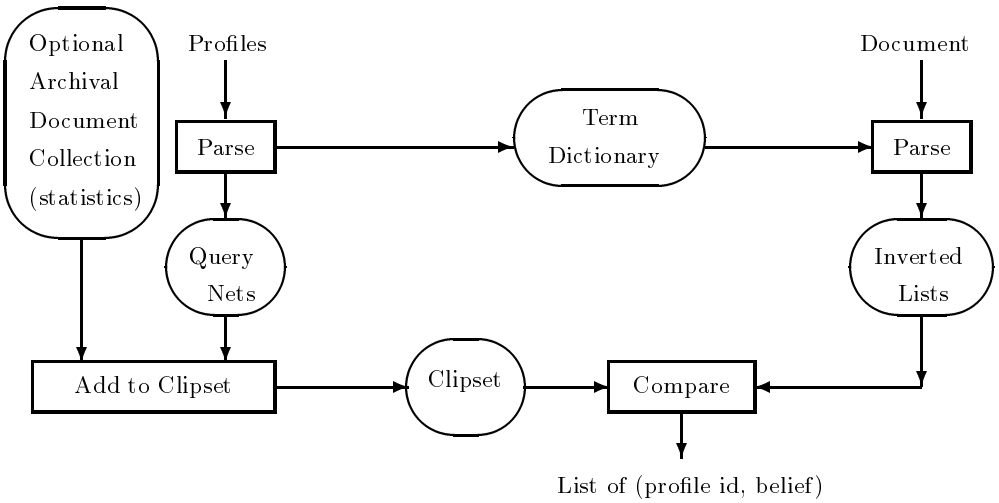

Figure 1: The architecture of the InRoute document filtering system. Ovals represent data structures. Boxes represent functions. Arrows show the flow of information.

appropriate user, and is then discarded. There is no reason to incur the I/O cost of writing documents or their indices to disk.

InRoute is designed around a "lightweight indexing" philosophy. Indexing speed is maximized by creating inverted lists only for terms that actually appear in one or more profiles. As a result, most of the tokens in a document are discarded as soon as they are recognized.

Documents are supplied, one at a time, by an application program. A lexical scanner identifies document markup, discards stopwords, and performs word stemming. Stems that do not appear in the profile term dictionary are discarded. Inverted lists are constructed, incrementally as tokens are encountered, for the remaining stems. When the document is parsed completely, the result is a set of inverted lists representing the document network for that document. Finally, each list is annotated with the belief that the term will contribute. Belief is calculated using a $tf.idf$ formula, as shown below.

$$ntf = 0.4 + 0.6 \cdot \frac{tf}{tf + 0.5 + 1.5 \cdot \frac{dl}{avg\_dl}} \quad (1)$$

$$idf = \frac{\log(\frac{C+0.5}{df})}{\log(C + 1.0)} \quad (2)$$

$$\mathrm{bel}_{\mathrm{term}}(t) = 0.4 + 0.6 \cdot ntf \cdot idf \quad (3)$$

where:

tf is the frequency of term t in the document,
dl is the document length,
avg_dl is the average document length in the collection,
C is the number of documents in the collection, and
df is the number of documents in which term t occurs.

Three of the statistics above are derived from the corpus as a whole: df, avg_dl, and C. Accurate values for these three statistics are known only after all documents are filtered, so filtering must be performed with estimates. Section 4 discusses the problem of obtaining accurate estimates.

Parsing a 3,000 byte document and later freeing the associated indices and data structures takes 0.02937 seconds (wall-clock time) on an otherwise idle DECStation 3000-600 (Alpha CPU, 175 MHz clock) with 64 megabytes of memory.

Document parsing speed is affected by the number of profiles, because inverted lists are built only for terms in the profile term dictionary. As more profiles are added, the vocabulary grows larger. Fortunately, adding a large number of profiles causes only a small increase in the size of the term dictionary [16], and therefore only a small decrease in document parsing speed.

3.4 Comparing a Document to Profiles

After a document is indexed, it can be compared to a clipset. Retrospective document retrieval systems owe their speed partially to indexing methods, such as inverted lists, that enable the system to consider only those documents that have terms in common with a query. A similar need exists for document filtering, because many profiles have nothing in common with most documents. Section 5 presents experiments with several different methods of selecting profiles.

Once a set of profiles is selected, each profile must be compared to the document. InRoute iterates through the selected profiles, determining for each the belief that the document satisfies the information need. The belief in a document for a particular query net is determined with depth-first evaluation. For each query term that occurs in the document, InRoute must locate the appropriate inverted list, lookup the belief associated with the term, and then combine the belief with the beliefs from other terms, according to the probabilistic operators being used [4]. If proximity operators are used, InRoute must also lookup the locations where the term occurs, intersect those with the locations of other proximate terms, and then compute the belief for the proximity operator dynamically, using the same $tf.idf$ formulas described above.

A profile is returned for a document if and only if it matches the query, *and* if the belief that the document sat-

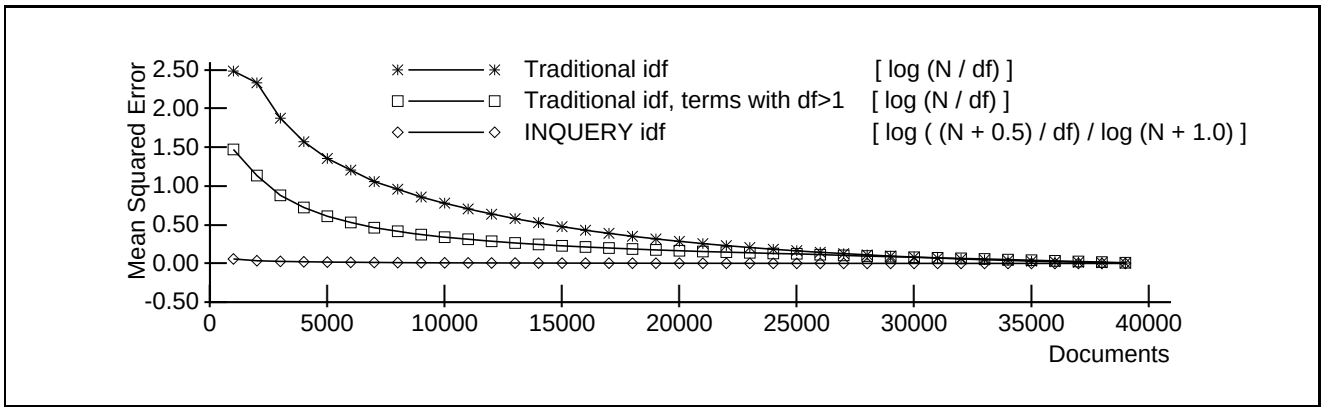

Figure 2: Mean squared error of *idf* estimates.

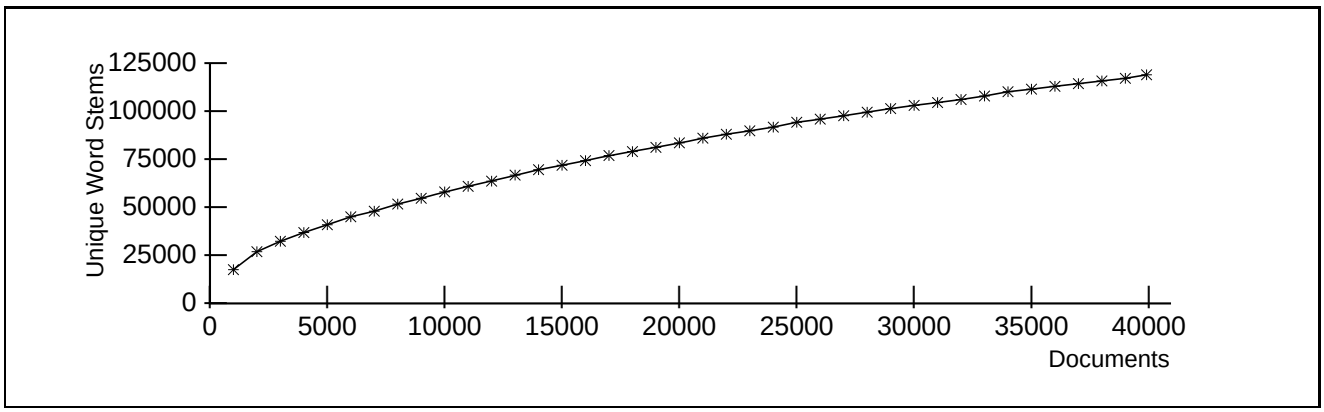

Figure 3: Growth of vocabulary while filtering WSJ '88.

isfies the information need exceeds a profile-specific, user-supplied dissemination threshold. This latter requirement is particularly important for probabilistic query operators. A document "matches" a weighted sum or probabilistic ("fuzzy") AND operator if even one query term is present, although the belief in the document is usually low. This behavior is rarely a problem in a ranked retrieval system with large sets of documents, because a low belief causes a document to appear low in the rankings. However, in a filtering environment, a low-scoring document may still be the best document encountered that day. If the system does not discard documents with low beliefs, users must either develop strictly Boolean queries, or wade through irrelevant documents on days when no relevant documents occur.

Filtering a 3,000 byte document for 1,000 Boolean profiles, each containing an average of 22 terms and operators, takes 0.024 seconds (wall-clock time) on an idle DECStation 3000-600. The time is proportional to the number of profiles; twice as many profiles takes twice as long. InRoute processes a 109 megabyte file of 39,906 Wall Street Journal documents against 1,000 Boolean profiles (extended Boolean model) in 25 minutes (wall-clock time), generating 616,487 matches. A similar experiment with 1,000 statistical profiles of similar complexity required 33 minutes.

Statistical profiles currently take longer to evaluate than Boolean profiles. Each statistical profile must be evaluated fully for each document, while evaluation of Boolean profiles stops when the first AND, OR or NOT condition is violated. Section 5 discusses approaches to evaluating statistical profiles more rapidly.

4 Effectiveness Experiments

One important difference between document filtering and document retrieval is how corpus-wide statistics like inverse document frequency (idf) and average document length are obtained. The effectiveness of current retrieval models depends upon accurate corpus statistics, which document retrieval systems gather while indexing the collection. In an online environment, where documents must be filtered as soon as they arrive, accurate corpus statistics are not available until after all of the documents have been filtered.

One approach used in some TREC experiments is to use statistics from another, presumably similar, corpus. For example, idfs from the TREC-3 training and test corpora produce nearly identical document rankings in the INQUERY retrieval system. This approach is *effective*, but it may be impractical in practice.

Obtaining idfs from a retrospective corpus can be expensive, particularly if queries include large numbers of proximity operators, as in [1]. Idfs for unindexed query fragments

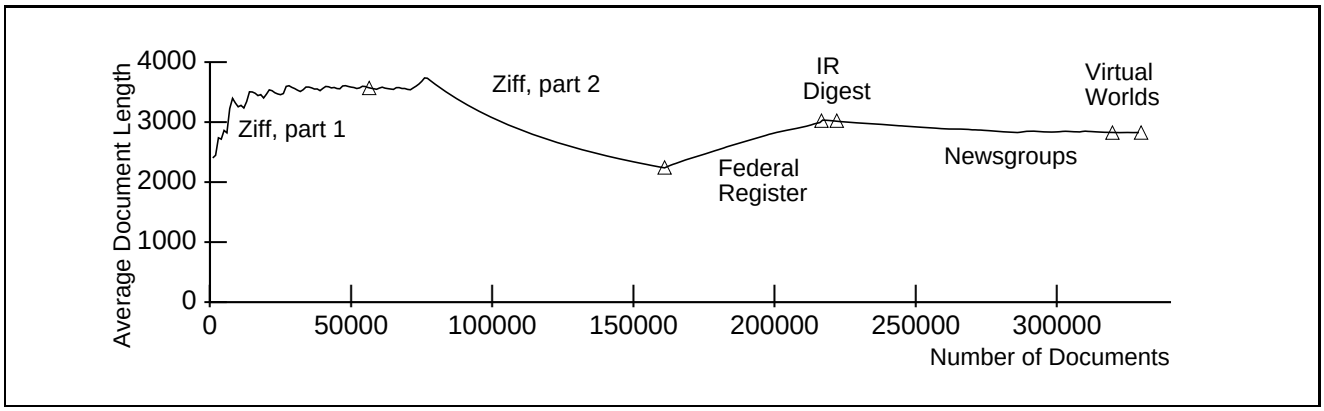

Figure 4: Changes in average document length as the TREC-4 Routing corpus is filtered. Document length is represented in bytes.

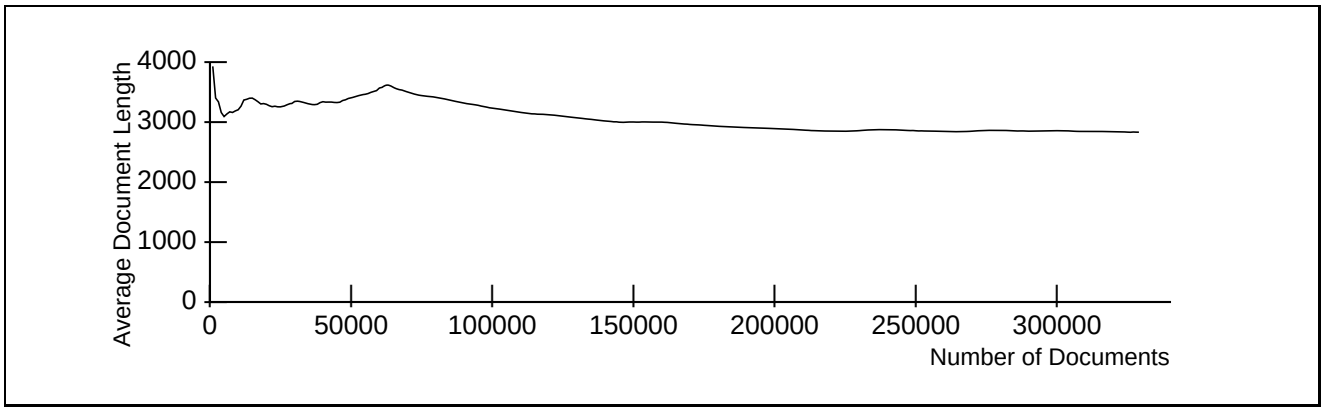

Figure 5: Changes in average document length as the TREC-4 Routing corpus is filtered. The subcollections are interleaved in this experiment. Document length is represented in bytes.

(e.g., proximity operators) can only be obtained by running queries against the retrospective collection. In an experiment with the INQUERY retrieval system, it took several hours to obtain the idfs for the proximity operators in 50 routing queries (set INQ203) used in the TREC-4 Routing task. This cost would be prohibitive in a "real world" setting.

An alternate approach, studied here, is to estimate corpus statistics dynamically, as each document is encountered. This approach has the advantage of being "low cost" and of not requiring a similar training corpus. Although the corpus statistics will initially be inaccurate, they will eventually converge to their "true" values for the corpus. The question is whether they will converge quickly enough.

The corpus statistics required for InRoute are inverse document frequency (idf) and average document length. We study idf on a small corpus, because it converges relatively quickly. We study average document length on a larger corpus, because it converges less quickly.

Figure 2 shows the convergence of idfs for terms in the 1988 Wall Street Journal corpus. Each curve shows the mean squared error (MSE) between estimated idf and true idf at 1,000 document increments. The top curve shows the MSE for a traditional method of computing idf. The bottom curve shows the MSE for the "scaled" idf used by InRoute and INQUERY (Equation 2).

Idfs converged rapidly to their true values (Figure 2), even as the vocabulary continued to grow (Figure 3). A "scaled" idf converges more rapidly, because it gives a more accurate estimate for terms that occur just once. An unscaled idf for terms that occur just once changes significantly as more documents are observed, while a scaled idf changes very little. If terms that occur just once are excluded (middle curve), the MSE for the traditional method is reduced by about half.

Figure 4 shows the convergence of average document length for the TREC-4 Routing corpus. It takes about 25,000 documents to reach a stable estimate, but the estimate then changes significantly whenever the document stream shifts from one subcollection to another.

The effect of shifting from one subcollection to another can be eliminated by interleaving the subcollections.[1] Figure 5 shows the convergence of average document length in a proportionally interleaved TREC-4 Routing corpus. It

[1]The documents could also be ordered by publication date, but doing so does not eliminate the "subcollection" effect, because the subcollections cover different periods of time.

Precision	Number of Documents Used Only For Training						
	0	1000	3000	5000	10000	15000	20000
at 5 docs	−18.7%	−16.1%	−11.8%	−8.1%	−3.1%	−3.7%	−3.1%
at 10 docs	−16.4%	−13.8%	−10.9%	−9.0%	−3.9%	−2.9%	−3.5%
at 15 docs	−16.0%	−11.9%	−9.9%	−7.0%	−3.5%	−2.2%	−1.1%
at 20 docs	−14.7%	−13.2%	−10.5%	−8.5%	−5.4%	−3.8%	−2.7%
at 30 docs	−13.6%	−12.2%	−10.0%	−9.1%	−5.7%	−3.5%	−3.4%
at 100 docs	−9.9%	−11.2%	−9.2%	−8.1%	−5.8%	−5.3%	−4.7%
at 200 docs	−2.8%	−9.1%	−8.1%	−7.3%	−5.3%	−5.3%	−5.3%
at 500 docs	+4.4%	−3.9%	−3.3%	−3.0%	−2.8%	−4.0%	−5.2%
11Pt Avg	−8.0%	−12.1%	−10.7%	−9.4%	−6.9%	−7.2%	−7.8%

Table 1: Effect on precision and average precision of training on (and then discarding) the first n documents; as compared with "perfect" idfs for the corpus.

takes about 20,000 documents to reach a stable estimate in this corpus, but the estimate is 15% above its eventual final value, and it continues to drift up and down, smoothly but by significant amounts, for another 100,000 documents.

An experiment with TREC-4 Routing queries and documents investigated the effects on recall and precision of learning corpus-wide statistics during filtering. These metrics were chosen because they were used in the TREC Routing track.

InRoute was run twice on the TREC-4 corpus (935 MB, 329,780 documents) and INQ203 Routing queries (50 queries, 50 terms and 200 proximity pairs each) [1]. In one run, corpus statistics were available *a priori* ("perfect" statistics). In the other, estimates were updated as each document was encountered ("learned" statistics). The experiment required dissemination thresholds that would disseminate at least 1,000 documents for each query. We used the document score that INQUERY assigned at rank 1,000, because it was conveniently available.

Learned corpus statistics produced a significant loss in average precision at all cutoffs and levels of recall (Table 1, Column "0"). The effect of inaccurate corpus statistics in the first few thousand documents is rather dramatic, given that the estimates converge to relatively accurate values after filtering only a small percentage of the corpus. However, analysis reveals that learned statistics produce substantially higher scores for documents filtered "early" than for documents filtered "later", when corpus statistics have converged. The "early" documents, with their overly generous scores, dominate the top of the rankings.

If the first several thousand documents are used only for training purposes (i.e., are not disseminated), the effect of learned corpus statistics on recall and precision is less significant (Table 1, columns "1000" to "20000"). For example, if 15,000 documents are used for training, corpus statistics produce a 2.2-5.3% loss in precision at cutoffs 5-500. This is a crude way of analyzing the effects of learning corpus statistics, because the baseline is based on all of the relevant documents, while the filtered set is missing whatever relevant documents were discarded during training. However, it confirms that, after the initial period of training, learned corpus statistics are effective for filtering.

5 Efficiency Experiments

Speed is an important characteristic of document filtering systems, and consequently techniques for optimizing Boolean filtering systems are well-known. Similar techniques for statistical document filtering are required.

Filtering a document involves profile selection and evaluation. Profile selection determines which profiles to evaluate; profile evaluation determines how well a document satisfies a profile. Both can be optimized, but we restrict our attention here to profile selection. For each document, the goal is to spend either no time or nearly no time on most of the profiles.

One approach, used for example in SIFT [14], is to index profiles with inverted lists. The terms in a document "retrieve" profiles during filtering. This approach works particularly well with the unstructured queries that characterize vector-space systems, because profile scores can be computed when inverted lists are merged.

Profile indexing is less effective with the structured queries that characterize inference network systems, because scores for structured queries cannot be computed when profile inverted lists are merged ("...we cannot simply turn the inference network 'upside down'..." [2]). In this case, profile indexing can be used only to identify profiles that are candidates for evaluation. Profile indexing may also be less effective on long Routing queries (e.g., Section 4), because a profile with many terms is more likely to have at least one in common with any document.

A new profile selection technique, *MinTerm Indexing*, solves this problem. Prior to filtering, each profile is analyzed to determine the number of document terms it must match before a document can exceed the dissemination threshold. The "optimal" document for a query term is one in which ntf approaches 1 (Equation 1). The MinTerm estimate is made by setting ntf to 1 for each query term, and then ordering sibling query net nodes by the estimated belief. When reordering is complete, the query net is traversed, accumulating belief values and counting query terms. When the accumulated belief exceeds the dissemination threshold, the *minimum* number of terms necessary to exceed the threshold is known. This information is stored in the profile index, and used during filtering. The profile is selected only if it matches a sufficient number of document terms.

The MinTerm estimate is obtained with an algorithm similar to algorithms that reorder and/or optimize unstructured queries (e.g., [3]) and Boolean queries (e.g., [15]). Reordering by optimal belief is perhaps a more general technique, because it applies to both unstructured queries and queries structured with a wide range of Boolean and probabilistic operators. However the important difference is that the query is not reordered to optimize query evaluation (although doing so is a good idea), but to find the minimum

	3,000 simple profiles 1988 WSJ corpus			50 complex profiles TREC-4 Routing corpus		
	No Index	Inverted Index	MinTerm Index	No Index	Inverted Index	MinTerm Index
Profiles Fully Evaluated	100%	24.2%	4.25%	100%	97.5%	74.9%
Total Filtering Time (h:mm)	1:28	0:55	0:41	5:26	5:31	2:53
Filtering Rate (MB / hour)	74	119	160	172	170	324
Avg Documents Disseminated Per Profile	77.8	77.8	77.8	999.5	999.5	922.0

Table 2: The effects of three profile selection techniques on document filtering.

number of terms that *must* match before a document could satisfy an information need.

MinTerm Indexing is implemented as a three stage filter. First, document terms "retrieve" profiles, using inverted lists. The number of terms matching each profile is determined as inverted lists are merged. Next, "retrieved" profiles that don't match enough document terms are discarded. Finally, the remaining profiles are evaluated completely, and any with scores below the dissemination threshold are discarded.

The speedup obtained with MinTerm Indexing increases as a profile's dissemination threshold increases. If the threshold is low, the minimum number of terms necessary to select a profile is one, reducing MinTerm Indexing to simple profile indexing.

MinTerm Indexing can be a *safe* or *unsafe* optimization, depending upon how it is used. If profiles are reanalyzed each time the idfs change, it is safe, i.e., guaranteed to select for a given document every profile that can possibly exceed the dissemination threshold. If idfs change, as when they are being learned, the MinTerm estimate may become wrong. Usually the MinTerm estimate will be an underestimate, causing no harm, because idfs can fall rapidly (increasing the actual MinTerms), but tend to rise slowly (decreasing the actual MinTerms). However, it may make sense to reanalyze profiles periodically, for example every few thousand documents, when idfs are being learned.

The relative effectiveness of these techniques is demonstrated in two experiments. In one experiment, the TREC 1988 Wall Street Journal corpus (109 MB, 39,906 documents) was filtered for a set of 3,000 simple, artificially-generated profiles (10 terms and 4 proximity pairs each). The dissemination threshold was set to yield about a 0.2% "hit" rate. In the second experiment, the TREC-4 Routing corpus (935 MB, 329,780 documents) was filtered for a set of 50 complex profiles (50 terms and 200 proximity pairs each). The dissemination threshold was set to yield about 1,000 documents per profile (a 0.3% "hit" rate), as is common in TREC Routing evaluations. In both experiments InRoute was learning corpus statistics, so profiles were reanalyzed and their MinTerm estimates updated every 1,000 documents. Table 2 summarizes the results.

With simple profiles (the 1988 WSJ experiment), simple profile indexing was a substantial improvement over evaluating all profiles. Filtering time was reduced by 37.5% without impacting effectiveness. MinTerm Indexing reduced filtering time by another 25%.

With complex profiles (the TREC-4 Routing experiment), simple profile indexing was slightly *worse* than evaluating all profiles. The computational cost of simple profile indexing provided little benefit, because most documents had a term in common with most of these routing profiles. However MinTerm Indexing, which considers the *number* of terms a document has in common with a profile, reduced filtering time by 47%.

MinTerm indexing was "unsafe" in these experiments, because corpus statistics were updated after each document but profile MinTerm estimates were updated after each 1,000 documents. In the 1988 WSJ experiment, the cost was the loss of one document from a set of 233,735. In the TREC-4 corpus, the cost was a much higher 76.5 documents per profile.

Most of the TREC-4 loss was due to experimental error. The algorithm that determined the number of terms a document and profile have in common did not consider the effect of duplicate terms in the profile. Duplicates are very rare in the 1988 WSJ profiles, so this error had no effect on the first experiment. Duplicates are quite common in the TREC-4 profiles, hence the "missed" documents in the second experiment.

6 Conclusion

This paper presents a new statistical filtering system called InRoute. Although InRoute is based on the inference network model, it is typical of most statistical filtering systems in some respects.

All statistical filtering systems that rely on corpus statistics such as idf or average document length are affected by inaccurate statistics. This paper shows that those effects can be pronounced, but that they seem limited to the first few thousand documents filtered. During this "training" period, a system that avoids corpus statistics, or that can learn to estimate them quickly, will have a decided advantage. After the statistics have converged, a statistical filtering system can be as effective as a comparable statistical document retrieval system.

Although all statistical filtering systems must select profiles for evaluation, different solutions are appropriate for structured and unstructured queries. This paper presents a new profile selection technique for structured queries, called MinTerm Indexing, that is effective and efficient. The effect on filtering speed is dramatic, narrowing the gap in speed between Boolean and statistical filtering systems.

Many open problems remain. The problem of acquiring corpus statistics dynamically is particularly rich, and this paper just scratches the surface. Obtaining statistics from another corpus is effective but often expensive, while learning them "on the fly" is less effective but cheap. These represent different ends of a spectrum; a variety of hybrid techniques are possible. However, any technique that learns statistics as documents are filtered will have the characteris-

tic that a document's score depends partially upon its position in the document stream. "Breaking news" might receive higher beliefs than later documents on the same subject, because idfs will decline, for example. This could make the system feel responsive to user interests, although in reality a document's score for a given profile would be somewhat less predictable. It is not at all clear how such a system might be evaluated.

Acknowledgements

I thank Eric Brown, Stephen Harding, and Sandhya Kasera for their assistance in the work described here. This research was partially supported by the NSF Center for Intelligent Information Retrieval at the University of Massachusetts, Amherst, by the National Science Foundation, Library of Congress, and Department of Commerce under cooperative agreement number EEC-9209623, and by NRaD contract number N66001-94-D-6054.

References

[1] J. Allan, L. Ballesteros, J. P. Callan, W. B. Croft, and Z. Lu. Recent experiments with inquery. In D. Harman, editor, *Proceedings of the Fourth Text REtrieval Conference (TREC-4)*. National Institute of Standards and Technology Special Publication, (to appear).

[2] N. J. Belkin and W. B. Croft. Information filtering and information retrieval: Two sides of the same coin? *Communications of the ACM*, 35(12):29–38, 1992.

[3] Chris Buckley and Alan F. Lewit. Optimization of inverted vector searches. In *Proceedings of the Eighth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 97–110, New York, NY, 1985. ACM.

[4] J. P. Callan, W. B. Croft, and J. Broglio. TREC and TIPSTER experiments with INQUERY. *Information Processing and Management*, 31(3):327–343, 1995.

[5] J. P. Callan, W. B. Croft, and S. M. Harding. The INQUERY retrieval system. In *Proceedings of the Third International Conference on Database and Expert Systems Applications*, pages 78–83, Valencia, Spain, 1992. Springer-Verlag.

[6] P. W. Foltz and S. T. Dumais. Personalized information delivery: An analysis of information filtering methods. *Communications of the ACM*, 35(12):51–60, 1992.

[7] D. Harman, editor. *Proceedings of the Fourth Text REtrieval Conference (TREC-4)*. National Institute of Standards and Technology Special Publication, Gaithersburg, MD, (to appear).

[8] K.H. Packer and D. Soergel. The importance of SDI for current awareness in fields with severe scatter of information. *Journal of the American Society for Information Science*, 30(3):125–135, 1979.

[9] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In D. K. Harman, editor, *The Third Text REtrieval Conference (TREC-3)*, Gaithersburg, MD, 1995. National Institute of Standards and Technology, Special Publication 500-225.

[10] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[11] H. R. Turtle and W. B. Croft. Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems*, 9(3):187–222, 1991.

[12] Howard R. Turtle and W. Bruce Croft. Efficient probabilistic inference for text retrieval. In *RIAO 3 Conference Proceedings*, pages 644–661, Barcelona, Spain, April 1991.

[13] M. F. Wyle and H. P. Frei. Retrieving highly dynamic, widely distributed information. In *Proceedings of the ACM SIGIR International Conference on Research and Development in Information Retrieval*, pages 108–115, Boston, MA, 1989. Association for Computing Machinery.

[14] T. Yan and H. Garcia-Molina. SIFT – A tool for wide-area information dissemination. In *Proc. USENIX Winter 1995 Technical Conference*, New Orleans, January 1995.

[15] J. A. Yochum. A high-speed text scanning algorithm utilizing least frequent trigraphs. In *Proceedings of the IEEE International Symposium on New Directions in Computing*, pages 114–121, Trondheim, Norway, 1985. IEEE.

[16] G. K. Zipf. *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*. Addison-Wesley, Reading, MA, 1949.