QueryTracker: An Agent for Tracking Persistent Information Needs

Gabriel Somlo Adele E. Howe
Computer Science Dept., Colorado State University
Fort Collins, CO 80523 USA
{somlo,howe}@cs.colostate.edu

Abstract

Most people have long term information interests. Current Web search engines satisfy immediate information needs. Specific sites support tracking of long term interests. We present an agent that satisfies a gap in these services. QueryTracker implements a search engine interface with state. A user's query and a learned alternative is automatically submitted daily to a search engine. A profile of the user's interest is constructed based on user relevance feedback. Daily search results are disseminated if they correspond closely to the user's profile. Changed pages are checked for relevant additions. We evaluate the impact of generating the alternate query and assess alternative methods of filtering the results in a month long user study. Results show that QueryTracker is effective at disseminating relevant documents over a period of time.

1. Introduction

The Web is an unprecedented source of information... assuming one can find what one wants. Although search engines are the tool of choice for answering immediate needs for information, many studies (e.g., [8]) have shown that users are relatively poor at using them; they tend to enter short (two to four word) queries, view few results, browse only the first page of results, and rarely reformulate unsuccessful queries.

Even then, search engines do not naturally satisfy on-going, persistent information needs. A Pew study [12] found that users will be much more dogged in their search for health information, starting at general search engines, expanding their scope and spending at least 30 minutes at each sitting on the search.

Many Web users have persistent information needs. As suggested by the Pew study, patients need to track the latest developments in treatment for chronic or potentially terminal illnesses. Portfolio managers (an increasingly large sector of the U.S. populace as individuals manage IRAs, 401-Ks, etc.) need to know news about the industries and stocks in their portfolios to track opportunities to buy and telltales to sell. Hobbyists want to know the latest about their pastime. Scientists must track the latest papers in their fields of expertise [2].

Tracking services are available for specific sites, e.g., New York Times or via tracking bots (e.g., "The Easy Bee" and "NewsHound"[3]). However, these services are limited in which sites are monitored, and they typically require the user to select a pre-defined category, use a restrictive query or focus on detecting changes. One of the most capable of these, the Copernic agent [5], monitors the contents of specified pages for changes and re-submits specified queries regularly, informing the user of changes in the results.

In this paper, we describe an agent, QueryTracker, that assists users in satisfying their long term information needs by acting as an intelligent intermediary between the user and the search engine. In addition to resubmitting queries and monitoring the results for changes, QueryTracker learns a much more complete picture of the user's needs through relevance feedback. The learned interest profile can then be used to automatically generate a query (to compensate for the difficulty users have in formulating queries) and to filter the results so that only results deemed relevant to the need are disseminated. Filtering is also applied to URLs that have been viewed before to determine whether changes are relevant to the need.

We test several aspects of QueryTracker's design in a user study. The user study compares the efficacy of the query generation to the original query and compares the performance of different filtering techniques. We find significant differences in the performance of the different methods, as well as differences due to individual user's information needs. Overall, the system appears to work well at presenting the user with relevant documents, continuing to do so long after the original query was submitted.

1.1. Related Work

QueryTracker brings together four capabilities: personalized tracking, automatic query generation, relevant change detection, and filtering. We briefly discuss the most relevant recent research on these three areas for Web applications similar to ours; the fourth is also covered in [15].

Most of the Web tracking programs are products or shareware. Thus, little information is available on their design. CiteSeer provided a tracking capability for researchers to monitor their field [2]. Users could describe their needs in terms of constraints (e.g., keywords or specific URLs) or as a set of papers that are representative of their interests. The system applied two relatedness measures (text and citation based) to determine whether a new document is relevant. The text measure is a thresholded TFIDF (Term Frequency * Inverse Document Frequency) scheme; the citation measure is analogous to TFIDF for citation counts. Thresholds are set by the system designer.

Two intelligent agents, WebAce and Syskill & Webert, provide profile guided query generation and filtering to support their users' long term information interests. WebAce [1] searches the Web for documents likely to be of interest to the user. The agent constructs queries from a user profile, submits them to a search engine, and then filters the results for dissemination to the user. The user profile is built by one of two unsupervised clustering methods on documents judged relevant by the user. Queries are generated from a cluster of documents based on the average term frequency (TF) of the terms contained in the documents of the cluster and the document frequency (DF), i.e., the number of documents in the cluster that contain a specified term. A set of k terms with the highest TF and another set of k terms with the highest DF are selected from the cluster. Document filtering depends on the clustering method used. For their Association Rule Hypergraph Partitioning, documents must exceed a support threshold; Principal Direction Divisive Partitioning discards documents that exceed a scatter value for the closest cluster.

Syskill & Webert [11] is a Web browsing/search assistant. It learns a set of profiles for each user, one for each topic, by soliciting positive/negative judgments for viewed pages. A simple Bayes classifier constructed from a subset of words on pages of interest determines whether particular pages would be interesting to the user. If the user asks the system to "Make Suggestions" of interesting pages, it explores links from an index or constructs queries to search engines. Queries are combinations of informative (words that appear in relevant, but not irrelevant pages) and frequently occurring words.

PersonalSearcher uses a profile of user interests to filter results returned from multiple search engines [7]. The user profile is a complex representation organized in a hierarchy of topics that include variable relevance levels and relatedness information. The richer representation is designed to support a range of services, but at present, supports on-demand searching and filtering.

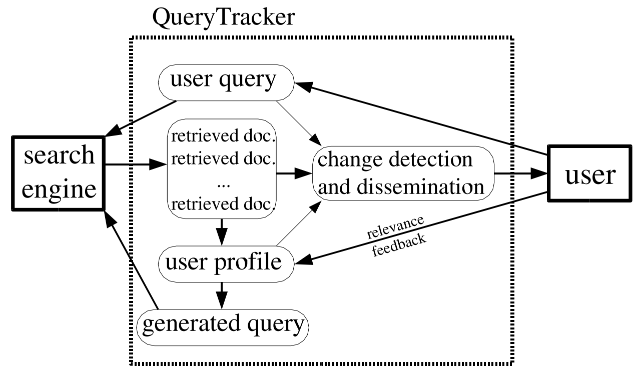

Figure 1. Architecture of QueryTracker

2. QueryTracker

The architecture of QueryTracker is presented in Figure 1. The users register their queries with the agent, which in turn repeatedly submits them to the search engine (every 24 hours). Every time the query is run through the search engine, results are filtered and disseminated back to the user. New results are examined for relevant content, and previously seen ones for relevant changes. The user may give relevance feedback to QueryTracker, by marking disseminated documents as relevant or non-relevant. From this feedback, a user profile is built. The profile can be used to guide the dissemination of subsequent results. Also, based on the user profile, an alternative to the user-supplied query is constructed and submitted to the search engine. The generated query may compensate for a poorly specified original query, either because the user was not able to construct the best query or because the user's need drifts over time.

It was our original design goal to limit user feedback to *relevant* examples. In earlier work, we had found little difference in some aspects of performance (query generation) when supplementing with negative feedback [16]. This, however, would have made it impossible for QueryTracker to distinguish between non-relevant documents and documents the user simply didn't have a chance to read yet (e.g., when many documents were disseminated at once).

2.1. Learning User Profiles

User profiles direct personalized information services. They are typically a compressed representation of documents previously judged to be relevant and sometimes those judged irrelevant as well. The most common representation for profiles based on textual documents is TFIDF vectors [14]. QueryTracker adopts a form of this representation modified for Web use [15].

Over time, topic descriptions are learned from user-supplied examples of relevant documents, which are added to the existing TFIDF vectors in what is known as *relevance feedback* [13]. Associated with each vector is a *dissemination threshold*, which is used when new documents are filtered by the agent: if the similarity between the new document and a profile exceeds the associated dissemination threshold, the document is considered relevant and shown to the user. We found that learning the appropriate dissemination threshold was critical to filtering performance and that one could be learned with relatively little feedback (i.e., 10 relevance judgments) [15]. Section 2.4 describes two methods for determining the dissemination threshold based on a user profile.

TFIDF vectors and their associated dissemination thresholds are known in the Information Retrieval (IR) literature as *filtering queries*. As with some other personalized Web Agents, such as Syskill & Webert, QueryTracker learns multiple topics for each user, one filtering query per interest area.

QueryTracker also maintains term frequency vectors for positive and negative feedback for each topic. These were used to support a second filtering technique based on a naive Bayes classifier.

2.2. Query Generation

QueryTracker supports two mechanisms for querying the search engine: use the original user query or generate a query based on the user profile (filtering query). In earlier work [16], we investigated several alternative methods of generating queries such as:

- select as in WebAce [1],
- select n most frequent terms from profile,
- select the n terms with the highest odd-ratio, which roughly is the log of the probability that the term appears in a document that is relevant divided by the probability that the term appears in a non-relevant document,
- select n terms with highest TFIDF.

Queries were generated based on a profile constructed from a previously judged document collection (TREC-5). Search engine results were compared to the original profile to determine relevance.

We found that the WebAce and TFIDF-based methods exhibited the best performance: good quality of hits and relatively high relevance. We also found that while relevance increased as the query length increased, the number of hits also decreased. This is not surprising as search engines are likely optimized for the most common queries, which tend to be short. Because we found TFIDF useful for the other components in QueryTracker and because it appeared to work slightly better, we employ TFIDF as the query generation technique.

2.3. Change Detection

Looking at a newer version of a document that has already been examined is only worthwhile if the changes made between versions (i.e., the text fragments that have been added to or removed from the original) are themselves relevant with respect to the information need that prompted retrieval of the original document. In other words, we wish to distinguish between changes in advertisements, banner ads, counters, etc. and real, meaningful changes to the content of the document itself.

Originally, we considered using UNIX `diff`, or one of its HTML-specific variants [6, 4] to extract the text of the changes between document versions and then convert it into a TFIDF vector. However, a similar effect can be obtained more directly and efficiently by performing a vector difference between the TFIDF vectors of the new and original versions of the document. The `diff` utility and its variants devote significant effort to finding the *location* in the document where changes occur. We are interested only in detecting the content and relevance of changes.

For this component of QueryTracker, we store the latest copy of each unique URL that was read by the user. Change detection is used on all URLs that have been encountered before. The difference obtained by subtracting the original document vector from the changed version's vector is an aggregate representation of modifications made to the original:

$$V_{\text{diff}} = V_{\text{new}} - V_{\text{orig}}$$

Terms with positive weights in V_{diff} represent content that was added to the original, while terms with negative weights represent text that was deleted from the original. Since we only allow TFIDF vectors with positive term weights (to simplify similarity computations and constrain their results to the interval $[0, 1]$), we express the change as the difference between two positive TFIDF vectors:

$$V_{\text{diff}} = V_{\text{add}} - V_{\text{del}}$$

V_{add} contains only terms of V_{diff} that have positive weights; V_{del} has only those that had negative weights in V_{diff}. We can now compute the relevance of additions and deletions made to the original document independently. This allows for more fine-grained dissemination decisions based not only on the relevance of the changes, but also on their direction (addition vs. deletion).

Based on earlier success with the method, QueryTracker will disseminate a new version of a previously encountered document if the added text (V_{add}) is found relevant with respect to either the original, user-supplied query, or, if available, with respect to the user profile.

2.4. Filtering and Dissemination

At this point, the results of the queries are collected and filtered to determine which should be disseminated. Recognizing that search engines tend to return more than what the user was looking for (e.g., sponsored links, low precision results), this step checks to what extent the search engine understood the information needs of the user. Search engines can return thousands of hits for queries (e.g., Google found 101,000 hits for query number 1 in our experiment); this stage removes those that do not satisfy one of four criteria for relevance.

For each query submitted to the search engine, we retain the first 100 results. New content is extracted from those results, either by establishing that a document is new, or by isolating the additions to the previous version. The four criteria used to determine whether to disseminate each result to the user are: similarity to original query, search engine rank, similarity to a TFIDF profile, and similarity to a Bayesian profile. The first two require no knowledge of the user or learning; the last two require a learned user profile. The four methods are described below. For changed documents, the text added to the new version is used to calculate the first value, as well as the third and fourth, if profiles exist. For new documents, values are calculated for the second and, if a user profile exists, for the last two criteria.

Whenever we use a similarity value and a threshold in our dissemination decision, we disseminate a document with the following probability:

$$dissem\_prob(sim, thr) = \begin{cases} 1, & \text{if } sim > thr \\ 1 - \frac{\sqrt{thr^2 - sim^2}}{thr}, & \text{otherwise} \end{cases}$$

A document is always disseminated if similarity exceeds the threshold. Otherwise, the document is disseminated with a probability that rapidly approaches 0 as similarity decreases. Because we are testing these as alternate methods, a document is disseminated if it passes on at least one criterion. In section 3, we test the efficacy of each of the methods.

2.4.1. Similarity to Original Query For changed documents, the key issue is whether the changes are *relevant*. Look-and-feel or ephemeral changes (e.g., advertising) should not matter. To determine relevance, TFIDF similarity between text that was newly added to the document and the original query supplied by the user is computed. We used a fixed threshold of 0.3 for the $dissem\_prob$ function, which seemed to work well in most cases. This method is the only one available for deciding dissemination of changed documents in the absence of a user profile.

2.4.2. Search Engine Rank Based on the convention that results will be listed by the search engine in what it perceives to be their decreasing relevance, we guarantee dissemination of the first 10 hits, given that they are new, by setting the threshold to be .9. For results ranked 11 and higher, we convert the rank to a pseudo similarity value ($\frac{1-rank}{max\_rank-1}$) and apply the $dissem\_prob$ function. Rank based dissemination only applies to new results and is performed, regardless of the presence or absence of a user profile.

2.4.3. Similarity to TFIDF Profile Once the user starts providing positive feedback, a TFIDF profile is constructed from the words in the relevant documents. We compute the similarity between the profile vector and the new content (whole document if new, or added content for changed document) and use the $dissem\_prob$ function described above to determine whether to disseminate the document. The threshold is learned adaptively while building the TFIDF profile vector (see Section 2.1).

2.4.4. Similarity to a Bayesian Profile In an early version, we requested only positive feedback, which made it difficult to determine whether a user had already seen and was disregarding a document. Once both positive and negative feedback was available to QueryTracker, we decided to take advantage of it and test an additional method for deciding document dissemination: a Naive Bayes Classifier (NBC) [10]. Probabilities of being in each of the two categories (relevant or not relevant) are estimated from the two term frequency vectors for the topic. A document is disseminated if its conditional probability of belonging to the relevant class is higher than the conditional probability of being in the non-relevant class. Pazzani and Billsus (1997) found a simple Bayes classifier to be quite effective for filtering in their Syskill & Webert system.

3. Experiment

To test the success of the different techniques (especially query generation and filtering) when working together for the agent, we performed a user study in which users were asked to submit and track queries for approximately one month. The users themselves defined their queries and made relevance judgments based on what *they* meant by it. We collected data from four people from our computer science department on ten queries, as follows:

1. educational+software+preschool+children+review
2. information+agent+filtering+tracking+text
3. text+image+information+retrieval+multimedia

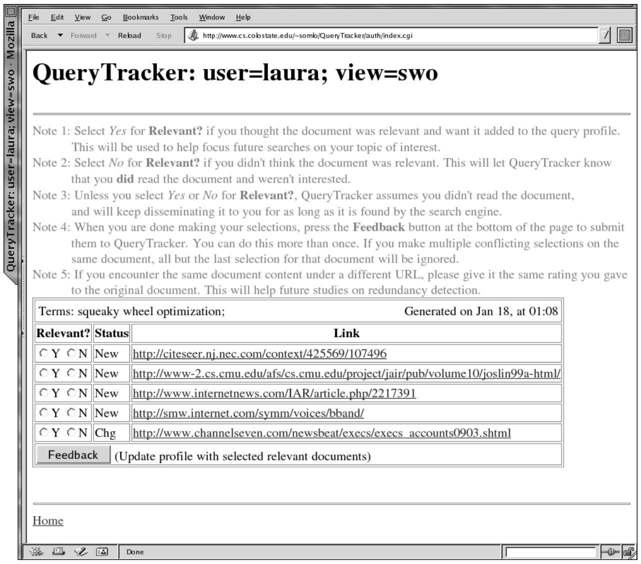

Figure 2. View of disseminated documents and feedback input form

4. oversubscribed+scheduling+AI+OR+algorithm
5. kid+friendly+resort+vacation+beach
6. scheduling+genetic+algorithms
7. squeaky+wheel+optimization
8. beveridge+electrostatic+speakers+audio
9. linux+dell+axim
10. music+piracy+riaa+campus+education

The topics covered a range of interests from professional to personal.

The queries were submitted to Google on a daily basis. Before the users supplied any feedback to QueryTracker, only the original queries were submitted, and dissemination was decided based on document rank (new documents) or similarity between added content and the user-supplied query (previously seen documents).

Users could access the QueryTracker Web site to monitor their queries at their convenience. They were not required to judge their queries daily. When the users accessed their results, they were asked to provide feedback on the disseminated documents by marking them either relevant or non-relevant. When documents were not marked one way or the other, it was assumed that users had not read them yet, and they continued to be treated as *new* rather than *previously seen* the next time they were returned by the search engine (usually the following day when the query was submitted). An example of the form used to solicit user feedback is shown in Figure 2.

As soon as one or more documents were supplied as positive feedback, a TFIDF profile was built based on them. This profile was used for two purposes. First, an alternative query was generated by selecting its top four terms by weight (see Section 2.2). This query was submitted to the search engine in addition to the original user query. Second, the TFIDF profile vector was used in the dissemination decision by comparing it to new text. By *new text*, we mean either the full text of a new document or the content added to a previously seen one.

As soon as both positive and negative feedback were available, the naive Bayes classifier was built for the topic and used to decide dissemination in addition to all the above-mentioned methods.

For each disseminated document, we recorded the type of query that led to its retrieval (**orig** for original query, **gen** for alternative query generated from TFIDF profile) and all methods that voted to disseminate the document (**rank** if the new document's rank was sufficiently high, **query** if changes from the previous version were similar to the original user query, **tfidf** if the new text was similar to the TFIDF profile, and **NBC** if the Bayes classifier determined the new text as relevant). New text was checked using all applicable methods and disseminated if at least one of them voted to disseminate.

For each query type and method described above, we computed the following values:

RF number of relevant documents disseminated via the query type or dissemination method in question,

NF number of non-relevant documents disseminated (false positives),

pseudo RM number of relevant documents missed by the query type or method in question, that were found by some other method (false negatives).

Based on these numbers, we computed precision and (pseudo-)recall as:

$$\text{precision} = \frac{RF}{RF + NF}$$

$$\text{recall} = \frac{RF}{RF + RM}$$

Please note that mathematically sound RM and recall cannot be computed without complete knowledge of (and random access to) the entirety of the documents indexed by the search engine. The values we compute using available data are, however, sufficient to compare our query generation and dissemination methods *to each other*.

Since recall and precision are usually inversely proportional to each other, we use a simplified version of the F-metric [9] to capture both equally. This metric will be referred to as LGF, in order to avoid confusion with ANOVA's F-value.

$$LGF = \frac{2 \cdot \text{recall} \cdot \text{precision}}{\text{recall} + \text{precision}}$$

					Query Method						Dissemination Method											
	Total Counts				Original			Generated			Rank			TFIDF			NBC			Query/Change		
q#	days	rel	¬rel	%	Rec	Prec	LGF	Rec	Prec	LGF	Rec	Prec	LGF	Rec	Prec	LGF	Rec	Prec	LGF	Rec	Prec	LGF
1	10	39	98	0.28	0.80	0.54	0.65	0.23	0.11	0.15	0.44	0.26	0.32	0.10	0.33	0.16	0.72	0.32	0.44	0.00	0.00	0.00
2	9	41	54	0.43	0.49	0.44	0.46	0.58	0.45	0.51	0.58	0.49	0.53	0.12	0.46	0.19	0.73	0.41	0.53	0.02	1.00	0.05
3	10	39	62	0.39	0.41	0.36	0.39	0.59	0.40	0.48	0.72	0.38	0.50	0.31	0.52	0.39	0.46	0.44	0.45	0.03	0.50	0.05
4	10	25	65	0.28	0.76	0.40	0.52	0.28	0.16	0.21	0.76	0.28	0.41	0.04	0.17	0.06	0.52	0.30	0.38	0.00	0.00	0.00
5	15	72	149	0.32	0.69	0.58	0.63	0.35	0.18	0.24	0.42	0.26	0.32	0.04	0.17	0.07	0.82	0.42	0.56	0.03	0.67	0.05
6	14	198	86	0.70	0.50	0.81	0.61	0.59	0.64	0.61	0.52	0.68	0.59	0.07	0.77	0.12	0.54	0.74	0.63	0.02	0.44	0.04
7	16	76	161	0.32	1.00	0.54	0.70	0.0	0.0	0.00	0.46	0.39	0.42	0.03	0.20	0.05	0.62	0.28	0.38	0.00	0.00	0.00
8	5	51	67	0.43	0.92	0.84	0.88	0.08	0.06	0.07	0.59	0.48	0.53	0.04	0.17	0.06	0.65	0.43	0.52	0.00	0.00	0.00
9	21	39	286	0.12	0.95	0.29	0.44	0.05	0.01	0.02	0.74	0.12	0.21	0.03	0.04	0.03	0.38	0.17	0.23	0.08	0.23	0.12
10	23	343	72	0.83	0.38	0.89	0.54	0.62	0.79	0.70	0.36	0.80	0.50	0.08	0.94	0.16	0.88	0.83	0.85	0.01	1.00	0.01

Table 1. Final counts and LGF performance metrics for tracked queries

Table 1 summarizes the results obtained for each of the ten queries monitored in this study. Under **total counts**, we show the number of days during which the query received user feedback, the total number of relevant and non-relevant documents that were gathered and the percentage of rated results that were relevant. Under **Query Method**, we present LGF values obtained for each of the two query generation methods (summarized across the dissemination methods). Finally, under **Dissemination**, we show LGF values obtained for each individual dissemination method (summarized across the query generation methods).

3.1. Should the original query be modified?

We wanted to assess whether the generated query was helping. As Table 1 shows, generated queries appear to be producing some links not produced by the original query. But the data suggests that in most cases (6 out of 10), the overall performance is lower than for the original query. In fact, a paired-sample t-test comparing LGF for the original query with that of the generated query produced a significant difference ($t = 2.62$, $p < 0.028$). The mean and standard deviation for the original query are 0.58 and 0.14. The mean and standard deviation for the generated query are 0.30 and 0.26.

The analysis suggests that generated queries perform worse than original ones. However, in four of the cases, the performance obtained using a generated query can match and even exceed the performance from the original query. For those queries on which the generated query excels (2, 3 and 10), it tends to do so because of improved recall. These queries also tend to have more relevant documents. One hypothesis is that over time, as more positive feedback is accumulated, the performance of the generated query improves. Using query #10 as an example (it has the most relevant feedback), we plot the LGF performance of each query method after each day when feedback was received from the query's user in Figure 3. Indeed, after 5 days of activity,

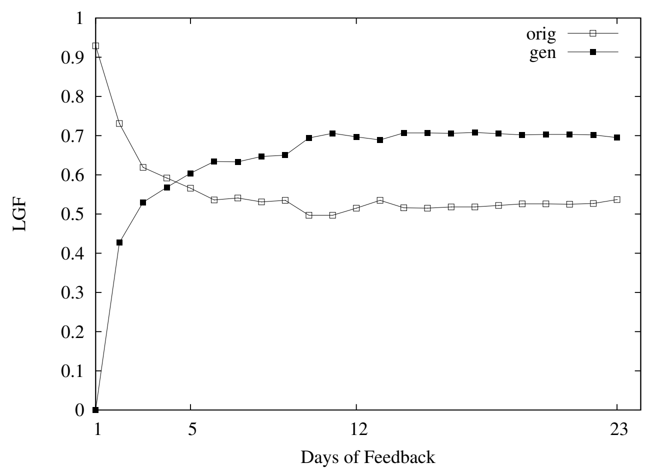

Figure 3. LGF performance of original query vs. generated alternative plotted over time for query #10

the performance of the generated query exceeds that of the user-supplied original. In other cases, the performance of the generated query also improves with feedback, but never manages to outperform the original. In looking at individual queries, it appears that narrowly focused queries (e.g., 7 and 8) cannot easily be improved, although, even for those, the new query does find some additional relevant links.

In conclusion, once a reliable TFIDF user profile has been learned, the automatically generated query built from its top-weighted terms can become quite useful. Also, there is very little overlap between relevant documents found via the original query and those resulting from the generated alternative. The latter is therefore useful at least because it extends the pool of candidates from which the filtering algorithm can disseminate potential relevant documents.

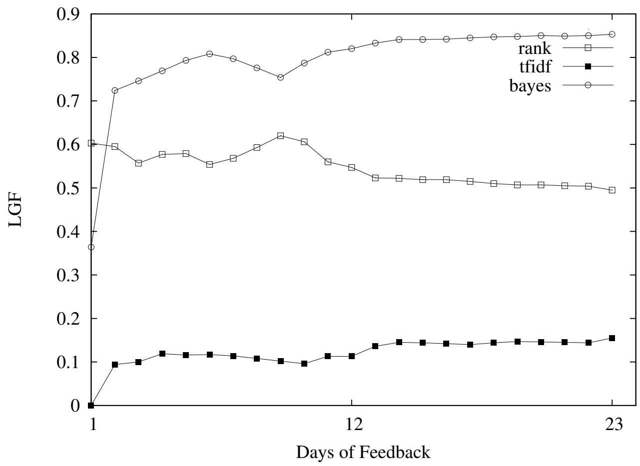

Figure 4. LGF performance comparison between dissemination methods over time for query #10

3.2. Which dissemination method is best?

The next key issue is filtering: which methods appear to be effective. Table 1 shows the recall, precision and LGF for the dissemination methods. We included results for **Query**, although it is only used as the default change relevance detection mechanism in the absence of a user profile. The data shows that it is a conservative filter providing high precision and low recall. We do not include it in our further evaluation, since we had relatively few changed documents (reflected in the 0.00 results), and most were detected using one of the two profile-based methods (**TFIDF** or **NBC**).

A one-way ANOVA comparing LGF for the three dissemination methods (**Rank**, **TFIDF**, **NBC**) produced a significant difference ($F = 22.10$, $p < 0.001$). Means and standard deviations were: 0.43 and 0.12 for **Rank**, 0.13 and 0.11 for **TFIDF**, and 0.50 and 0.17 for **NBC**. On LGF, **NBC** clearly dominates **TFIDF**, but **Rank** does only slightly worse.

All three methods achieve comparable precision. Surprisingly, **TFIDF** has a rather low recall as compared to the other two methods. This is likely because **TFIDF** has a slower learning process than NBC – which also benefits from the negative feedback that our TFIDF profile does not use. Performance over time for all three methods is illustrated using query #10 in Figure 4. It can be observed how **NBC** learning is quite fast, **Rank** performance is relatively flat, and **TFIDF** improves, albeit slowly.

To complete the picture, we searched for an interaction effect between dissemination method and the query that was submitted (original or generated). Running a two-way ANOVA with LGF as dependent variable, and query and

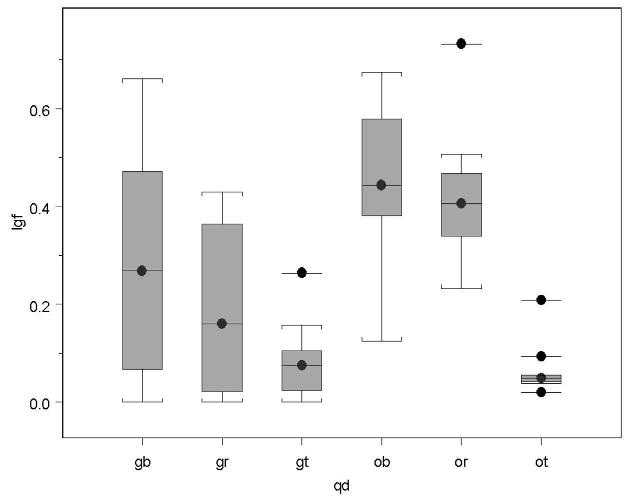

Figure 5. Boxplot of LGF performance for the six combinations of dissemination and query generation methods

dissemination methods as the parameters, we obtained $F = 10.11, p < 0.003$ for the query method, $F = 19.95, p < 0.001$ for the dissemination method, and $F = 3.22, p = 0.05$ for the interaction effect between query and dissemination methods. Based on the summary data (boxplot of the six conditions is shown in Figure 5), apparently, TFIDF dissemination is largely unaffected by the query generation, whereas Rank dissemination is boosted more, relatively speaking, for the original query than for the generated one.

4. Conclusions and Future Work

We designed, implemented and evaluated a user agent, QueryTracker, that learns about the information needs of users via relevance feedback and keeps track of them daily using a search engine. QueryTracker provides several services: automatic query submission, query generation, relevant change detection and filtering of results. We evaluated the agent in a user study to assess how query generation and several filtering methods perform relative to each other.

Query generation based on the learned profile has good potential; the technique led to relevant results that had not been obtained by the original query. However, it appears to require significant amounts of feedback before becoming effective, especially for more precise queries.

On dissemination, we found that TFIDF is rather slow-learning when compared to NBC and is often outperformed by the search engine's ranking. We think the main reason for this is that NBC has the advantage of using negative feedback to significantly speed up learning.

From the experimental results, it appears that some of the methods need more judgments to perform well. In talking to users, they expressed some frustration in not obtaining good results quickly for precise queries. To address these concerns, for future work, we will look at jump-starting the process by allowing users to submit known relevant URLs with their original queries. These documents will be used to seed the profile.

Additionally, anecdotal evidence from users suggests that some information needs may be more precise than others. For those, the original query may be the best approach, and additional filtering may be desirable. We will expand the study to determine whether these situations can be identified automatically and methods adjusted accordingly.

5. Acknowledgments

The authors wish to thank Evelyn Mitchell for suggesting *QueryTracker* as the name of our system. We also wish to thank the anonymous reviewers for their careful comments on the submitted version. Adele Howe was supported by the National Science Foundation under Grant No. IIS-0138690. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] D. Boley, M. Gini, R. Gross, E. Han, K. Hastings, G. Karypis, V. Kumar, M. Bamshad, and J. Moore. Document categorization and query generation on the world wide web using webace. *AI Review*, 13(5-6):365–391, 1999.

[2] K. Bollacker, S. Lawrence, and C. Giles. A systems for automatic personalized tracking of scientific literature on the web. In *Proceedings of the Fourth ACM Conference on Digital Libraries*, pages 105–113, Berkeley, CA, USA, 1999.

[3] BotSpot. Tracking Bots. www.botspot.com/BOTSPOT/Windows/Tracking_Bots/index.html.

[4] Y. Chen, F. Douglis, H. Huang, and K. Vo. Topblend: An efficient implementation of htmldiff in java. In *Proceedings of the World Conference on the WWW and Internet (WebNet 2000)*, San Antonio, TX, USA, 2000.

[5] I. Copernic Technologies. Copernic agent professional. www.copernic.com/en/products/agent/professional.html.

[6] F. Douglis, T. Ball, Y. Chen, and E. Koutsofios. The at&t internet difference engine: Tracking and viewing changes on the web. *World Wide Web*, 1(1):27–44, 1998.

[7] D. Godoy and A. Amandi. A user profiling architecture for textual-bsed agents. In *Proceedings of the Fourth Argentine Sympoisum on Artificial Intelligence*, Sante Fe, Argentina, Sept. 2002.

[8] B. Jansen, A. Spink, and T. Saracevic. Real life, real users, and real needs: a study and analysis of user queries on the web. *Information Processing and Management*, 36(2):207–227, 2000.

[9] D. Lewis and W. Gale. A sequential algorithm for training text classifiers. In *Proceedings of the 17th International ACM-SIGIR Conference on Research and Development in Information Retrieval*, Dublin, Ireland, 1994.

[10] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[11] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill & webert: Identifying interesting web sites. In *Proceedings of the 1996 National Conference on Artificial Intelligence (AAAI-96)*, Portland, OR, USA, 1996.

[12] P. I. Project. Search engines: A pew internet project data memo. http://pewinternet.org/reports/toc.asp?Report=64, Jul. 2002.

[13] J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall, 1971.

[14] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1988.

[15] G. Somlo and A. Howe. Adaptive lightweight text filtering. In *Proceedings of the Fourth International Symposium on Intelligent Data Analysis (IDA 2001)*, Lisbon, Portugal, 2001.

[16] G. Somlo and A. Howe. Using web helper agent profiles in query generation. In *Proceedings of the Second International Joing Conference on Autonomous Agents and Multi Agent Systems*, Melbourne, Australia, 2003.