tapraid5/8n-jasis/8n-jasis/8n-orig/8n0754-01a | heckt | S=8 | 4/2/01 | 14:56 | Art: JA-1645 | Input-tam(tam)

FEATURES: Real-Time Adaptive Feature and Document Learning for Web Search

Zhixiang Chen, Xiannong Meng, and Richard H. Fowler
Department of Computer Science, University of Texas–Pan American, 1201 West University Drive, Edinburg, TX 78539-2999. E-mail: chen@cs.panam.edu, meng@cs.panam.edu, fowler@panam.edu

Binhai Zhu
Department of Computer Science, Montana State University, Bozeman, MT 59717. E-mail: bhz@cs.montana.edu

In this article we report our research on building FEATURES—an intelligent web search engine that is able to perform real-time adaptive feature (i.e., keyword) and document learning. Not only does FEATURES learn from the user's document relevance feedback, but it also automatically extracts and suggests indexing keywords relevant to a search query and learns from the user's keyword relevance feedback so that it is able to speed up its search process and to enhance its search performance. We design two efficient and mutual-benefiting learning algorithms that work concurrently, one for feature learning and the other for document learning. FEATURES employs these algorithms together with an internal index database and a real-time meta-searcher to perform adaptive real-time learning to find desired documents with as little relevance feedback from the user as possible. The architecture and performance of FEATURES are also discussed.

1. Introduction

As the World Wide Web rapidly evolves and grows, web search has come to provide an interface between the human users and the vast information on the web in people's daily life. There have been a number of popular and successful general-purpose or meta search engines such as AltaVista [a], Yahoo! [b], Google [c], MetaCrawler [d], Dogpile [e], and Inference Find [f]. Many of the existing engines support personalization (or customization) with the help of predefined user profiles or a collection of customizable parameters such as suggestions about keywords to include or exclude, language choices, document locations, etc. These functions can help a search engine find more relevant documents for the user. User profiles are often automatically created by means of *cookies*, client-side *digital traces* or tracking of user's browsing patterns (Bollacker, Lawrence, & Giles, 1999; Widyantoro, Ioerger, & Yu, 1999; Meng & Chen, 1999), or manually created by users themselves. Profiles can be used either at the server side or client side. In most cases, the collection of customizable parameters is fixed. In other cases, some of those parameters can be automatically generated by tracking the recent browsing processes of a user. For example, a search engine can compile a list of suggested keywords based on its internal ranking and the user's most recent browsing contents for use in future search. In essence, the nature of the personalization (or customization) are *static* in the sense that it is defined before a search process and is not able to support real-time adaptive learning from the user's relevance feedback through interactive refinements.

One approach for the next generation of intelligent search engines is that they be built on top of existing search engine design and implementation techniques. They may be built by integrating intelligent components with one general-purpose search engine or with a collection of general-purpose search engines through meta-searching. An intelligent search engine would use the search results of the general-purpose search engines as its starting search space, from which it would adaptively learn from the user's feedback to boost and to enhance the search performance and the relevance accuracy. It may use feature extraction, document clustering and filtering, and other methods to help an adaptive learning process. Recent research on web communities (Kleinberg, 1999; Gibson, Kleinberg, & Raghavan, 1998; Chakrabarti et al., 1998) has used a short list of web pages returned by a search engine as a starting set for further expansion of search. There has been considerable effort applying machine learning to web search-related applications, e.g., scientific article locating and user profiling (Bol-

Received August 30, 2000; revised January 25, 2001; accepted January 25, 2001

© 2001 John Wiley & Sons, Inc.

Author Proof

lacker, Lawrence, & Giles, 1998; 1999; Lawrence, Bollacker, & Giles, 1999), focused crawling (Rennie & McCallum, 1999), collaborative filtering (Nakamura & Abe, 1998; Billsus & Pazzani, 1998), and user preference boosting (Freund, Iyer, Schapire, & Singer, 1998). An adaptive real-time search algorithm without an index, which is basically a focused search starting at some given url and crawling within some neighboring documents, is given in Ikeji and Fotouhi (1999).

FEATURES is part of our research on building an intelligent search engine (Chen, Meng, Zhu, & Fowler, 2000; Chen & Meng, 2000). In this article, document features are limited to keywords that are used to index documents, but our approach may be applied to other cases of document features. Given a search engine S, we use two new concepts, dynamic features and dynamic vector space, to explore the search result $R(q, u)$ returned by S for any query q and any user u. Our strategy is that we use the dynamic features that are relevant to the query q to map the whole document search space to a substantially smaller subspace, the dynamic vector space, that is relevant to the query. We present a feature-learning algorithm that extracts a small set of the dynamic features, i.e., the most relevant indexing keywords to the search query at the moment, and suggests those keywords to the user for him/her to judge whether they are indeed relevant or not. The feature-learning algorithm works concurrently with the document-learning algorithm operating on relevance judgments to retrieve relevant documents for the user. Both learning algorithms help each other to speed up the search process and enhance search performance. An internal index database is used in which each document is indexed using about 300 keywords. We also design and implement a meta-searcher for FEATURES through real-time meta-searching, parsing, and indexing. FEATURES uses an internal index database, a real-time meta-searcher, and learning algorithms to perform real-time adaptive learning for web search to retrieve relevant documents requiring the least possible feedback.

The rest of this article is organized as follows. In section 2, we discuss the necessity of adaptive learning from relevance feedback for web search. In section 3, we introduce two new concepts, the dynamic features and the dynamic vector space, which are relevant to a search query. In section 4, we present the feature-learning and the document-learning algorithms and strategies for feature ranking and document ranking as well as a method for simulating equivalence queries. In section 5, we explain the design and implementation of FEATURES. The performance of FEATURES is also discussed. We conclude the article in section 6.

2. Should We Learn From Relevance Feedback or Should We Not?

The *static* nature of search engines can be understood formally as follows. Let $\mathcal{W}$ denote the collection of web documents, $\mathcal{T}$ denote the set of time values, $\mathcal{Q}$ the set of all possible queries, and $\mathcal{U}$ the set of all users. Mathematically, a search engine S can be understood as a map f_S as follows:

$$f_S: \mathcal{Q} \times \mathcal{U} \times \mathcal{T} \Rightarrow 2^{\mathcal{W}}.$$

That is, given any query $q \in \mathcal{Q}$, any user $u \in \mathcal{U}$, and any time $t \in \mathcal{T}$, $f_S(q, u, t)$ is a subset of documents in $\mathcal{W}$. We say that a search engine S is static, if

$$f_S(q, u, t) = f_S(q, u', t')$$

for any $q \in \mathcal{Q}$, $u, u' \in \mathcal{U}$, and $t, t' \in \mathcal{T}$ with $|t - t'| \leq B$ for some constant B. As far as we understand, the existing general-purpose search engines (or meta-search engines) are static, because within certain time intervals (say, intervals between updates of the index database or search strategies), the search result of the engine is dependent on the query only.

The static nature of the existing search engines makes it very difficult, if not impossible, to support the *dynamic* changes of the user's search interests. The following scenario is not typical. Let us suppose that X is a computer scientist and his daily work and web search are all computer science-oriented. However, one day X wanted to search for several good geometry reference books for his child in high school to read. All his personalization (or customization) is heavily computer science-oriented and thus provides no help but misleading search directions. Turning off the augmented features of personalization (or customization) is a simple and nice trick. This time X received *high school*-related pages and *geometry*-related pages. Sadly, the high school-related pages are not related to high school geometry and the geometry related pages are oriented for computer science professionals, because the ranking and search strategy of the search engine is for general purpose. AQ: 1

The augmented features of personalization (or customization) certainly help a search engine to increase its search performance; however, their ability is very limited. An intelligent search engine should be built on top of existing search engine design and implementation techniques. It should use the search result of the general-purpose search engines as its starting search space, from which it would adaptively learn in real-time from the user's relevance feedback to boost and enhance search performance and relevance accuracy. With the ability to perform real-time adaptive learning from relevance feedback, the search engine is able to learn a user's search interest changes or shifts, and thus provides the user with improved search results.

Relevance feedback is the most popular query reformation method in information retrieval (Salton, Wong, & Yang, 1975; Baeza-Yates & Ribeiro-Neto, 1999). It is essentially an adaptive learning process from the document examples judged by the user as relevant or irrelevant. It requires a sequence of iterations of relevance feedback to search for the desired documents. As it is known through the research of Salton, Wong, & Yang (1975 and Salton (1971),

Author Proof

a single iteration of similarity-based relevance feedback usually produces improvements from 40% to 60% in the search precision, evaluated at certain fixed levels of the recall and averaged over a number of user queries.

Some people have the opinion that web search users are not willing to try iterations of relevance feedback to search for their desired documents. However, the authors think otherwise. It is not a question whether the web search users are not willing to try iterations of relevance feedback to perform their search. It is a question whether we can build an adaptive learning system that supports high search precision increase with several iterations of relevance feedback. The web search users may have no patience to try more than a couple of dozens of iterations of relevance feedback. But, if a system has a 20% or so search precision increase with about five or fewer iterations of relevance feedback, are the users willing to use such a system? The authors believe that the answer will be "yes."

As matter of fact, the authors conducted a survey about the answers to the above question among a group of 62 students in the summer of 2000, and 96% answered "yes." The survey was conducted as follows. We selected the students in two summer classes as the targeted survey objects. We explained the purpose of the survey. We also explained the concepts of relevance feedback and precision and recall as well. To make sure that the students understood those concepts we demonstrated several examples of the real-time search with relevance feedback using the working prototype of WebSail (Chen et al., 2000) that we have implemented. We then asked the students the following questions: *If a system has a 20% or so search precision increase with one iteration of relevance feedback, do you want to use such a system?* The same question was repeated with the number of iterations being 2, 3, . . . , 10, and greater than 10, respectively. We counted how many students answered "yes" and computed the average values. We found that 96% of the students would like to use such a system with five or fewer iterations of relevance feedback if a 20% or so search precision may be achieved. We used the 20% precision increase in our survey because we believe that this increase is reasonable and can be achieved practically.

It is certainly not trivial at all to implement adaptive relevance feedback for web search. Because an adaptive learning process needs to keep the history of the learning, the search engine must use a separate thread for each individual search request and must maintain the thread until the search ends. A search thread usually needs to have its own working space and is very time and space consuming. Imagine that a general-purpose search engine needs to process hundreds or thousands of search requests in every minute. It would be too difficult to maintain so many threads in every single minute and to answer each search request very efficiently. Other difficulties include designing innovative real-time adaptive learning algorithms with as little relevance feedback from the user as possible.

3. Dynamic Features Versus Dynamic Vector Space

In spite of the World Wide Web's size and the high dimensionality of web document indexing features, the traditional vector space model in information retrieval (Salton et al., 1975; Salton, 1989; Baeza-Yates & Ribeiro-Neto, 1999) has been used for web document representation and search. However, to implement real-time adaptive learning with limited computing resource, here, at an Ultra™ One Sun workstation, we cannot apply the traditional vector space model directly. Recall that back in 1998, the AltaVista system was running on 20 multiprocessor machines, all of them having more than 130 Giga-Bytes of RAM and over 500 Giga-Bytes of disk space (Baeza-Yates & Ribeiro-Neto, 1999). We need a new model that is efficient enough both in time and space for FEATURES and other web search implementations with limited computing resources. The new model may also be used to enhance the computing performance of a web search system even if enough computing resources are available.

We now examine indexing in web search. Again, in this article, we use keywords as document indexing features. Let X denote the set of all indexing keywords for the whole web (or, practically, a portion of the whole web). Given any web document d, let $I(d)$ denote the set of all indexing keywords in X that are used to index d with nonzero values. Then we have the following two properties:

1. The size of $I(d)$ is substantially smaller than the size of X. Practically, $I(d)$ can be bounded by a constant. The rationale behind this is that in the simplest case we do not need to use all the keywords in d to index it. In our implementation of FEATURES, we use about 300 keywords to index documents in its internal database and at most 64 automatically generated keywords to index a document retrieved through meta-search.
2. For any search process related to the search query q, let $D(q)$ denote the collection of all the documents that match q, then the set of indexing keywords relevant to q, denoted by $F(q)$, is

$$F(q) = \bigcup_{d \in D(q)} I(d).$$

Although the size of $F(q)$ varies from different queries, it is still substantially smaller than the size of X, and might be bounded by a few hundreds or a few thousands in practice.

Definition 3.1.

Given any search query q, we define $F(q)$, which is given in (b) above, as the set of dynamic features relevant to the search query q.

Definition 3.2.

Given any search query q, the dynamic vector space $V(q)$ relevant to q is defined as the vector space that is

Author Proof

constructed with all the documents in $D(q)$ (as given in (b) above) such that each of those document is indexed by the dynamic features in $F(q)$.

For any query q, FEATURES first finds the set of documents $D(q)$ that match the query q. It finds $D(q)$ with the help of a general-purpose search strategy through searching its internal database, or through meta-searching AltaVista [a] when no matches are found within its internal database. It then finds the set of dynamic features $F(q)$, and later constructs the dynamic vector space $V(q)$. Once $D(q)$, $F(q)$, and $V(q)$ have been found, FEATURES starts its adaptive learning process from the user's document and feature relevance feedback. More precisely, it uses two learning algorithms in parallel to achieve its goal of learning: One algorithm adaptively learns from the documents judged by the user as relevant or irrelevant examples; the other extracts and suggests a small set of keywords that are most relevant to the search query up to the moment, and learns from the keywords judged by the user as relevant or irrelevant. The feature-learning algorithm helps the document-learning algorithm to increase the ranks of relevant documents, while the document-learning algorithm helps the feature-learning algorithm to increase the ranks of the relevant features.

Example 3.3.

We now give examples to show the significance of our approach of dynamic features and dynamic vector spaces. Here we use an example query "Zhixiang Chen" because it is related to a big collection of web pages due to the popularity of the family name "Chen," and of course, it is related to the web pages of one of authors. On January 12, 2001, we queried the search engine AltaVista with the example query and found out that there are 247,371 related pages. If we follow the traditional vector space approach, then the number of terms (or keywords) used to index those pages would be huge. To get some idea about this number, we simply counted the number of keywords that occurred in all Zhixiang Chen's 267 web pages that reside at the web server of the Computer Science Department, the University of Texas–Pan American. We found 12,347 keywords. When those 247,371 pages are considered, it is obvious that the number of keywords occurred in those pages is much bigger than 12,347. That is, any of those pages would be indexed with many more than 12,347 keywords, or in other words, the dimensionality of the vector space for those pages is much bigger than 12,347.

Now, let us consider dynamic indexing features and dynamic vector space. The top four web pages related to the query "Zhixiang Chen" were

d_1 = www.cs.panam.edu/chen/researchProfile.html
d_2 = cs-people.bu.edu/zchen
d_3 = www.uidaho.edu/micro-biology/faculty/chen.html
d_4 = sunsite.informatik.rwth-aachen.de/dblp/db/indices/a-tree/Chen:Zhixiang.html

The dynamic indexing features that were automatically generated by our system for those four pages are

$I(d_1)$ = {algorithm, chen, class, computing, data, edu, field, information, learning, machine, mining, network, neural, new, panam, paper, profile, project, recently, research, retrieval, search, seek, strategies, visualization, web, working, www, zhixiang}
$I(d_2)$ = {american, chen, class, computer, department, edinburg, edu, find, pan, panam, people, profile, research, science, texas, university, usa, zchen, zhixiang}
$I(d_3)$ = {acid, action, biology, chen, class, defense, faculty, home, interest, mechanism, micro, page, plant, regulation, research, response, salicylic, transcriptional, uidaho, www, zhixiang}
$I(d_4)$ = {ameur, bibliography, chen, class, dblp, foued, home, hpsearch, informatik, list, page, publication, rwth, search, server, sunsite, zhixiang}

The set of dynamic features for those four web pages is

F("Zhixiang Chen") = $I(d_1) + I(d_2) + I(d_3) + I(d_4)$ = {acid, action, algorithm, american, ameur, bibliography, biology, chen, class, computer, computing, data, dblp, department, defense, edinburg, edu, faculty, field, find, foued, home, hpsearch, informatik, information, interest, learning, list, machine, mechanism, micro, mining, networks, neural, new, page, pan, panam, paper, people, plant, profile, projects, publication, recently, regulation, research, response, retrieval, rwth, salicylic, science, search, seek, server, strategies, sunsite, texas, transcriptional, university, usa, uidaho, visualization, web, working, www, zchen, zhixiang}

We should point out that the strategy we used to find dynamic features is based on parsing out keywords in the head, title, and the first few sentences of the web pages. A variety of other strategies may be designed, and we plan to study those in the future research. The point is that the number of dynamic features is much smaller than the number of traditional indexing features. Hence, the approach of dynamic features and the vector space is practically feasible for real-time adaptive learning in web search.

4. Feature Learning and Document Learning

As we have investigated (Chen, Meng, & Fowler, 1999; Chen et al., 2000; Chen & Meng, 2000), intelligent web search can be modeled approximately as an adaptive learning process such as online learning (Angluin, 1987; Littlestone, 1988), where the search engine acts as a learner and the user as a teacher. The user sends a query to the engine, the engine uses the query to search the index database, and returns a list of document urls that are ranked according to a ranking function. Then, the user provides relevance feedback, and the engine uses the feedback to improve its next search and returns a refined list of document urls. The learning (or search) process ends when the engine finds the desired documents for the user. Conceptually, a query en-

Author Proof

tered by the user can be understood as the logical expression of the collection of the documents the user wants. A list of document urls returned by the engine can be interpreted as an approximation to the collection of the desired documents.

Rocchio's similarity-based relevance feedback algorithm, one of the most popular query reformation methods of information retrieval (Rocchio, 1971; Ide, 1971; Salton, 1989; Baeza-Yates & Ribeiro-Neto, 1999), is in essence adaptive supervised learning from examples (Salton & Buckley, 1990; Lewis, 1991). We showed (Chen & Zhu, 2000) that for any of the four typical similarity measures (inner product, cosine coefficient, dice coefficient, and Jaccard coefficient) listed in Salton (1989), Rocchio's similarity-based relevance feedback algorithm has a worst-case lower bound that is at least linear in the dimensionality of the Boolean vector space. More precisely, we showed that in the n-dimensional Boolean vector space model, if the initial query vector is $\mathbf{0}$, then when any of the four typical similarity measures (inner product, dice coefficient, cosine coefficient, and Jaccard coefficient) is used, Rocchio's similarity-based relevance feedback algorithm makes at least n mistakes when used to search for a collection of documents represented by a monotone disjunction of at most k relevant features (or indexing terms). When an arbitrary Boolean initial query vector is used, it makes at least $(n + k - 3)/2$ mistakes to search for the same collection of documents. Our linear lower bound holds for arbitrary classification threshold and updating coefficients used at each step of the algorithm. Because the linear lower bound was proved based on the worst case analysis, they *may not* affect the effective applicability of the similarity-based relevance feedback algorithm. On the other hand, the lower bound helps us understand the algorithm well so that we may find new strategies to improve its performance or design new learning algorithms with better performance.

4.1. The General Setting of Learning

For each particular search query q, with the help of certain general-purpose search strategies, FEATURES first finds the three sets; the general matching document set $D(q)$, the dynamic feature set $F(q)$, and the dynamic vector space $V(q)$. Let $F(q) = \{K_1, \ldots, K_n\}$ such that each K_i denotes a dynamic feature (i.e., an indexing keyword). The two learning algorithms of FEATURES maintain a common weight vector $\mathbf{w} = (w_1, \ldots, w_n)$ for dynamic features in $F(q)$. The components of $\mathbf{w}$ have non-negative real values. The feature-learning algorithm uses $\mathbf{w}$ to extract and learn the most relevant features. The document-learning algorithm also uses $\mathbf{w}$ to classify documents in $D(q)$ as relevant or irrelevant.

One should note that during the learning process both learning algorithms update the common weight vector $\mathbf{w}$ concurrently. Of course, both algorithms need to be equipped with efficient ranking functions. Moreover, we also need to provide a good strategy to simulate the equivalence query for the document-learning algorithm, because the user in reality cannot serve as a *real* teacher as modeled in online learning.

Example 4.1.1. We continue example 3.3 with the example query "Zhixiang Chen." For simplicity let us consider the top four pages returned by AltaVista on January 12, 2001. Please recall that AltaVista found a total of 247,371 relevant pages at the time. According to example 3.3, the set of dynamic features for those four web pages is

F("Zhixiang Chen") $= I(d_1) + I(d_2) + I(d_3) + I(d_4) =$ {acid, action, algorithm, american, ameur, bibliography, biology, chen, class, computer, computing, data, dblp, department, defense, edinburg, edu, faculty, field, find, foued, home, hpsearch, informatik, information, interest, learning, list, machine, mechanism, micro, mining, networks, neural, new, page, pan, panam, paper, people, plant, profile, projects, publication, recently, regulation, research, response, retrieval, rwth, salicylic, science, search, seek, server, strategies, sunsite, texas, transcriptional, university, usa, uidaho, visualization, web, working, www, zchen, zhixiang}

This means that we need to use a common weight vector $\mathbf{w} = (w_1, w_2, \ldots, w_{64})$ for computing the weights of all the 68 dynamic features. For example, w_1 represents the weight of "acid," and w_{44} represents the weight of "publication." Remember that the features are sorted in lexicographical order.

4.2. The Feature Learning Algorithm FEX (Feature EXtraction)

For any dynamic feature $K_i \in F(q)$ with $1 \leq i \leq n$, we define the rank of K_i as

$$h(K_i) = h_0(K_i) + w_i.$$

$h_0(K_i)$ is the initial rank for K_i. Recall that K_i is some indexing keyword. With the feature ranking function h and common weight vector $\mathbf{w}$, *FEX* extracts and learns the most relevant features as follows.

Algorithm FEX. At stage $s \geq 0$, it first sorts all the dynamic features in $F(q)$ with the ranking function h and extracts 10 top-ranked features and suggests them to the user for her to judge their relevance to the query q. When it receives the feature relevance feedback from the user, then for each feature K_i judged by the user as relevant, it promote K_i by setting $w_i = pw_i$; for each feature judged by the user as irrelevant it demotes it by setting $w_i = w_i/d$. Here, both p and d are, respectively, feature promotion and demotion parameters, and they are tunable.

Example 4.2.1. We continue example 4.1.1. The feature learning algorithm FEX will use features judged by the user

Author Proof

to promote or demote their weights. For example, when the feature"acid" is judged by the user as relevant, then its weight w_1 will be promoted and hence the score of the web page d_3 will be increased. When the feature "publication" is judged by the user as irrelevant, its weight w_{44} will be demoted and hence the score of the web page d_4 will be decreased.

4.3. The Document-Learning Algorithm TW2

The algorithm TW2, a tailored version of Winnow2 (Littlestone, 1988), was designed and successfully implemented in our recent projects WebSail and Yarrow (Chen et al., 2000; Chen & Meng, 2000). Winnow2 sets all initial weights to 1, but TW2 sets all initial weights to 0 and has a different promotion strategy accordingly. Another substantial difference between TW2 and Winnow2 is that TW2 accepts document examples that may not contradict its current classification to promote or demote its weight vector. One must notice that Winnow2 only accepts examples that contradict its current classification to perform promotion or demotion. The rationale behind setting all the initial weights to 0 is not as simple as it looks. The motivation is to focus attention on the propagation of the influence of the relevant documents, and use irrelevant documents to adjust the focused search space. Moreover, this approach is computationally feasible because existing effective document-ranking mechanisms can be coupled with the learning process.

Algorithm TW2 (The tailored Winnow2). TW2 uses the common weight vector **w** and a real-valued threshold θ to classify documents in $D(q)$. Initially, all weights have value 0. Let $\alpha > 1$ be the promotion and demotion factor. TW2 classifies documents whose vectors $x = (x_1, \ldots, x_n)$ satisfy $\sum_{i=1}^{n} w_i x_i > \theta$ as relevant, and all others as irrelevant. If the user provides a document that contradicts the classification of TW2, then we say that TW2 makes a mistake. Let $w_{i,b}$ and $w_{i,a}$ denote the weight w_i before the current update and after, respectively. When the user responds with a document which may or may not contradict to the current classification, TW2 updates the weights in the following two ways

- Promotion: For a document judged by the user as relevant with vector $x = (x_1, \ldots, x_n)$, for $i = 1, \ldots, n$, set

$$w_{i,a} = \begin{cases} w_{i,b}, & \text{if } x_i = 0, \\ \alpha, & \text{if } x_i = 1 \text{ and } w_{i,b} = 0, \\ \alpha w_{i,b}, & \text{if } x_i = 1 \text{ and } w_{i,b} \neq 0. \end{cases}$$

- Demotion: For a document judged by the user as irrelevant with vector $x = (x_1, \ldots, x_n)$, for $i = 1, \ldots, n$, set $w_{i,a} = w_{i,b}/\alpha$.

In contrast to the linear lower bounds proved for Rocchio's similarity-based relevance feedback algorithm (Chen & Zhu, 2000), the above learning algorithm has surprisingly small mistake bounds for learning any collection of documents represented by a disjunction of a small number of relevant features. The mistake bounds are independent of the dimensionality of the indexing features. For example

- To learn a collection of documents represented by a disjunction of at most k relevant features (or indexing keywords) over the n-dimensional boolean vector space, TW2 makes at most $[\alpha^2 A/(\alpha - 1)\theta] + (\alpha + 1)k \ln_\alpha \theta - \alpha$ mistakes, where A is the number of dynamic features occurred in the learning process.
- When, in average, l out of k relevant features (or indexing keywords) appear as dynamic features for any relevant document judged by the user during the learning process, the bound in Theorem 4.3.1 is improved to $\alpha^2 A/(\alpha - 1)\theta + [(\alpha + 1)^k/l] \ln_\alpha \theta - \alpha$ in average, where A is the number of dynamic features occurred in the learning process.

The actual implementation of the learning algorithm TW2 requires the help of document ranking and equivalence query simulation given in the following two subsections.

Example 4.3.1. We continue example 4.1.1. The document learning algorithm TW2 will use document examples judged by the user to promote or demote the weights of the dynamic features in those examples. For example, when the document d_3 is judged as relevant by the user, then the algorithm TW2 will promote the weights for all dynamic features in $I(d_3)$ = {acid, action, biology, chen, class, defense, faculty, home, interest, mechanism, micro, page, plant, regulation, research, response, salicylic, transcriptional, uidaho, www, zhixiang}. Thus, any document with dynamic features in $I(d_3)$ will also be promoted. When the document d_1 is judged as irrelevant by the user, then the algorithm TW2 will demote the weights for all dynamic features in $I(d_1)$ = {algorithm, chen, class, computing, data, edu, field, information, learning, machine, mining, network, neural, new, panam, paper, profile, project, recently, research, retrieval, search, seek, strategies, visualization, web, working, www, zhixiang}. Consequently, any document with dynamic features in $I(d_1)$ will be demoted.

4.4. Document Ranking

Let g be a ranking function independent of TW2 and FEX. We define the ranking function f for TW2 as follows. For any web document $d \in D(q)$ with vector $x_d = (x_1, \ldots, x_n) \in V(q)$

$$f(d) = \gamma_d[g(d) + \beta_d] + \sum_{i=1}^{n} w_i x_i,$$

g remains constant for each document d during the learning process of TW2. Various strategies can be used to define g; e.g., PageRank (Brin & Page, 1998), classical tf-idf scheme, vector spread, or cited-based rankings (Yuwono & Lee,

Author Proof

tapraid5/8n-jasis/8n-jasis/8n-orig/8n0754-01a | heckt | S=8 | 4/2/01 | 14:56 | Art: JA-1645 | Input-tam(tam)

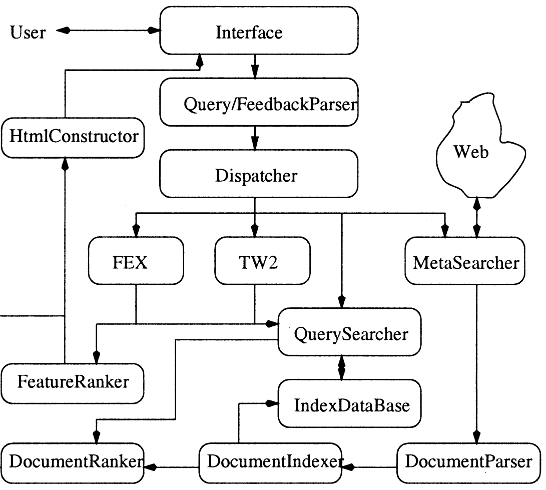


FIG. 1. Architecture of FEATURES.

1996). The two additional tuning parameters are used to do individual document promotions or demotions of the documents that have been judged by the user. Initially, let $\beta_d \geq 0$ and $\gamma_d = 1$. γ_d and β_d can be updated in the similar fashion as w_i is updated by TW2.

4.5. *Equivalence Query Simulation*

The DocumentRanker of FEATURES uses the ranking function f to rank the documents, and the HtmlConstructor returns the top-10–ranked documents to the user. These top-10–ranked documents represent an approximation to the classification made by TW2. The quantity 10 can be replaced by, say, 25 or 50. But it should not be too large for two reasons: The user may only be interested in a very small number of top-ranked documents, and the display space is limited for visualization. The user can examine the short list of documents and can end the search process, or if some documents are judged misclassified, document relevance feedback can be provided. Sometimes, in addition to the top-10–ranked documents, the system may also provide the user with a short list of other documents below the top 10. Documents in the second short list may be selected randomly. The motivation for the second list is to give the user some better view of the classification made by the learning algorithm.

5. The FEATURES

5.1. *The Architecture*

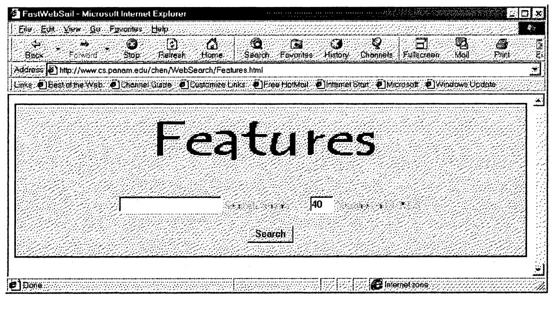
FEATURES is implemented on an Ultra One Sun Workstation using 27 Giga-bytes hard disk storage on an IBM R6000 workstation. It is a multithreaded program coded in F1 C++. Its architecture is shown in Figure 1. FEATURES maintains an internal index database with about 834,000 documents, each of which is indexed with about 300 indexing keywords. Besides its internal index database, it has a MetaSearcher that queries AltaVista when needed. The documents retrieved through meta-search are parsed and indexed by the DocumentParser and the DocumentIndexer that work in real-time. The two learning algorithms FEX and TW2 update the common weight vector **w** concurrently. The major components and their functions are explained in the next subsection.

5.2. *How FEATURES Works*

FEATURES has an interface as shown in Figure 2. Using F2 this interface, the user can enter a query and specify the number of document urls to be returned. Having entered query information, she then starts FEATURES. FEATURES invokes its Query/FeedbackParser to parse the query information, document-relevance feedback, or feature-relevance feedback. Then, Dispatcher decides whether the current task is an initial search process or a learning process. If it is an initial search process, Dispatcher first calls QuerySearcher to find the relevant documents within its internal index database. The relevant documents found by QuerySearcher are then passed to DocumentRanker, which ranks the documents and sends them to HtmlConstructor. HtmlConstructor finally generates html content to be shown to the user.

If QuerySearcher fails to find any documents relevant to the query within IndexDataBase, FEATURES calls its MetaSearcher to query AltaVista and retrieve a list of documents. The length of the list is determined by the user. Once the list of the top-matched documents is retrieved, FEATURES calls its DocumentParser and DocumentIndexer to parse retrieved documents, collate them, and index them with at most 64 indexing keywords. The indexing keywords are automatically extracted from the retrieved documents by DocumentParser. The indexed documents will be cached in IndexDataBase and also sent to DocumentRanker and later to HtmlConstructor to be displayed to the user.

Usually, HtmlConstructor shows the top-10–ranked documents, plus the top-10–ranked features, to the user for her to judge document relevance and feature relevance. However, for the initial search, HtmlConstructor shows only the top-ranked documents. The format of presenting the top-10–ranked documents together with the top-10–ranked features is shown in Figure 3. In this format, each document url F3 and each feature are preceded by radio buttons for the user


FastWebSail - Microsoft Internet Explorer

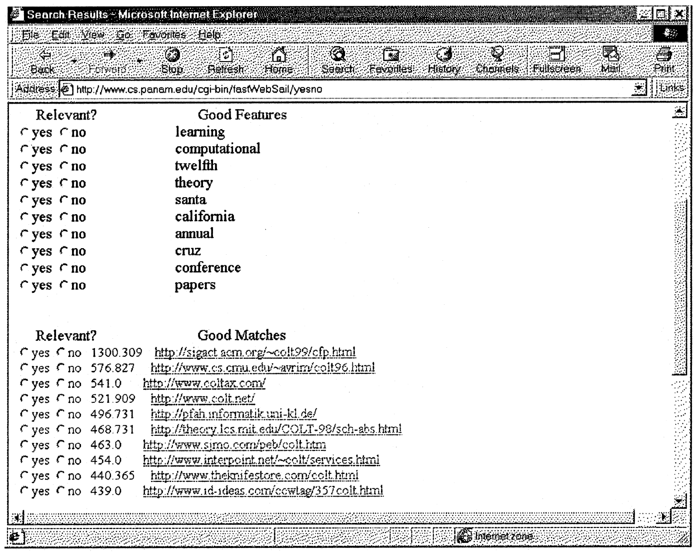
Address http://www.cs.panam.edu/chen/WebSearch/Features.html

Features

40
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print
Links Best of the Web Channel Guide Customize Links Free HotMail Internet Start Microsoft Windows Update
Search
Done
Internet zone

FIG. 2. Interface of FEATURES.

Author Proof


Search Results - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print
Address http://www.cs.panam.edu/cgi-bin/fastWebSail/yesno
Links
Internet zone

Relevant?	Good Features
◯ yes ◯ no	learning
◯ yes ◯ no	computational
◯ yes ◯ no	twelfth
◯ yes ◯ no	theory
◯ yes ◯ no	santa
◯ yes ◯ no	california
◯ yes ◯ no	annual
◯ yes ◯ no	cruz
◯ yes ◯ no	conference
◯ yes ◯ no	papers

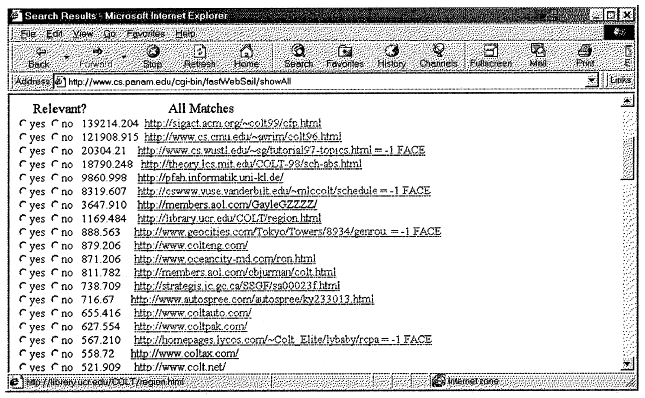
Relevant?		Good Matches
◯ yes ◯ no	1300.309	http://sigact.acm.org/~colt99/cfp.html
◯ yes ◯ no	576.827	http://www.cs.cmu.edu/~avrim/colt96.html
◯ yes ◯ no	541.0	http://www.coltax.com/
◯ yes ◯ no	521.909	http://www.colt.net/
◯ yes ◯ no	496.731	http://pfah.informatik.uni-kl.de/
◯ yes ◯ no	468.731	http://theory.lcs.mit.edu/COLT-98/sch-abs.html
◯ yes ◯ no	463.0	http://www.sjmo.com/peb/colt.htm
◯ yes ◯ no	454.0	http://www.interpoint.net/~colt/services.html
◯ yes ◯ no	440.365	http://www.theknifestore.com/colt.html
◯ yes ◯ no	439.0	http://www.id-ideas.com/ccwtag/357colt.html

FIG. 3. Initial query result for "colt."

to indicate whether the document or the feature is relevant. F4 The search processes shown in Figures 3 and 4 were performed on March 7, 2000. The query word was "colt" and the desired web documents were those related to "computational learning theory." After two iterations with a total of four relevant and irrelevant documents and five relevant and irrelevant features judged by the user as feedback, all the "colt" related web documents among the initial 100 matched documents were moved to the top-10 positions. The clickable urls could have been selected to view the documents so that the user could make his/her judgment more accurately. After providing relevance feedback, he/she can submit the feedback to FEATURES, view all the document urls, or enter a new query to start a new search process.

If the current task is a learning process from the user's document and feature relevance feedback, Dispatcher sends the feature relevance feedback information to the feature learner FEX and the document relevance feedback information to the document learner TW2. FEX uses the relevant and irrelevant features as judged by the user to promote and demote the related feature weights in the common weight vector **w**. TW2 uses the relevant and irrelevant documents judged by the user as positive and negative examples to promote and demote the weight vector. TW2 also performs individual document promotion or demotion for those judged documents. Once FEX and TW2 have finished promotions and demotions, the updated weight vector **w** is sent to QuerySearcher and to FeatureRanker. FeatureRanker re-ranks all the dynamic features that are then sent to HtmlConstructor. QuerySearcher searches IndexDataBase to find the matched documents that are then sent to DocumentRanker. DocumentRanker re-ranks the matched documents and then sends them to HtmlConstructor to select documents and features to be displayed.

Author Proof


Search Results - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print
Address http://www.cs.panam.edu/cgi-bin/fastWebSail/showAll
Links
http://library.ucr.edu/COLT/region.html
Internet zone

Relevant?		All Matches
◯ yes ◯ no	139214.204	http://sigact.acm.org/~colt99/cfp.html
◯ yes ◯ no	121908.915	http://www.cs.cmu.edu/~avrim/colt96.html
◯ yes ◯ no	20304.21	http://www.cs.wustl.edu/~sg/tutorial97-topics.html = -1 FACE
◯ yes ◯ no	18790.248	http://theory.lcs.mit.edu/COLT-98/sch-abs.html
◯ yes ◯ no	9860.998	http://pfah.informatik.uni-kl.de/
◯ yes ◯ no	8319.607	http://cswww.vuse.vanderbilt.edu/~mlccolt/schedule = -1 FACE
◯ yes ◯ no	3647.910	http://members.aol.com/GayleGZZZZ/
◯ yes ◯ no	1169.484	http://library.ucr.edu/COLT/region.html
◯ yes ◯ no	888.563	http://www.geocities.com/Tokyo/Towers/8934/genrou = -1 FACE
◯ yes ◯ no	879.206	http://www.colteng.com/
◯ yes ◯ no	871.206	http://www.oceancity-md.com/ron.html
◯ yes ◯ no	811.782	http://members.aol.com/cbjurman/colt.html
◯ yes ◯ no	738.709	http://strategis.ic.gc.ca/SSGF/sa00023f.html
◯ yes ◯ no	716.67	http://www.autospree.com/autospree/ky233013.html
◯ yes ◯ no	655.416	http://www.coltauto.com/
◯ yes ◯ no	627.554	http://www.coltpak.com/
◯ yes ◯ no	567.210	http://homepages.lycos.com/~Colt_Elite/lybaby/rcpa = -1 FACE
◯ yes ◯ no	558.72	http://www.coltax.com/
◯ yes ◯ no	521.909	http://www.colt.net/

FIG. 4. Result for "colt" after two Iterations, four examples, and five features judged.

Example 5.2.1. When we queried AltaVista with "colt" on March 7, 2000, it found 182,130 relevant pages. For example, the following two pages were among the top-10 pages returned

d_1 = http://sigact.acm.org/cot99/COLT99.html
d_2 = http://www.colteng.com

The dynamic features automatically generated by our system for those two pages are

$I(d_1)$ = {acm, annual, california, class, computational, conference, cruz, july, learning, org, region, registration, santa, sigact, theory, twelfth, university}
$I(d_2)$ = {choice, clients, colteng, com, contractor, corporation, creative, engineering, innovative, people, quote, solutions, striving, welcome}

Obviously, d_1 is relevant to "computational learning theory," while d_2 is not. When d_1 is judged as relevant and d_2 as irrelevant, the document learning algorithm TW2 will

Table 1. ●●●

$R_{precision}$	(50,10)	(50,20)	(100,10)	(100,20)	(150,10)	(150,20)	(200,10)	(200,20)
AltaVista	0.36	0.30	0.36	0.30	0.36	0.30	0.36	0.30
FEATURES	0.48	0.40	0.51	0.44	0.52	0.46	0.52	0.46

promote all the pages with dynamic features in $I(d_1)$ and demote those with dynamic features in $I(d_2)$. After this iteration of the algorithm TW2, dynamic features such as "learning, computational, theory, conference, papers," were among the top-ranked features and then presented to the user to judge for their relevance by the feature learning algorithm FEX. When "learning, computational, theory" were judged by the user as relevant features, all the pages with them as dynamic features would be promoted by the algorithm FEX.

5.3. *The Performance: FEATURES Versus AltaVista*

We have made FEATURES open for public access. Interested readers can access it via the url given at the end of the article and check its performance. Although FEATURES is still in its early stages, its actual performance is promising. In order to provide some measure of system performance we have made a comparison between AltaVista [a] and FEATURES. Our evaluation is based on the following approach similar to Recall and Precision.

For a search query q, let A denote the set of documents returned by the search engine (either AltaVista or FEATURES), and let R denote the set of documents in A that are relevant to q. For any integer m with $1 \le m \le |A|$, define R_m to be the set of documents in R that are among the top-m–ranked documents according to the search engine. Now, we define the relative Recall R_{recall} and the relative Precision $R_{precision}$ as follows

$$R_{recall} = \frac{|R_m|}{|R|}$$

$$R_{precision} = \frac{|R_m|}{m}$$

We have conducted experiments for 100 search queries, each of which was sent to both AltaVista and FEATURES. The results returned from the two engines were examined manually to calculate R_{recall} and $R_{precision}$. We selected a query based on the following conditions: It should have a long list (say, more than 1,000) of relevant pages. Its relevant pages should have different "clusters" or "groups." Both conditions are easy to verify through querying AltaVista. For example, the query "colt" has 182,130 relevant pages according to the query made on March 7, 2000. Those pages can divided into a group related to "computational learning theory," a group related to "horses," a group related to "guns," a group related to "corporation and engineering," and so on. It is rather easy to find 100 words satisfying the conditions. Among those we selected are "colt, memory, chip, language, high school geometry, michael jordan, harvard, utpa, architecture." The number of keywords in our queries varies from one to four, and the average number is 1.75. During those experiments, the authors served as users and evaluated the documents returned by the search engines. For each query, the set R of relevant pages is defined and understood by the authors. For example, for the query "colt," the R is defined as the set of documents related to "computational learning theory." For the query "chip," the R is defined as the set of documents related to "computer chips." For the query "harvard," the R is defined as the set of documents related to "Harvard University." Since each query we selected has a huge number of relevant pages, in order to perform the experiments we limit R within a short list of top (say, 100)-ranked documents returned by the search engine. We actually read and examined the contents of all the documents within this short list to find R. We followed the same approach to find the set R_m for each query, i.e., we read and examined the contents of the top-m–ranked documents to find those that are indeed relevant to the query.

We summarize the average relative Recall R_{recall} and the average relative Precision $R_{precision}$ in Tables 1 and 2. One may notice that FEATURES has better performance by these measures. T1-T2

In the tables, each column is labeled by a pair of integers $(|A|, m)$, where $|A|$ is the total number of documents retrieved by the engine for the given search query and m is used to define the relative Recall and Precision. Because of the nonadaptive nature of AltaVista, it is obvious that AltaVista has the same relative Precision for each query when the value of m is the same. The average relative Recall and

Table 2. ●●●

R_{recall}	(50,10)	(50,20)	(100,10)	(100,20)	(150,10)	(150,20)	(200,10)	(200,20)
AltaVista	0.37	0.51	0.30	0.42	0.29	0.39	0.29	0.39
FEATURES	0.67	0.96	0.42	0.80	0.37	0.71	0.37	0.67

Author Proof

tapraid5/8n-jasis/8n-jasis/8n-orig/8n0754-01a | heckt | S=8 | 4/2/01 | 14:56 | Art: JA-1645 | Input-tam(tam)

Precision values are calculated for Features after an average of five interactive refinements.

6. Concluding Remarks and Future Work

Web search has come to people's daily life as the web evolves. Designing practically effective web search algorithms is a challenging task. It calls for innovative methods and strategies from many fields including machine learning. As we pointed out, web search can be understood in some sense as adaptive learning from queries. However, few learning algorithms are ready to be used in web search because of a number of realistic requirements. In general, the fundamental question about any learning algorithm is certainly its applicability to real-world problems. In the particular case of web search, we are interested in learning algorithms that can effectively increase the search performance with a very small number (say, five) of relevance AQ: 2 feedback data, because users do not have their patience to try too many iterations of learning. We think that the most challenging problem regarding intelligent web search is whether we can build an adaptive learning system that supports high search precision increase with several iterations of relevance feedback from the user.

As part of our efforts toward building an intelligent system for web search, we have designed and implemented Features. It utilizes two adaptive learning algorithms that work concurrently in real-time, one of which extracts and learns the most relevant dynamic features from the user's feature relevance judgments, and the other which learns the most relevant documents from the user's document relevance judgments. Features is still in its early stages and needs to be improved and enhanced in many aspects.

In the future, we plan to improve the interface of Features. Right now, the system displays the urls of the documents. If the user wants to know the contents of the document, he/she needs to click the url to download the content. We plan to display the url of a document together with a good preview of its content. We also want to highlight those indexing keywords in the preview and allow them to be clickable for feature learning.

We also plan to apply clustering techniques to increase the performance of our system. It is easy to observe that in most cases documents that are relevant to a search query can be divided into a few different clusters or groups. Recall that documents relevant to "colt" can be divided into clusters related to "computational learning theory," "guns," "horse," "corporation engineering," etc. We believe that document clustering techniques such as graph spectral partitioning can be used to reduce the number of the iterations of the learning process and to increase the performance of the system.

At its current stage, Features can only query AltaVista for its meta-search purpose. We plan to expand its meta-search capability to allow parallel queries to several existing search engines. We want to use collaborative recommendation methods to collate the search results of the parallel queries.

Because adaptive learning is incremental and depends on the history of a learning process to improve learning performance, Features creates and maintains a specific thread for each web-search process. When a search process is finished, its related thread will be terminated. It may not be easy to employ an adaptive learning algorithm at a popular web-server side. Because a popular web server may have thousands of user accesses in every minute, innovative thread management methods are needed in order to maintain many threads for individual search processes.

Acknowledgments

We thank the two anonymous referees for their valuable comments and questions that have helped us to revise our paper. The work in this paper is supported in part by a NASA grant NAG9-1169, a 1999–2000 UTPA Faculty Research Council grant, and a 1999–2000 UTPA Scholars grant. Part of Binhai Zhu's work was completed while he was affiliated with the Computer Science Department, City University of Hong Kong, China.

URL References

[a] AltaVista: www.altavista.com
[b] Yahoo!: www.yahoo.com
[c] Google: www.google.com
[d] MetaCrawler: www.metacrawler.com
[e] Inference Find: www.infind.com
[f] Dogpile: www.dogpile.com
[g] WebSail: www.cs.panam.edu/chen/WebSearch/WebSail.html
[h] Yarrow: www.cs.panam.edu/chen/WebSearch/Yarrow.html
[i] Features: www.cs.panam.edu/chen/WebSearch/Features.html

References

Angluin, D. (1987). Queries and concept learning. Machine Learning, 2, 319–432.

Baeza-Yates, R., Ribeiro-Neto, B. (1999). Modern Information Retrieval. Addison-Wesley. AQ: 3

Billsus, D., & Pazzani, M.J. (1998). Learning collaborative information filters. In Proceedings of the Fifteenth International Conference on Machine Learning. AQ: 4

Bollacker, K., Lawrence, S., & Giles, C.L. (1998). Citeseer: An autonomous web agent for automatic retrieval and identification of interesting publications. In Proceedings of the Second International Conference on Autonomous Agents (pp. 116–113). New York: ACM Press. AQ: 5

Bollacker, K., Lawrence, S., & Giles, C.L. (1999). A system for automatic personalized tracking of scientific literature on the web. In Proceedings of the Fourth ACM Conference on Digital Libraries (pp. 105–113). New York: ACM Press. AQ: 5

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine. In Proceedings of the Seventh World Wide Web Conference. AQ: 6

Chakrabarti, S., Dom, B., Raghavan, P., Rajagopalan, S., Gibson, D., & Kleinberg, J. (1998). Automatic resource compilation by analyzing hyperlink structure and associated text. In Proceedings of the Seventh World Wide Web Conference (pp. 65–74). AQ: 6

Author Proof

Chen, Z., & Meng, X. (2000). Yarrow: A real-time client site meta search learner. In Proceedings of the AAAI 2000 Workshop on Artificial Intelligence for Web Search (pp. 12–17). AQ: 6

Chen, Z., & Zhu, B. (2000). Some formal analysis of the Rocchio's similarity-based relevance feedback algorithm. In D. Lee & S.-H. Teng (Eds.), Proceedings of the Eleventh International Symposium on Algorithms and Computation, Lecture Notes in Computer Science 1969 (pp. 108–119). Springer-Verlag. AQ: 7

Chen, Z., Meng, X., & Fowler, R.H. (1999). Searching the web with queries. Knowledge and Information Systems, 1, 369–375.

Chen, Z., Meng, X., Zhu, B., & Fowler, R. (2000). Websail: From on-line learning to web search. In Proceedings of the 2000 International Conference on Web Information Systems Engineering [the full version will appear in Journal of Knowledge and Information Science, the special issue of WISE'00] (pp. 192–199). AQ: 6

Freund, Y., Iyer, R., Schapire, R., & Singer, Y. (1998). An efficient boosting algorithm for combining preferences. In Machine Learning: Proceedings of the Fifteenth International Conference. AQ: 6

Gibson, D., Kleinberg, J., & Raghavan, P. (1998). Inferring web communities from link topology. In Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia. AQ: 6

Ide, E. (1971). New experiments in relevance feedback. In G. Salton (Ed.), The Smart System—Experiments in Automatic Document Processing (pp. 337–354). Englewood Cliffs, NJ: Prentice-Hall.

Ikeji, A., & Fotouhi, F. (1999). An adaptive real-time web search engine. In Proceedings of the ACM CIKM'99 Workshop on Web Information and Data Management. AQ: 6

Kleinberg, J. (1999). Authoritative sources in a hyperlinked environment. Journal of ACM 46, 604–632.

Lawrence, S., Bollacker, K., & Giles, C.L. (1999). Indexing and retrieval of scientific literature. In Proceedings of the Eighth ACM International Conference on Information and Knowledge Management. AQ: 6

Lewis, D. (1991). Learning in intelligent information retrieval. In Proceedings of the Eighth International Workshop on Machine Learning (pp. 235–239). AQ: 6

Littlestone, N. (1988). Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. Machine Learning, 2, 285–318.

Meng, X., & Chen, Z. (1999). Personalize web search using information on client's side. In Advances in Computer Science and Technologies (pp. 985–992). International Academic Publishers. AQ: 8

Nakamura, A., & Abe, N. (1998). Collaborative filtering using weighted majority prediction algorithms. In Machine Learning: Proceedings of the Fifteenth International Conference. AQ: 6

Rennie, J., & McCallum, A. (1999). Using reinforcement learning to spider the web efficiently. In Proceedings of the Sixteenth International Conference on Machine Learning. AQ: 6

Rocchio, J. (1971). Relevance feedback in information retrieval. In G. Salton (Ed.), The Smart Retrieval System—Experiments in Automatic Document Processing (pp. 313–323). Englewood Cliffs, NJ: Prentice-Hall.

Salton, G. (1971). The performance of interactive information retrieval. Information Processing Letters, 1, 35–41.

Salton, G. (1989). Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley. AQ: 9

Salton, G., & Buckley, C. (1990). Improving retrieval performance by relevance feedback. Journal of the American Society for Information Science, 41, 288–297.

Salton, G., Wong, A., & Yang, C. (1975). A vector space model for automatic indexing. Comm of ACM, 18, 613–620. AQ: 10

Widyantoro, D., Ioerger, T., & Yu, J. (1999). An adaptive algorithm for learning changes in user interests. In Proceedings of the Eight ACM International Conference on Information and Knowledge Management (pp. 405–412).

Yuwono, B., & Lee, D. (1996). Search and ranking algorithms for locating resources on the world wide web. In Proceedings of the International Conference on Data Engineering (pp. 164–171). AQ: 11

Author Proof