WebSail: From On-line Learning to Web Search

Zhixiang Chen
Department of Computer Science
University of Texas-Pan American
Edinburg, TX 78539, USA
chen@cs.panam.edu

Xiannong Meng
Department of Computer Science
University of Texas-Pan American
Edinburg, TX 78539, USA
meng@cs.panam.edu

Binhai Zhu
Department of Computer Science
City University of Hong Kong
Kowloon, Hong Kong, China
bhz@cs.cityu.edu.hk

Richard H. Fowler
Department of Computer Science
University of Texas-Pan American
Edinburg, TX 78539, USA
fowler@panam.edu

Abstract

In this paper we investigate the applicability of on-line learning algorithms to the real-world problem of web search. Consider that web documents are indexed using n Boolean features. We first present a practically efficient on-line learning algorithm TW2 to search for web documents represented by a disjunction of at most k relevant features. We then design and implement WebSail, a real-time adaptive web search learner, with TW2 as its learning component. WebSail learns from the user's relevance feedback in real-time and helps the user to search for the desired web documents. The architecture and performance of WebSail are also discussed.

1. Introduction

Nowadays there are a number of search engines for people to search for their desired web documents. Each of the existing search engines has a unique interface and an index database covering a different portion of the web. They have proved both useful and popular. In general, when using a search engine the user needs to repeatedly refine her query as she does not have enough knowledge to formulate the query precisely. Usually the search engine returns tremendously many urls of web documents that are irrelevant, forcing the user to manually sift through the long list to locate the desired documents.

The way the user uses a search engine is much like a dialogue between the user and the engine: The user sends a query to the engine, and the engine uses the query to search the index database and returns a list of document urls. Then, the user provides the engine relevance feedback, and the engine uses the feedback to improve its next search and returns a refined list of document urls. The dialogue ends when the engine finds the desired documents for the user. Note that conceptually a query entered by the user can be understood as the logical expression of the collection of the documents wanted by the user. A list of document urls returned by the engine can be interpreted as an approximation to the collection of the desired documents. This type of scenario is very similar to the process of on-line learning with queries [1, 17], when the user acts as a teacher and the engine as a learner.

Unfortunately, in reality, the user is not qualified as a teacher as modeled in on-learning [1, 17]. Although the average user (at the common sense level) knows what kinds of documents she wants, it is difficult, if not impossible, for her to inform the search engine what is wanted in an easy way so that the search engine understands. Even when disjunctions or conjunctions of keywords are chosen as the way of expressing the search goal as existing search engines do, the user may not know what set of keywords she should use to define the collection of the desired documents precisely. On the other hand, having received relevance feedback from the user, the search engine needs to find an efficient way to use the feedback so that it can, in real-time, improve the result of its next search.

From the perspective of machine learning, the fundamental question about any learning algorithm is of course its applicability to real-world problems. Especially when the real-world problem of web search is concerned, few theoretically well-established learning algorithms are ready to use, not only because the user cannot be modeled as a teacher, but also because the user may make mistakes and has no patience to try more than a couple of dozens of interactions. Other practical factors such as real-time computing, indexing and ranking are involved as well.

In this paper, we first investigate the applicability of on-line learning algorithms to web search. Since existing search engines support queries represented by disjunctions or conjunctions of features, we especially want to know how to use on-line learning algorithms to search for a collection of documents represented by a disjunction of at most k relevant features. The conjunction case can be coped with similarly. We choose Winnow2 [17] as the starting point of our investigation because it has error-tolerant ability and an inherent ranking mechanism (the inner product of the weight vector and the document vector) as well as small updating complexity. It also has the best known mistake bounds for learning disjunctions of at most k relevant features for small k. However, the mistake bounds of Winnow are still too high for practical use in web search. For example, with tuning parameters $\alpha = 1.5$ and $\theta = \frac{n}{k}$, the mistake bound of Winnow2 is $13k + 14k \ln \frac{n}{k}$ for learning a disjunction of at most k relevant features over the Boolean vector space $\{0, 1\}^n$. As it is known, usually a huge vocabulary of keywords is used to index documents in web search. If, for example, the vocabulary has just 10,000 keywords, then about 600 refinements (or mistakes) are needed when Winnow2 is used to search for documents represented by a disjunction of at most 5 relevant keywords. Obviously, these refinements are too many for any user.

We examine the properties of document indexing, and design a tailored version TW2 of Winnow2 in section 3 for the real-world problem of web search. In search for a collection of documents represented by a disjunction of at most k relevant features, the mistake bound of TW2 is at most $\frac{\alpha}{\alpha-1}\frac{\alpha A}{\theta} + (\alpha+1)k \log_\alpha \theta - \alpha$, where α is the promotion and demotion parameter, θ is the threshold, and A is total number of distinct features used to index all the positive examples (documents judged by the users as relevant) received in the search process. Other bounds are also obtained for tolerating feature errors and for the average number of relevant features occurred in each positive example. In practice A is very small, especially when meta-search is concerned. For example, in the project Yarrow [7] that we have implemented, A is usually smaller than 640 when the top 100 matches are needed. Even if we assume $A = 10,000$ for the purpose of comparison, for $\alpha = 1.5$ and $\theta = \frac{A}{5}$, TW2 makes about about 256 mistakes in the worst case for learning a disjunction of at most 5 relevant features. For $A \leq 1,000$, the worst case mistake bound of TW2 is 194 for learning the same disjunction.

We have implemented a real-time adaptive web search learner WebSail [d] with TW2 as its learning component. Interested readers can access WebSail via its url given at the end of the paper. WebSail learns from the user's relevance feedback and helps the user to search for the desired documents with as little relevance feedback as possible. It is implemented on a Sun Ultra one workstation with storage of 27 Giga-bytes hard disk on an IBM R6000 workstation. It has an internal index database and a meta-search component through AltaVista [a]. Each document in the internal database is indexed using about 300 keywords. When the user performs a search process, WebSail first searches its internal database. If no matches can be found for the query within the internal database, then it turns to its meta-search component to receive the matched documents through AltaVista [a] and then performs the learning process locally.

There have been considerable efforts applying machine learning to web search related applications, for example, scientific article locating and user profiling [3, 4, 15], focused crawling [20], and collaborative filtering [19].

The remaining part of this paper is organized as follows. In section 2, we examine the similarity between on-line learning and web search, and discuss what properties a learning algorithm should have in order to be applicable to web search. In section 3 we discuss some difference between on-learning approach to web search and the similarity-based relevance feedback algorithm in information retrieval. In section 4, we examine the property of web document indexing and design the learning algorithm TW2, a tailored version of Winnow2 [17] for web search. In section 5, we discuss practical issues such as document ranking and equivalence query simulation regarding the actual employment of TW2 as a learning component in WebSail [d]. We also discuss the architecture and performance of WebSail. We conclude the paper in section 6.

2. On-Line Learning vs. Web Search

In the on-line learning model [1, 17] with equivalence queries, the goal of a learner for learning a concept class $\mathbf{C}$ over the domain $\mathbf{Z}$ is to learn any unknown target concept $c_t \in \mathbf{C}$ that has been fixed by a teacher. In order to obtain information about c_t, the learner can ask the teacher equivalence queries by proposing hypotheses h from a fixed hypothesis space $\mathbf{H}$ over $\mathbf{Z}$ with $\mathbf{C} \subseteq \mathbf{H}$. If $h = c_t$, then the teacher says "yes", so the learner succeeds. If $h \neq c_t$, then the teacher responds with an example x in $(c_t - h) \cup (h - c_t)$ for some $x \in \mathbf{Z}$. In such a case, we say that the algorithm make a mistake. x is called a positive example if it is in c_t and a negative example otherwise. Each new hypothesis issued by the leaner may depend on the earlier hypotheses and the examples received so far. A learner exactly learns $\mathbf{C}$, if for any target concept $c_t \in \mathbf{C}$, it can learn c_t. We say that a class $\mathbf{C}$ is polynomial time learnable if a learner can exactly learn any target concept in $\mathbf{C}$ and the time required by the learner is polynomial in the size of the domain and the size of the target concept. Besides the time complexity of the algorithm, we are also interested in the total number of mistakes the algorithm may make in order to learn any target concept $c_t \in \mathbf{C}$. It is easy to see that the number of equivalence queries needed by the learning algorithm to learn the target concept is one plus the number of mistakes the algorithm may make during its learning process.

We now consider how to use on-line learning from equivalence queries to approach the problem of web search. We use the vector space model [21, 22, 2] to represent documents. The vector space may consist of boolean vectors. It may also consist of discretized vectors, for example the frequency vector of the indexing keywords. A target concept is a collection of documents, which is equivalent to the set of vectors of the documents in the collection. The learner is the search engine and the teacher is the user. The goal of the search engine is to find the target concept in "real-time" with a minimal number of mistakes (or equivalence queries).

Let us define an *interaction* between the user and the search engine as the process that starts at the time the user provides her feedback (or search query at the very beginning of the search) to the search engine and ends at the time the search engine displays the search result back to the user. At each interaction, it is not unreasonable to assume that the user can on the average judge five documents as relevant or irrelevant to provide relevance feedback to the search engine. On the average no user would like to perform more than a dozen of interactions. Hence, we consider that on the average the user may judge about 60 documents as relevance feedback to the search engine during the entire search process. Since existing search engines support disjunctions or conjunctions of features as query formation that is practically acceptable by the average user, in this paper we will focus on the problem of searching for collections of web documents represented by disjunctions of relevant features. Conjunctions can be coped with similarly. Based on the above analysis, a learning algorithm L should have the following properties in order to be applicable to the real-world problem of web search:

- *L should have the ability to tolerate errors such as feature errors and classification errors.*
- *L should have a "practically small" mistake bound. For example, when used to search for documents represented by a disjunction of at most k relevant features, its mistake bound should be very small, say, about 60 for $k = 5$.*
- *L should have a built-in ranking mechanism to move the most relevant documents to the top and the least relevant to the bottom.*
- *Finally, its computation for each interaction should be performed in time linear in the dimensionality of the vector space, or a few dozens of seconds in practice.*

In reality, the user is definitely not qualified as a "teacher" as modeled in on-line learning [1, 17]. She does not know the logical representation of the target concept, nor how to answer equivalence queries. However, it is reasonable to assume that the user can judge whether a particular web document is relevant or not to her search, though she may also make mistakes in this aspect. It should be pointed out that there are cases in which the user may not be able to tell whether a web document is relevant or not. Such cases are beyond the scope of this paper and should be studied in future research.

3. On-Line Learning vs. Similarity-Based Relevance Feedback

One should distinguish our on-line learning approach to web search from the similarity-based relevance feedback algorithm in information retrieval [21, 2, 12, 11, 13]. The central idea of relevance feedback is to improve search performance for a particular query by modifying the query step by step, based on the user's judgments of the relevance or irrelevance of some of the documents retrieved. In the vector space model [21, 22], both documents and queries are represented as vectors in a discretized vector space. In this case, relevance feedback is essentially an on-line learning algorithm: A query vector and a similarity measure are used to classify documents as relevant and irrelevant; the user's judgments of the relevance or irrelevance of some the classified documents are used as examples for updating the query vector as a linear combination of the initial query vector and the examples judged by the user. Especially, when the inner product similarity is used, relevance feedback is just a Perceptron-like learning algorithm [16, 9].

There are many different variants of relevance feedback in information retrieval. The most popular one is Rocchio's similarity-based relevance feedback algorithm [13, 11, 21], which works in a step by step adaptive refinement fashion as follows. Starting at an initial query vector q_1, the algorithm searches for all the documents d such that d is *very close* to q_1 according to the similarity m, ranks them by $m(q, d)$, and finally presents a short list of the top ranked documents to the user. The user examines the returned list of documents and judges some of the documents as relevant or irrelevant. At step $t \geq 1$, assume that the list of documents the user judged is $x_1, \ldots, x_{t-1}$. Then, the algorithm updates its query vector as $q_t = \alpha_{t_0} q_1 + \sum_{j=1}^{t-1} \alpha_{t_j} x_j$, where the coefficients $\alpha_{t_j} \in R$ for $j = 0, 1, \ldots, t-1$. At step $t + 1$, the algorithm uses the updated query vector q_t and the similarity m to search for relevant documents, ranks the documents according to m, and presents the top ranked documents to the user. In practice, a threshold θ is explicitly (or implicitly) used to select the highly ranked documents. Practically, the coefficients α_{t_j} may be fixed as 1, -1 or 0.5 [2, 21].

Some formal analysis about Rocchio's similarity-based relevance feedback was given in [9]. It was proved in [9] that in the Boolean vector space model, for any of the four typical similarities (inner product, dice coefficient, cosine coefficient, and Jaccard coefficient), Rocchio's similarity-based relevance feedback algorithm makes at least $(n+k-3)/2$ mistakes when used to search for a collection of documents represented by a monotone disjunction of at most k relevant features over the n-dimensional Boolean vector space $\{0,1\}^n$. The lower bounds are independent of the choices of the threshold and coefficients that the algorithm may use in updating its query vector and making its classification.

The lower bounds established in [9] for Rocchio's similarity-based relevance feedback algorithm is based on the worst case analysis, hence they may not affect the algorithm's effective applicability to the real-world problems despite of their theoretical significance. On the other hand, the formal analysis motivates us to design new learning algorithms for information retrieval. As the first step, we design TW2 in the next section. TW2 has a provable better mistakes bounds for learning disjunctions of at most k relevant features for very small k when the inner product similarity measure is used.

4. The Algorithm TW2

When a set of n Boolean-valued features are used to index web documents, a document is represented as a vector in the n-dimensional space $\{0,1\}^n$. Given any document d, let $v_d(x_1,\ldots,x_n)$ denote its vector representation. Define

$$\begin{aligned} MD[x_{i_1},\ldots,x_{i_s}] &= \{d | v_d(x_1,\ldots,x_n) \\ &\qquad \Longrightarrow x_{i_1} \vee \cdots \vee x_{i_s}\}, \\ MD(k) &= \{MD[x_{i_1},\ldots,x_{i_s}] | 1 \le s \le k\}, \end{aligned}$$

In other words, $MD[x_{i_1},\ldots,x_{i_s}]$ is a collection of documents whose vectors satisfy the monotone disjunction of $x_{i_1},\ldots,x_{i_s}$, and $MD(k)$ is the class of all collections of documents represented by monotone disjunctions of at most k relevant feature variables. Using machine learning techniques, several efficient algorithms have been constructed in [8] for searching any collection of documents in $MD(k)$.

Many theoretically efficient algorithms exist for learning disjunctions of at most k relevant features. But few meet the applicability requirements as discussed in section 2. Winnow2 [17] meets all the requirements except that its mistake bound, though optimal within a constant factor, is still too high for web search users. Because of the $k\log_2\frac{n}{k}$ lower bound [17] for learning a disjunction of k relevant features in the n-dimensional Boolean vector space, it is essentially impossible to improve the upper bound of Winnow2 in general. However, we can exploit the properties of web document indexing, and seek opportunities to reduce the bound in the concrete case of web search.

Definition 4.1. *Given any feature x and any document d, x is said to be an indexing feature for the document d, if the corresponding component of the feature x in the document vector v_d is* 1.

Although a huge collection of features (in the simplest case, keywords) are needed and used to index web documents, for each particular document d, the number of its indexing features is relatively small. One can easily note that a web hypertext document may have several hundreds of distinct keywords, while a good dictionary may have over 100,000 words. One may argue that there are long documents. This is certainly true. But as far as indexing is concerned, not all words in a long document are needed to index it. Instead, a small portion of the words may be used. To the end, indexing is closely related to classification. The depth of the Yahoo! [b] classification tree is about 30. Also for the efficiency consideration of web crawling, normally only the first a few kilobytes of a document is extracted, when it is long. The current version of the http protocol allows the creator of a web document to list keywords as meta-attributes in a document for the purpose of indexing or classification. People may list about a couple of dozens of keywords. If everyone adds meta-attributes to her web documents, then the challenging problem of document indexing or classification would be resolved with automatic extraction of those keywords. As far as the authors understand, e-commerce related web documents tend to aggressively employ those kinds of meta-attributes. This may influence average web users to follow the practice.

When the user queries a search engine, the engine finds the matched documents, ranks them and returns the top ranked ones to the user. Various strategies have been studied for document ranking to move the most relevant documents to the top and the least relevant to the bottom. For example, the classical tf-idf scheme, vector spread, or cited-based rankings [23]; the algorithm for locating authorities and hubs [14, 6, 10] which works effectively for document ranking related to broad topic queries; and PageRank [5] which is the key ranking method for Google [b]. A learning algorithm for web search should be constructed to take advantage of the existing ranking mechanisms. More precisely, when the user queries a search engine, the learning algorithm should use the list returned by the search engine as its initial search space, it then uses relevance feedback from the user to effectively propagate the user's preference within the initial list of documents returned and other documents in the database as well. Because the user is really interested in a short list of the top matched (or relevant) documents, the learning algorithm should at the beginning

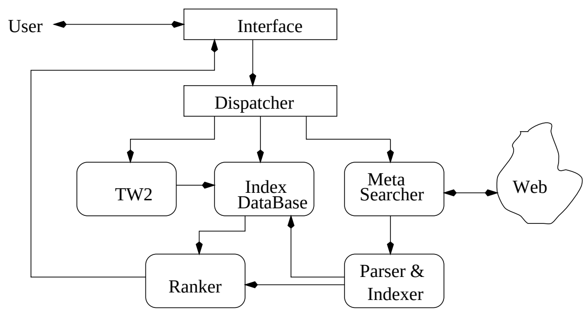


Figure 1. Architecture of WebSail

pay more attention to the propagation of the influence of the relevant documents judged by the user.

We now design TW2, a tailored version of Winnow2, to exploit the particular nature of web search. We also present four mistake bounds for TW2 in this section. The actual implementation of TW2 along with document ranking and equivalence query simulation will be given in the next section. The key difference between TW2 and Winnow2 is that Winnow2 sets all initial weights to 1, while TW2 sets all initial weights to 0 and thus it has a different promotion strategy accordingly. 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 can be effectively implemented because existing effective document ranking mechanisms can be coupled with the learning process as discussed in next section.

The Algorithm TW2 (The Tailored Winnow2). *TW2 maintains non-negative real-valued weights $w_1, \ldots, w_n$ for features $att_1, \ldots, att_n$, respectively. It also maintains a real-valued threshold θ. 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 to 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 contradicts 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*

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

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

Let A denote the total number of distinct indexing features of all the relevant documents that are judged by the user during the learning process. The following theoretical mistake bounds are obtained for TW2.

- *To learn a collection of documents represented by a disjunction of at most k relevant features over the n-dimensional Boolean vector space, TW2 makes at most $\frac{\alpha^2 A}{(\alpha-1)\theta} + (\alpha+1)k \ln_\alpha \theta - \alpha$ mistakes.*
- *When on the average l out of k relevant features appear as indexing features for any relevant document judged by the user during the learning process, the bound in Theorem 3.2 is improved to $\frac{\alpha^2 A}{(\alpha-1)\theta} + \frac{(\alpha+1)k}{l} \ln_\alpha \theta - \alpha$ on the average.*

TW2 has the ability to tolerate feature errors and classification errors. Given an example x and a Boolean value b, we say the example x paired with the classification value b, denoted by $< x, b >$, has t feature errors with respect to a target concept c, if t is the minimal number of features or bits of x that have to be changed so that b is true if and only if $x \in c$. The number of feature errors with respect to a target concept for a sequence of examples paired with their classification values $< x_1, b_1 >, \ldots, < x_m, b_m >$ is is simply the total number of feature errors for the all pairs $< x_i, b_i >$. When a learning algorithm learns a target concept c, the feature errors occurred in the learning process is the number of feature errors with respect to c for the sequence of pairs $< x_1, b_1 >, \ldots, < x_m, b_m >$, where each x_i is an example received by the algorithm and b_i is the classification value of x_i judged by the user. Given an example x and a Boolean value b, a classification error for the pair $< x, b >$ with respect to the target concept c is that b is true but $x \notin c$ or b is false but $x \in c$. As pointed out in [18], a classification error can be compensated with at most k feature errors (or attribute errors in their term) for any target concept represented by a disjunction of k relevant features. Hence, we only state feature error-tolerant bounds for TW2. The following error-tolerant mistake bound is obtained for TW2.

- *Let Z denote the total number of feature errors occurred during the learning process of a collection of documents represented by a disjunction of at most k relevant features over the n-dimensional Boolean vector space, TW2 makes at most $\frac{\alpha^2 A}{(\alpha-1)\theta} + (\alpha+1)k \ln_\alpha \theta + (\alpha+1)Z - \alpha$ mistakes.*

When the user judges a document example x as relevant, the indexing features of x may contain more than one relevant feature of the target concept. Thus, a better mistake bound is also obtained for TW2 in the following.

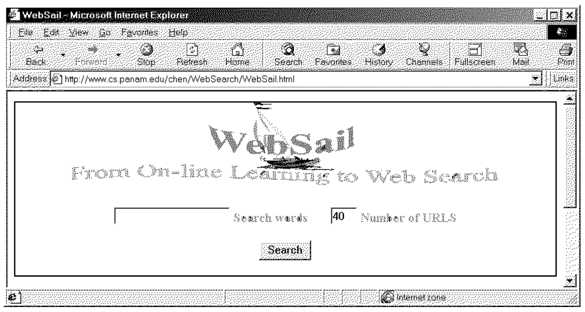

WebSail - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print
Address http://www.cs.panam.edu/chen/WebSearch/WebSail.html
Links
WebSail
From On-line Learning to Web Search
Search words
40
Number of URLS
Search
Internet zone

Figure 2. Interface of WebSail

- *When on the average l out of k relevant features appear as indexing features for any relevant document judged by the user during the learning process, then TW2 has an improved mistake bound* $\frac{\alpha^2 A}{(\alpha-1)\theta} + \frac{(\alpha+1)k}{l}\ln_\alpha \theta + (\alpha+1)Z - \alpha$.

5. The WebSail

We now present the design and implementation of WebSail and discuss how TW2 is used as its learning component. As we mentioned in the previous section, TW2 must be used with the help of document ranking and equivalence query simulation.

5.1. Document Ranking

Let $w = (w_1, \ldots, w_n)$ be the weight vector maintained by TW2. Let g be a ranking function independent of TW2. We define the ranking function f for TW2 as follows: For any web document d with vector $x_d = (x_1, \ldots, x_n)$,

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

g remains constant for each document d during the learning process of TW2. Various strategies can be used to define g, for example, PageRank [5], the classical tf-idf scheme, vector spread, or cited-based rankings [23]. The two additional tuning parameters are used to perform individual document promotions or demotions of the documents that have been judged by the user as relevance feedback during the learning process of TW2. The motivation for individual document promotions or demotion is that when the status of a document is determined because of the user's judgment, it should be placed closer to the top than the rank supported by the weighted sum of TW2 if it is relevant or placed closer to the bottom otherwise. 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.

5.2. Equivalence Query Simulation

WebSail uses the ranking function f to rank the documents classified by TW2 and returns the top R relevant documents to the user. These top R ranked documents represent an approximation to the classification made by TW2. The user can examine this short list of the R documents. If she feels satisfactory with the result then she can end the search process. If she finds that some documents in the list are misclassified, then she can indicate some of those misclassified documents to TW2 as relevance feedback. Because normally the user is only interested in the top 10 to 50 ranked documents, R can be tuned between 10 and 50. In order to provide a better view of the classification made by TW2, sometimes the system may give the user a second list of R documents which are ranked below top R. One way for selecting documents in the second list is to randomly select R documents ranked below top R. Another way is to select the bottom ranked R documents.

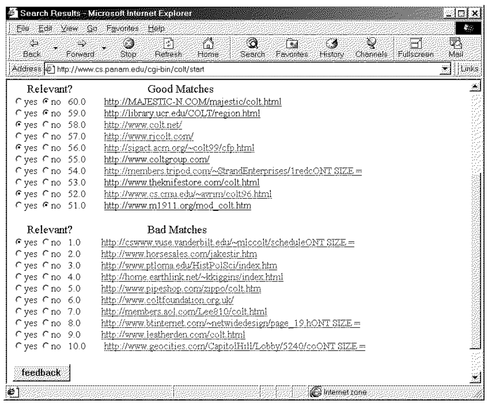
5.3. The Architecture of WebSail

WebSail is a real-time adaptive web search learner we designed and implemented during the winter break of 1999-2000. It is implemented on a Sun Ultra-1 workstation with storage of 27 Giga-bytes hard disk on an IBM R6000 workstation. Our goal of the project is to show that TW2 not only works in theory but also works in practice. WebSail employs TW2 as its learning component and is able to help the user to search for the desired documents with as little relevance feedback as possible. Its architecture is given in Figure 1. A demonstration version of WebSail is available for interested readers to access via the url given at the end of the paper.

5.4. How WebSail Works

WebSail has an interface as shown in Figure 2 to allow the user to enter her query and to specify the number of the top matched document urls to be returned. WebSail maintains an internal index database of 834,000 documents. Each of those documents is indexed with about 300 keywords. It also has a meta-search component to query AltaVista [a] whenever needed. When the user enters a query and starts a search process, WebSail first searches its internal index database. If no relevant documents can be found within its database then it receives a list of top matched documents externally with the help of its meta-search component. WebSail displays the search result to the user in a format as shown in Figure 3.

Also as shown in Figure 3, at each interaction we provide the top R (normally 10) and the bottom R ranked document urls. Each document url is preceded by two radio buttons for


Search Results - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address
Links
feedback
Internet zone

Address: http://www.cs.panam.edu/cgi-bin/colt/start

Relevant?		Good Matches
yes [✓ no]	60.0	http://MAJESTIC-N.COM/majestic/colt.html
yes [✓ no]	59.0	http://library.ucr.edu/COLT/region.html
[✓ yes] no	58.0	http://www.colt.net/
yes no	57.0	http://www.rjcolt.com/
[✓ yes] no	56.0	http://sigact.acm.org/~colt99/cfp.html
yes no	55.0	http://www.coltgroup.com/
yes no	54.0	http://members.tripod.com/~StrandEnterprises/1redcONT SIZE =
yes no	53.0	http://www.theknifestore.com/colt.html
[✓ yes] no	52.0	http://www.cs.cmu.edu/~avrim/colt96.html
yes [✓ no]	51.0	http://www.m1911.org/mod_colt.htm

Relevant?		Bad Matches
[✓ yes] no	1.0	http://cswww.vuse.vanderbilt.edu/~mlccolt/scheduleONT SIZE =
yes no	2.0	http://www.horsesales.com/jakestir.htm
yes no	3.0	http://www.ptloma.edu/HistPolSci/index.htm
yes no	4.0	http://home.earthlink.net/~kkiggins/index.html
yes no	5.0	http://www.pipeshop.com/zippo/colt.htm
yes no	6.0	http://www.coltfoundation.org.uk/
yes no	7.0	http://members.aol.com/Lee810/colt.html
yes no	8.0	http://www.btinternet.com/~netwidedesign/page_19.hONT SIZE =
yes no	9.0	http://www.leatherden.com/colt.html
yes no	10.0	http://www.geocities.com/CapitolHill/Lobby/5240/coONT SIZE =

Figure 3. Initial Query Result for “Colt"

the user to indicate whether the document is relevant to the search query or not[1]. The document urls are clickable for viewing the actual document contents so that the user can judge whether a document is relevant or not more accurately. After the user clicks a few radio buttons, she can click the feedback button to submit the feedback to TW2. WebSail has a function to parse out the feedback provided by the user when the feedback button is clicked. Having received the feedback from the user, TW2 updates its weight vector w and also performs individual document promotions or demotions. It then re-ranks the documents and displays the top R and the bottom R document urls to the user.

At each interaction, the dispatcher of WebSail parses query or relevance feedback information from the interface and decides which of the following components should be invoked to continue the search process: TW2, or Index Database Searcher, or Meta-Searcher. When meta-search is needed, Meta-Searcher is called to query AltaVista [a] to receive a list of the top matched documents. the Meta-Searcher has a parser and an indexer that work in real-time to parse the received documents and to index each of them with at most 64 keywords. The received documents, once indexed, will also be cached in the index database. After this, WebSail uses TW2 and the ranking function f to process relevance feedback. This kind of real-time client side meta-search learning feature has been expanded in our project Yarrow [7].

5.5. The Performance

The actual performance of WebSail is promising. We use the following relative Recall to measure the performance of WebSail: For any query q, the relative Recall is

$$R_{Recall}(q) \quad = \quad \frac{R_{20}}{R},$$

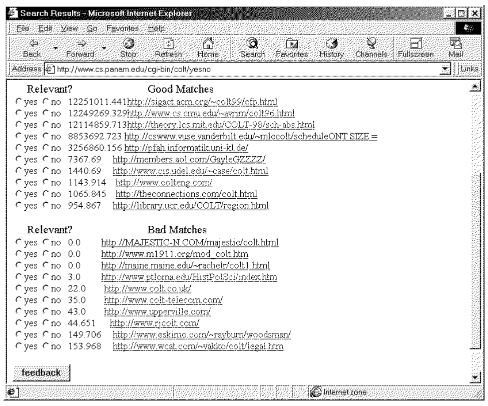
where R is the total number of relevant documents among the list of m retrieved documents, and R_{20} is the number of relevant documents ranked among the top 20 positions in the final search result of the search engine. We have selected 100 queries to calculate the average relative Recall of WebSail. Each query is represented by a collection of at most 5 words. For each query, we tested WebSail with the returning document number m as 50, 100, 150, 200, respectively. For each test, we recorded the number of interactions used and the number of documents judged by the user. The relative Recall was calculated based on manual examination of the relevance of the returned documents. Our experiments reveal that WebSail achieves an average of 0.95 relative Recall with an average of 3.72 interactions and an average of 13.46 documents judged as relevance feedback.

6. Concluding Remarks

Web search, an interface between the human users and the vast information gold mine of the web, 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 on-line 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 the real-world problems such as web search. Our goal in this paper is to take Winnow2 [17] as a starting point to investigate, in theory and in practice, the applicability of the well-studied learning algorithms to the real-world problem of web search. We design a tailored version TW2 of Winnow2, which has small enough mistake bounds for practical application to web search. We have designed and implemented a real-time adaptive web search learner WebSail with TW2 as its learning component. WebSail shows that TW2 indeed works effectively in practice.

Because on-line learning is incremental and depends on the history of a learning process to improve learning performance, WebSail creates and maintains a specific thread for each web search process. When a search process is finished, its related thread will be terminated. We believe from our experience that on-line learning algorithms such as TW2 can be well employed at the client side as a plug-in component of the web browser to effectively help the user search the web. However, it may not be very realistic to employ an on-line learning algorithm at a popular web server side. Because a

[1]The search process shown in Figures 3 and 4 was performed on January 25, 2000. The query word is *“colt"* and the desired web documents are those related to “computational learning theory". After 3 interactions and 9 relevant and irrelevant documents judged by the user as relevance feedback, all the colt related web documents among the initial 100 matched ones were moved to the top 10 positions.


Search Results - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail
Address
Links
feedback
Internet zone

```
http://www.cs.panam.edu/cgi-bin/colt/yesno

   Relevant?               Good Matches
○ yes ○ no  12251011.441http://sigact.acm.org/~colt99/cfp.html
○ yes ○ no  12249269.329http://www.cs.cmu.edu/~avrim/colt96.html
○ yes ○ no  12114859.713http://theory.lcs.mit.edu/COLT-98/sch-abs.html
○ yes ○ no  8853692.723 http://cswww.vuse.vanderbilt.edu/~miccolt/scheduleONT SIZE =
○ yes ○ no  3256860.156 http://pfah.informatik.uni-kl.de/
○ yes ○ no  7367.69     http://members.aol.com/GayleGZZZZ/
○ yes ○ no  1440.69     http://www.cis.udel.edu/~case/colt.html
○ yes ○ no  1143.914    http://www.colteng.com/
○ yes ○ no  1065.845    http://theconnections.com/colt.html
○ yes ○ no  954.867     http://library.ucr.edu/COLT/region.html

   Relevant?               Bad Matches
○ yes ○ no  0.0         http://MAJESTIC-N.COM/majestic/colt.html
○ yes ○ no  0.0         http://www.m1911.org/mod_colt.htm
○ yes ○ no  0.0         http://maine.maine.edu/~rachelr/colt1.html
○ yes ○ no  3.0         http://www.ptloma.edu/HistPolSci/index.htm
○ yes ○ no  22.0        http://www.colt.co.uk/
○ yes ○ no  35.0        http://www.colt-telecom.com/
○ yes ○ no  43.0        http://www.upperville.com/
○ yes ○ no  44.651      http://www.rjcolt.com/
○ yes ○ no  149.706     http://www.eskimo.com/~rayburn/woodsman/
○ yes ○ no  153.968     http://www.wcat.com/~vakko/colt/legal.htm
```

Figure 4. Refined Result for “colt" after 3 Interactions and 9 Examples

popular web server may have thousands of users accessing it in every single minute, it cannot afford to maintain too many threads for individual search processes.

URL References:

[a] AltaVista: www.altavista.com.
[b] Yahoo!: www.yahoo.com.
[c] Google: www.google.com.
[d] WebSail:
www.cs.panam.edu/chen/WebSearch/WebSail.html.
[e] Yarrow:
www.cs.panam.edu/chen/WebSearch/Yarrow.html.

References

[1] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–432, 1987.

[2] R. Baeza-Yates and B. Riberiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[3] K. Bollacker, S. Lawrence, and C. L. Giles. Citeseer: An autonomous web agent for automatic retrieval and identification of interesting publications. In *Proceedings of the Second International Conference on Autonomous Agents*, pages 116–113, New York, 1998. ACM Press.

[4] K. Bollacker, S. Lawrence, and C. L. Giles. A system for automatic personalized tracking of scientific literature on the web. In *Proceedings of the Fourth ACM Conference on Digital Libraries*, pages 105–113, New York, 1999. ACM Press.

[5] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the Seventh World Wide Web Conference*, 1998.

[6] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J. Kleinberg. Automatic resource compilation by analyzing hyperlink structure and associated text. In *Proceedings of the Seventh World Wide Web Conference*, pages 65–74, 1998.

[7] Z. Chen and X. Meng. Yarrow: A real-time client site meta search learner. Proceedings of the AAAI 2000 Workshop on Artificial Intelligence for Web Search, July 2000.

[8] Z. Chen, X. Meng, and R. H. Fowler. Searching the web with queries. *Knowledge and Information Systems*, 1:369–375, 1999.

[9] Z. Chen and B. Zhu. Some formal analysis of the rocchio's similarity-based relevance feedback algorithm. Technical Report CS-00-22, Dept. of Computer Science, University of Texas-Pan American, March 2000.

[10] D. Gibson, J. Kleinberg, and P. Raghavan. Inferring web communities from link topology. In *Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia*, 1998.

[11] E. Ide. Interactive search strategies and dynamic file organization in information retrieval. In G. Salton, editor, *The Smart System - Experiments in Automatic Document Processing*, pages 373–393, Englewood Cliffs, NJ, 1971. Prentice-Hall Inc.

[12] E. Ide. New experiments in relevance feedback. In G. Salton, editor, *The Smart System - Experiments in Automatic Document Processing*, pages 337–354, Englewood Cliffs, NJ, 1971. Prentice-Hall Inc.

[13] J. J.J. Rocchio. Relevance feedback in information retrieval. In G. Salton, editor, *The Smart Retrieval System - Experiments in Automatic Document Processing*, pages 313–323, Englewood Cliffs, NJ, 1971. Prentice-Hall, Inc.

[14] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of ACM*, 46(5):604–632, 1999.

[15] S. Lawrence, K. Bollacker, and C. L. Giles. Indexing and retrieval of scientific literature. In *Proceedings of the Eighth ACM International Conference on Information and Knowledge Management*, 1999.

[16] D. Lewis. Learning in intelligent information retrieval. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 235–239, 1991.

[17] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[18] W. Maass and M. Warmuth. Efficient learning with virtual threshold gates. *Information and Computation*, 141(1):66–83, 1998.

[19] A. Nakamura and N. Abe. Collaborative filtering using weighted majority prediction algorithms. In *Machine Learning: Proceedings of the Fifteenth International Conference*, 1998.

[20] J. Rennie and A. McCallum. Using reinforcement learning to spider the web efficiently. In *Proceedings of the Sixteenth International Conference on Machine Learning*, 1999.

[21] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1989.

[22] G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Comm. of ACM*, 18(11):613–620, 1975.

[23] B. Yuwono and D. Lee. Search and ranking algorithms for locating resources on the world wide web. In *Proceedings of the International Conference on Data Engineering*, pages 164–171, New Orleans, USA, 1996.