WebSail: From On-line Learning to Web Search

Zhixiang Chen* Xiannong Meng* Binhai Zhu† Richard H. Fowler*

**Department of Computer Science, University of Texas-Pan American*

Edinburg, TX 78539, USA. Emails: chen@cs.panam.edu

meng@cs.panam.edu, fowler@panam.edu

†*Department of Computer Science, Montana State University*

Bozeman, MT 59717, USA. Email: bhz@cs.montana.edu.

Abstract

In this paper we report our research on building *WebSail* – an intelligent web search engine that is able to perform real-time adaptive learning. *WebSail* learns from the user's relevance feedback, so that it is able to speed up its search process and to enhance its search performance. We design an efficient adaptive learning algorithm $TW2$ to search for web documents. *WebSail* employs $TW2$ together with an internal index database and a real-time meta-searcher to perform real-time adaptive learning to find desired documents with as little relevance feedback from the user as possible. The architecture and performance of *WebSail* are also discussed.

Keywords: web search, adaptive learning, relevance feedback, vector space, document ranking.

1 Introduction

In this paper, we investigate the applicability of adaptive learning algorithms to web search. We design a tailored version $TW2$ of the well-known algorithm Winnow2 [10] for the particular purpose of web search. We have implemented a real-time adaptive web search learner WebSail [d] with $TW2$ as its learning component. 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. WebSail has been 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 of about 800,000 documents 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 [2, 9], and collaborative filtering [11, 1]. Our work related to this paper can be found in [6, 7, 5].

The rest of the paper is organized as follows. In section 2, we examine the property of web document indexing and design the learning algorithm $TW2$ for web search. In section 3, 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 4.

2 The Algorithm $TW2$

When a set of n binary-valued index terms (or keywords) $T_1, \ldots, T_n$ are used to index web documents, a document is represented as a vector in the n-dimensional binary vector space $\{0,1\}^n$. Given any document d, let $v_d(x_1, \ldots, x_n)$ denote its vector representation, where for $i = 1, \ldots, n$, x_i is a binary-valued variable for the index term T_i. If $x_i = 1$ in the vector v_d, then the document d has the index term T_i, otherwise d does not. Define

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

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 index terms. Many *theoretically efficient* algorithms exist for learning disjunctions of at most k relevant index terms. But few are applicable to web search, because a user may have no patience to try, say, more than 10 iterations of learning. The challenging question for us is whether we can design an efficient learning algorithm that can yield significant search precision increase with about four to five iterations of learning from the user's relevance feedback. Moreover, the algorithm must be *practically efficient* to process its learning tasks.

Definition 2.1. *Given any index term x and any document d, x is said to be an index term for the document d if d has x, or in other words, the corresponding component of the index term x in the document vector v_d is* 1.

Although a huge collection of index terms (in the simplest case, keywords) are needed and used to index web documents, for each particular document d, the number of its index terms 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 few kilobytes of a document is extracted, when it is long.

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 four different ranking methods in [12] and PageRank in [3] which is the key ranking method for Google [c]. 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 pay more attention to the propagation of the influence of the relevant documents judged by the user.

We introduce $TW2$, a tailored version of Winnow2, to exploit the particular nature of web search. As in Winnow2, $TW2$ maintains non-negative real-valued weights $w_1, \ldots, w_n$ for index terms $T_1, \ldots, T_n$, respectively. It also maintains a real-valued threshold θ. $TW2$ classifies a document d with its document vector $v_d = (x_1, \ldots, x_n)$ as relevant if

$$w_1 x_1 + \cdots + w_n x_n \geq \theta,$$

and irrelevant otherwise. $TW2$ also uses the value $w_1 x_1 + \cdots + w_n x_n$ to rank the document d. In contrast to the fact that Winnow2 sets all initial weights to 1, $TW2$ sets all initial weights to 0 and has a different promotion process accordingly. Another substantial difference between $TW2$ and and Winnow2 is that $TW2$ accepts document examples that may not contradict to its current classification to promote or demote its weight vector. One must notice that Winnow2 only accepts examples that contradict to its current classification to perform promotion or demotion. The rationale behind setting all the initial weights to 0 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 realistic because existing effective document ranking mechanisms can be coupled with the learning process as discussed in next section.

```
Algorithm $TW2(\mathbf{w}_0, \alpha, \theta)$:
(i) Inputs:
        $\mathbf{w}_0$ is the non-negative initial weight vector
        $\alpha > 1$ is the updating factor
        $\theta \geq 0$ is the classification threshold
(ii) Set $k = 0$.
(iii) Classify and rank documents with $\mathbf{w}_k$ and $\theta$.
(iv) While (the user judged the relevance of a document d) do {
        for $i = 1, \ldots, n$, do {
            /* $\mathbf{w}_k = (w_{1,k}, \ldots, w_{n,k})$, $v_d = (x_1, \ldots, x_n)$ */
            if ($x_i \neq 0$) {
                if ($w_{ik} \neq 0$) set $w_{i,k+1} = w_{i,k}$ else set $w_{i,k+1} = 1$ /* adjustment */
                if (d is relevant)   /* promotion */
                    set $w_{i,k+1} = \alpha w_{i,k+1}$
                else   /* demotion */
                    set $w_{i,k+1} = \frac{w_{i,k+1}}{\alpha}$
            }
        }
    }
(v) If the user has not judged any document in the k-th step, then stop. Otherwise, let $k = k + 1$ and go
to step (iv).
/* The end of the algorithm TW2 */
```

At each iteration, if the user's judgment of a document contradicts to the algorithm's classification, we say that $TW2$ makes a mistake. Let A denote the total number of distinct index terms of all the relevant documents that have been 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 index terms over the n-dimensional binary vector space, $TW2$ makes at most $\frac{\alpha^2 A}{(\alpha-1)\theta} + (\alpha+1)k \log_\alpha \theta - \alpha$ mistakes.*
- *When on the average l out of k relevant index terms appear as index terms for any relevant document judged by the user during the learning process, the bound in Theorem 3.2 is improved to $\frac{\alpha^2 A}{(\alpha-1)\theta} + \frac{(\alpha+1)k}{l} \log_\alpha \theta - \alpha$ on the average.*

As our work in [8, 4] indicated, $TW2$ has substantially better worst-case performance than the popular Rocchio's similarity-based relevance feedback algorithm in learning disjunctions of a small number of relevant index terms, and the multiplicative weight updating method can boost the usefulness of an index term exponentially.

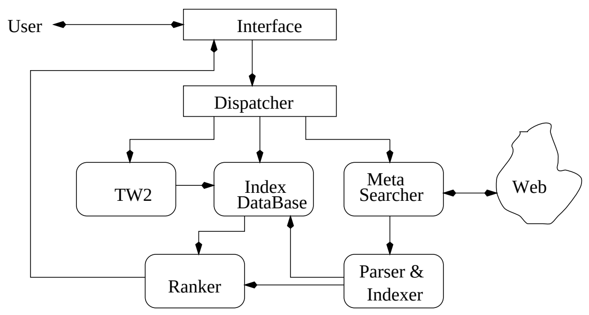


Figure 1: Architecture of WebSail

3 The WebSail

3.1 Document Ranking

Let $\mathbf{w} = (w_1, \ldots, w_n)$ 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 $v_d = (x_1, \ldots, x_n)$,

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

g remains constant for each document d during the learning process of $TW2$. Various strategies can be used to define g, for example, PageRank [3], the classical tf-idf scheme, vector spread, or cited-based rankings [12]. 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 $TW2$. 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 $TW2$.

3.2 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 as an approximation to the classification made by $TW2$. The user can examine documents in this short list. If she feels satisfactory with the result then she can end the search process. Otherwise, she can judge some 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.

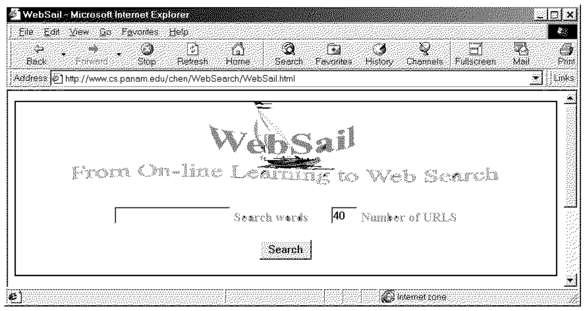

WebSail - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print
Address http://www.cs.panam.edu/chen/WebSearch/WebSail.html
Links
WebSail
From On-line Learning to Web Search
Search words
40
Number of URLS
Search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

Figure 2: Interface of WebSail

3.3 The Architecture of WebSail

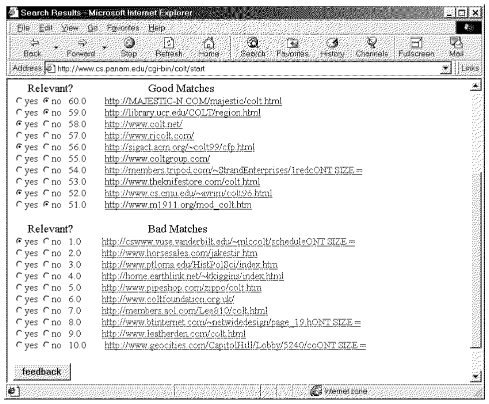
WebSail is a real-time adaptive web search learner we have designed and implemented on a Sun Ultra one workstation with storage of 27 Giga-bytes hard disk on an IBM R6000 workstation. Our goal of the project is to show that $TW2$ not only works in theory but also works in practice. WebSail employs $TW2$ 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.

3.4 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 about 800,000 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 iteration we provide the top R (normally 10) and the bottom R ranked document urls. Each document url is preceded by two radio buttons for 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 $\mathbf{w}$ and also performs individual

[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 iterations 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
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"

document promotions or demotions. It then re-ranks the documents and displays the top R and the bottom R document urls to the user.

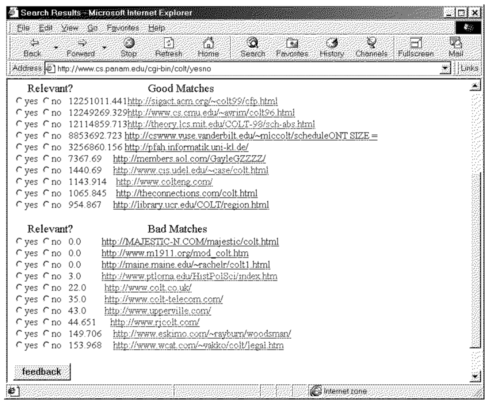
At each iteration, the dispatcher of WebSail parses query or relevance feedback information from the interface and decides which of the following components should be invoked to continue the search process: $TW2$, or Index Database Searcher, or Meta-Searcher. When meta-search is needed, Meta-Searcher 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.

3.5 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

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

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 keywords. 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 iterations 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 iterations and an average of 13.46 documents judged as relevance feedback.



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

```
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 Iterations and 9 Examples

4 Concluding Remarks and Future Work

Designing practically effective web search algorithms is a challenging task. It calls for innovative methods and strategies from many fields including machine learning. However, few learning algorithms are ready to be used in web search because of a number of realistic requirements. We feel that the fundamental question about intelligent web search is whether one can design an adaptive learning to achieve significant search precision increase with about *four* to *five* iterations of learning from the user's relevance feedback. 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.

In the future we plan to expand WebSail to work in the non-binary vector space. Algorithm $TW2$ is not suitable for learning in the non-binary vector space. We need to design new adaptive learning algorithms and some progresses have been made in [4]. Because web search users are usually reluctant to try many iterations to search for the desired documents, real-time clustering techniques and index term learning techniques may be coupled with $TW2$ to reduce the number of iterations required for a web search process. In our recent project FEATURES [7] the coupling of $TW2$ with index term learning shows promising performance increase.

Acknowledgment. The work in this paper is supported in part by a NASA grant NAG9-1169. The authors thank Sergio Reyes and Joe Noyola who worked in the early stage of implementing WebSail. Under the support of a 1999-2000 UTPA Faculty Research Council grant they implemented a C++ class template to generate html files. Part of the third author's work was done when he was at City University of Hong Kong, Hong Kong, China. Finally, we thank the anonymous referees for their valuable comments to help us to revise this paper.

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.

References

[1] D. Billsus and M. J. Pazzani. Learning collaborative information filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998.

[2] K. Bollacker, S. Lawrence, and C. Lee 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.

[3] 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.

[4] Z. Chen. Multiplicative adaptive algorithms for user preference retrieval. submitted for publication, February, 2001.

[5] Z. Chen and X. Meng. Yarrow: A real-time client site meta search learner. In *Proceedings of the AAAI 2000 Workshop on Artificial Intelligence for Web Search*, pages 12–17, Austin, July 2000.

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

[7] Z. Chen, X. Meng, R.H. Fowler, and B. Zhu. FEATURES: Real-time adaptive feature and document learning for web search. *Journal of the American Society for Information Science*, in press, 2001.

[8] Z. Chen and B. Zhu. Some formal analysis of the rocchio's similarity-based relevance feedback algorithm. In D.T. Lee and Shang-Hua Teng, editors, *Proceedings of the Eleventh International Symposium on Algorithms and Computation, Lecture Notes in Computer Science 1969*, pages 108–119. Springer-Verlag, 2000.

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

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

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

[12] B. Yuwono and D.L. 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.