Learning to Query the Web

William W. Cohen and **Yoram Singer**
AT&T Labs
600 Mountain Ave., Murray Hill, NJ 07974
{wcohen,singer}@research.att.com

Abstract

The World Wide Web (WWW) is filled with "resource directories"—*i.e.*, documents that collect together links to all known documents on a specific topic. Keeping resource directories up-to-date is difficult because of the rapid growth in online documents. We propose using machine learning methods to address this problem. In particular, we propose to treat a resource directory as a list of positive examples of an unknown concept, and then use machine learning methods to construct from these examples a definition of the unknown concept. If the learned definition is in the appropriate form, it can be translated into a query, or series of queries, for a WWW search engine. This query can be used at a later date to detect any new instances of the concept. We present experimental results with two implemented systems, and two learning methods. One system is interactive, and is implemented as an augmented WWW browser; the other is a batch system, which can collect and label documents without any human intervention. The learning methods are the RIPPER rule learning system, and a rule-learning version of a new online weight allocation algorithm called the *sleeping experts* prediction algorithm. The experiments are performed on real data obtained from the WWW.

Introduction

The World Wide Web (WWW) is currently filled with documents that collect together links to all known documents on a topic; henceforth, we will refer to documents of this sort as *resource directories*. While resource directories are often valuable, they are difficult to create and maintain. Maintenance is especially problematic since the rapid growth in on-line documents makes it difficult to keep a resource directory up-to-date.

We propose using machine learning methods to address the resource directory maintenance problem. In particular, we propose to treat a resource directory as an extensional definition of an unknown concept—*i.e.*, documents pointed to by the resource list will be considered positive examples of the unknown concept, and all other documents will be considered negative examples of the concept. Machine learning methods can then be used to construct from these examples an *intensional* definition of the concept. If an appropriate learning method is used, this definition can be translated into a query for a WWW search engine, such as Altavista, Infoseek or Lycos. If the query is accurate, then re-submitting the query at a later date will detect any new instances of the concept that have been added to the WWW domain.

We will present experimental results on this problem with two implemented "query-learning" systems. One is an interactive system—an augmented WWW browser that allows the user label documents, and then to learn a search query from previously labeled documents. This system is useful in locating documents similar to those in a resource directory, and thus making it more comprehensive. The other is a batch system which repeatedly learns queries from examples, and then collects and labels documents using these queries. In labeling examples, this system assumes that the original resource directory is complete, and hence can only be used with a nearly exhaustive initial resource directory; however, it can operate without human intervention.

In the experiments we performed, these systems are shown to produce usefully accurate intensional descriptions of concepts: in two of three tests problems, the concepts produced were much more comprehensive than manually-constructed lists, with only a modest loss in precision.

This problem is a variant of the problem of *relevance feedback*, which is well-studied in information retrieval. One novel aspect of the work in this paper (other than the WWW-based setting) is that we will focus, as much as is practical, on learning methods that are *independent* of the search engine used to answer a query. This emphasis seems natural in a WWW setting, as there are currently a number of general-purpose WWW search engines, all under constant development, none clearly superior to the others, and none directly supporting relevance feedback (at the time of this writing); hence it seems inappropriate to rely too heavily on a single search engine.

Two initial implementations

A batch system

The first implementation is a Perl script that runs as a "batch" system—*i.e.*, it requires no user intervention. The input of the batch system is a list of URL's that correspond to the positive examples of an unknown concept. The batch system has two outputs: an intensional representation of the unknown concept, and a set of example documents that include all of the positive examples plus a sample of negative examples.

The procedure used to accomplish this is summarized in Figure 1. Three subroutines are used. The first, **Learn**, learns a concept from a sample. The only assumption made by the query-learning system about the learning subsystem is that the hypothesis of the learning subsystem is in disjunctive normal form (DNF), where the primitive conditions test for the presence of words. For example, a DNF hypothesis learned from a resource list on college basketball might be: $(\text{college} \wedge \text{basketball}) \vee (\text{college} \wedge \text{hoops}) \vee (\text{NCAA} \wedge \text{basketball})$. Henceforth we will call each term (conjunction) in this DNF a "rule".

A set of k rules can be easily converted to k search queries, each of which consists of a conjunction of words—a query format that is supported by practically every search engine. The restriction, therefore, makes the system largely independent of the search engine used.

The second subroutine used by the query-learning system, **CorrespondingQuery**, converts a single rule to a query for the search engine being used. Some knowledge about the search engine is clearly needed to appropriately encode the query; however, because most search engines use similar formats for queries, adding the knowledge needed to support a new search engine is usually straightforward.[1]

The final subroutine, **Top-k-Documents**, submits a query to a search engine and collects the top k documents returned. Again, some knowledge about the search engine is needed to perform this task.

The basic procedure followed by the batch query-learner is to repeatedly learn a set of rules, convert these rules to queries, and then use incorrect responses to these queries as negative examples. The premise behind this approach is that the responses to learned queries will be more useful than randomly selected documents in determining the boundary of the concept. Although this simple method works reasonably well, and can be easily implemented with existing search engines, we suspect that other strategies for collecting examples may be competitive or superior; for instance, promising results have been obtained with "uncertainty sampling" (Lewis and Gale 1994) and "query by committee" (Seung *et al.* 1992; Freund *et al.* 1992; Dagan and Engelson 1995).

[1] Some search engines can handle more expressive queries—*e.g.* queries that require terms to appear near each other, or queries that contain word stems like "comput*". Most advanced queries are not currently supported by the existing **CorrespondingQuery** routine. One exception are queries containing conditions that check for the *absence* (rather than the presence) of words, such as $(\text{basketball} \wedge \neg\text{college} \wedge \neg\text{NCAA})$. These can be used if both the learning subsystem and the query system allow it, but were not used in any of the experiments of this paper.

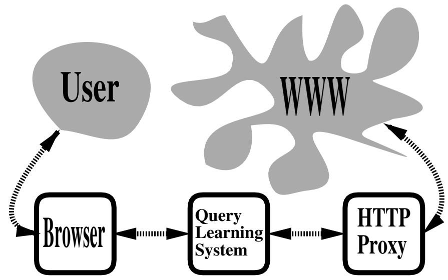

Figure 2: Implementation of the interactive query-learning system. HTML pages delivered from the HTTP proxy server to the browser are augmented by adding certain additional special links. Requests corresponding to these special links are trapped by the query learning system, which performs the appropriate action (e.g., learning or searching) and returns an HTML page summarizing the result of the action to the browser.

A few final details require some discussion.

Constraining the initial query: To construct the first query, a large set of documents were used as *default negative* examples. A "default negative example" is treated as a ordinary negative example unless it has already been labeled as positive example, in which case the example is ignored. We used 363 documents collected from a cache used by our labs' HTTP proxy server as default negative examples.

Termination: In the current implementation, the process of learning rules and then collecting negative examples is repeated until some resource limit set by the user is exceeded. Currently the user can limit the number of negative examples collected, and the number of times the learning subsystem is called.

Avoiding looping: It may be that on a particular iteration, no new documents are collected. If this occurs, then the training data on the next iteration will be the same as the training data on the previous iteration, and the system will loop. To avoid this problem, if no new documents are collected on a cycle, heuristics are used to vary the parameters of the learning subsystem for the next cycle. In the current implementation, two heuristics are followed: if the hypothesis of the learning subsystem is an empty ruleset, then the cost of a false negative is raised; otherwise, the cost of a false positive is raised. The proper application of these heuristics, of course, depends on the learning subsystem being used.

An interactive system

The batch system assumes that every document not on the resource list is a negative example. This means

```
Program Batch-Query-Learner(positive-URLs,SearchEngine)
    PosSample := {(d, +) : URL(d) ∈ positive-URLs}
    NegSample := {(d, −) : d was "recently accessed" and d ∉ positive-URLs}
    Sample := PosSample ∪ NegSample
    repeat
        RuleSet := Call Learn(Sample)
        for each r ∈ RuleSet do
            q := Call CorrespondingQuery(r,SearchEngine)
            Response := Call Top-k-Documents(q,SearchEngine)
            for each document d ∈ (Response − positive-URLs) do Sample := Sample ∪(d, −)
        endfor
        if no new documents collected then adjust parameters of Learn
        else reset parameters of Learn to default values
    until some resource limit exceeded (see text);
end
```

Figure 1: Batch query-learning system

that it cannot be used successfully unless one is confident that the initial set of documents is reasonably complete. In practice this is seldom the case. For this reason, we also implemented an interactive query-learning system, which does not assume completeness of an initial set of positive examples. Instead, it relies on the user to provide appropriate labels.

The interactive system does not force any particular fixed sequence for collecting documents and labeling them; instead it is simply an augmented WWW browser, which allows the user to label the document being browsed, to invoke the learning subsystem, or to conduct a search using previously learned rules.

The architecture of the interactive system is shown in Figure 2. The user's interface to the query-learning system is implemented as a separate module that is interposed between a WWW browser and an HTTP proxy server. This module performs two main jobs. First, every HTML document that is transmitted from the proxy server to the browser is augmented, before being sent to the browser, by adding a small amount of text, and a small number of special links at the beginning of the document. Second, while most HTTP requests generated by the browser are passed along unmodified to the proxy server, the HTTP requests that are generated by clicking on the special inserted links are trapped out and treated specially.

This implementation has the advantage of being browser-independent. Following current practice in this research area, we have given this system a catchy name–*Swimsuit*.[2]

Functionally, the special links act as additional "control buttons" for the browser—similar to the buttons labeled "Back" and "Net Search" on the Netscape browser. By clicking on special links, the user can classify pages, invoke the learning subsystem, and so on. The user's view of the interactive system is illustrated in Figure 3.

The special additional links are:

Document labeling: The *yes* link and *no* link allow the user to classify the current page as a positive (respectively negative) example of the current class.

Invoking the learner: The *learn* link returns a form that allows the user to set options for the actual learning subsystem and/or invoke the learner on the current class. The behavior of this link can be easily changed, so that different learning subsystems can be used in experiments. As in the batch system, learning is normally constrained by using default negative examples. This means that reasonable rules can often be found even if only a few positive examples are marked.

Searching: The *search* link returns a list of previously learned rules. Clicking on any rule will submit the corresponding query to the currently selected search engine, and return the result.

Configuration and help: The *set options* link returns a form that allows the user to change the current class (or to name a new class), or to change the current search engine; the *review previous* link returns an HTML page that lists all previously marked examples of the current class; and the *help* link returns a help page.

Learning Subsystems

Two learning subsystems have been integrated with the system: RIPPER, a propositional rule learner that is related to FOIL (Quinlan 1990), and a rule-learning version of "Sleeping experts". *Sleeping experts* is a new prediction algorithm that combines ideas used for online prediction (Freund and Schapire 1995) with the infinite attribute model of Blum (1990).

These algorithms have different strengths and weaknesses. RIPPER implicitly assumes that examples are i.i.d.—which is not the case for samples collected via browsing or by the batch query-learning system. However, formal results suggest that sleeping experts will perform well even on datasets that are selected in a

[2] Surfing While Inducing Methods to Search for URLs by Included Terms.

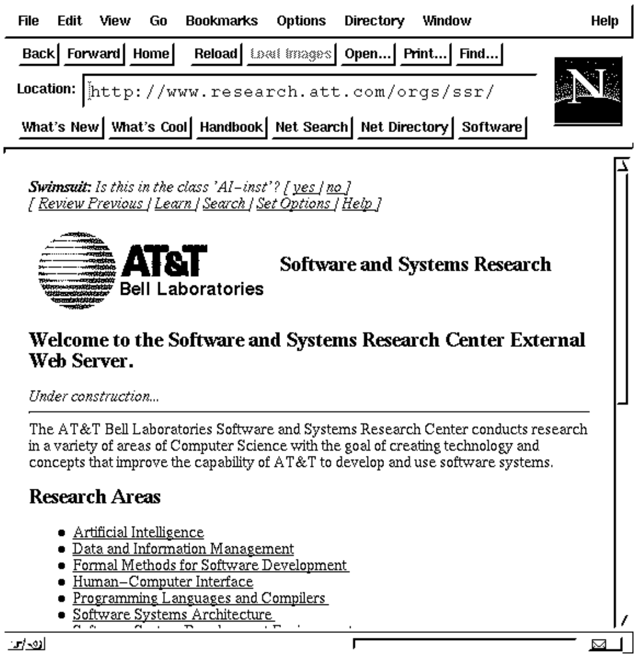

Figure 3: User interface to the query-learning system. The user's view of the query-learning system is a set of special links that appear at the top of each HTML page. Clicking on these links allows the user to perform operations such as classifying a document or invoking the learning subsystem.

non-random manner. The sleeping experts algorithm is also largely incremental, which is potentially an advantage is this setting. On the other hand, sleeping experts uses a more restricted hypothesis space, and cannot learn large rules, whereas RIPPER can (at least in principle).

RIPPER

Briefly, RIPPER builds a set of rules by repeatedly adding rules to an empty ruleset until all positive examples are covered. Rules are formed by greedily adding conditions to the antecedent of a rule with an empty antecedent until no negative examples are covered. After a ruleset is constructed, a optimization postpass massages the ruleset so as to reduce its size and improve its fit to the training data. A combination of cross-validation and minimum-description length techniques are used to prevent over-fitting. In previous experiments, RIPPER was shown to be comparable to C4.5rules (Quinlan 1994) in terms of generalization accuracy, but much faster for large noisy datasets. For more detail, see Cohen (1995a).

The version of RIPPER used here was extended to handle "set-valued features", as described in (Cohen 1996). In this implementation of RIPPER, the value of a feature can be a *set* of symbols, rather than (say) a number or a single symbol. The primitive conditions that are allowed for a set-valued feature F are of the form $c \in F$ or (optionally) $c \notin F$, where c is any constant value that appears as a value of F in the dataset. This leads to a natural way of representing documents: a document is represented by a single feature, the value of which is the set of all *tokens*[3] appearing in the document.

A second extension to RIPPER allows the user to specify a *loss ratio* (Lewis and Catlett 1994). A loss ratio indicates the ratio of the *cost* of a false negative error to the *cost* of a false positive error. By changing the loss ratio, the user can affect the recall and precision[4] of a learned classifier.

One additional modification to RIPPER was also made specifically to improve performance on the query-learning task. The basic RIPPER algorithm is biased toward producing simple, and hence general, conjunctions; for example, for RIPPER, when a conjunction of conditions is specific enough to cover no negative examples, no further conditions will be added. This bias appears to be inappropriate in learning queries, where the concepts to be learned are typically extremely specific. Thus, we added a postpass to RIPPER that adds to each rule all[5] conditions that are true for every positive example covered by the rule. (We note that a similar scheme has been investigated in the context of the "small disjunct problem" (Holte *et al.* 1989).) The postpass implements a bias towards specific rules rather than general rules.

Sleeping Experts

In the past few years there has been a growing interest in online prediction algorithms. The vast majority of these prediction algorithms are given a pool of fixed "experts"—each of which is a simple, fixed, classifier—

[3] In the experiments, documents were tokenized by deleting e-mail addresses, HTML special characters, and HTML markup commands; converting punctuation to spaces; converting upper to lower case; removing words from a standard stop-list (Lewis 1992, page 117) and finally treating every remaining sequence of alphanumeric characters as a token. To keep performance from being degraded by very large documents, we only used tokens from the first 100 lines of a file. This also approximates the behavior of some search engines, which typically index only the initial section of a document.

[4] *Recall* is the fraction of the time that an actual positive example is predicted to be positive by the classifier, and *precision* is the fraction of the time that an example predicted to be positive is actually positive. For convenience, we will define the precision of a classifier that always prefers the class negative as 1.00.

[5] Actually, the number of conditions added was limited to a constant k—in the experiments below, to $k = 20$. Without this restriction, a rule that covers a group of documents that are nearly identical could be nearly as long as the documents themselves; many search engines do not handle gracefully very long queries.

and build a *master algorithm*, which combines the classifications of the experts in some manner. Typically, the master algorithm classifies an example by using a weighted combination of the predictions of the experts. Building a good master algorithm is thus a matter of finding an appropriate weight for each of the experts.

Formal results show that by using a multiplicative weight update (Littlestone and Warmuth 1994), the master algorithm is able to maintain a set of weights such that the predictions of the master algorithm are almost as good as the best expert in the pool, even for a sequence of prediction problems that is chosen by an adversary. In some cases even stronger results can be shown—for instance, some algorithms can be guaranteed to perform nearly as well as the best weighted combination of the experts.

The *sleeping experts* algorithm is a procedure of this type. It is based on two recent advances in multiplicative update algorithms. The first is a weight allocation algorithm called Hedge, due to Freund and Schapire (1995), which is applicable to a broad class of learning problems and loss functions. The second is the *infinite attribute model* of Blum (1990). In this setting, there may be any number of experts, but only a few actually post predictions on any given example; the remainder are said to be "sleeping" on that example. A multiplicative update algorithm for the infinite attribute model based on Winnow (Littlestone 1988) has also been implemented (Blum 1995).

Below we summarize the sleeping experts procedure, which combines the Hedge algorithm with the infinite attribute model to efficiently maintain an arbitrarily large pool of experts with an arbitrary loss function.

The master algorithm. Pseudo-code for the algorithm is shown in Fig. 4. The master algorithm maintains a *pool*, which is a set recording which experts have been active on any previous example, and a set of weights, denoted by $\mathbf{p}$, for every expert in the pool. At all times, all weights in $\mathbf{p}$ are nonnegative. However, the weights need not sum to one. At each time step t, the learner is given a new instance x_t to classify; the master algorithm is then given a set W_t of integer indices, which represent the experts that are active (*i.e.*, not "sleeping") on x_t. The prediction of expert i on x_t is denoted by y_i^t. Based on the experts in W_t, the master algorithm must make a prediction for the class of x_t, and then update the pool and the weight set $\mathbf{p}$.

To make a prediction, the master algorithm decides on a distribution $\tilde{\mathbf{p}}$ over the active experts, which is determined by restricting the set of weights $\mathbf{p}$ to the set of active experts W_t and normalizing the weights. We denote the vector of normalized weights by $\tilde{\mathbf{p}}$, where $\tilde{p}_i^t = p_i^t / \sum_{j \in W_t} p_j^t$. The prediction of the master algorithm is $F_\beta(\sum_{i \in W_t} \tilde{p}_i^t y_i^t))$. We use $F_\beta(r) = \ln(1 - r + r\beta)/(\ln(1 - r + r\beta) + \ln((1 - r)\beta + r))$, the function used by Vovk (1990) for predicting binary sequences.

Each active expert i then suffers some "loss" l_i^t. In the implementation described here, this loss is 0 if the expert's prediction is correct and 1 otherwise.

Next, the master algorithm updates the weights of the *active* experts based on the losses. (The weight of the experts who are asleep remains the same, hence we implicitly set $\forall i \notin W_t : \; p_i^{t+1} = p_i^t$.) When an expert is first encountered its weight is initialized to 1. At each time step t, the master algorithm updates the weights of the active experts as follows,

$$\forall i \in W_t : \; p_i^{t+1} = \frac{1}{Z}\, p_i^t \, U_\beta(l_i^t) \;\; , \tag{1}$$

where Z is chosen such that $\sum_{i \in W_t} p_i^t = \sum_{i \in W_t} p_i^{t+1}$. The "update function" U_β is any function satisfying (Cesa-Bianchi *et al.* 1993) $\beta^r \le U_\beta(r) \le 1 - (1 - \beta)r$. In our implementation, we used the linear update, $U_\beta(r) = 1 - (1 - \beta)r$, which is simple to implement and it avoids expensive exponentiations.

Analysis of this algorithm is beyond the scope of this paper and will be presented elsewhere. Briefly, if one defines the loss of the master algorithm to be the average loss with respect to the distribution $\{\tilde{p}_i^t | i \in W_t\}$, the cumulative loss of the master algorithm over all t can be bounded relative to the loss suffered by the best possible fixed weight vector. These bounds hold for *any* sequence of examples $(x_1, y_1), \ldots, (x_t, y_t)$, in particular, the bounds hold for sequences whose instances are not statistically independent.

The pool of experts. It remains to describe the experts used for WWW page classification. In our experiments each expert is corresponds to a *sparse phrase* that appears in a document. That is, if ω_i is the ith token appearing in the document, each expert is of the form $\omega_{i_1}\omega_{i_2}\ldots\omega_{i_k}$ where $1 \le i_1 < i_2 < \ldots i_{k-1} < i_k$ and $i_k - i_1 < n$. This is a generalization of the word n-gram model[6] model. For each sparse phrase we construct two *mini-experts*, one which persistently predicts 0 (not in the class), and one that persistently predicts 1.

Extracting rules from experts. Finally, heuristics are used to construct rules based on the weights constructed by the sleeping experts algorithm. We constructed a rule for each expert that predicts 1 and that has a large weight. This is done by scanning the weights of the combined experts (each combined expert containing two mini-experts) and selecting those which have large weight. More formally, an expert i is used to construct a rule if $p_i^T / \sum_{j \in Pool} p_j^T \ge w_{min}$, where T is the number of training examples, and w_{min} is a weight threshold for extracting experts. In practice, we have found that most of the weight is often concentrated on a few experts (typically less than 8), and hence the

[6]Note that our goal is to *classify* WWW documents; hence each expert representing a phrase is used to predict the classification of the document in which it appears, rather than the next token (word).

Parameters: $\beta \in [0,1]$, number of examples T
Initialize: $Pool \leftarrow \phi$
Do for $t = 1, 2, \ldots, T$

1. Receive a set of active experts W_t and their predictions $\{y_i^t \,|\, i \in W_t\}$.
2. Initialize: $\forall i \in W_t \wedge i \notin Pool :\ p_i^t = 1$.
3. Define a distribution for the set of active experts:
$$\forall i \in W_t,\ \tilde{p}_i^t = \frac{p_i^t}{\sum_{j \in W_t} p_j^t}.$$
4. Predict: $F_\beta(\sum_{i \in W_t} \tilde{p}_i^t y_i^t)$.
5. Compute loss values $\{l_i^t \in [0,1] \,|\, i \in W_t\}$ based on y_i^t and true class.
6. Update weights: $p_i^{t+1} = p_i^t\, U_\beta(l_i^t)$.
7. Renormalize weights: $p_i^{t+1} = p_i^{t+1} \frac{\sum_{i \in W_t} p_i^t}{\sum_{i \in W_t} p_i^{t+1}}$.
8. Update: $Pool \leftarrow Pool \bigcup W_t$.

Figure 4: The Sleeping Experts Prediction Algorithm.

number of experts extracted is not too sensitive to particular choices of w_{min}. We used $w_{min} = 0.0625$ and set the learning rate β to be 0.5 in the experiments described below.

Typically, the "heavy" experts correspond to phrases that frequently appear in documents labeled as positive examples; however, they may also appear in many of the negative labeled documents. We therefore examined the mini-experts of each extracted expert and selected those experts which are statistically correlated only with the positive examples. We define the average prediction $\hat{p}_i$ of expert i, based on its two mini-experts $(i,0)$ and $(i,1)$, to be $\hat{p}_i = F_\beta(p_{i,0}/(p_{i,0}+p_{i,1}))$. An expert is finally chosen to be used as a rule if its average prediction is larger than p_{min}. In the experiments we used $p_{min} = 0.95$ as the default value, and increased or decreased this threshold to encourage proportionally more or fewer positive predictions.

Finally, as was done with RIPPER, we add to each rule the list of all tokens that appear in *all* positive documents covered by a rule. We also remove all rules that have strictly fewer conditions than another rule in the set. The result is a rule set where each rule is of the form $w_{i_1} \wedge w_{i_2} \wedge \ldots \wedge w_{i_k}$. Although the sleeping experts algorithm treats this as a list of tokens, all within a given proximity, we currently treat it simply as a conjunction of features in constructing queries; clearly, this is suboptimal for search engines which support proximity queries.

Experimental results

We have evaluated the system with three resource directories.

ML courses. This list is part of an excellent machine learning resource maintained by David Aha[7]. It contains pointers to 15 on-line descriptions of courses.

AI societies. This is a WWW page jointly maintained by SIGART, IJCAI, and CSCSI. It contains pointers to nine AI societies.

Jogging strollers. This is a list of pointers to discussions of, evaluations of, and advertisements for jogging and racing strollers.

Our initial goal was to find resource directories that were exhaustive (or nearly so) containing virtually all positive examples of some narrow category. Our hope was that systematic experiments could then be carried out easily with the batch system. However, finding such a resource turned out to be much harder than we expected.

We began with the *ML course* problem, which as a narrow section of a frequently-used resource we expected to be comprehensive; however, preliminary experiments showed that it was not. (The first query constructed by the batch system using RIPPER [8] retrieved (from Altavista) 17 machine learning course descriptions in the first 20 documents; however, only 5 of these were from the original list.) Our next try at finding a comprehensive resource directory was the *AI societies* problem; this directory had the advantage (not shared by the ML course directory) that it explicitly stated a goal of being complete. However, similar experiments showed it to be quite incomplete. We then made an effort to construct a comprehensive list with the *jogging strollers* problem. This effort was again unsuccessful, in spite of spending about two hours with

[7]http://www.aic.nrl.navy.mil/~aha/research/machine-learning.html

[8]For those interested in details, this query was (course $\wedge$ machine $\wedge$ instructor $\wedge$ learning).

Problem	Initial Size	Initial Rcl.	Final Size	Iterations Lrn/Srch	Doc. Labeled
ML Cours.	11	0.24	46	5 / 4	82
AI Soc.	8	0.16	51	5 / 5	81
Jog. Strol.	5	0.25	20	4 / 6	24

Table 1: Summary of experiments with interactive system

Problem	#Iterations Allowed	Doc. Per Query(k)	Doc. Collected
ML courses	10	30	199
AI societies	20	30	126
Jog. Stroll.	10	30	29

Table 2: Summary of experiments with batch system

existing browsers and search engines on a topic deliberately chosen to be rather narrow.

We thus adopted the following strategy. With each problem we began by using the interactive system to expand an initial resource list. After the list was expanded, we invoked the batch system to collect additional negative examples and thus improve the learned rules.

Experiments with the interactive system

We used the interactive system primarily to emulate the batch system;[9] in particular, we did not attempt to uncover any more documents by browsing, or hand-constructed searches. However, we occasionally departed from the script by varying the parameters of the learning subsystem (in particular, the loss ratio), changing search engines, or examining varying numbers of documents returned by the search engines. We repeated the cycle of learning, searching, and labeling the results, until we were fairly sure that no new positive examples would be discovered by this procedure.

Table 1 summarizes our usage of the interactive system. We show the number of entries in each initial directory, the *recall*[10] of the initial directory relative to the final list that was generated, as well as the number of times a learner was invoked, the number of searches conducted, and the total number of pages labeled.[11]

To summarize, the interactive system appears to be very useful in the task of locating additional relevant documents from a specific class. In each case the number of known relevant documents was at least quadrupled. The effort involved was modest: our use of the interactive system generally involved labeling a few dozen pages, waiting for the results of a handful of searches, and invoking the learner a handful of times.[12]

Experiments with the batch system

In the next round of experiments, we invoked the batch system for each of these problems. Table 2 shows the resource limit set for each of these problems (the column "#Iterations Allowed" indicates how many times the learning subsystem could be called), the number of documents k that were collected for each query, and the total number of documents collected by the batch system (not including the initial set of 363 default negative examples). The resource limits used do not reflect any systematic attempt to find optimal limits. In each case, RIPPER was used as the learning system.

We then carried out a number of other experiments using the datasets collected by the batch system. One goal was simply to measure how successful the query-learning systems are in constructing an accurate intensional definition of the resource directories. To do this we re-ran both RIPPER and sleeping experts on the datasets constructed by the batch system, executed the corresponding queries, and recorded the recall and precision of these queries relative to the resource directory used in training. To obtain an idea of the tradeoffs that are possible, we varied the number of documents k retrieved from a query and parameters of the learning subsystems (for RIPPER, the loss ratio, and for sleeping experts, the threshold $p_{\min}$.) Altavista was used as the search engine.

The results of this experiment are shown in the graphs of Figure 5. (The first three graphs show the results for the individual classes and the second graph, somewhat less readable, shows the results for all three classes together.) Generally, sleeping experts generates the best high-precision classifiers. However, its rulesets are almost always larger than those produced by RIPPER; occasionally they are much larger. This makes them more expensive to use in searching and is the primary reason that RIPPER was used in the experiments with the batch and interactive systems.

The constructed rulesets are far from perfect, but this is to be expected. One difficulty is that neither of the learners perfectly fit the training data; another is that the search engine itself is incomplete. However, it seems quite likely that even this level of performance is enough to be useful. It is instructive to compare these hypotheses to the original resource directories that were used as input for the interactive system. The original directories all have perfect precision, but relatively poor recall. For the *jogging strollers* problem, both the learners are able to obtain nearly twice the recall (48% *vs* 25%) at 91% precision. For the *AI so-*

[9] The difference, of course, being that positive and negative labels were assigned to new documents by hand, rather than assuming all documents not in the original directory are negative.

[10] Note that the precision of the initial directory is 1 by definition.

[11] We count submitting a query for each rule as a single search, and do not count the time required to label the initial positive examples. Also, we typically did not attempt to label every negative example encountered in the search.

[12] In these experiments the time required by the learner is typically well under 30 seconds on a Sun 20/60.

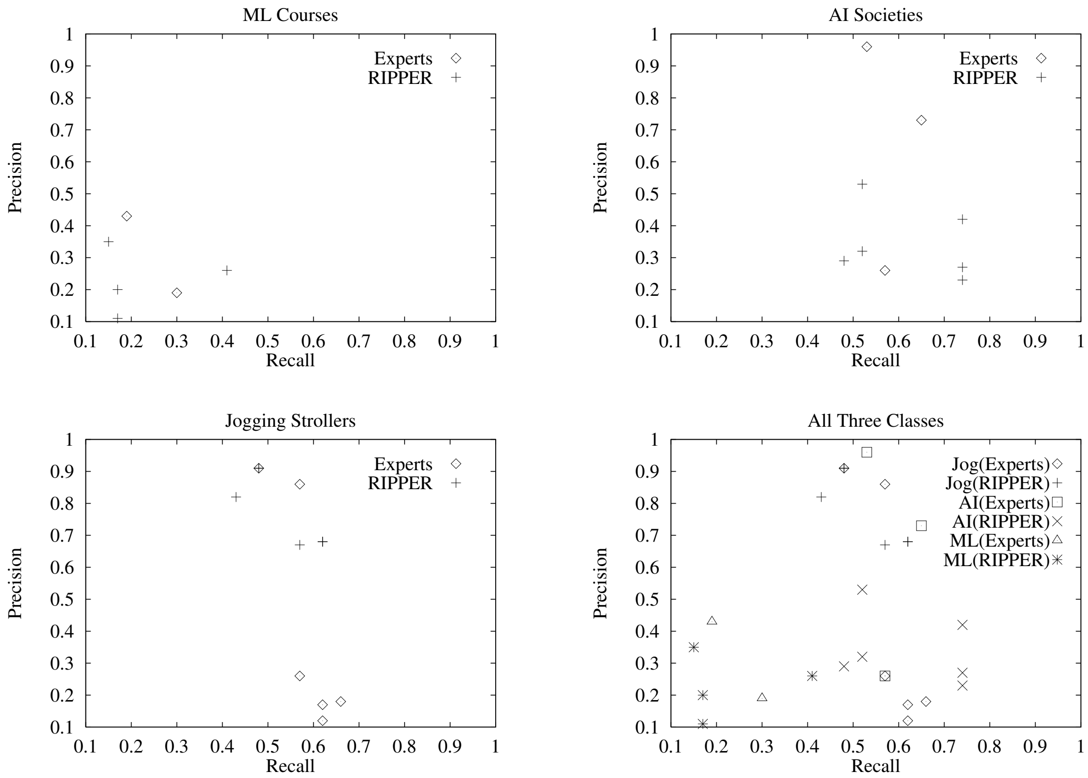

Figure 5: Precision-Recall tradeoffs for the three problems studied

	RIPPER			Sleeping Experts		
Problem	%Error	Recall	Precision	%Error	Recall	Precision
ML courses	9.37	0.98	0.92	9.27	0.96	0.94
AI societies	4.89	0.99	0.95	4.17	1.00	0.96
Jogging Strollers	1.36	1.00	0.99	1.59	1.00	0.98

Table 3: Results of the generalization error study

cieties problem, both learners obtain more than three times the recall at 94% precision or better. (RIPPER obtains 57% *vs* 16% recall with 94% precision, and sleeping experts obtains 53% recall at 96% precision.) Only for the *ML courses* problem are the hypotheses arguably worse than the original resource directory.

We also conducted a generalization error experiment on the datasets. In each trial, a random 80% of the dataset was used for training and the remainder for testing. A total of 50 trials were run for each dataset, and the average error rate, precision and recall on the test set (using the default parameters of the learners) were recorded.

The results are shown in Table 3. However, since the original sample is non-random, these numbers should be interpreted with great caution. Although the results suggest that significant generalization is taking place, they do not demonstrated that the learned queries can fulfill their true goal of facilitating maintenance by alerting the maintainer to new examples of a concept. This would require a study spanning a reasonable period of time.

Related work

As formulated, the learning-to-query problem has been reduced to learning to classify documents using the representation of (possibly monotone) DNF hypotheses. Previous work with rule learning systems has shown that the DNF representation sometimes yields reasonable results on text categorization problems (Apté *et al.* 1994; Cohen 1996). Furthermore, other experiments have shown that results nearly as good can be obtained if only *monotone* DNF is allowed (Cohen 1995c; 1996). The experiments of this paper also use monotone DNF in learning, but the learning is not done from a large random sample; rather, the examples used for learning are collected from repeatedly querying a search engine with previously learned rules.

Previous researchers (*e.g.*, (Salton *et al.* 1983; 1985)) have also considered the problem of relevance feedback for boolean query engines. Here an initial query (formulated by a user) is submitted to a search engine, and then modified based on labels collected from the user, after which the cycle is repeated. One difference in this paper is that we begin with a relatively large set of positive examples, rather than an initial query. A second difference is that we make use of an automated procedure to collect additional examples without user intervention, thus, making an assumption about the completeness of the current set of examples. This assumption is reasonable in this context, but not in a general information retrieval setting. The learning algorithms used in this paper are also novel in a relevance feedback setting.

Another point of difference with earlier work in information retrieval is that our experimental results are on actual WWW documents. This is perhaps important, because unlike most of the document collections used in IR experiments, the WWW is very heterogeneous, containing many types of documents. For example, we have noted that it is difficult for our learning subsystems to distinguish between a description of a machine learning course (for example) and a document containing descriptions of many courses, include one about machine learning; in general, we suspect that distinguishing between relevant documents and references to relevant documents may be a difficult problem. Such difficulties would not arise in a more homogeneous collection of documents, such as a collection of abstracts.

At least two other research projects have augmented Web browsers with learning capabilities; however in each case the goals of learning were different. WebWatcher (Armstrong *et al.* 1995) learns to advise the user about which links to follow to achieve some desired goal—a task quite different from the one considered in this paper. Syskill & Webert (Pazzani *et al.* 1995) learns to classify pages with respect to a user's interests; this task is more similar to the one we considered. However, the end goal of this system is to rank documents on existing hotlists according to user interest, not to facilitate resource directory maintenance. After learning, the Syskill & Webert system can also construct a Lycos query that retrieves additional "interesting" documents; however, this (single) query is constructed by using a very simple heuristic procedure, and is not intended to be an accurate intensional definition of a concept.

Conclusions and future work

The World Wide Web (WWW) is currently filled with *resource directories*—documents that collect together links to all known documents on a specific topic. Keeping resource directories up-to-date is difficult due to the rapid growth in on-line documents.

This paper has investigated the use of machine learning methods as an aid in maintaining resource directories. A resource directory is treated as an exhaustive list of all positive examples of an unknown concept, thus yielding an extensional definition of the concept. Machine learning methods can then be used to construct from these examples an *intensional* definition of the concept. The learned definition is in DNF form, where the primitive conditions test the presence (or even the absence) of particular words. This representation can be easily converted to a series of queries that can be used to search for the original documents—as well as new, similar documents that have been added recently to the WWW.

We described two systems which were implemented to test these ideas, both of which make minimal assumptions about the search engine. The first is a batch system which repeatedly learns a concept, generates an appropriate set search queries, and uses the queries to collect more negative examples. An advantage of this procedure is that it can collect hundreds of examples with no human intervention; however, it can only be used if the initial resource list is complete (or nearly so). The second is an interactive system. This systems augments an arbitrary WWW browser with the ability to label WWW documents and then learn search-engine queries from the labeled documents. It can be used to perform the same sorts of sequences of actions as the batch system, but is far more flexible. In particular, keeping a human user "in the loop" means that positive examples not on the original resource list can be detected. These examples can be added to the resource list, both extending the list and improving the quality of the dataset used for learning. In experiments, these systems were shown to produce usefully accurate intensional descriptions of concepts. In two of three tests problems, the concepts produced had substantially higher recall than manually-constructed lists, while still attaining precision of greater than 90%.

The results reported in this paper suggest a number of topics for further research. The evaluations in this paper were of somewhat limited scope. Evaluation of the learning methods used in this study could also be conducted on artificial data, perhaps along the lines of the studies conducted in the Text Retrieval and Classification (TREC) meetings (Harman 1995). This would allow more rigorous comparative evaluation of learning methods, albeit in a somewhat more artificial setting.

Several future research goals involve improving the interaction between the learned rule and the various search engines. First, the addition of new search engines would be simplified by an automatic procedure to construct the transformation from a rule (produced by the learning subsystem) to a query (for the search engine). Second, since none of the search engines is clearly superior to the others, it may be better to use the results of all the available search engines rather than rely on a single engine; one possibility would be to use learning techniques to combine the output of different engines. Lastly, several search engines support complex queries such as proximity queries. Since our learning methods (or extensions (Cohen 1995b)) take into account the proximity of the words appearing in a document, we plan to modify the current rule extraction schemes to support such queries.

References

Chidanand Apté, Fred Damerau, and Sholom M. Weiss. Automated learning of decision rules for text categorization. *ACM Transactions on Information Systems*, 12(3):233–251, 1994.

R. Armstrong, D. Frietag, T. Joachims, and T. M. Mitchell. WebWatcher: a learning apprentice for the world wide web. In *Proceedings of the 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, Stanford, CA, 1995. AAAI Press.

Avrim Blum. Learning boolean functions in a infinite attribute space. In *22nd Annual Symposium on the Theory of Computing*. ACM Press, 1990.

Avrim Blum. Empirical support for WINNOW and weighted majority algorithms: results on a calendar scheduling domain. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

Nicolò Cesa-Bianchi, Yoav Freund, David P. Helmbold, David Haussler, Robert E. Schapire, and Manfred K. Warmuth. How to use expert advice. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 382–391, May 1993. Submitted to the Journal of the ACM.

William W. Cohen. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

William W. Cohen. Learning to classify English text with ILP methods. In Luc De Raedt, editor, *Advances in ILP*. IOS Press, 1995.

William W. Cohen. Text categorization and relational learning. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

William W. Cohen. Learning with set-valued features. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Portland, Oregon, 1996.

Ido Dagan and Shaun Engelson. Committee-based sampling for training probabilistic classifiers. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Taho, California, 1995. Morgan Kaufmann.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37. Springer-Verlag, 1995. A long version will appear in JCSS.

Y. Freund, H.S. Seung, E. Shamir, and N. Tishby. Information, prediction, and query by committee. In *Advances in Neural Informations Processing Systems 5*, pages 483–490, San Mateo, CA, 1992. Morgan Kaufmann.

Donna Harman. Overview of the second text retrieval conference (TREC-2). *Information Processing and Management*, 3:271–289, 1995.

Robert Holte, Liane Acker, and Bruce Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Detroit, Michigan, 1989. Morgan Kaufmann.

David Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In *Machine Learning: Proceedings of the Eleventh Annual Conference*, New Brunswick, New Jersey, 1994. Morgan Kaufmann.

David Lewis and William Gale. Training text classifiers by uncertainty sampling. In *Seventeenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1994.

David Lewis. Representation and learning in information retrieval. Technical Report 91-93, Computer Science Dept., University of Massachusetts at Amherst, 1992. PhD Thesis.

Nick Littlestone and Manfred Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212–261, 1994.

Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4), 1988.

M. Pazzani, L. Nguyen, and S. Mantik. Learning from hotlists and coldlists: towards a WWW information filtering and seeking agent. In *Proceedings of AI Tools Conference*, Washington, DC, 1995.

J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3), 1990.

J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1994.

G. Salton, C. Buckley, and E. A. Fox. Automatic query formulations in information retrieval. *Journal of the Americal Society for Information Science*, 34(4):262–280, 1983.

G. Salton, E. A. Fox, and E. Voorhees. Advances feedback methods in information retrieval. *Journal of the Americal Society for Information Science*, 36(3):200–210, 1985.

H.S Seung, M. Opper, and H. Sompolinsky. Query by committee. In *Proceedings of the Fifth Workshop on Computational Learning Theory*, pages 287–294, San Mateo, CA, 1992. Morgan Kaufmann.

V. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371–383. Morgan Kaufmann, 1990.