



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F 17/30, 19/00	A1	(11) International Publication Number: WO 97/38377 (43) International Publication Date: 16 October 1997 (16.10.97)
<p>(21) International Application Number: PCT/US97/05355</p> <p>(22) International Filing Date: 9 April 1997 (09.04.97)</p> <p>(30) Priority Data: 60/015,231 10 April 1996 (10.04.96) US</p> <p>(71) Applicant (for all designated States except US): AT & T CORP. [US/US]; 131 Morristown Road, Basking Ridge, NJ 07920 (US).</p> <p>(72) Inventors; and (75) Inventors/Applicants (for US only): COHEN, William, W. [US/US]; 178 Belmont Avenue, North Plainfield, NJ 07060 (US). SINGER, Yoram [IL/US]; 36 Columbus Avenue, New Providence, NJ 07974 (US).</p> <p>(74) Agents: REDMOND, Joseph et al.; Morgan & Finnegan, L.L.P., 1299 Pennsylvania Avenue, N.W., Washington, DC 20004 (US).</p>	<p>(81) Designated States: CA, JP, KP, KR, US, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).</p> <p>Published <i>With international search report.</i></p>	
<p>(54) Title: A SYSTEM AND METHOD FOR FINDING INFORMATION IN A DISTRIBUTED INFORMATION SYSTEM USING QUERY LEARNING AND META SEARCH</p>		
<p>(57) Abstract</p> <p>An information retrieval system finds information in a (DIS) Distributed Information System, (the Internet) using query learning and meta search (figure 2) for adding documents to resource directories contained in the DIS. A selection means (figure 4; yes/no link) generates training data characterized as positive and negative examples of a particular class of data residing in the DIS. A learning means (figure 4; learn link) generates from the training data at least one query that can be submitted to any one of a plurality of search engines for searching the DIS to find "new" items of the particular class. An evaluation means (figure 4; review previous link) determines and verifies that the new item(s) is a new subset of the particular class and adds or updates the particular class in the resource directory.</p>	<pre> 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 Corresponding Query(r.Search Engine) Response := Call Top-k- Documents(q.Search Engine) 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 </pre>	

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakistan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

COHEN 6-1

A SYSTEM AND METHOD FOR FINDING INFORMATION
IN A DISTRIBUTED INFORMATION SYSTEM USING QUERY
LEARNING AND META SEARCH

Notice

This document discloses source code for
implementing the invention. No license is granted
directly, indirectly or by implication to the source
code for any purpose by disclosure in this document ,
5 except copying for informational purposes only or as
authorized in writing by the assignee under suitable
terms and conditions.

Related Application

10 Provisional Application, Serial Number
60/015,231, filed April 10, 1996 and assigned to the
same assignee as that of the present invention.

Background of the Invention

(1) Field of the Invention

COHEN 6-1

This invention relates to information retrieval systems. More particularly, the invention relates information retrieval in distributed information system, e.g Internet using query learning and meta search.

(2) Description of the Prior Art

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 can be difficult to create and maintain. Maintenance is especially problematic because the rapid growth in on-line documents makes it difficult to keep a resource directory up-to-date.

This invention proposes to describe 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

COHEN 6-1

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

5 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

10 the concept that have been added. We will present experimental results on this problem with two implemented systems. One is an interactive system---an augmented WWW browser that allows the user label any document, and to learn a search query from previously

15 labeled examples. This system is useful in locating documents similar to those in a resource directory, thus making it more comprehensive. The other is a batch system which repeatedly learns queries from examples, and then collects and labels pages using

20 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.

COHEN 6-1

Prior art related to machine learning methods includes the following:

USP 5278980 issued January 11, 1994 discloses an information retrieval system and method in which an operator inputs one or more query words which are used to determine a search key for searching through a corpus of a document, and which returns any matches between the search key and the corpus of a documents as a phrase containing the word data matching the query word(s), a non-stop (content) word next adjacent to the matching work data, and all intervening stop-words between the matching word data and the next adjacent non-stop word. The operator, after reviewing one or more of the returned phrases can then use one or more of the next adjacent non-stop words as new query words to reformulate the search key and perform a subsequent search through the document corpus. This process can be conducted iteratively, until the appropriate documents of interest are located. The additional non-stop words for each phrase are preferably aligned with each other (e.g., columination) to ease viewing of the "new" content words.

COHEN 6-1

Other prior art related to machine learning methods is disclosed in the references attached to the specification as Appendix 1.

None of the prior art discloses a system and method of adding documents to a resource directory in a distributed information system by using a learning means to generate from training data a plurality of items as positive and/or negatives examples of a particular class and using a learning means to generate at least one query that can be submitted to any of a plurality of methods for searching the system for a new item, after which the new item is evaluated by learning means with the aim of verifying that the new item is a new subset of the class.

Summary of the Invention

An information retrieval system finds information in a Distributed Information System (DIS), e.g. the Internet using query learning and meta search for adding documents to resource directories contained in the DIS. A selection means generates training data

COHEN 6-1

characterized as positive and negative examples of a particular class of data residing in the DIS. A learning means generates from the training data at least one query that can be submitted to any one of a plurality of search engines for searching the DIS to find "new" items of the particular class. An evaluation means determines and verifies that the new item(s) is a new subset of the particular class and adds or updates the particular class in the resource directory.

10 Description of the Drawing

Fig. 1 is a representation of a prior art distributed information system which implements the principles of the present invention.

15 Fig. 2 is a listing of pseudo code for a batch-query -learner incorporating the principles of the present invention in the system of Fig. 1.

Fig. 3 is a representation of an interactive query-learning system incorporating the principles of the present invention.

COHEN 6-1

Fig. 4. is a user interface to the query learning system of the present invention.

Fig. 5 is a listing of pseudo code for an on line prediction algorithm incorporating the principles
5 of the present invention.

Fig. 6 is a Table summarizing experiments with the learning system of Fig. 3.

Fig. 7 is a Table summarizing experiments with the learning system of Fig. 2.

10 Fig. 8 is a graph of data showing the results of precision-recall tradeoff for the three problems studied the batch query learning system of Fig. 2.

15 Fig. 9 is a Table of results of a generalization error study for the learning systems of Fig. 2 and Fig. 5.

COHEN 6-1

Description of Preferred Embodiments

The problem addressed by the present invention is a variant of the problem of relevance feedback, which is well-studied in information retrieval. One novel aspect of the present invention (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 application); hence it seems inappropriate to rely too heavily on a single search engine. The current implementation can use any of several search engines. A second motivation for investigating search engine independent learning methods is that there are many search engines accessible from the WWW that index databases partially or entirely separate from the WWW. As WWW browsers and the Common Gateway Interface (CGI) now provide a nearly uniform interface to many search engines, it seems

COHEN 6-1

reasonable to consider the problem of designing general-purpose relevance feedback mechanisms that require few assumptions to be made about the search engine.

5 A distributed information system 10, e.g.,
the Internet to which the invention is applicable is
shown in Fig. 1 The Internet is further described in
the text "How The Internet Works" by Joshua Eddings,
published by Ziff Davis, 1994. The system includes a
10 plurality of processors 12 and related databases 14
coupled together through routers (not shown) for
directing messages among the processors in accordance
with network protocols. Each processor and related
database is coupled to a plurality of users through
15 servers (not shown). The users may originate messages
for purposes of communication with other users and/ or
search the system for information using search
engines.

 The initial research goal was to implement a
20 WWW-based query-learning system in the system of Fig. 1
and support meaningful experimentation to provide a
qualitative evaluation of the difficulty of the task.

COHEN 6-1

To conduct this initial evaluation two different systems were implemented: one designed for batch use, and the other designed for interactive use, as will be described hereinafter.

5 A Batch System

 The first implementation is a Perl script that runs as a ``batch'' system-- it requires no user intervention. The input of the batch system is a list of Uniform Resource Locators (URL's) that correspond
10 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.

15 The procedure used to accomplish this is shown in Fig 2. Three subroutines are used. The first, Learn comprehends a concept from a sample. The only assumption made by the query-learning system about the learning system is that the hypothesis of the learning
20 system is in disjunctive normal form (DNF), where the

COHEN 6-1

primitive conditions test for the presence of words.
For example, a DNF hypothesis learned from a resource
list on college basketball might be:

(college \wedge basketball) \vee (college \wedge hoops) \vee (NCAA \wedge
5 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
10 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
15 query-learning system, Corresponding Query, 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
20 formats for queries, adding the knowledge needed to

COHEN 6-1

support a new search engine is usually straightforward. Some search engines can handle more expressive queries---queries that require terms to appear near each other, or queries that contain word stems like
5 ``comput*\$*\$''. Most advanced queries are not currently supported by the existing Corresponding Query routine. One exception are queries containing conditions that check for the absence (rather than the presence) of words, such as (basketball \wedge -college \wedge -NCAA). These
10 can be used if both the learning system and the query system allow it, but were not used in any of the experiments of this invention.

The final subroutine, Top-k-Documents}, submits a query to a search engine and collects the top
15 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
20 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

COHEN 6-1

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
5 collecting examples may be competitive or superior; for
instance, promising results have been obtained with
''uncertainty sampling. See Lewis and Gale (16) and
query-learning by committee. See Seung et al (25). Also
see Dagan and Engelson (10).

10 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
15 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,
20 the process of learning rules and then collecting

COHEN 6-1

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 system is called.

5 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
10 this problem, if no new documents are collected on a cycle, heuristics are used to vary the parameters of the learning system for the next cycle. In the current implementation, two heuristics are followed: if the
15 hypothesis of the learning system is an empty rule set, 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 system being used.

An Interactive System

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

COHEN 6-1

means that it cannot be successfully used unless one is confident that the initial set of documents is reasonably complete. Our experience so far is that this is seldom the case. For this reason, we also
5 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
10 particular fixed sequence for collecting documents and labeling; instead it is simply an augmented WWW browser, which allows the user to label the document being browsed, to invoke the learning system, or to conduct a search using previously learned rules.

15 The architecture of the interactive system is shown in Fig. 3. 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.
20 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,

COHEN 6-1

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
5 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,
an acronym Surfing While Inducing Methods to Search
10 for URLs or SWIMSUIT has been assigned to the 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
15 invoking the learning system.

Functionally, the special links inserted by
the query-learning interface act as additional
''control buttons'' for the browser---similar to the
buttons labeled ''Back'' and ''Net Search'' on the
20 Netscape browser. By clicking on special links, the
user can classify pages, invoke the learning system,

COHEN 6-1

and so on. The user's view of the interactive system is shown in Fig. 4.

The special links are:

Document labeling: The yes link and no link
5 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
10 actual learning system and/or invoke the learner on the
current class. The behavior of this link can be easily
changed, so that different learning systems can be used
in experiments. As in the batch system, learning is
normally constrained by using default negative
15 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

COHEN 6-1

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 anew 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.

10 Learning Systems

Two learning systems have been integrated with the system: RIPPER, a propositional rule learner that is related to FOIL, see Quinlan (21), and a rule-learning version of "Sleeping experts". Sleeping experts is a new prediction algorithm that combines ideas from used for online prediction, see Freund (11) with the infinite attribute model of Blum (3).

These algorithms have different strengths and weaknesses. RIPPER implicitly assumes that examples are i.i.d---which is not the case for samples collected

COHEN 6-1

via browsing or by the batch query-learning system.
However, formal results suggest that sleeping experts
will perform well even on data sets that are selected
in a non-random manner. The sleeping experts algorithm
5 is also largely incremental, which is potentially an
advantage in 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).

10 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
15 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
20 minimum-description length techniques are used to
prevent overfitting. In previous experiments, RIPPER

COHEN 6-1

was shown to be comparable to C4.5rules, Quinlan (22) in terms of generalization accuracy, but much faster for large noisy datasets. For more detail, see Cohen (8).

5 The version of RIPPER used here was extended to handle ``set-valued features'', as described in Cohen (9). 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
10 conditions that are allowed for a set-valued feature F are of the form $c \in F$, 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
15 of which is the set of all tokens appearing in the document.. 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
20 from a standard stoplist, Lewis (17) 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

COHEN 6-1

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.

5 A second extension to RIPPER allows the user to specify a loss ratio, see Lewis and Catlett (14). A loss ratio indicates the ratio of the cost of a false negative error to the cost of a false positive error; the goal of learning is to minimize total
10 misclassification cost, rather than simply the number of errors, on unseen data. Loss ratios in RIPPER are implemented by changing the weights given to false positive errors and false negative errors in the pruning and optimization stages of the learning
15 algorithm.

 One additional modification to RIPPER was also made specifically to improve performance on the query-learning task. The basic RIPPER algorithm is heavily biased toward producing simple, and hence
20 general, conjunctions; for example, for RIPPER, when a conjunction of conditions is specific enough to cover no negative examples, no further conditions will be

COHEN 6-1

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 of rule l conditions that are true for every positive covered by the rule. 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 gracefully handle very long queries. We note that a similar scheme has been investigated in the context of the "small disjunct problem", see Holte (14). The postpass implements a bias towards specific rules rather than general rules.

Sleeping Experts

In the past years there has been a growing interest in online prediction algorithms. The vast majority of the prediction algorithms are given a pool of fixed "experts"---each of which is a simple, fixed, classifier---and build a master algorithm, which combines the classifications of the experts in some

COHEN 6-1

manner. Typically, the master algorithm classifies an
example by using a weighted combination of the
predictions of the experts. Building a good master
algorithms thus a matter of finding an appropriate
5 weight for each of the experts. Formal results show
that by using a multiplicative weight update, see
Littlestone (18), 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
10 expert in the pool, even for a sequence of prediction
problems that is chosen by an adversary.

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
15 allocation algorithm called Hedge, due to Freund and
Schapire, see Freund (11), which is applicable to a
broad class of learning problems and loss functions.
The second is thenfinite attribute model of Blum (3) .
In this setting, there may be any number of experts,
20 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,

COHEN 6-1

Littlestone(19) has also been implemented, see Blum
(4).

Below we summarize the sleeping experts
procedure, which combines the Hedge algorithm with the
5 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
10 Fig. 5. 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} will be non-negative. However, the weights
15 need not sum to one. At each time step t , the learner
is given a new instance χ_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 χ_t . The prediction of expert i on χ_t is
20 denoted

COHEN 6-1

by

$$y_i^t$$

5 . Based on the experts in W_t , the master algorithm must make a prediction for the class of χ_t , and then update the pool and the weight set p .

To make a prediction, the master algorithm decides on a distribution \bar{p} over the active experts, which is determined by restricting the set of weights p to the set of active experts W_t , and normalizing the weights.

10 We denote the vector of normalized weights by \bar{p} ,

where
$$F_3\left(\sum_{i \in W_t} \bar{p}_i^t y_i^t\right)$$

. The prediction of the master algorithm is

15

25

. We use $F_3(r) = \ln(1 - r + r\beta) / (\ln(1 - r + r\beta) + \ln((1 - r)\beta^3 + r))$, the function used by Vovk [1990] for predicting binary sequences.

COHEN 6-1

 l_i^t

. In the implementation described here, this loss is 0 if the expert's prediction is correct and 1 otherwise.

5 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

$$\forall i \in W_t : p_i^{t+1}$$

10). 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,

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

COHEN 6-1

where Z is chosen such that

$$\sum_{i \in W} p_i^t = \sum_{i \in W} p_i^{t+1}$$

. The "update function" U_β is any function satisfying
 [Cesa-Binachi et al., 1993] $\beta^t \leq U_\beta(r) \leq 1 - (1 - \beta)r$. In our
 5 implementation, we used the linear update. $U_\beta(r) = 1 - (1 - \beta)r$, which is simple to implement and it avoids
 expensive exponentiations.

Briefly, if one defines the loss of the
 master algorithm to be the average loss with respect to
 10 the distribution

$$\{p_i^t | i \in W\},$$

, 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
 15 for any sequence of examples $(x_1, y_1), \dots, (x_t, y_t)$, in
 particular, the bounds hold for sequences whose
 instances are not statistically independent.

COHEN 6-1

The Pool of Experts

It remains to describe the experts used for WWW page classification. In our experiments each expert is corresponds to a space that appears in a document. That is, if ω_i is the i th token appearing in the document, each expert is of the form $\omega_{i_1}\omega_{i_2}\dots\omega_{i_k}$ where $1 \leq i_1 < i_2 < \dots < i_{j+1} < i_k$ and $i_k - i_1 < n$. This is a generalization of the ngram\footnote model. Note that our goal is to classify WWW documents; hence each ngram expert is used to predict the classification of the document in which it appears, rather than the next token (word). For each ngram we construct two mini-experts, one which always predicts 0 (not in the class), and one that always predicts 1. The loss of each mini-expert is either % 0 or 1 depending on the actual classification of the document.

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 predicts that 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

$$P_i^T / \sum_{j \in \text{Pool}} P_j^T \geq w_{\min}$$

COHEN 6-1

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 few experts, and hence the 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 p_i of expert i , based on its two mini-experts $(i,0)$ and $(i,1)$, to be $p_i = F_3(p_{i,0}/(p_{i,0}+p_{i,1}))$. 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 $w_{i1} \wedge w_{i2} \wedge \dots \wedge w_{ik}$.

COHEN 6-1

Although the sleeping experts algorithm treats this as an ngram, we currently treat it simply as a conjunction of features: clearly, this is suboptimal for search engines which support proximity queries.

5 Experimental Results

 We have evaluated the system with three resource directories.

 ML courses is part of an excellent machine learning resource maintained by David Aha¹.
10 This list contained (at the time the experiments were conducted) pointers to 15 on-line descriptions of courses.

 AI societies is a WWW page jointly maintained by SIGART, IJCAI, and CSCSI. It contains pointers to
15 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
20 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

<http://www.aic.nrl.navy.mil/~aha/research/machine-learning.html>

COHEN 6-1

batch system. However, finding such a resource turned out to be much harder than we expected.

We began with the MLcourse 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 etrieved (from Altavista) 17 machine learning course descriptions in the first 20 documents; however, only 5 of these were from the original list.). For those interested in details, this query was

(course \wedge machine \wedge instructor \wedge learning)

Our next try at finding a comprehmachine learning course descriptions in the first 20 documents; ensive 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 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

COHEN 6-1

list was expanded, we invoked the batch system to collect additional negative examples and thus improve the learned rules.

5 Experiments With The Interactive System

We used the interactive system primarily to emulate the batch system; 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. 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 system (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.

Fig. 6 summarizes our usage of the interactive system. We show the number of entries in each initial directory, the term *Recal* is the fraction of the time that an actual positive example is predicted to be positive by the classifier, and the term *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

COHEN 6-1

1.00 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. 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.

5
10

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 a handful of searches, and invoking the learner a handful of times. In these experiments the time required by the learner is typically well under 30 seconds on a Sun 20/60.

15

20

Experiments With The Batch System

In the next round of experiments, we invoked the batch system for each of these problems. Fig. 7 shows the resource limit set for each of these problems (the column ``\#Iterations Allowed`` indicates how many times the learning system 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

25

30

COHEN 6-1

default negative examples). The resource limits used do not reflect any systematic attempt to find optimal limits. However, for the last two problems, the learner seemed to "converge" after a few iterations, and
5 output a single hypothesis (or in one case alternate between two variants of a hypothesis) on all subsequent iterations.) In each case, RIPPER was used as the learning system.

We then carried out a number of other
10 experiments using the datasets collected by the batch system. One goal was simply to measure how successful the learning systems are in constructing an accurate intensional definition of the resource directories. To do this we re-ran the learning systems on the datasets
15 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
20 documents k retrieved from a query and parameters of the learning systems (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
25 in the graphs of Figure 8. The first three graphs show the results for the individual classes and the second graph shows the results for all three classes together. Generally, sleeping experts generates the best high-precision classifiers. However, its rulesets are

COHEN 6-1

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
5 and interactive systems.

The constructed rulesets are far from
perfect, but this is to be expected. One difficulty is
that the neither of the learners perfectly fit the
training data; another is that the search engine itself
10 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
15 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 societies* problem, both
learners obtain more than three times the recall at 94%
20 precision or better. (RIPPER obtains 57% vs 15% recall
with 94% precision.

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
25 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.

COHEN 6-1

The results are shown in Fig. 9. However, since the original sample is non-random, these numbers should be interpreted with great caution. Although the results suggest that significant
5 generalization is taking place, they do not demonstrate 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.

10 Summary

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
15 up-to-date is difficult because of the rapid growth in on-line documents. This invention describes 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
20 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
25 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
30 to the WWW.

COHEN 6-1

Two systems were implemented to test these ideas, both of which make minimal assumptions about the search engine. One 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 produce usefully accurate intensional descriptions of concepts. In two of three test problems, the concepts produced had substantially higher recall than manually-constructed lists, while attaining precision of greater than 90%

In support of the invention, and in particular the description of Preferred Embodiment, the following Appendices are included in the application:

COHEN 6-1

Appendix 1. A list of references cited in the application by reference numeral.

Appendix 2. A copy of a README file which describes the source code implementing the presently-preferred embodiment of the invention.

5

Appendix 3. Commented source code written in perl for the presently-preferred embodiment of the invention.

Appendix 4. A copy of the documentation for the OreO shell tool which was used in the implementation of the presently-preferred embodiment.

10

COHEN 6-1

References

1. (Apté, et al., 1994) 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, 5 1994.
2. (Armstrong et al., 1995) R. Armstrong, D. Frietag, T. Joachims, and T.M. Mitchell. WebWatcher: a learning apprentice for the world wide web. In 10 *Proceedings of the 1995 AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*. Stanford, CA, 1995. AAAI Press.
3. (Blum, 1990) Avrim Blum. Learning boolean 15 functions in a infinite attribute space. In *22nd Annual Symposium on the Theory of Computing*. ACM Press, 1990.
4. (Blum, 1990) Avrim Blum. Empirical support 20 for WINNOWER and weighted majority algorithms: results on a calendar scheduling domain. In *Machine Learning: Proceedings of the Twelfth International Conference, Lake Tahoe, California, 1995*. Morgan Kaufmann.
5. (Cesa-Bianchi et al., 1993) Nicolò Cesa- 25 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. 30 Submitted to the Journal of the ACM.
6. (Cohen, 1995a) William W. Cohen. Fast 35 effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference, Lake Tahoe, California, 1995*. Morgan Kaufmann.
7. (Cohn, 1995b) William W. Cohen. Learning to 40 classify English text with ILP methods. In Luc De Raedt, editor, *Advances in ILP*. IOS Press, 1995.

COHEN 6-1

- 5 8. (Cohen, 1995c) William W. Cohen. Text categorization and relational learning. In *Machine Learning: Proceedings of the Twelfth International Conference, Lake Tahoe, California, 1995*. Morgan Kaufmann.
- 10 9. (Cohen, 1996) William W. Cohen. Learning with set-valued features. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence, Portland, Oregon, 1996*.
- 15 10. (Dagan and Engelson, 1995) Ido Dagan and Shaun Engelson. Committee-based sampling for training probabilistic classifiers. In *Machine Learning: Proceedings of the Twelfth International Conference, Lake Tahoe, California, 1995*. Morgan Kaufmann.
- 20 11. (Freund and Schapire, 1995) 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 23-27. Springer-Verlag, 1995. A long version will appear in JCSS.
- 25 12. (Freund et al., 1992) 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.
- 30 13. (Harman, 1995) Donna Harman. Overview of the second text retrieval conference (TREC-2). *Information Processing and Management*, 3:271-289, 1995.
- 35 14. (Holte, 1989) 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.
- 40

COHEN 6-1

15. (Lewis and Catlett, 1994) 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.
16. (Lewis and Gale, 1994) 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.
17. (Lewis, 1992) David Lewis. Representation and learning in information retrieval. Technical Report 91-93, Computer Science Dept., University of Massachusetts at Amherst, 1992. PhD Thesis.
18. (Littlestone and Warmuth, 1994) Nick Littlestone and Manfred Warmuth. The weighted majority algorithm. *Information and Computation*, 108(2):212-261, 1994.
19. (Littlestone, 1988) Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4), 1988.
20. (Pazzani, et al., 1995) 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.
21. (Quinlan, 1990) J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3), 1990.
22. (Quinlan, 1990) J. Ross Quinlan. *C4,5: programs for machine learning*. Morgan Kaufmann, 1994.
23. (Salton, et al., 1983) G. Salton, C. Buckley, and E.A. Fox. Automatic query formulations

COHEN 6-1

- in information retrieval. *Journal of the American Society for Information Science*, 34(4):262-280, 1983.
- 5 24. (Salton, et al., 1985) G. Salton, E.A., Fox, and E. Voorhees. Advances feedback methods in information retrieval. *Journal of the American Society for Information Science*, 36(3):200-210, 1985.
- 10 25. (Seung, et al., 1992) 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.
- 15 26. (Vovk, 1990) V. Vovk. Aggregating strategies. In *Proceedings of the Third Annual Workshop on Computational Learning Theory*, pages 371-383. Morgan Kaufmann, 1990.

COHEN 6-1

Appendix 2. A copy of a README file which describes the source code implementing the presently-preferred embodiment of the invention.

APPENDIX 2

5 README Page 1

```
=====
utilities
=====
```

10 oreo.pl
- some utilities for handling http requests with oreosh

```
=====
```

general routines for use with oreosh

15 my hope is that these can be used for
other purposes as well....

```
=====
```

connect.pl

- simple two-way connection using oreosh.
mostly there as an example.

trap-request.pl

20 - traps http requests that match a given regexp
and handles the specially--specifically a given
program is invoked and its output (which should
be html) is returned to the client.

markup.pl

25 - allows you to insert arbitrary stuff at the top
of html documents, before the client gets a hold of
them.

cache.pl, cache-util.pl

- implements a simple local cache for .html pages

30 track.pl

- buggy routine to track state of the browser

```
=====
the form-labeling program
=====
```

COHEN 6-1

launch.csh

- invokes this cascade of oreosh creatures
client | trap-request | markup | cache | proxy
The client should connect to port 8111 of radish.

5 autosurf.pl

- loads in cache and cycles through it randomly,
filtering by the current class definition.

ss-marker.pl

10 - invoked by the markup.pl daemon, this inserts
an appropriate header into html documents

ss-main.pl

15 - this traps the requests included in ss-marker.pl
and handles them specially. The request handling
is done in the files below. All local data is stored
in -user/.ss/cache or -user/.ss/data.

COHEN 6-1

Appendix 3. Commented source code written in perl for the presently-preferred embodiment of the invention.

COHEN 6-1

Appendix 4. A copy of the documentation for the OreO shell tool which was used in the implementation of the presently-preferred embodiment.

APPENDIX 4

5 Developing an OreO Agent

Table of Contents

- Types of OreO Agents
- Library Routines
- OreO Shell API
- 10 • Future Directions

OreO Agents

15 By some measurements, OreO agents appear to function as servers in that they support connections from multiple clients and provide services to these clients. In this respect, the design of an OreO Agent (agent) uses the same techniques as designing any other network-based server.

20 We define a *connection* as consisting of two socket; one to the "upstream" client and one to the "downstream" server. The OreO shell (oreosh) is responsible for setting up these connections and making them available to the actual processing code. An OreO agent is thus a combination of the OreO shell and some processing code: in the simplest case, a plain OreO shell acts as a simple pass-thru mechanism. The agent 25 receives HTTP request data on the client socket, and HTTP response data on the server socket.

OreO Agents expect to see HTTP proxy requests. The HTTP proxy protocol simply specifies 30 that URLs presented in the various HTTP methods will be absolute; normally, an HTTP server don't see the scheme and host address/port number portion of the URL. These proxy requests are then forwarded to the host specified by the OREO_PROXY environment variable (specified as a <hostaddress>:"<port number>tuple). 35

Since the agent "speaks" the HTTP proxy protocol on both its upstream and downstream side, OreO agents may be nested in a manner similar to Unix pipelines.

COHEN 6-1

When designing an agent, we can utilize several different designs. These are

- whether connection should be processes serially or in parallel
- whether an new process is generated for each connection.

The above results in four different agent models, which we discuss below. Our use of them process is influenced by the canonical Posix process model, which supports (at present) a single thread of control per process. The design of an agent will change dramatically for those systems (like Windows/NT) that provide multiple threads of control per process.

Serial connections, multiple processes

In this model, a new process is generated for each connection and the shell waits for this process to finish before `accept()`ing another connection. This model is useful when the agent code requires sequential access to a shared resource, and no mechanism exists to synchronize shared access to that resource. This form of processing is enabled by specifying the `-l` switch to the OreO shell.

Parallel connection, multiple process

In this model, the shell guarantees a new process for each connection, but the shell immediately returns to `accept()` another incoming connection. This provides maximum parallelism, but not necessarily optimum thrupt. The application must synchronize access to shared, writeable resources such as databases, files, etc.

In both instances, the shell supports different ways to process the HTTP request and response. The agent author can choose to filter either the HTTP request, the HTTP response, or both. If only the request or response stream is desired, the shell takes responsibility for forwarding the other.

The shell supports this via the following command line arguments.

- `-i` process the HTTP request stream
- `-o` process the HTTP response stream
- `-b` process the request and response stream

If no arguments are specified, the OreO shell simply copies its input and output from the client to the server, and vice versa.

When filtering the request stream, the shell arranges to connect the client socket to the standard

COHEN 6-1

input (stdin) of the child process, and the server socket to the standard output (stdout). This is reversed for processing the response stream. Connecting the sockets in this way permits the use of pre-existing Unix style filters as processing code, or creating processing code in various script languages that easily read and write stdin and stdout.

Processes that read from the client side normally will never see EOF, since the client is waiting on that channel to receive the HTTP response. Therefore, the shell intervenes on the process's behalf, and sends a SIGTERM when EOF is seen on the HTTP response stream. Processes that read the response stream will see EOF when the server closes the connection; at this point, the socket to the client can be closed after the data has been written.

If only one of -i or -o are specified, the shell takes responsibility for processing the other side of the connection.

20 **Single process for all connections, serial processing**

In this model, a single process (the co-process) is generated by the shell upon startup; the shell still generates the connections, but passes these connections to the co-process via an IPC mechanism. The shell does not wait for the IPC to be acknowledged, but rather passes an identifier that uniquely identifies the particular pair of sockets corresponding to this connection. Once the co-process has taken control of these connections, the co-process acknowledges this to the shell, and the shell closes its copy of the sockets (this is necessary since the client side will never see an EOF on its socket if multiple processes have this connection open).

35 **Single process for all connections; parallel processing**

This implementation works exactly as described above under the serial processing case, but the co-process manipulates each connection in parallel instead of sequentially. Note that it is the responsibility of the co-process to implement sequential vs. parallel processing; the shell is always asynchronous with respect to transferring connections to the co-process.

COHEN 6-1

Library Routines

This version of the OreO shell packages several functions into a library into a library (liboreo.a). These routines are used both by the OreO shell and by the OreO Shell API functions. These routines are documented here as an aid to those who wish to program OreO agents at a low level interface.

```

5      typedef int Socket;
      int makeArgv(char * string, char * av[], int
10     maxargs)
      Takes as input a text string, and returns a
      vector of pointers that no point to individuals tokens
      in that string (where a token is defined to be a series
15     of non-white-space characters separated by a series of
      white-space characters. White-space characters are
      spaces and tabs. Returns the number of tokens, which
      will be <= the max number of strings allowed. The
      caller must allocate space for the vector of pointers.
20     int readn(Socket s, void * buffer, unsigned
      int size)
      Like read(), but guarantees that size bytes
      are read before returning.
      int written(Socket, void * buffer, unsigned
25     int size)
      Like write(), but guarantees the specified
      number of bytes will be written before the call
      returns. This is important because of protocol
      buffering and flow control, since it is very possible
30     that the write() call will return less than number of
      bytes requested.
      int RecvRights ( Socket IPSock, Socket *
      client, Socket * server)
      This call returns a socket corresponding to
      connections to the client and downstream server. This
35     call hides the mechanisms used to retrieve these
      sockets; such mechanisms are guaranteed to be different
      across operator systems, and may change from release to
      release.

Signal Handling
40     Agent writers should not have to worry about
      signal handling; in fact, a correct implementation
      relies on the default signal handling behavior as
      specified by the POSIX signal handling mechanisms.

```

COHEN 6-1

OreO Shell API

In order to facilitate the creation of OreO agents, we have defined a higher-level API than that presented by the Winsock API. We call this the OreO shell API. This API presents the notion of a connection that can be created and deleted. Each connection contains our two Sockets, and a variable indicating the state of the connection; uninitialized, processing the request, processing the response, or terminating. This API either supports agents written using the co-process model, or agents that receive their sockets on stdin or stdout.

The following example is a rudimentary example of using the Shell API to implement an agent that could be invoked via the `-b` switch.

```

15      ConnectionPtr cp = newOSHConnection(
        StdioConnection);
        // process the request
20      while (nn = OSHConnectionRead(cp->browser,
        buffer,
                sizeof buffer) > 0)
                (void)OSFConnectionWrite( cp->proxy,
        buffer, n);
        while (nn = OSHConnectionRead( cp->proxy,
25      buffer,
                sizeof buffer) > 0
                OSFConnectionWrite( cp->client,
        buffer,n);
        deleteOSHConnection(&cp);
30      This named code would be suitable for
        generating a program to be used as a co-process; in
        this case, the connection would be created by a call to
        newOSHConnection( IPConnection )

```

Future Directions

This is the first version of a UNIX (POSIX) release. Future releases will buffer in implementation details; however, the interfaces defined above will not change, nor will the implementation defined by the OreO Shell API.

One notion is to re-implement the OreO shell as an agent analogue of the internet `inetd`. In this version, the shell would initialize via a configuration mechanism that would indicate a specific port number, a process to run, and how that process should be started. The shell would accept connections on all such port

COHEN 6-1

numbers, and generate the appropriate sequence of commands to start the appropriate agents.

5 An alternative would be to re-implement the shell as a "location broker" for agents, in the style of the DEC RPC daemon. Processes would connect to the Agent daemon, and request services; if available, the daemon would redirect these requests to the appropriate agent. This would probably require a change to the HTTP proxy protocol model.

APPENDIX 3

```
#!/bin/csh

set d="/home/wcohen/code/ss"

killall oreosh

setenv SS_LEARNER "Sd/ss-learn-ripper.pl"

setenv OREO_PROXY radish.research.att.com:8112
oreosh -p 8111 -b "Sd/trap-request.pl ^GET.*QXZYQZ-SS\S* Sd/ss-main.pl"

setenv OREO_PROXY radish.research.att.com:8113
oreosh -p 8112 -b "Sd/markup.pl -e Sd/ss-marker.pl"

setenv OREO_PROXY radish.research.att.com:8000
oreosh -p 8113 -b "Sd/cache.pl"
```

Dec 22 11:08 1995 markup.pl Page 1

```
#!/usr/local/bin/perl

*****
* markup.pl -- insert some text into the top of each HTML page
*
* syntax:
*   markup.pl <text-to-insert>
*   markup.pl -f <file-containing-text-to-insert>
*   markup.pl -e <file-containing-text-to-insert>
*
* -f inserts contents of a file
* -e executes a file and inserts result
* in text, %U is replaced with current document's URL
*****

$ssdir="/home/wcohen/code/ss/";
require "$ssdir/oreo.pl";

$debug=0;

*****
* establish connections

print STDERR "$0: connecting...\n" if $debug;
open(CLIENT, "+<&0");
close(STDIN);
select(CLIENT) ; $| = 1 ;
open(PROXY, "+>&1");
select(PROXY); $| = 1 ;
close(STDOUT);
print STDERR "$0: connections established...\n" if $debug;

*****
* get request, forward to proxy

print STDERR "$0: getting request...\n" if $debug;
$request = &get_request(CLIENT);
print STDERR "$0: sending request:\n" if $debug;
print STDERR "==== begin request =====\n" if $debug;
print STDERR $request if $debug;
print STDERR "==== end request =====\n" if $debug;
print PROXY $request;

*****
*figure what type of file this is...
$URL = &requested_URL($request);
$content = &URL_type($URL) if $URL;

print STDERR "$0: URL=$URL\n" if $debug;
print STDERR "$0: type is $content\n" if $debug;

*****
*construct insertion string

SWITCH: {
  ($ARGV[0] =~ /^-f(.+)$/) && do { $insert = 'cat $1'; last SWITCH; };
}
```

Dec 08 12:18 1995 markup.pl Page 2

```

(SARGV[0] eq "-f")      ;; do { $insert = "cat SARGV[1]"; last SWITCH: };
(SARGV[0] == /^-e(.-)S/) ;; do { $insert = "S1"; last SWITCH: };
(SARGV[0] eq "-e")     ;; do { $insert = "SARGV[2]"; last SWITCH: };
$insert = SARGV[0];
}
$insert == s/\W/SURL/g;

*****
*send proxy's message to client, with $insert
*inserted after the end of the title

while (<PROXY>) {
  if ($content eq "html" ;; /(.*</title>)(.*S)/i) {
    print CLIENT "S1\n$insert\nS2\n";
    $content = "augmented_html"; *never insert twice
  } else {
    print CLIENT;
  }
}

```


Dec 5 16:43 1995 trap-request.pl Page 1

```
#!/usr/local/bin/perl

*****
* trap-request.pl --- invoke a special process for certain requests
*
* syntax: trap-request.pl <form-re> <generator-program>
*   traps out requests matching the regexp <form-re>.
*   then invokes <generator-program>. First argument
*   of generator-program is stuff that matches the form-re,
*   and second argument is data part of request (if any).
*   Generator program writes html to output.
*****

$ssdir="/home/wcohen/code/ss/";
require "$ssdir/oreo.pl";

$debug=0;

*****
* figure out what to trap and how to handle it

$special_re = $ARGV[0];
$generator= $ARGV[1];

print STDERR "$0: handle requests matching '$special_re'\n" if $debug;
print STDERR "$0: response generator is '$generator'\n" if $debug;

*****
* set up connections

print STDERR "$0: connecting...\n" if $debug;
open(CLIENT, "+<&0");
close(STDIN);
select(CLIENT) ; $| = 1 ;
open(PROXY, "+>&1");
select(PROXY); $| = 1 ;
close(STDOUT);
print STDERR "$0: connections established...\n" if $debug;

*****
* get request and echo it

print STDERR "$0: getting request...\n" if $debug;
$request = &get_request(CLIENT);
print STDERR "$0: handling request:\n" if $debug;
print STDERR "---- begin request ----\n" if $debug;
print STDERR $request if $debug;
print STDERR "----- end request -----\n" if $debug;

*****
* process request

if ($request =~ /($special_re)/) {
    local($save);

    $match_string = $1;
```

Dec 4 16:48 1995 trap-request.pl Page 3

```
Srequest =- /\n\r?\n(.*)$/;
Sform_arguments = $1;
send_header(CLIENT);
Ssave = S; $i = 1;
open(RESPONSE, "Sgenerator 'Smatch_string' Sform_arguments;")
  | die("can't invoke generator");
while (<RESPONSE>) {
  print CLIENT;
}
close(RESPONSE);
Si = Ssave;
} else {
  print PROXY Srequest;
  while (<PROXY>) {
    print CLIENT S_;
  }
}
}
```

Dec 02 13:49 1995 cache.pl Page 1

```
#!/usr/local/bin/perl

#####
# cache.pl --- implement a simple cache
#
# for now--restricted to html documents.  must be invoked with -i
# option to avoid contention for cache
#
#####

$ssdir="/home/wcohen/code/ss/";
require "$ssdir/oreo.pl";
require "$ssdir/cache-util.pl";

$debug=0;

#####
# initialize cache variables, etc

($cachedir,$maxcachesize)=&init_cache;
print STDERR "$0: max=$maxcachesize, cachedir=$cachedir\n" if $debug;

#####
# set up connections

open(CLIENT, "<&0");
close(STDIN);
select(CLIENT) ; $| = 1 ;
open(PROXY, ">&1");
select(PROXY); $| = 1 ;
close(STDOUT);

#####
# get request and echo it

$request = &get_request(CLIENT);
print STDERR "$0: handling request:\n" if $debug;
print STDERR "==== begin request =====\n" if $debug;
print STDERR $request if $debug;
print STDERR "===== end request =====\n" if $debug;

#####
# process request

$URL = &requested_URL($request);
$type = &URL_type($URL) if $URL;

print STDERR "$0: type=$type URL=$URL\n" if $debug;

if ($URL) {
    $cache_listing = &load_cache_listing;
    $filename = &url2file(&lookup_url($URL,$cache_listing));
    if ($filename && -e $filename && !($request =~ /Pragma: no-cache/)) {
        print STDERR "$0: cache has URL in $filename\n" if $debug;
        #replay the response found in cache
        open(RESPONSE,"<$cachedir/$filename") || die("can't open cached file");
    }
}

```

Dec 22 13:49 1995 cache.pl Page 2

```

while(<RESPONSE>) {
    print CLIENT;
}
close(RESPONSE);
} else {
    if ($filename) {
        print STDERR "$0: removing file $filename...\n" if $debug;
        #request was that cache not be used
        #so expunge the old cache entry
        unlink("$cachedir/$filename");
        #system "rm -f $filename" && &complain;
        for ($i=0; $i<=$cache_listing; $i++) {
            $cache_listing[$i] = "" if $cache_listing[$i] eq "SURL\n";
        }
        #get an answer from the proxy, record in tempfile
        print STDERR "$0: cache has no URL SURL\n" if $debug;
        open(TEMP, ">$cachedir/TEMP") || die("can't open cache temp file");
        print PROXY $request;
        while(<PROXY>) {
            print CLIENT;
            print TEMP;
        }
        close(TEMP);
    }
    #update_cache(SURL, $cache_listing);
} else {
    print STDERR "$0: not a cacheable response\n" if $debug;
    #not a request--so handle it normally
    print PROXY $request;
    while(<PROXY>) {
        print CLIENT;
    }
    close(TEMP);
}

sub update_cache #($url, $listing)
{
    local($url, $listing) = @_;
    local($filename);

    print STDERR "$0: caching $url\n" if $debug;

    #delete url from cache and append to the end
    for ($i=0; $i<=$#listing; $i++) {
        $listing[$i] = "" if $listing[$i] =~ /$url$/;
    }
    $listing[$#listing+1] = "$url\n";

    # print STDERR "$0: cache + $url:\n" if $debug;
    # #show_cache($listing) if $debug;

    # truncate cache to appropriate size
    print STDERR "$0: new cache has $#listing, limit is $maxcachesize\n"
        if $debug;
    if ($#listing >= $maxcachesize) {

```

Dec 22 10:49 1998 cache.pl Page 3

```

    Sndel = $#listing-$maxcachesize;
    for (Si=0; Si<Sndel; Si--) {
        Sfilename = `url2file($listing[$i]);
        unlink("$Scachedir/$Sfilename");
        *system "rm -f $Scachedir/$Sfilename" && &complain;
        $listing[$i] = "";
    }
}
# save the cache
open(LISTING, ">$Scachedir/LISTING") || die("can't write cache listing file");
print LISTING join("", grep(/./, @listing));
close(LISTING);

# move the response file to appropriate place
if (-e "$Scachedir/TEMP" && -s "$Scachedir/TEMP") {
    Sfilename = `url2file($url);
    print STDERR "S0: url->file $Sfilename\n" if $sdebug;
    rename("$Scachedir/TEMP", "$Scachedir/$Sfilename") ; &complain;
}
}

sub complain
{
    print STDERR "S0: command fails!\n";
}

sub show_cache +(cache)
{
    local(@listing) = @_;

    print STDERR "-----\n";
    print STDERR join("", grep(/./, @listing));
    print STDERR "-----\n";
}

```

Dec 14 15:22 1995 oreo.pl Page 1

```

#!/usr/local/bin/perl
*****
#
# oreo.pl -- Perl routines to be used with the creosh
#
*****

sub requested_URL *(request)
{
    local($request) = $_[0];

    ($request =- /^GET http:(.*) HTTP\/1\.\.0\/) ? $1 : "";
}

sub URL_type *(url)
{
    local($URL) = $_[0];
    local($stype);

    $stype="html" if ($URL =- /\s/ ! $URL =- /html?s/);
    $stype="html" if ($URL =- /cgi-bin/);
    $stype;
}

sub get_request *(client)
{
    local($client) = $_[0];
    local($request,$content_length);

    while(<$client>) {
        $request .= $_;
        $content_length = $1 if (/^Content-length: (.*)$/);
        last if /\r?\n$/;
    }
    while ($content_length-->0) {
        $request .= (getc $client);
    }
    $request;
}

sub send_header *(client)
{
    local($client) = $_[0];

    print $client "HTTP/1.0 200 OK\r\n";
    print $client "MIME-version: 1.0\r\n";
    print $client "Content-type: text/html\r\n";
    print $client "\r\n";
}

# return a true value
1;

```

Dec 13 11:13 1995 cache-util.pl Page 1

```
#!/usr/local/bin/perl

#####
# cache-util.pl --- utilities for cache.pl
#####

sub init_cache
(
    local($ss,$sdir,$smax);

    $smax = 40;
    $ss="ENV{'HOME'}/.ss";
    if (!(-e "$ss")) {
        mkdir("$ss",0777);
    }
    $sdir="$ss/cache";
    if (!(-e "$sdir")) {
        mkdir("$sdir",0777);
    }
    ($sdir,$smax);
)

sub load_cache_listing
(
    local($!listing);
    open(LISTING,"<$sdir/LISTING");
    $!listing = <LISTING>;
    close(LISTING);
    $!listing;
)

sub lookup_url $(url,$!listing)
(
    local($surl,$!listing) = @_;
    foreach $scached_url ($!listing) {
        return $surl if ($scached_url eq "$surl\n");
    }
    return 0;
)

sub url2file $(url)
(
    local($surl) = @_;
    $surl -- s(\.){#S}g;
    $surl -- s(\?){#Q}g;
    $surl -- s(\&){#A}g;
    $surl -- s(\n){#N}g;
    $surl;
)

1;
```

Dec 6 16:43 1995 ss-help.pl Page 1

```
#!/usr/local/bin/perl
*****
*
* ss-help.pl -- help routines
*
*****

require "Sssdir/ss-util.pl";
Sdebug=0;

*****
* invoke ripper on these examples

sub help_command *(datadir,class,url)
{
    local($datadir,$class,$url) = @_;

    print <<END_OF_TEXT:
<html>
<title>Help screen</title>
<h2>Help</h2>
No on-line help is available yet.  If you want an explanation
of this system contact
<a href="mailto:wcohen@research.att.com">
William Cohen
</a>
END_OF_TEXT
        &send_foot($url):
}

1;
```


Dec 6 16:48 1995 ss-label.pl Page 1

```
#!/usr/local/bin/perl
*****
:
* ss-label.pl -- routines to label an html document
*****

require "$ssdir/cache-util.pl";
require "$ssdir/ss-util.pl";

*****
= label a URL, getting the actual document from
= the cache

sub label_command *(datadir, class, url, label)
{
    local($datadir, $class, $url, $label) = @_;
    local($filename, $title);

    ($cachedir, $maxcachesize) = @init_cache;
    print STDERR "S0: max=$maxcachesize, cachedir=$cachedir\n" if $debug ;

    #try to find the url in the cache
    %cache_listing = %load_cache_listing;
    $filename = %url2file(%lookup_url($url, %cache_listing));

    if (!$filename) {
        %send_error("can't find anything in cache for URL $url. Try reloading, then re-la.
    } else {
        #set up subdirectories of labeled URL's
        mkdir("$datadir/classes/$class/Y", 0777)
            unless (-e "$datadir/classes/$class/Y");
        mkdir("$datadir/classes/$class/N", 0777)
            unless (-e "$datadir/classes/$class/N");

        #copy cache file to new label file
        open(CACHE, "<$cachedir/$filename") || die("can't open cached file");
        open(LABEL, ">$datadir/classes/$class/$label/$filename")
            || die("can't open label file for write");
        while(<CACHE>) {
            print LABEL;
        }
        close(CACHE);
        close(LABEL);

        #record that URL was labeled
        open(EXAMPLELISTING, ">>$datadir/classes/$class/LISTING.$label")
            || die("can't append to listing file");
        print EXAMPLELISTING $url, "\n";
        close(EXAMPLELISTING);

        if (-e "$datadir/classes/$class/COUNTER") {
            open(COUNTER, "<$datadir/classes/$class/COUNTER")
                || die("can't read counter");
            $count = <COUNTER>; chop($count);
            close(COUNTER);
        } else {
```

Dec 6 15:48 1995 ss-label.pl Page 2

```
        Scount = 0;
    }
    open(COUNTER, ">sdatar/dir/classes/$class/COUNTER")
        || die("can't write counter");
    print COUNTER "--$count.\n";
    close(COUNTER);

    #send acknowledgement
    $title = `html_title("$sdatar/dir/classes/$class/$label/$filename");
    print <<END_OF_TEXT:

<html>
<title>Label acknowledgement</title>
<body>
<h2>Label acknowledgement</h2>
<p><strong>Received</strong>: label of
`quote`strong`$label`quote (class `quote`strong`$class`quote)
for the document entitled `quote`strong`$title`quote
and located at http:$url.
</p>
END_OF_TEXT
    `send_foot($url);
    } * else filename was found in cache
}

l;
```

Dec 2 12:08 1995 ss-learn-ripper.pl Page 1

```
#!/usr/local/bin/perl
*****
#
# ss-learn-ripper.pl -- routines to allow user to learn search commands
#   for SS using Ripper
#
# A file should define these routines:
#
#   &learn_command(datadir,class,url) -- generates the HTML page
#   that the user sees when he clicks on the "learn" option.
#   The generated page is printed to STDOUT. Generally it will
#   contain, somewhere, this special anchor:
#       http://QQXYZQQ-SS-Surl-invoke-learner
#   When accessed this causes &invoke_learner (see below) to be run.
#   It might also contain a form with the special action
#       http://QQXYZQQ-SS-Surl-learner-options-form
#   which, when submitted, causes the process_learner_form to
#   be invoked on its submitted arguments.
#
#   &invoke_learner(datadir,class,url) -- runs the learner and
#   generates an HTML page indicating status of the run command.
#
#   &process_learner_form(datadir,class,url,options) -- handles
#   the setting of run-time options for the learning system.
#
*****

require "Sssdir/ss-util.pl";

$debug=0;

# send an options form for the learner, and a link that invokes the learner

sub learn_command #(datadir,class,url)
{
    local($datadir,$class,$url) = @_ ;
    local($schk,$sychk,$sropts,$snt,$spwt,$swts);

    print "<html><title>Invoke Rule Learner</title>\n";
    print "<body><h2>Invoke Rule Learner</h2>\n";

    print "Click <a href=\"http://QQXYZQQ-SS-Surl-invoke-learner\"> here\n";
    print "</a> to invoke the rule learner on class &quot;$class&quot;.\n";
    print "This may take a minute or two.\n";
    print "<hr>\n";

    &options = &read_options($datadir,$class);

    #options form
    print "<h2>Set learning options</h2>\n";
    print "<form action=\"http://QQXYZQQ-SS-Surl-learner-options-form\">\n";
    print "<p>The options below modify RIPPER's behavior in learning\n";
    print "the class &quot;$class&quot;.\n";

    $schk = $sychk = "";
    $schk = "checked" if (grep(/rare No/, &options));

```

Dec 3 10 18 1998 ss-learn-ripper.pl Page 2

```

$ychk = "checked" unless $snchk;
print "<p><strong> Assume the class is rare: </strong>\n";
print " <input name=\"rare\" type=\"radio\" $ychk value=\"Yes\">Yes</input>\n";
print " <input name=\"rare\" type=\"radio\" $snchk value=\"No\">No</input>\n";
print "</p>\n";

if (($swts = grep(/^weights/, $options)) {
    ($spwt, $snwt) = ($swts =~ /^weights (\S-) (\S-)/);
} else {
    ($spwt, $snwt) = (1, 1);
}
print "<p><strong> Weight for positive examples: </strong>\n";
print " <input name=\"pwt\" type=\"text\" value=\"$spwt\"></input>\n";
print "<br>\n";
print "<p><strong> Weight for negative examples: </strong>\n";
print " <input name=\"nwt\" type=\"text\" value=\"$snwt\"></input>\n";
print "</p>\n";

($sropts) = grep(/^command_line .*$/, $options);
$sropts = $sropts =~ s/^command_line //;
$sropts = "-v" unless $sropts;
print STDERR "ropts: $sropts\n" if $sdebug;
print "<p><strong>RIPPER command-line options: </strong>\n";
print " <input name=\"ripper\" type=\"text\" size=40 value=\"$sropts\"></input>\n";
print "</p>\n";
print " <input type=\"submit\" value=\"Set options\"> </input>\n";
print " <input type=\"reset\" value=\"Reset\"> </input>\n";
print "</form>\n";
&send_foot($url);
}
:

# invoke ripper on these examples

sub invoke_learner *($datadir, $class, $url)
{
    local($sdatadir, $sclass, $surl) = @_;
    local($options, $sdefneg, $sriopts, $scom, $swd, $ssave, $spwt, $snwt, $swts);

    #autoflush buffers
    $save = $|; $| = 1;
    $swd = "$sdatadir/classes/$sclass";

    print "<html>\n";
    print "<title>Output of RIPPER rule learner</title>\n";
    print "<body>\n";
    print "<h3>Output of RIPPER rule learner</h3>\n";

    print "<p>Preparing data for RIPPER...\n";

    #read in options
    $options = &read_options;

    unless (grep(/rare No/, $options)) {
        $sdefneg = "$sdatadir/defneg.data";
        if (!(-e $sdefneg) || !(-s $sdefneg)) {

```

Doc 8 12:08 1995 ss-learn-ripper.pl Page 3

```

#create a data file for default negative examples
local($findcom) =
    "find /usr/local/www/cache/http \\( -name '*.html' -o -name '*.htm' \\)";
print "Extracting negative examples from proxy cache...\n";
print STDERR "findcom: $findcom\n";
open(DEFNEG, ">$defneg") || die("can't create defneg");
open(CACHELIST, "$findcom -print:") || die("can't list proxy cache");
while(<CACHELIST>) {
    chop;
    print "<dd> <code>$_</code>\n";
    $file2example($datadir, "$_", "N", DEFNEG) if (-e $_);
}
close(CACHELIST);
close(DEFNEG);
}

if (($swts) = grep(/^weights/, $options)) {
    ($spwt, $snwt) = ($swts == /^weights (\S+) (\S-)?/);
} else {
    ($spwt, $snwt) = (1, 1);
}

open(DATA, ">$wd/web.data") || die("can't write to web.data");
$prepare_examples($datadir, $class, DATA, "Y", "$spwt");
$prepare_examples($datadir, $class, DATA, "N", "$snwt");
close(DATA);
#create names file
open(NAMES, ">$wd/web.names") || die("can't write to web.names");
print NAMES "Y.N.\n";
print NAMES "WORDS: set.\n";
close(NAMES);
print "prepared.\n</p>";

print "<p>Invoking RIPPER...</p>\n";
print "<pre>\n";
#invoke ripper--read output from a pipe
($ripropts) = (join(" ", $options) == /command_line (.*)$/);
print STDERR "use defneg: $defneg\n" if $debug;
print STDERR "ripper options: $ripropts\n" if $debug;
$scatcom = "cat $wd/web.data $defneg | clean-data -c Y -s $wd/web";
open(RIPOUT, "$scatcom | ripper -a given -s $ripropts $wd/web|")
    || die("can't execute ripper");
while(<RIPOUT>) {
    print;
}
close(RIPOUT);
print "</pre>\n";

#update counter
open(COUNTER, ">$datadir/classes/$class/COUNTER")
    || die("can't write to counter");
print COUNTER "0\n";
close(COUNTER);

#finish off html file
$send_foot($url);

```

Dec 2 13:08 1995 ss-learn-ripper.pl Page 4

```

S! = Ssave:
)

sub prepare_examples +(Sdatadir, Sclass, Sout, Slabel, Swt)
{
    local(Sdatadir, Sclass, Sout, Slabel, Swt) = @_ ;
    local(Sfile);

    opendir(LABELED, "Sdatadir/classes/Sclass/Slabel")
    !! die("can't list labelled files");
    while (Sfile = readdir(LABELED)) {
        if (Sfile =~ /[^\s]/) {
            print STDERR "S0: will label file Sfile as Slabel\n" if Sdebug;
            &file2example(Sdatadir,
                "Sdatadir/classes/Sclass/stopwords",
                "Sdatadir/classes/Sclass/Slabel/Sfile",
                Slabel, Sout, Swt);
        }
    }
    closedir(LABELED);
}

#####

sub process_learner_form +(datadir, class, url, options)
{
    local(Sdatadir, Sclass, Surl, Soptions) = @_ ;
    local($rare, $pwt, $nwt, $command_line);

    ($rare) = ($options =~ /[?&]rare=([^\&\/]*)[\&\/]/);
    ($pwt) = ($options =~ /[?&]pwt=([^\&\/]*)[\&\/]/);
    ($nwt) = ($options =~ /[?&]nwt=([^\&\/]*)[\&\/]/);
    ($command_line) = ($options =~ /[?&]ripper=(.*)$/);
    $command_line =~ tr/+// ;
    $command_line =~ s/%21/!/g;
    print STDERR "S0: rare=$rare\n" if Sdebug;
    print STDERR "S0: command_line=$command_line\n" if Sdebug;

    &write_options(Sdatadir, Sclass,
        "rare $rare",
        "weights $pwt $nwt",
        "command_line $command_line");

    print "<html>\n";
    print "<title>New option acknowledgement</title>\n";
    print "<body>\n";
    print "<h2>New option acknowledgement</h2>\n";
    print "<p><strong>Received</strong>: the following options\n";
    print "have been set for class &quot;<strong>$class</strong>&quot;:\n";
    print "</p>\n";
    print "<p> <dd> Assume the class is rare: <strong>$rare</strong> </p>\n";
    print "<p> <dd> Example weights: <strong>$pwt for pos, $nwt for neg</strong> </p>\n";
    print "<p> <dd> Options to RIPPER: <strong>$command_line</strong> </p>\n";
    print "<p> Click <a href=\"http://QQXYZQQ-ss-surl-invoke-learner\"> here\n";
    print "</a> to invoke the rule learner with these options.\n";
    print "This may take a minute or two.\n";
}

```

Dec 3 12:03 1996 ss-learn-ripper.pl Page 5

```
    isend_foot($url);
}

sub read_options
{
    local($datadir,$class) = @_ ;

    open(OPTIONS,"<$datadir/classes/$class/RIPPER.OPTS");
    %options = <OPTIONS>;
    close(OPTIONS);
    print STDERR "options: %options\n" if $debug;
    %options;
}

sub write_options
{
    local($datadir,$class,%options) = @_ ;

    open(OPTIONS,">$datadir/classes/$class/RIPPER.OPTS");
    .. die("can't write to options file");
    print OPTIONS join("\n",%options),"\n";
    close(OPTIONS);
}

1;
```

Dec 6 16:48 1995 ss-main.pl Page 1

```

#!/usr/local/bin/perl
*****
*
* ss-main.pl -- main routines for handling labeling commands
*
* invoked by trap-request.pl, so arguments are <match-string>
*   and <form-arguments>... only match-string is used
*
*****

$ssdir="/home/wcohen/code/ss";
require "$ssdir/ss-util.pl";
require "$ssdir/ss-label.pl";
require "$ssdir/ss-review.pl";
require "$ssdir/ss-options.pl";
require "$ssdir/ss-search.pl";
require "$ssdir/ss-help.pl";

#load the user-defined learning program
require "ENV('SS_LEARNER')";

$debug=0;

*****
# figure out class

($datadir,$class) = &init_ss;

*****
# figure out intended command, and perform it

$match = $ARGV[0];
print STDERR "ss-main.pl was invoked for match '$match'\n" if $debug;
if ($match == /GET.*QXZYQO-SS-(.*)-label-([NY])$/) {
    &label_command($datadir,$class,$1,$2);
} elsif ($match == /GET.*QXZYQO-SS-(.*)-review$/) {
    &review_command($datadir,$class,$1);
} elsif ($match == /GET.*QXZYQO-SS-(.*)-learn$/) {
    &learn_command($datadir,$class,$1);
} elsif ($match == /GET.*QXZYQO-SS-(.*)-invoke-learner$/) {
    &invoke_learner($datadir,$class,$1);
} elsif ($match == /GET.*QXZYQO-SS-(.*)-learner-options-form(?:.*)$/) {
    &process_learner_form($datadir,$class,$1,$2);
} elsif ($match == /GET.*QXZYQO-SS-(.*)-search$/) {
    &search_command($datadir,$class,$1);
} elsif ($match == /GET.*QXZYQO-SS-(.*)-ss-class-form(?:.*)$/) {
    &process_ss_class_form($datadir,$class,$1,$2);
} elsif ($match == /GET.*QXZYQO-SS-(.*)-set-ss-options$/) {
    &set_ss_options_command($datadir,$class,$1);
} elsif ($match == /GET.*QXZYQO-SS-(.*)-help$/) {
    &help_command($datadir,$class,$1);
} else {
    &send_error("Unknown command -- match was $match");
}

```


Dec 8 15:48 1998 ss-main.pl Page 2

Dec 18 13:40 1995 ss-marker.pl Page 1

```
#!/usr/local/bin/perl
*****
*
* ss-marker.pl -- generates the text used to mark up html forms by markup.pl
*
* invoked by oreosh -b markup.pl
*
*****

Sdebug=0;

$ssdir="/home/wcohen/code/ss";
require "$ssdir/ss-util.pl";

print STDERR "$0: writing marker...\n" if $debug;
($datadir,$class) = &init_ss;
print STDERR "$0: class is $class...\n" if $debug;

print <<END_OF_INSERT;
<i>
<strong>Swimsuit:</strong>
Is this in the class '$class'?
[ <a href="http://QXZYQZ-SS-%U-label-Y"> yes </a>
  <a href="http://QXZYQZ-SS-%U-label-N"> no </a>
] <br>
[ <a href="http://QXZYQZ-SS-%U-review"> Review Previous </a>
  <a href="http://QXZYQZ-SS-%U-learn"> Learn </a>
  <a href="http://QXZYQZ-SS-%U-search"> Search </a>
  <a href="http://QXZYQZ-SS-%U-set-ss-options"> Set Options </a>
  <a href="http://QXZYQZ-SS-%U-help"> Help </a> ]
</i>
<br>
END_OF_INSERT

print STDERR "$0: marker for class $class written\n" if $debug;
```

Dec 6 16:48 1995 ss-options.pl Page 1

```
#!/usr/local/bin/perl
*****
#
# ss-options.pl -- routines to allow user to set options
# for the labeling system
#
*****

require "Sssdir/ss-util.pl";

Sdebug=1;

*****
# process a [Set Options] command from the
# inserted .html page

sub set_ss_options_command *(datadir,class,url)
{
    local($datadir,$class,$url) = @_ ;
    local($old_class,$engine,$salt_engine);

    #send head of control panel
    print "<html><title>Control Panel</title>\n";
    print "<body><h2>Control Panel</h2><hr>\n";
    print "<form action=\"http://QXZYQZQ-SS-$url-ss-class-form\">\n";

    #the change classes section of the form
    #print a menu of old classes
    print " <strong> Change the current class</strong> to an old class:\n";
    print " <select name=\"oldclass\" size=1>\n";
    print " <option selected> $class\n";
    opendir(CLASSDIR,$datadir/classes) || die("can't list classes");
    while($old_class = readdir(CLASSDIR)) {
        next if ($old_class =~ /\.\.|\.\.?/);
        next if ($old_class eq $class);
        print " <option>$old_class\n";
    }
    closedir(CLASSDIR);
    print " </select>\n";
    #print an option to enter a new class
    print " or a new class: <input name=\"newclass\"> </input>\n";
    print " <p>For now, please do not include punctuation or spaces\n";
    print " in class names.</p>\n";

    #a change search-procedure form
    $engine = $load_engine_name($datadir);

    print "<hr>\n";
    print "<form action=\"http://QXZYQZQ-SS-$url-ss-class-form\">\n";
    #print a menu of known engines
    print " <strong> Change the current search engine</strong>: \n";
    print " <select name=\"engine\" size=1>\n";
    print " <option selected> $engine\n";
    opendir(ENGINEDIR,$datadir/engines) || die("can't list engines");
    while($salt_engine = readdir(ENGINEDIR)) {
        next if ($salt_engine =~ /\.\.|\.\.?/);

```

Dec 5 16:48 1998 ss-options.pl Page 2

```

    next :if ($Salt_engine == /-S/); #skip backups
    next :if ($Salt_engine eq $engine);
    print "    <option>Salt_engine\n";
}
closedir(ENGINEDIR);
print " </select>\n";
# submit buttons
print " <hr>\n";
print " <input type=\"submit\" value=\"Change values\"></input>\n";
print " as indicated above, or\n";
print " <input type=\"reset\" value=\"reset\"> </input> the form.\n";
print "</form>\n";

    $send_foot($url);
}

*****
# process the form associated with the
# [Set Options] page

sub process_ss_class_form {($datadir,$class,$options)
{
    local($datadir,$class,$url,$options) = @_;
    local($oldclass,$newclass,$oldengine,$newengine);

    print STDERR "$0: options = $options\n" if $debug;
    ($oldclass) = ($options =~ /[\?&]oldclass=([\^\&\/]*)([\&\/]\/)/);
    ($newclass) = ($options =~ /[\?&]newclass=([\^\&\/]*)([\&\/]\/)/);
    ($newengine) = ($options =~ /[\?&]engine=(.*)$/);
    print STDERR "$0: oldcl=$oldclass\n" if $debug;
    print STDERR "$0: newcl=$newclass\n" if $debug;
    print STDERR "$0: engine=$engine\n" if $debug;
    $next_class = $newclass || $oldclass;
    unless (($next_class eq $class)) {
        print STDERR "$0: changing class, $class->$newclass\n" if $debug;
        open(CLASS, ">$datadir/CLASS") || die("can't write current class");
        print CLASS "$next_class\n";
        close(CLASS);
        $class = $next_class;
    }
    $oldengine = $load_engine_name($datadir);
    unless (($newengine eq $oldengine)) {
        print STDERR "$0: changing engine to $newengine\n" if $debug;
        open(ENGINE, ">$datadir/ENGINE") || die("can't write current engine");
        print ENGINE "$newengine\n";
        close(ENGINE);
    }
}

print "<html><title>New option acknowledgement</title>\n";
print "<body><h2>New option acknowledgement</h2>\n";
print "<p><strong>Changed</strong>: the current class\n";
print "is now &quot;$class&quot;</p>\n";
print "<p><strong>Changed</strong>: the current search engine\n";
print "is now &quot;$newengine&quot;</p>\n";

```

Dec 6 16:46 1995 ss-options.pl Page 3

```
    send_foot: Surl;;  
};  
  
1:
```

Dec 6 16:48 1995 ss-review.pl Page 1

```
#!/usr/local/bin/perl
*****
#
# ss-review.pl -- routines to review previously labeled documents
*****

require "Sssdir/ss-util.pl";
require "Sssdir/cache-util.pl";

sub review_command *(datadir,class,Surl)
{
    local($datadir,$class,$Surl) = @_;

    print "<html><title>Listing for class $class</title><body>\n";
    print "<h2>Previously marked examples of "$class"</h2>\n";
    &review_examples("Y","Positive");
    &review_examples("N","Negative");
    &send_foot($Surl);
}

sub review_examples *(short,long)
{
    local($slab,$slabname) = @_;
    local($title,$Surl,$file,$found);

    print "<h3>$slabname Examples</h3>\n<ul>\n";
    if (-e "$datadir/classes/$class/LISTING.$slab") {
        $found = open(LIST,"sort $datadir/classes/$class/LISTING.$slab | uniq |")
        || die("can't read/sort/uniq classes");
    }

    if (!$found) {
        print "<li> <i>No examples marked</i></li>\n";
    } else {
        while($Surl = <LIST>) {
            chop $Surl;
            $file = $url2file($Surl);
            $title = $html_title("$datadir/classes/$class/$slab/$file");
            print "<li><a href=\"$Surl\"> $title </a> </li>\n";
        }
        close(LIST);
    }
    print "</ul>\n";
}

1;
```

Dec 6 17:10 1995 ss-search.pl Page 1

```
#!/usr/local/bin/perl
*****
*
* ss-search.pl -- routines to allow user to learn search commands
*
*****

$ssdir="/home/wcohen/code/ss";
require "$ssdir/ss-util.pl";

$debug=1;

*****
* process the form associated with the
* [Set Options] page

sub search_command { (datadir, class, url)
{
    local($datadir, $class, $url) = @_;
    local($fields, $rule);
    local($engine) = $load_engine_name($datadir);

    print "<html>\n";
    print "<title>Ways to search for class &quot;$class&quot;</title>\n";
    print "<body>\n";
    print "<h3>Ways to search for the class &quot;$class&quot;</h3>\n";
    if (!( -e "$datadir/classes/$class/web.hyp" )) {
        print "<li><i>No rules have been learned</i>\n";
    } else {
        #Access counter
        open(COUNTER, "<$datadir/classes/$class/COUNTER")
            || die("can't read counter");
        $count = <COUNTER>; chop($count);
        close(COUNTER);
        $count = "No" if (!$count);
        $verb = ($count==1) ? "example has" : "examples have";
        open(RULES, "$datadir/classes/$class/web.hyp") || die("can't read rules file");
        print "<p>$count $verb been labeled since these rules were learned.\n</p>";
        print "<p>The current search engine is $engine.</p>\n";
        print "<ul>\n";
        while (<RULES>) {
            $fields = split;
            $rule = ();
            if ($fields[0] eq "Y") {
                #collect terms from rule
                for ($i=6; $i<=$#fields; $i+=3) {
                    if ($fields[$i-1] eq '-') {
                        push($rule, ("+" . $fields[$i]));
                    } else {
                        push($rule, ("- " . $fields[$i]));
                    }
                }
                print "<li> ($fields[1] right/$fields[2] wrong) ";
                print STDERR "$datadir/engines/$engine $rule\n" if $debug;
                print "$datadir/engines/$engine $rule";
            }
        }
    }
}
}
```

Dec 6 17:10 1995 ss-search.pl Page 2

```
        print "</li>\n";
    } elsif ($fields[0] eq "N") {
        print "<li> ($fields[1] right/$fields[2] wrong) ";
        print "not covered by the rules above.\n</li>\n";
    }
}
close(RULES);
print "</ul>\n";

#send_foot($url);
}
1;
```


Dec 8 17:11 1995 ss-util.pl Page 1

```
#!/usr/local/bin/perl
*****
=
= ss-util.pl -- misc utilities
=
*****

Sdebug=0;

Sssroot="`ENV{'HOME'}`/..ss";

*****
* initialize directories, return directory&class

sub init_ss
{
    local($Sss,$Sdata,$Scl);
    $Sss = $Sssroot;
    if (!(-e "$Sss")) {
        mkdir("$Sss",0777) || die("can't create '$Sss' directory");
    }
    $Sdata = "$Sss/data";
    if (!(-e $Sdata)) {
        mkdir($Sdata,0777) || die("can't create data directory");
    }
    if (!(-e "$Sdata/engines")) {
        print STDERR "SO: need an $Sdata/engines directory\n" if $Sdebug;
        mkdir("$Sdata/engines",0777)
            || die("can't create engine directory $Sdata/engines");
    }
    if (!(-e "$Sdata/ENGINE")) {
        open(ENGINE,">$Sdata/ENGINE") || die("can't create $Sdata/ENGINE");
        print ENGINE "webcrawler\n";
        close(ENGINE);
    }
    if (open(CLASS,"<$Sdata/CLASS")) {
        $Scl = <CLASS>; chop($Scl);
        close(CLASS);
    } else {
        $Scl = "cool";
    }
    if (!(-e "$Sdata/classes")) {
        mkdir("$Sdata/classes",0777);
    }
    if (!(-e "$Sdata/classes/$Scl")) {
        mkdir("$Sdata/classes/$Scl",0777);
    }
    ($Sdata,$Scl);
}

*****
* figure out the title of an html document

sub html_title *(filename)
{
    local($file) = $_[0];

```

Dec 8 17:11 1995 ss-util.pl Page 2

```

local($n,$start,$title);

#get up to first N lines
$n = 50;
open(HTML,"<$file") || die("can't open HTML file $file");
while (($s_ = <HTML>) && $n--) {
    chop;
    $start .= ($s_ . " ");
}
close(HTML);

$title = $1 if ($start =~ /<title>(.*?)<\/title>/i);
$title = "Apparently Untitled" if (!$title);
* print STDERR "title is $title\n";
$title;
}

*****
* send an error message

sub send_error *(message)
{
    print "<html>\n";
    print "<title>Error message</title>\n";
    print "<body>\n";
    print "<p>Error: $_[0]\n";
    print "</body>\n";
    print "</html>\n";
}

*****
* print a file footer

sub send_foot *(url)
{
    local($url) = $_[0];

    print <<END_OF_TEXT:
<hr>
<p><i>
[ <a href="http:$url"> Resume Browsing </a>
| <a href="http://QOXYZQO-SS-$url-review"> Review Previous </a>
| <a href="http://QOXYZQO-SS-$url-learn"> Learn </a>
| <a href="http://QOXYZQO-SS-$url-search"> Search </a>
| <a href="http://QOXYZQO-SS-$url-set-ss-options"> Set Options </a>
| <a href="http://QOXYZQO-SS-$url-help"> Help </a> ]
</i></p>
</body>
</html>
END_OF_TEXT
}

*****
* convert a file to an example for ripper
*

```

Dec 8 17:11 1995 ss-util.pl Page 3

*if wt is present it should be an argument ":x"

```

sub file2example =(datadir,stopwordfile,file,label,outhandle[,wt])
{
    local($datadir,$stopwordfile,$file,$label,$out,$wt) = @_ ;
    local($lines,$stopwords,$extra_stopwords,$extra_stopword_pattern);

    unless ($Sstopword_pattern) {
        open(STOPWORDS,"$datadir/stopwords") || die("can't find stopwords file");
        $stopwords = <STOPWORDS>;
        close(STOPWORDS);
        chop($stopwords);
        $Sstopword_pattern = "\\b(" . join("|",$stopwords) . ")\\b";
    }
    if ($Sstopwordfile == -e $stopwordfile) {
        *class-specific stopwords
        open(STOPWORDS,"<$stopwordfile") || die("can't find stopwords file $stopwordfile");
        $extra_stopwords = <STOPWORDS>;
        close(STOPWORDS);
        chop($extra_stopwords);
        $extra_stopword_pattern = "\\b(" . join("|",$extra_stopwords) . ")\\b";
    } else {
        $extra_stopword_pattern = "";
    }

    open(FILE,$file) || die("can't open html file $file");
    *skip header
    while(<FILE>) {
        last if /^\\r?\\n$/;
    }
    $lines=0;
    while(<FILE>) {
        last if $lines++ > 100;

        *delete e-mail addresses and stopwords
        s/$Sstopword_pattern//g;
        s/$extra_stopword_pattern//g if $extra_stopword_pattern;
        s/^w*@([w\\.])*//g;

        *delete HTML special characters
        s/&.*/g;

        *convert to lowercase
        tr/A-Z/a-z/;

        *delete complete HTML commands of the form <...>
        s/<[^>]*>/g;

        * now print what's left, again
        * deleting stuff between <...>'s

        if (s/<[^>]*>/) {
            *open without close
            *remove non-alphanumerics

```

Dec 8 17:11 1995 ss-util.pl Page 4

```

        tr/a-z0-9\n/ /c;
        print $out $_;
        $open_bracket = 1;
    } elsif ($open_bracket && s/^[^<]*>//) {
        *close without open
        *remove non-alphanumerics
        tr/a-z0-9\n/ /c;
        print $out $_;
        $open_bracket = 0;
    } elsif (!$open_bracket) {
        *remove non-alphanumerics
        tr/a-z0-9\n/ /c;
        print $out $_;
    }
}
print $out "\n$label$wt.\n";
close(FILE);
}

sub load_engine_name $datadir
{
    local($datadir) = $_[0];
    local($engine);

    open(ENGINE, "<$datadir/ENGINE") || die("can't local search engine");
    chop($engine = <ENGINE>);
    close(ENGINE);
    $engine;
}

1;

```

COHEN 6-1

5 While the invention has been shown and described with respect to preferred embodiments, various modifications can be made therein without departing from the spirit and scope of the invention, as described in the specification and defined in the claims, as follows:

I claim:

COHEN 6-1

1. A method of adding new documents to a resource list of existing documents, comprising the steps of:

5 learning selection information which selects the documents on the resource list;

making a persistent association between the selection information and the resource list;

using the selection information to select a set of documents which the information specifies; and

10 adding new documents to the resource list, the new documents being added belonging to a subset of the selected set of documents which contains documents which are not already on the resource list.

2. The method set forth in claim 1 wherein the step of adding documents comprises the steps of:

15 interactively determining whether a document in the subset should be added to the resource list; and

adding the document only if it has been determined that the document should be added.

20 3. The method set forth in claim 2 further comprising the steps of:

COHEN 6-1

using a document for which it has been determined that the document should not be added together with documents on the resource list to learn new selection information; and

5 associating the new selection information with the resource list.

4. The method set forth in claim 1 wherein the step of learning the selection information comprises the steps of:

10 learning a rule for which the documents on the resource list are positive examples;
translating the rule into a query; and
in the step of using the selection information, using the query to select the set of
15 documents.

5. The method set forth in any of claims 1 through 4 wherein:

the system in which the method is practiced has access to a plurality of searching means;
20 the step of learning the selection information learns a plurality of queries as required by the plurality of searching means; and

COHEN 6-1

the step of using the selection information to select a set of documents uses the plurality of queries in the plurality of searching means.

5 6. The method set forth in claim 5 wherein:
the system in which the method is practiced has access to the world wide web; and
the searching means are searching means in the world wide web.

10 7. An improved web page of a type which contains a list of documents, the improvement comprising:
selection information associated with the web page which selects documents having content which is similar to the documents on the list, whereby the list
15 of documents on the web page may be updated using the selection information.

20 8. Apparatus for making a resource list of documents which have contents belonging to the same class,
the apparatus comprising:

COHEN 6-1

a first list of documents, all of which have contents belonging to the class;

a second list of documents, none of which have contents belonging to the class;

5 learning means responsive to the first list of documents and the second list of documents for learning selection information which specifies documents whose contents belong to the class;

10 means responsive to the selection information for finding the documents whose contents belong to the class, using the documents to make the resource list, and making a persistent association between the selection information and the resource list.

9. The apparatus set forth in claim 8
15 further comprising:

first interactive means for indicating whether a given document is to be added to the first list or the second list.

10. The apparatus set forth in claim 9
20 further comprising:

second interactive means for activating the learning means.

COHEN 6-1

11. The apparatus set forth in claim 10
further comprising:

third interactive means for activating the
means for finding the documents.

5 12. The apparatus set forth in any of claims
9 through 11 wherein:

the apparatus is used in a system which
includes a document browser; and

10 the interactive means of the claim are
implemented in the document browser.

13. In an information system which stores
related data and information as items for a plurality
of interconnected computers accessible by a plurality
of users, a method for finding items of a particular
15 class residing in the information system comprising the
steps of:

a) identifying as training data a plurality
of items characterized as positive and/or negative
examples of the class;

COHEN 6-1

b) using a learning technique to generate from the training data at least one that can be submitted to any of a plurality of methods for searching the information system;

5 c) submitting said query to at least one search method and collecting any new item(s) as a response to the query;

d) evaluating the new item(s) by a learned model with the aim of verifying that the new item(s) is
10 indeed a new subset of the particular class; and

e) presenting the new subset of the new item(s) to a user of the system.

14. The method of claim 13 wherein the information system is a distributed information system
15 (DIS) and the items are documents collected in resource directories in the DIS.

15. The method of claim 14 wherein step a) the positive examples are a set of documents in the resource directories and the negative examples are a

COHEN 6-1

selection of documents obtained by using the process of steps a-d.

16. The method of claim 15 wherein step b) the query is (i) a conjunction of terms which must
5 appear in a document as a positive example; (ii) contains all the terms appearing in the training data covered by the query, and (iii) learned by the system using a propositional rule-learning or prediction algorithm method.

10 17. The method of claim 16 wherein step d) a learning technique generates from the training data a learned model that computes a score for the new item(s), such that the new item(s) which has a low probability of being classified within the particular
15 class.

18. The method of claim 17 further comprising the step of providing a user on the system an ordered list of the new item(s) according to the score assigned by the learned model.

COHEN 6-1

19. The method of claim 17 further comprising the step of providing a user by electronic mail or facsimile an ordered list of the new item(s) having a score exceeding a threshold probability.

5 20. The method of claim 17 further comprising the step of using an batch process to identify documents as positive or negative examples of the search concept.

 21. The method of claim 17 further
10 comprising the step of using an interactive process to identify documents as positive examples of the search concept by browsing the distributed information system.

 22. The method of claim 17 further
15 comprising the step of resubmitting a query to the system to detect any new item added to the system and related to the query.

 23. An information system which stores
20 related data and information as items for a plurality of interconnected computers accessible by a plurality of users for finding items of a particular class

COHEN 6-1

residing in the information system using query learning and meta search, comprising:

- 5
- a) means for identifying as training data in the system a plurality of items characterized as positive and/or negative examples of the class;
- 10
- b) means for using a learning technique to generate from the training data at least one query that can be submitted to any of a plurality of search engines for searching the information system;
- 15
- c) means for submitting said query to at least one search engine and collecting any new item(s) as a response to the query;
- 20
- d) means for evaluating the new item(s) by the at least one search engine with the aim of verifying that the new item(s) is indeed a new subset of the particular class; and

COHEN 6-1

e) means for presenting the new subset of the new item(s) to a user of the system.

24. The system of claim 23 wherein the information system is a distributed information system (DIS) and the items are documents stored in resource
5 directories in the DIS.

25. The system of claim 24 wherein the positive examples are a set of items in the resource directories and the negative examples are a selection
10 of documents obtained by the search engine in responding to the query.

26. The system of claim 25 the query is (i) a conjunction of terms which must appear in a document as a positive example; (ii) contains all the terms
15 appearing in the training data covered by the query, and iii) learned by the system using a propositional rule-learning or prediction algorithm method.

27. The system of claim 26 wherein step the learning technique generates from the training data a
20 learned model that computes a score for the new

COHEN 6-1

item(s), such that the new item(s) which has a high probability of being classified within the particular class will be assigned a higher score than the new item(s) which has a low probability of being classified
5 within the particular class.

28. The system of claim 27 further comprising means for providing a user on the system an ordered list of the new item(s) according to the score assigned by the learned model.

10 29. The system of claim 27 further comprising means for providing a user by electronic mail or facsimile an ordered list of the new item(s) having a score exceeding a threshold probability.

15 30. The system of claim 27 further comprising means for using a batch process to select documents as positive examples of the search concept.

20 31. The system of claim 27 further comprising means for using an interactive process to identify documents as positive examples of the query by browsing the distributed information system.

COHEN 6-1

32. The system of claim 27 further comprising means for resubmitting a query to the system to detect any new item added to the system and related to the query.

5 33. An article of manufacture comprising:

a computer useable medium having computer readable program code means embodied therein for finding items of a particular class residing an information system which stored related data and information as items for a plurality of interconnected computers accessible by a plurality of users, the computer readable program code means in said article of manufacture comprising:

15 a) program code means for identifying as training data a plurality of items characterized as positive and/or negative examples of the class;

b) program code means for using a learning technique to generate from the training data at least one query that can be

20

COHEN 6-1

submitted to any of a plurality of methods
for searching the information system;

5 c) program code means for submitting
said query to at least one search method and
collecting any new item(s) as a response to
the query;

10 d) program code means for evaluating
the new item(s) by the at least one search
method with the aim of verifying that the new
item(s) is indeed a new subset of the
particular class; and

e) program code means for presenting
the new subset of new item(s) to a user of
the system.

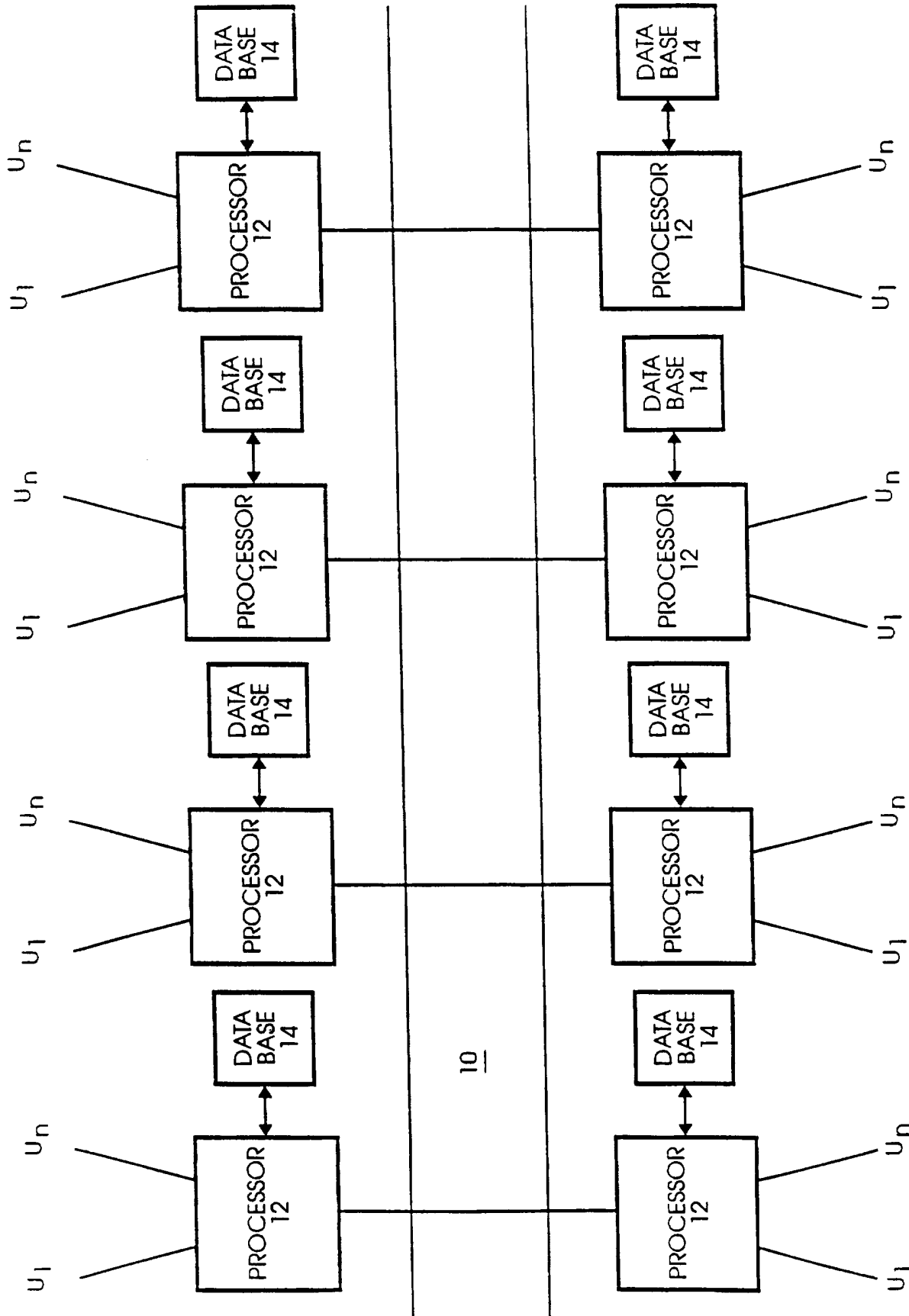


FIG. 1 PRIOR ART

2/7

```
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 U NegSample
  repeat
    RuleSet := Call Learn(Sample)
    for each r ∈ RuleSet do
      q := Call Corresponding Query(r.Search
      Engine)
      Response := Call Top-k-
      Documents(q.Search Engine)
      for each document d ∈ (Response -
      positive-URLs) do Sample := Sample
      U(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
```

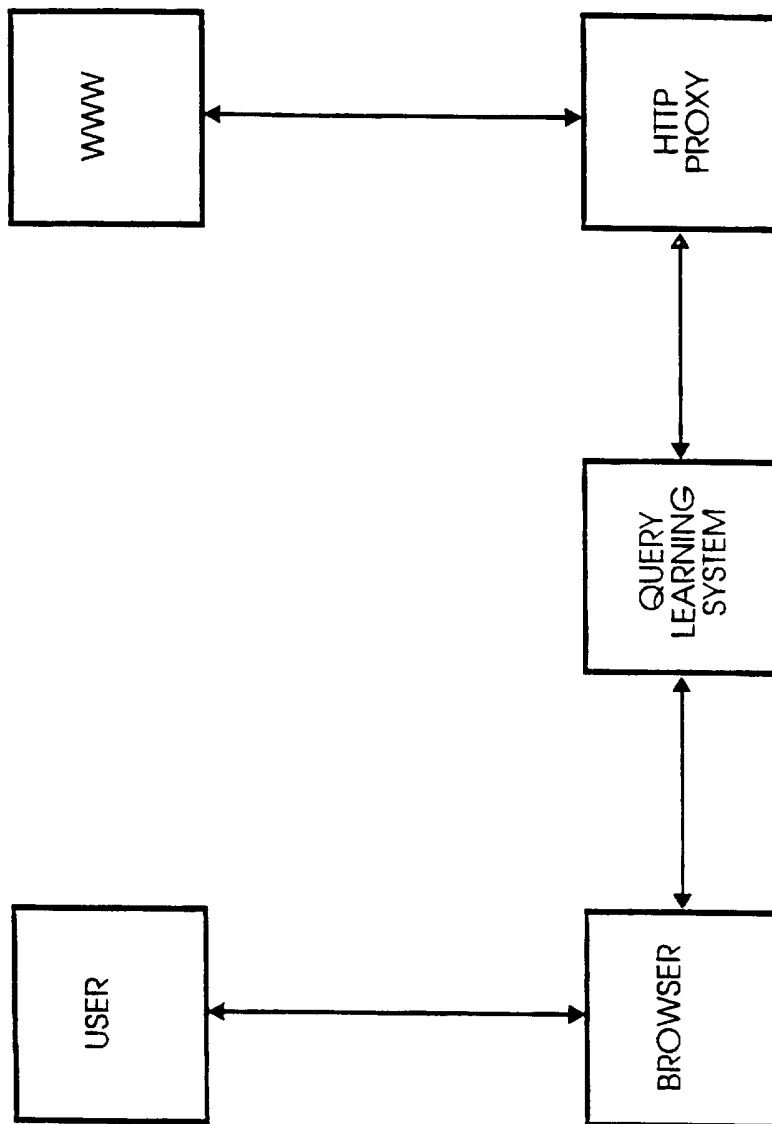


FIG. 3


File Edit View Go Bookmarks Options Directory Window Help

Back Forward Home Reload Local Images Open... Print... Find...

Location:

What's New What's Cool Handbook Net Search Net Directory Software

Swimsuit: Is this in the class 'AI-inst'? [Yes/No](#)
[Review Previous](#) [Learn / Search / Set Options / Help](#)



AT&T Software and Systems Research
 Bell Laboratories

Welcome to the Software and Systems Research Center External Web Server.

Under construction...

The AT&T Bell Laboratories Software and Systems Research Center conducts research in a variety of areas of Computer Science with the goal of creating technology and concepts that improve the capability of AT&T to develop and use software systems.

Research Areas

- [Artificial Intelligence](#)
- [Data and Information Management](#)
- [Formal Methods for Software Development](#)
- [Human-Computer Interface](#)
- [Programming Languages and Compilers](#)
- [Software Systems Architecture](#)

[F1001](#)

FIGURE 4

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

1. Receive a set of active experts W_t and their predictions $\{y_i^t \mid i \in W_t\}$.
2. Initialize: $\forall i \in W_t \wedge i \in Pool : p_i^t = 1$.
3. Define a distribution for the set of active experts:

$$\forall i \in W_t, \bar{p}_i^t = p_i^t \frac{\beta}{\sum_{j \in W_t} p_j^t}$$
4. Predict $F_3(\sum_{i \in W_t} \bar{p}_i^t y_i^t)$.
5. Compute loss values $U_i^t \in [0, 1] \mid i \in W_t$ based on y_i^t and true class.
6. Update weights: $p_i^{t+1} = p_i^t U_3(I^t)$.
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: $Pool = Pool \cup W_t$.

Figure 5. The Sleeping Experts Prediction Algorithm

Problem	Initial		Final Size	# Interactions		# Documents Labeled
	Size	Precision		Learn	Search	
ML Courses	11	0.24	46	5	4	82
AI Societies	8	0.16	51	5	5	81
Jogging Strollers	5	0.25	20	4	6	24

FIGURE 6 Summary of experiments with interactive system

Problem	#Iterations Allowed	# Documents Per Query (<i>k</i>)	# Documents Collected
ML Courses	10	30	199
AI Societies	20	30	126
Jogging Strollers	10	30	29

FIGURE 7 Summary of experiments conducted with batch system

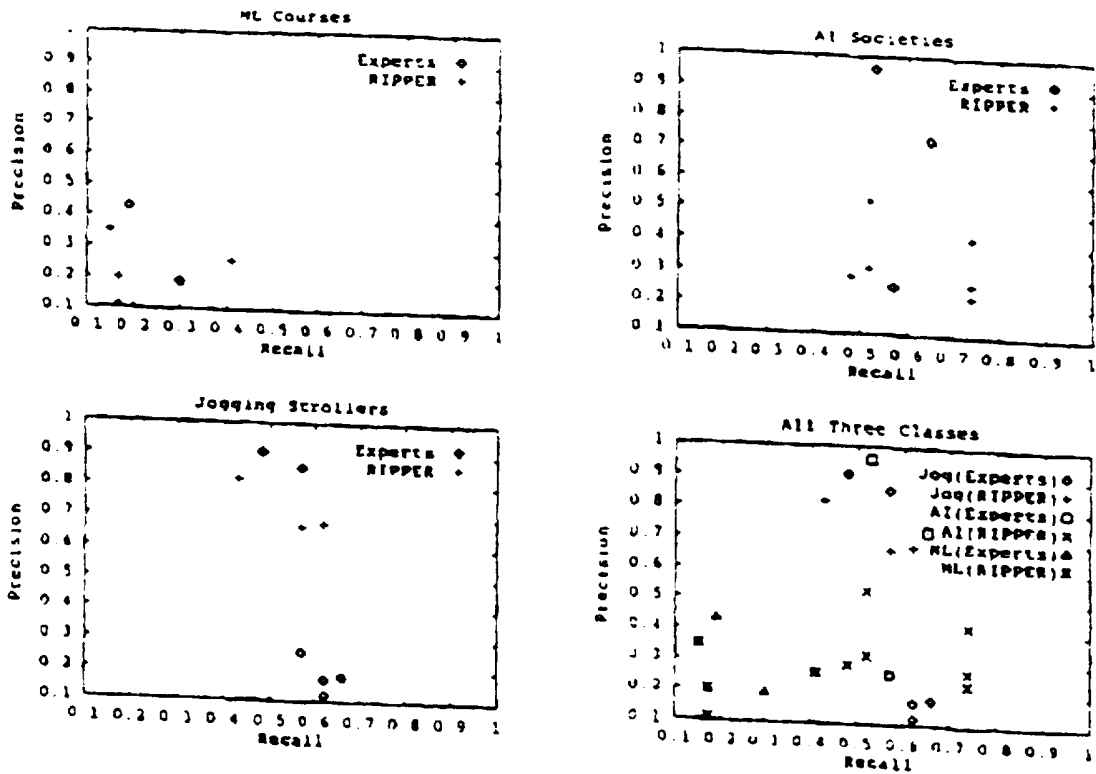


Figure 8 Precision-Recall tradeoffs for the three problems studied

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

Table 3: Results of the generalization error study

FIG. 9

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/05355

A. CLASSIFICATION OF SUBJECT MATTER IPC(6) :G06F 17/30; G06F 19/00 US CL :395/601, 610; 395/793 According to International Patent Classification (IPC) or to both national classification and IPC		
B. FIELDS SEARCHED Minimum documentation searched (classification system followed by classification symbols) U.S. : 395/601, 610; 395/793 Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched Electronic data base consulted during the international search (name of data base and, where practicable, search terms used) Please See Extra Sheet.		
C. DOCUMENTS CONSIDERED TO BE RELEVANT		
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
YP	US, 5,572,643, A (JUDSON) 05 November 1996, col. 2, lines 29-53.	1-33
YP	US, 5,530,852 A (MESKE Jr. ET AL) 25 June 1996, col. 2, lines 1-55	1-33
E	US 5,623,652 A (VORA ET AL) 22 April 1997, col. 3, lines 25-64.	1-33
Y	J. KUNZE, IS&T UC Berkely, February 1995, "Functional Recomendations for Internet Locators", page 9.	7-32
Y	K. SOLLINS, Xerox Corporation, December 1994, "Functional Requirements for Uniform Resource Names", pages 1-3	1-33
<input checked="" type="checkbox"/> Further documents are listed in the continuation of Box C. <input type="checkbox"/> See patent family annex.		
A	document defining the general state of the art which is not considered to be of particular relevance	*T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
E	earlier document published on or after the international filing date	*X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
L	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	*Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
O	document referring to an oral disclosure, use, exhibition or other means	*Z* document member of the same patent family
P	document published prior to the international filing date but later than the priority date claimed	
Date of the actual completion of the international search 20 JUNE 1997		Date of mailing of the international search report 01 AUG 1997
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231		Authorized officer <i>Thomas G. Black</i> THOMAS G. BLACK
Facsimile No. (703) 305-3230		Telephone No. (703) 305-4900

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/05355

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
Y	T. BERNERS-LEE, CERN, June 1994, "Universal Resource Identifiers in WWW", pages 2-10,	1-33
Y	R. FIELDING, UC Irving, June 1995, "Relative Uniform Resource Locators", pages 1-3	1-33
Y	M. HORTON ET AL, AT&T Bell Laboratories, December 1987, "Standard for Interchange of USENET Messages", pages 1-8.	19 & 29
Y	T. BERNERS ET AL, Xerox Corporation, December 1994, Uniform Resource Locators (URL), pages 1-25.	1-33
Y	DONALD H. JONES, IEEE Expert Magazine, December 1995, "A Model for Commerce on the World Wide Web" pages 54-59.	1-33

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US97/05355

B. FIELDS SEARCHED

Electronic data bases consulted (Name of data base and where practicable terms used):

APS, PROQUEST, IEEE, DIALOG

search terms: web browser, online search?, http, document? list?, internet?, html?,