Intelligent Web Agents that Learn to Retrieve and Extract Information

Tina Eliassi-Rad[1] and Jude Shavlik[2]

[1]Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Box 808, L-560, Livermore, CA 94551, USA.
Email: eliassi@llnl.gov.[†]

[2]Computer Sciences Department, University of Wisconsin-Madison, 1210 West Dayton Street, Madison, WI 53717, USA.
Email: shavlik@cs.wisc.edu.[‡]

Abstract. We describe systems that use machine learning methods to retrieve and/or extract *textual* information from the Web. In particular, we present our *Wisconsin Adaptive Web Assistant* (*WAWA*), which constructs a Web agent by accepting user preferences in form of instructions and adapting the agent's behavior as it encounters new information. Our approach enables WAWA to rapidly build instructable and self-adaptive Web agents for *both the information retrieval* (*IR*) and *information extraction* (*IE*) tasks. WAWA uses two neural networks, which provide adaptive capabilities for its agents. User-provided instructions are compiled into these neural networks and are modified via training examples. Users can create these training examples by rating pages that WAWA retrieves, but more importantly our system uses techniques from reinforcement learning to internally create its own examples. Users can also provide additional instruction throughout the life of an agent. Empirical results on several domains show the advantages of our approach.

Keywords. Instructable and adaptive software agents, Web mining, machine learning, neural networks, information retrieval, information extraction

1 Introduction

The rapid growth of information on the World Wide Web has boosted interest in using machine learning techniques to solve the problems of retrieving and extracting *textual* information from the Web [43,47]. The *information-retrieval*

[†] This work was done while the first author was at the Computer Sciences Department of the University of Wisconsin-Madison.

[‡] This research was supported in part by NLM Grant 1 R01 LM07050-01, NSF Grant IRI-9502990, and UW Vilas Trust.

(*IR*) learners attempt to model a user's preferences and return Web documents "matching" those interests. The *information-extraction* (*IE*) learners attempt to find patterns that fill a user-defined *template* (or questionnaire) with correct pieces of information.

We discuss several noted IR and IE learners in this chapter. Among the IR learners, many different machine learning techniques have been used ranging from a Bayesian classifier in *Syskill and Webert* [30] to our use of theory-refinement and reinforcement learning in *WAWA-IR* [10,11,40,41]. The breath of investigated approaches for IE learners basically falls into three categories: (*i*) systems that use hidden Markov models [2,15,20,31,38], (*ii*) systems that use relational learners [5,14,42], and (*iii*) systems that use theory-refinement techniques (such as our *WAWA-IE*) [10,12,13].

Our system, WAWA (short for *Wisconsin Adaptive Web Assistant*), interacts with its user and the Web to build an intelligent agent for retrieving and/or extracting information. It has two sub-systems: (*i*) an information retrieval sub-system, called *WAWA-IR*; and, (*ii*) an information extraction sub-system, called *WAWA-IE*. WAWA-IR is a general search-engine agent, which can be trained to produce specialized and personalized IR agents. WAWA-IE is a general extractor system, which creates specialized agents that extract pieces of information from documents in the domain of interest.

WAWA builds its agents based on ideas from the theory-refinement community within machine learning [28,29,45]. First, the user-provided domain knowledge is "compiled" into "knowledge based" neural networks [45]. Then, this prior knowledge is refined whenever training examples become available. By using theory refinement, we are able to find an appealing middle ground between non-adaptive agent programming languages and systems that solely learn user preferences from training examples. On one hand, utilizing user's prior knowledge enables WAWA's agents to perform reasonably well initially. On the other hand, since WAWA's agents are learners,[1] they do not rely on the user's prior knowledge to be correct.

This chapter is organized as follows. We present the fundamental operations of WAWA's agents in Section 2. WAWA's information-retrieval (IR) system along with other IR learners are discussed in Section 3. In Section 4, we present WAWA's information-extraction (IE) system and other notable IE systems. Section 5 describes some future directions. Finally, Section 6 summarizes the material in this chapter.

2 The Core of WAWA Agents

In this section, we briefly review the fundamental operations of a WAWA agent, which are used in both the WAWA-IR and the WAWA-IE agents [11].

[1] This learning ability makes WAWA's agents arguably "intelligent" since they can adapt their behavior due to both users' instructions and the feedback they get from their environments.

The knowledge base of a WAWA agent is centered around two basic functions: SCORELINK and SCOREPAGE (see Figure 1). If given highly accurate such functions, standard heuristic search would lead to effective retrieval of text documents: the best-scoring links would be traversed and the highest-scoring pages would be collected.

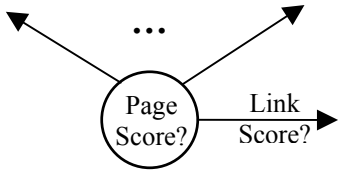

Fig. 1. Central Functions of WAWA's Agents Score Web Pages and Hyperlinks

Users are able to tailor an agent's behavior by providing advice about the above functions. This advice is "compiled" into two "knowledge based" neural networks [45] implementing the functions SCORELINK and SCOREPAGE. These functions, respectively, guide the agent's wandering within the Web and judge the value of the pages encountered. Subsequent reinforcements from the Web (*e.g.*, encountering dead links) and any ratings of retrieved pages that the user wishes to provide are, respectively, used to refine the link- and page-scoring functions.

A WAWA agent's SCOREPAGE network is a supervised learner [26]. That is, it learns through user-provided training examples and advice. A WAWA agent's SCORELINK network is a reinforcement learner [44]. This network automatically creates its own training examples [40,41], though it can also use any user-provided training examples and advice. Hence, our design of the SCORELINK network has the important advantage of producing self-tuning agents since training examples are created by the agent itself.

The user-provided instructions is mapped into the SCOREPAGE and SCORELINK networks using a Web-based language, called *advice*. An expression in our advice language is an instruction of the basic form:

`when` *`preconditions`* `then` *`actions`*

The *preconditions* refer to aspects of the contents and structure of Web pages. The *actions* specify the *goodness* of a page or a link when the preconditions are met.

WAWA extracts its input features from either HTML or plain-text Web pages. These input features[2] constitute the primitives in its advice language, which can be combined through logical and numerical operators to create more complicated advice constructs. Table 1 lists some of WAWA's extracted input features. The features *anywhereOnPage*(<*word*>) and *anywhereInTitle*(<*word*>) take a word as input and return true if the word was on the page or inside the title of the page, respectively. WAWA captures a large number of its features by sliding a fixed-

[2] See Eliassi-Rad [10] for a full description of WAWA's input features.

size[3] window across a page one word at a time. In particular, WAWA defines most of the features representing a page with respect to the current center of this sliding window, *e.g.* the *isNthWordInTitle*(<*N*>, <*word*>) feature is true when the given *word* is in the N^{th} word (from the left) on a page's title. Moreover, WAWA also has two bags of words of size *10* around the sliding window which allows it to capture instructions such as *when "Green Bay" is near "Packers" then show page*. Besides the input features related to words and their positions on the page, a WAWA agent's input vector also includes various other features, such as the length of the page, the date the page was created and modified (should the page's server provide that information), whether the sliding window is currently inside emphasized HTML text, the number of words in the title or URL, how many words mentioned in advice are present in the title or URL, etc.

Table 1. Sample Extracted Input Features

anywhereOnPage(<*word*>)
anywhereInTitle(<*word*>)
...
isNthWordInTitle(<*N*>, <*word*>)
...
centerWordInWindow(<*word*>)
...
numberOfWordsInTitle()
numberOfAdviceWordsInTitle()
...
insideEmphasizedText()
timePageWasLastModified()

A key feature of WAWA's advice language is its ability to capture abstract concepts (*e.g.*, names) through variables. To understand how variables are used in WAWA, assume that we wish to use the system to create a home-page finder. We might wish to give such a system some (very good) advice like: When the title of the page contains the phrase "*?FirstName ?LastName* 's Home Page", show me the page. The leading question marks (*?*) indicate variables that are bound upon receiving a request to find a specific person's home page. The use of variables allows the same advice to be applied to the task of finding the home pages of any number of different people.

Advice is compiled into the SCOREPAGE and SCORELINK networks using a variant of the KBANN algorithm [45]. The mapping process is analogous to compiling a traditional program into machine code, but our system instead compiles advice rules into an intermediate language expressed using neural networks. This provides the important advantage that our "machine code" can automatically be refined based on feedback provided by either the user or the

[3] Typically, the sliding window contains *15* words.

Web. Namely, we can apply the backpropagation algorithm [34] to learn from the training set.

We will illustrate the mapping of an advice rule with variables through an example, which also illustrates the powerful *phrase*[4] construct in WAWA's advice language. Suppose we are given the following advice rule: *When the phrase "Professor ?FirstName ?LastName" is on the page, show me the page*. During advice compilation, WAWA maps the phrase by centering it over the sliding window (Figure 2). In this example, our phrase is a sequence of three words, so it maps to three positions in the input units corresponding to the sliding window (with the variable *?FirstName* associated with the center of the sliding window).

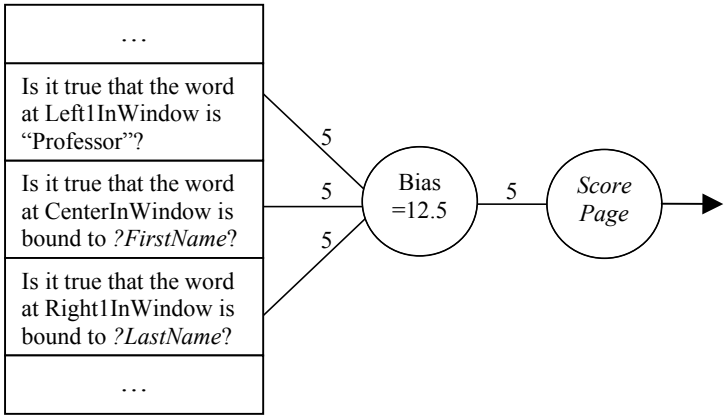

Fig. 2. Mapping Advice into SCOREPAGE Network

The variables in the input units are bound outside of the network and the units are turned on only when there is a match between the bindings and the words in the sliding window. So, if the bindings are:

?FirstName ← "Joe"
?LastName ← "Smith"

then, the input unit "*Is it true that the word at CenterInWindow is bound to ?FirstName*?" will be true (*i.e.*, set to 1) only if the current word in the center of the window is "Joe." WAWA then connects the referenced input units to a newly created (sigmoidal) hidden unit, using weights of value 5. Next, the bias (*i.e.*, the threshold) of the new hidden unit is set such that all the required predicates must be true in order for the weighted sum of its inputs to exceed the bias and produce an activation of the hidden unit near 1. Some additional zero-weighted links are also added to this new hidden unit, to further allow subsequent learning, as is standard in KBANN. Finally, WAWA links the hidden unit into the output unit with a weight determined by the rule's action. The mapping of advice rules without variables follows the same process except that there is no variable-binding step.

[4] A phrase is a sequence of consecutive words.

3 Retrieving Information from the Web

Information retrieval (IR) systems take as input a set of documents (*a.k.a.* the corpus) and a query (usually consisting of a bunch of keywords or keyphrases). The ultimate goal of an IR system is to return *all* and *only* the documents that are relevant to the given query. To achieve this goal, IR learners attempt to model a user's preferences and return Web documents "matching" those interests.

This section describes our design for creating specialized/personalized intelligent agents for retrieving information from the Web, an experimental study of our system, and some other recently developed IR learners.

3.1 WAWA-IR

WAWA's IR system is a general search-engine agent that through training can be specialized/personalized. Table 2 provides a high-level description of WAWA-IR.

Table 2. WAWA's Information Retrieval Algorithm

Unless they have been saved to disk in a previous session, create the *ScoreLink* and *ScorePage* neural networks by reading the user's initial advice (if any).

Either (a) start by adding user-provided URLs to the search queue; or (b) initialize the search queue with URLs that will query the user's chosen set of Web search engine sites.

Execute the following concurrent processes.

Independent Process #1
While the search queue is not empty nor the maximum number of URLs visited,

- Let *URLtoVisit* = pop(search queue).
- Fetch *URLtoVisit*.
- Evaluate *URLtoVisit* using the *ScorePage* network.
- If score is high enough, insert *URLtoVisit* into the sorted list of best pages found.
- Use the score of *URLtoVisit* to improve the predictions of the *ScoreLink* network.
- Evaluate the hyperlinks in *URLtoVisit* using *ScoreLink* network (but, only score those links that have not yet been followed this session).
- Insert these new URLs into the (sorted) search queue if they fit within its max-length bound.

Independent Process #2
Whenever the user provides additional advice, insert it into the appropriate neural network.

Independent Process #3
Whenever the user rates a fetched page, utilize this rating to create a training example for the *ScorePage* neural network.

The basic operation of WAWA-IR is heuristic search, with our SCORELINK network acting as the heuristic function. Rather than solely finding one goal node, we collect the 100 pages that SCOREPAGE rates highest. The user can choose to seed the queue of pages to fetch in two ways. She can either specify a set of starting URLs or provide a simple query that WAWA-IR converts into "query" URLs. These "query" URLs are then sent to a user-chosen subset of selectable search engine sites (currently ALTAVISTA, EXCITE, GOOGLE, HOTBOT, INFOSEEK, LYCOS, TEOMA, and YAHOO).

There are three ways to train WAWA-IR's two neural networks: (*i*) system-generated training examples, (*ii*) advice from the user, and (*iii*) user-generated training examples.

Before fetching a page *P*, WAWA-IR predicts the value of retrieving *P*. This "predicted" value of *P* is based on the text surrounding the hyperlink to *P* and some global information on the "referring" page (*e.g.*, the title, the URL, etc). After fetching and analyzing the actual text of *P*, WAWA-IR re-estimates the value of *P*. Any differences between the "before" and "after" estimates of *P*'s score constitute an error that can be used by backpropagation [34] to improve the SCORELINK neural network.[5]

In addition to the above system-internal method of automatically creating training examples, the user can improve the SCOREPAGE and SCORELINK neural networks in two ways. One, the user can provide additional advice. Observing the agent's behavior is likely to invoke thoughts of good additional instructions. A WAWA-IR agent can accept new advice and augment its neural networks at any time. It simply adds to a network additional hidden units that represent the compiled advice, a technique whose effectiveness was demonstrated on several tasks [25]. Providing additional hints can rapidly and drastically improve the performance of a WAWA-IR agent, provided the advice is relevant. Maclin and Shavlik [25] showed that their algorithm is robust when given advice incrementally. When "bad" advice was given, the agent was able to quickly learn to ignore it.

Although more tedious, the user can also rate pages as a mechanism for providing training examples for use by backpropagation. This can be useful when the user is unable to articulate why the agent is misscoring pages and links. This standard learning-from-labeled-examples methodology has been previously investigated by other researchers, *e.g.*, Pazzani *et al.* [30]. However, we conjecture that most of the improvement to WAWA-IR's neural networks, especially to SCOREPAGE, will result from users providing advice. In our personal experience, it is easy to think of simple advice that would require a large number of labeled examples in order to learn purely inductively. In other words, one advice rule typically covers a large number of labeled examples. For example, a rule such as *when* (*"404 file not found"*) *then avoid showing page* will cover all pages that contain the phrase "404 file not found".

[5] This type of training is *not* performed on the pages that constitute the initial search queue.

3.2 WAWA-IR: An Experimental Study

To evaluate WAWA's IR system, we built a home-page finder agent by using WAWA's advice language. We chose the task of building a home-page finder because of an existing system named AHOY! [39], which provides a valuable benchmark. Ahoy! uses a technique called Dynamic Reference Sifting, which filters the output of several Web indices and generates new guesses for URLs when no promising candidates are found.

For our home-page finder, we wrote a simple interface layered on top of WAWA-IR. Then, by using our advice language's *variables*, we wrote 76 general advice rules related to home-page finding, many of which are slight variants of others (e.g., with and without middle names or initials).[6] Specializing WAWA-IR for this task and creating the initial general advice took only one day, plus we spent parts of another 2-3 days tinkering with the advice using 100 examples of a "training set" that we describe below. This step allowed us to manually refine our advice – a process, which we expect will be typical of future users of WAWA-IR.

To run experiments that evaluate WAWA-IR, we randomly selected 215 people from Aha's list of machine learning (ML) and case-based reasoning (CBR) researchers (*www.aic.nrl.navy.mil/~aha/people.html*).[7] Table 3 lists the best performance of WAWA-IR's home-page finder and the results from AHOY!, and two different HOTBOT versions. The first HOTBOT version performs the engine's specialized search for people; we use the name given on Aha's page for these queries. In the second HOTBOT version, we provide the search engine with a general-purpose disjunctive query, which contains the person's last name as a *required* word, and all the likely variants of the person's first name. The latter is the same query that WAWA-IR initially sends to five of its search engines (namely, ALTAVISTA, EXCITE, INFOSEEK, LYCOS, and YAHOO). For our experiments, we only look at the first 100 pages that HOTBOT returns and assume that few people would look further into the results returned by a search engine. Besides reporting the percentage of the 100 test set home-pages found, we report the average ordinal position (*i.e.*, rank) *given that a page is found*, since WAWA-IR, AHOY!, and HOTBOT all return sorted lists.

Table 3. Empirical Results: WAWA-IR *vs* AHOY! and HOTBOT

System	% Found	Mean Rank Given Page Was Found
WAWA-IR with 76 advice rules	92%	1.3
AHOY!	79%	1.4
HOTBOT person search	66%	12.0
HOTBOT general search	44%	15.4

These results provide strong evidence that the version of WAWA-IR, specialized into a home-page finder by adding simple advice, produces a better home-page

[6] The complete list of these advice rules appears in Eliassi-Rad [10].
[7] See Eliassi-Rad and Shavlik [10] for full description of our methodology and other results.

finder than does the proprietary people-finder created by HOTBOT or by AHOY!. The difference (in percentage of home-pages found) between WAWA-IR and HOTBOT in this experiment is statistically significant at the 99% confidence level. The difference between WAWA-IR and AHOY! is statistically significant at the 90% confidence level. These results illustrate that we can build an effective agent for a web-based task quickly.

The above experiments were performed in 1998, at which time GOOGLE did not exist publicly. To compare our IR system with GOOGLE, we ran new experiments with our home page finder in 2001. Table 4 compares the best performances of WAWA-IR's home page finder seeded with and without GOOGLE to the results from GOOGLE (run by itself). The WAWA-IR experiment seeded without GOOGLE uses the following search engines: AltaVista, Excite, InfoSeek, Lycos, and Teoma. For these experiments, we trained the WAWA-IR agent with reinforcement learning, supervised learning, and all 76 home-page finding advice rules.

Table 4. Empirical Results: Two Different WAWA-IR Agents *vs* GOOGLE

System	% Found	Mean Rank ± Variance Given Page Was Found
WAWA-IR with GOOGLE	96%	1.12 ± 0.15
GOOGLE	95%	2.01 ± 16.64
WAWA-IR without GOOGLE	91%	1.14 ± 0.15

WAWA-IR seeded with GOOGLE is able to slightly improve on GOOGLE's performance by finding 96 of the 100 pages in the test set. WAWA-IR seeded without GOOGLE is not able to find more home pages than GOOGLE because the aggregate of the five search engines used is not as accurate as GOOGLE. In particular, GOOGLE appears to be quite good at finding home pages due to its *PageRank* scoring function, which globally ranks a Web page based on its location in the Web's graph structure and not on the page's content [4].

WAWA-IR experiments seeded with and without GOOGLE have the advantage of having a lower mean rank and variance than GOOGLE. We attribute this difference to WAWA-IR's learning ability, which is able to bump home pages to the top of the list. Finally, this set of experiments show how WAWA-IR can be used to personalize search engines by reorganizing the results they return as well as searching for nearby pages that score high.

3.3 Other IR Learners

Like WAWA, *Syskill and Webert* [30], and *WebWatcher* [18] are Web agents that use machine learning techniques. They, respectively, use a Bayesian classifier and a reinforcement learning -- TFIDF hybrid to learn about interesting Web pages and hyperlinks. Unlike WAWA, these systems are unable to accept (and refine) advice, which usually is simple to provide and can lead to better learning than rating or manually visiting many Web pages.

Drummond *et al.* [9] have created a system which assists users browsing software libraries. Their system learns unobtrusively by observing users' actions.

Letizia [21] is a system similar to Drummond *et al*.'s that uses lookahead search from the current location in the user's Web browser. Compared to WAWA, Drummond's system and Letizia are at a disadvantage since they cannot take advantage of advice given by the user.

WebFoot [42] is a system similar to WAWA, which uses HTML page-layout information to divide a Web page into segments of text. WAWA uses these segments to create an expressive advice language and extract input features for its neural networks. *WebFoot*, on the other hand, utilizes these segments to extract information from Web pages. Also, unlike WAWA, WebFoot only learns via supervised learning.

CORA [24] is a domain-specific search engine on computer science research papers. Like WAWA, it uses reinforcement-learning techniques to efficiently spider the Web [32]. CORA's reinforcement learner is trained off-line on a set of documents and hyperlinks which enables its *Q*-function to be learned via dynamic programming since both the reward function and the state transition function are known. WAWA's training, on the other hand, is done on-line. WAWA uses temporal-difference methods to evaluate the reward of following a hyperlink. In addition, WAWA's reinforcement-learner automatically generates its own training examples and is able to accept and refine user's advice. CORA's reinforcement-learner is unable to perform either of these two actions. To classify text, CORA uses naive Bayes in combination with the EM algorithm [8], and the statistical technique "shrinkage" [22,23]. Again, unlike WAWA, CORA's text classifier learns only through training examples and cannot accept and refine advice.

4 Extracting Information from the Web

Information extraction (IE) is the process of pulling desired pieces of information out of a document, such as the author of an article. Unfortunately, building an IE learners requires either a large number of annotated examples[8] or an expert to provide sufficient (and correct) knowledge about the domain of interest. Both of these requirements make it time-consuming and difficult to build an IE system. Similar to the IR case, we use WAWA's theory-refinement mechanism to build an IE system, namely *WAWA-IE*. By using theory refinement, we are able to strike a balance between needing a large number of labeled examples and having a complete (and correct) set of domain knowledge.

This section describes WAWA-IE, experimental results on WAWA-IE, and some other recently developed IE learners.

4.1 WAWA-IE

WAWA-IE takes advantage of the intuition that IR and IE are nearly inverse problems of each other. We illustrate this intuition with an example. Assume we have access to an *effective* home-page finder, which takes as input a person's

[8] By annotated examples, we mean the result of the tedious process of reading the training documents and tagging each extraction by hand.

name and returns her home page. The inverse of such an IR system is an IE system that takes in home pages and returns the names of the people to whom the pages belong. By using a *generate-and-test* approach to information extraction, we are able to utilize what is essentially an IR system to address the IE task. In the *generate* step, the user first specifies the slots to be filled (along with their part-of-speech tags or parse structures), and WAWA-IE generates a large *list of candidate extractions* from the document. Each entry in this list of candidate extractions is one complete set of slot fillers for the user-defined extraction template. In the *test* step, WAWA-IE scores each possible entry in the list of candidate extractions. The candidates that produce scores that are greater than a system-defined threshold are returned as the extracted information. A critical component of WAWA-IE is an intelligent selector that eliminates the need to create an exhaustive list of all possible candidate bindings.

The first step WAWA-IE takes (both during training and after) is to generate all possible fillers for each *individual* slot for a given document. Fillers can be individual words or phrases. Individual words are collected by using Brill's tagger [3], which annotates each word in a document with its part-of-speech tag. For each slot, we collect every word in the document that has a POS tag that matches a tag assigned to this variable somewhere in the IE task's advice. For cases where a variable is associated with a phrase, we apply a sentence analyzer called Sundance [33] to each document. We then collect those phrases that match the specified parse structure for the extraction slot and also generate all possible subphrases of consecutive words (since Sundance only performs a crude shallow parsing).

At this point, we typically have lengthy lists of candidate fillers for each slot, and we need to focus on generating good *combinations* that fill *all* the slots. Obviously, this process can be combinatorially demanding. To reduce this computational complexity, WAWA-IE contains several methods (called *selectors*) for creating complete assignments to the slots from the lists of individual slot bindings. WAWA-IE's selectors range from suboptimal and cheap (like simple random sampling from each individual list) to optimal and expensive (like exhaustively producing all possible combinations of the individual slot fillers). Among its heuristically inclined selectors, WAWA-IE has a modified WalkSAT algorithm [37], a modified GSAT algorithm [37], a hill-climbing algorithm with random restarts [35], and a statistically-oriented selector [10].

In our modified WalkSAT algorithm, we build the list of combination-slots candidate extractions for a document by randomly selecting an item from each extraction slot's list of individual-slot candidates. This produces a combination-slots candidate extraction that contains a candidate filler for each slot in the template. If the score produced by the SCOREPAGE network is high enough (*i.e.*, over 9 on a -10 to 10 scale) for this set of variable bindings, then we add this combination to the list of combination-slots candidates. Otherwise, we repeatedly and randomly select a slot in the template. Then, with probability p, we randomly select a candidate for the selected slot and add the resulting combination-slots candidate to the list of combination-slots candidates. With probability 1-p, we iterate over *all* possible candidates for this slot and add the candidate that

produces the highest network score for the document to the list of combination-slots candidates.[9]

To build a WAWA-IE agent, the user provides the following information:

1. The set of on-line documents from which the information is to be extracted.
2. The extraction slots like speaker names, etc.
3. The possible part-of-speech (POS) tags (*e.g.*, noun, verb, etc) or the parse structures (*e.g.*, noun phrase, verb phrase, etc) for each extraction slot.
4. A set of advice rules containing variables which refer to the extraction slots.[10]
5. A set of annotated examples, *i.e.*, training documents in which extraction slots have been marked by hand.

In one of our case studies, we want to extract names of proteins and their subcellular locations from the yeast database of Ray and Craven [31]. One of our advice rules for this task is: *When the phrase "?ProteinName/Nphrase •/Vphrase ?LocationName/Nphrase" appears in the document, then score it very high.* The variables *?ProteinName* and *?LocationName* represent the protein names and their subcellular structures. The "/Nphrase" trailing the variables indicates the required parse structure of the variables ("Nphrase" refers to a noun phrase). The "•/Vphrase" matches any verb phrase. The precondition of this rule matches phrases such as "UBC6 localizes to the endoplasmic reticulum."

Figure 3a shows the process of building a trained IE agent. Since (usually) only positive training examples are provided in IE domains, we first need to generate some negative training examples. To this end, we run the training documents through the candidate generator and selector described above. In this step, the heuristic used in the candidate selector scores each possible training extraction on the *untrained* SCOREPAGE network. By untrained, we mean a network containing only compiled (initial) advice and without any further training via backpropagation and labeled examples. The effect of using the untrained SCOREPAGE network is that the generated list contains informative negative examples (*i.e.*, near misses). This is due to the fact that the user-provided prior knowledge rates these "near miss" training extractions highly (as if they were true extractions).

After the negative examples are collected, we train the SCOREPAGE neural network using these negative examples and all the provided positive examples. By training the network to recognize (i.e., produce a high output score for) a correct extraction in the context of the document as a whole [41], we are able to take advantage of the global layout of the information available in the documents of interest.

Figure 3b depicts the steps a trained IE agent takes to produce extractions. For each entry in the list of combination-slots extraction candidates, we first bind the variables to their candidate values. Then, we perform a forward propagation on the trained SCOREPAGE network and output the score of the network for the test document based on the candidate bindings. If the output value of the network is

[9] See Eliassi-Rad [10] for complete details on all the selectors.

[10] Actually, the user does not have to explicitly provide the extraction slots and their POS tags separately from advice since they can be extracted from the advice rules.

greater than the system-defined threshold,[11] we record the bindings as an extraction. Otherwise, these bindings are discarded.

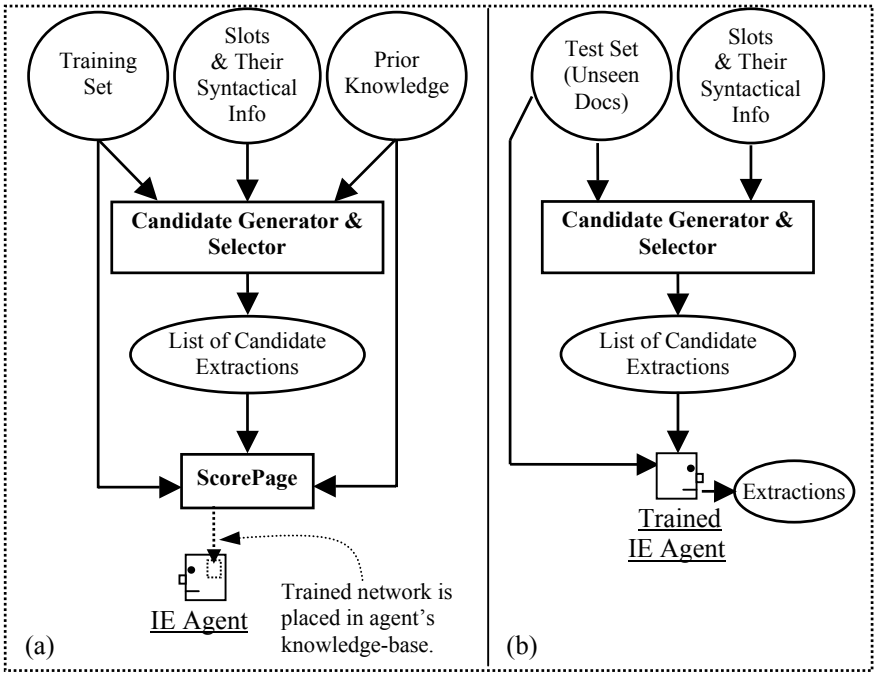

Fig. 3. (a) Building a Trained IE agent (b) Testing a Trained IE Agent

4.2 WAWA-IE: An Experimental Study

We evaluate WAWA-IE on a task involving extraction of protein names and their locations within the cell from the yeast protein-localization domain produced by Ray and Craven [31]. [12] This domain is a collection of abstracts from biomedical articles on yeast. In this study, the fillers for extraction slots depend on each other because a single abstract can contain multiple proteins and locations. Hence, for each document, a single list of <*protein*, *location*> pairs is extracted. We followed Ray and Craven's methodology for our experiments on this domain and used their "tuple-level" method for measuring accuracy (where a tuple is an instance of a protein and its location).

WAWA-IE is given 12 advice rules in BNF [1] notation about a protein and its subcellular location. None of the advice rules are written with the specifics of the yeast data set in mind. It took us about half a day to write these rules and we did not *manually* refine these rules over time.

For our information measures, we use *precision*, *recall*, and the F_1*-measure.* P*recision* (P) is the ratio of the number of correct fillers extracted to the total number of fillers extracted, and *recall* (R) is the ratio of the number of correct

[11] During training, WAWA-IE computes a *task-specific threshold* on the output of the SCOREPAGE network by analyzing results on some "tuning" examples.
[12] See Eliassi-Rad [10] for experimental details on two other IE domains.

fillers extracted to the total number of fillers in correct extraction slots [46]. An ideal system has a precision and recall of 100%. The commonly used F_1-*measure* combines precision and recall using the following formula:

$$F_1 = (2 \times Precision \times Recall) / (Precision + Recall)$$

The F_1-measure is more versatile than either precision or recall for explaining relative performance of different systems, since it takes into account the inherent tradeoff that exists between precision and recall.

Ray and Craven [31] split the yeast data set into five disjoint sets and ran 5-fold cross-validation for their experiments. We use the same folds with WAWA-IE and compare our results to theirs. Figure 4 illustrates the difference in the *test-set* F_1-measure between our modified WalkSAT selector and the exhaustive candidate selector (where *all* possible negative exampls are used). The horizontal axis depicts the percentage of negative training examples used during the learning process, and the vertical axis depicts the F_1-measure of the trained IE-agent on the test set. WAWA-IE is able to achieve very good performance by using less than 20% of the negative training candidates, which demonstrates that we can intelligently select good training examples (and, hence, reduce training time).

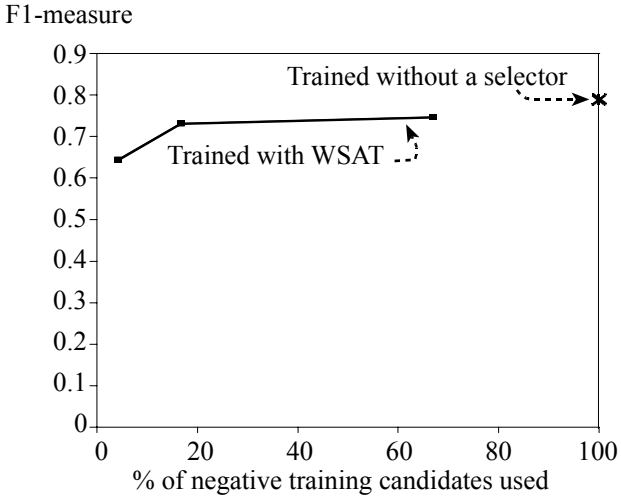

Fig. 4. F_1-measure on the Test Set vs. Percentage of Negative Training Candidates Used for Different Selector Algorithms

In F_1-measures, WAWA-IE's trained agents outperform the untrained agents by approximately 50% (results not shown). This further demonstrates that WAWA-IE is able to refine initial advice.

Figure 5 shows the precision and recall curves for (a)WAWA-IE's trained agent with the modified WalkSAT selector (using 17% of the negative examples), (b) WAWA-IE's trained agent without a selector (*i.e.*, using all the negative training examples), and (c) the system of Ray and Craven.

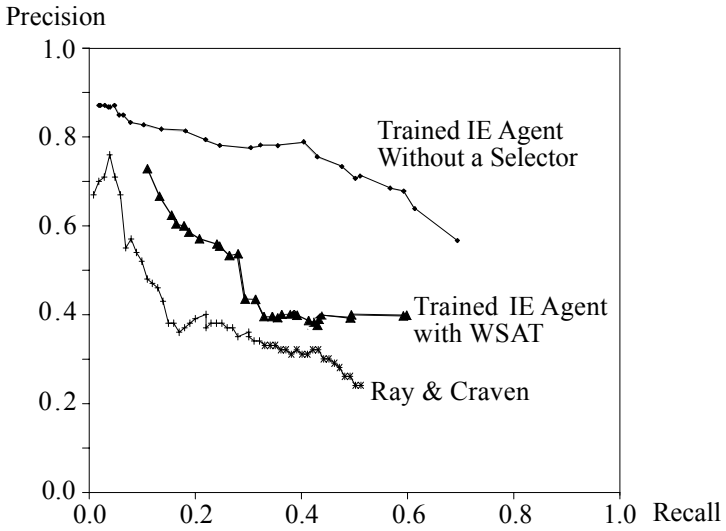

Fig. 5. Precision/Recall Curves

The trained IE agent without any selector algorithm produces the best results. But it is computationally very expensive, since it needs to take the cross-product of all entries in the lists of individual-slot candidates. The trained IE agent with modified WalkSAT selector performs quite well, still outperforming Ray and Craven's system.

Our results both illustrate the value of using theory refinement for IE and justify using an intelligent candidate-selection algorithm to reduce the computational burden of our "IE via IR" approach, which uses a computationally demanding generate-and-test strategy. WAWA-IE with the "modified WalkSAT" selector is able to improve on the state of the art using only 17% of the possible negative training candidates during training. Finally, recall that we also use our variant of WalkSAT during *testing*. Thus, Figure 5 also shows that we obtain good precision and recall without needing to exhaustively score every possible candidate.

4.3 Other IE Learners

We were able to find only one other system in the literature that applies theory refinement to the IE problem. Feldman *et al.*'s IE system [11] takes a set of *approximate* IE rules and uses training examples to incrementally revise the inaccuracies in the initial rules. Their revision algorithm uses heuristics to find the place and type of revision that should be performed. Unlike WAWA-IE's advice rules, their IE rules provide advice on how to refine existing rules. Also, their system manipulates IE rules directly, whereas WAWA-IE compiles rules into neural networks and uses standard neural training to refine the rules. Finally, their approach is to suggest possible revisions to the human user, whereas WAWA-IE's approach is to make the revisions automatically.

Most IE systems break down into two groups. The first group uses some kind of relational learning to learn extraction patterns [5,14,43]. The second group learns parameters of hidden Markov models (HMMs) and uses the HMMs to extract

information [2,15,20,31,38]. Recently, Freitag and Kushmerick [16] combined wrapper induction techniques [19] with the AdaBoost algorithm [36] to create an extraction system named *BWI* (short for Boosted Wrapper Induction). Their system out-performed many of the relational learners and was competitive with systems using HMMs and WAWA-IE.

Leek [20] uses HMMs for extracting information from biomedical text. His system uses a lot of initial knowledge to build the HMM model before using the training data to learn the parameters of HMM. However, his system is not able to refine the knowledge.

Several authors use statistical methods to reduce the need for a lot of training examples. Freitag and McCallum [15] use HMMs to extract information from text. They employ a statistical technique called "shrinkage" to get around the problem of not having sufficient labeled examples. Seymore *et al.* [38] also use HMMs to extract information from on-line text. They get around the problem of not having sufficient training data by using data that is labeled for another purpose in their system. Similarly, Craven and Kumlien [6] use "weakly" labeled training data to reduce the need for labeled training examples.

One advantage of our system is that we are able to utilize prior knowledge, which reduces the need for a large number of labeled training examples. However, we do not depend on the initial knowledge being 100% correct. We believe that it is relatively easy for users to articulate some useful domain-specific advice (especially when a user-friendly interface is provided that converts their advice into the specifics of WAWA's advice language). The second advantage of our system is that the entire content of the document is used to estimate the correctness of a candidate extraction. This allows us to learn about the extraction slots *and* the documents in which they appear. The third advantage of WAWA-IE is that we are able to utilize the untrained SCOREPAGE network to produce some informative negative training examples (*i.e.*, near misses).

5 Future Directions

In order to better understand what people would like to say to an instructable Web agent (such as WAWA) and improve our advice language accordingly, we need to build more personalized and easily customized intelligent Web agents.

We would like to embed WAWA into a major, existing Web browser, thereby minimizing new interface features that users must learn in order to interact with our system. Related to this, we would like to develop methods whereby WAWA can automatically infer plausible training examples by observing users' normal use of their browsers [17].

In our IE domains, we would like to incorporate the candidate generation and selection steps directly into our connectionist framework, whereby we would use the current SCOREPAGE network to find new candidate extractions during the training process. Finally, an interesting area of research would be to explore theory-refinement techniques on different supervised learning algorithms (such as support vector machines [7], HMMs, and relational learners).

6 Conclusion

We argue that a promising way to create useful intelligent agents is to involve both the user's ability to do direct programming (*i.e.*, provide approximately correct instructions of some sort), along with the agent's ability to accept and automatically create training examples. Due to the largely unstructured nature and the size of the Web, such a hybrid approach is more appealing than ones solely based on either non-adaptive agent programming languages or users that rate or mark the desired extractions from a large number of Web pages.

WAWA utilizes the user's knowledge about the task at hand to build agents that retrieve and extract information. Three important characteristics of WAWA's agents are (*i*) their ability to receive instructions and refine their knowledge-bases through learning (hence, the instructions provided by the user need not be perfectly correct), (*ii*) their ability to receive the user's advice continually, and (*iii*) their ability to create informative training examples.

We first present and evaluate WAWA's information-retrieval system, which provides an appealing approach for creating personalized information-finding agents for the Web. A central aspect of our design is that a machine learner is at the core. Users create specialized agents by articulating their interests in our advice language. WAWA-IR compiles these instructions into neural networks, thereby allowing for subsequent refinement. The system both creates its own training examples (via reinforcement learning) and allows for supervised training should the user wish to rate the information a WAWA-IR agent finds. This process of continuous learning makes WAWA-IR agents (self) adaptive. Our "home-page finder" case study demonstrates that we can build an effective agent for a web-based task quickly.

We also describe and evaluate a system for using theory refinement to perform information extraction. WAWA's information-extraction system uses a neural network, which accepts advice containing variables, to rate candidate variable bindings in the content of the document as a whole. Our extraction process first generates a large set of candidate variable bindings for each slot, then selects a subset of the possible slot bindings via heuristic search, and finally uses the trained network to judge which are "best." Those bindings that score higher than a system-computed threshold are returned as the extracted information. By using theory refinement, we are able to take advantage of prior knowledge in the domain of interest and produce some informative training examples, both of which lead to an increase in the performance of the IE agent. Our experiments on the Yeast protein-localization domain illustrates that we are able to compete with state-of-the-systems. Also, we empirically show the benefits of using intelligent algorithms for selecting possible candidates for multiple slots.

We also briefly reviewed other approaches to the IR and IE tasks that are based on machine learning techniques. These systems, including ours, demonstrate the promise of using machine learning to make sense of the vast resourses that is the World-Wide Web.

References

1. Aho A., Sethi R., Ullman, J. (1986). *Compilers, Principles, Techniques and Tools*, Addison Wesley.
2. Bikel D., Schwartz R., Weischedel R. (1999). An Algorithm That Learns What's in a Name, *Machine Learning: Special Issue on Natural Language Learning*, **34**, 211–231.
3. Brill E. (1994). Some advances in rule-based part of speech tagging, *Proc. of AAAI-94 Conference*, 722–727.
4. Brin S., Page L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, **30**, 107-117.
5. Califf M.E. (1998). *Relational Learning Techniques for Natural Language Information Extraction*. Ph.D. Thesis, Department of Computer Sciences, University of Texas, Austin, TX.
6. Craven M., Kumlien J. (1999). Constructing biological knowledge-bases by extracting information from text sources, *Proc. of ISMB-99*, 77–86.
7. Cristianini N., Shawe-Taylor J. (2000). *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge University Press.
8. Dempster A., Laird N., Rubin D. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm, *Journal of the Royal Statistical Society*, **39**, 1–38.
9. Drummond C., Ionescu D., Holte R. (1995). A learning agent that assists the browsing of software libraries, Technical Report TR-95-12, University of Ottawa, Ottawa, Canada.
10. Eliassi-Rad T., (2001). *Building Intelligent Agents that Learn to Retrieve and Extract Information*, Ph.D. Thesis, Computer Sciences Department. University of Wisconsin, Madison, WI.
11. Eliassi-Rad T., Shavlik J. (2001). A system for building intelligent agents that learn to retrieve and extract information, Appears in the *International Journal on User Modeling and User-Adapted Interaction, Special Issue on User Modeling and Intelligent Agents.*
12. Eliassi-Rad T., Shavlik J. (2001). A theory-refinement approach to information extraction. *Proc. of ICML-01 Conference*, 130–137.
13. Feldman R., Liberzon Y., Rosenfeld B., Schler J., Stoppi J. (2000). A framework for specifying explicit bias for revision of approximate information extraction rules. *Proc. Of KDD-00 Conference*, 189–197.
14. Freitag D. (1998). *Machine Learning for Information Extraction in Informal Domains*, Ph.D. thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

15. Freitag D., McCallum A. (1999). Information extraction with HMMs and shrinkage, *Workshop Notes of AAAI-99 Conference on Machine Learning for Information Extraction*, 31–36.
16. Freitag D., Kushmerick N. (2000). Boosted wrapper induction, *Proc. AAAI-00 Conference*, 577–583.
17. Goecks J., Shavlik J. (2000). Learning users' interests by unobtrusively observing their normal behavior, *Proc. of IUI-2000*, 129–132.
18. Joachims T., Freitag D., Mitchell T. (1997). WebWatcher: A tour guide for the World Wide Web, *Proc. of IJCAI-97 Conference*, 770–775.
19. Kushmerick N. (2000). Wrapper Induction: Efficiency and expressiveness, *Artificial Intelligence*, **118**, 15–68.
20. Leek T., (1997). *Information Extraction Using Hidden Markov Models*, Masters Thesis, Department of Computer Science & Engineering, University of California, San Diego.
21. Lieberman H. (1995). Letzia: An agent that assists Web browsing, *Proc. of IJCAI-95 Conference*, 924–929.
22. McCallum A., Rosenfeld R., Mitchell T. (1998). Improving text classification by shrinkage in a hierarchy of classes, *Proc. of ICML-98 Conference*, 359–367.
23. McCallum A., Nigam K. (1998). A comparison of event models for naive Bayes text classification, *Workshop Notes of AAAI-98 Conference on Learning for Text Categorization*, 41–48.
24. McCallum A., Nigam K., Rennie J., Seymore K. (1999c). Building domain-specific search engines with machine learning techniques, *AAAI-99 Spring Symposium*, Stanford University, CA, 28–39.
25. Maclin R., Shavlik, J. (1996). Creating Advice-Taking Reinforcement Learners, *Machine Learning*, **22**, 251–281.
26. Mitchell T. (1997). *Machine Learning*, McGraw-Hill.
27. National Library of Medicine (2001). *The MEDLINE Database*, http://www.ncbi.nlm.nih.gov/PubMed/.
28. Ourston D., Mooney R. (1994). Theory Refinement: Combining Analytical and Empirical Methods. *Artificial Intelligence*, **66**, 273–309.
29. Pazzani M., Kibler D. (1992). The Utility of Knowledge in Inductive Learning. *Machine Learning*, **9**, 57–94.
30. Pazzani M., Muramatsu J., Billsus D., (1996). Syskill & Webert: Identifying interesting Web sites. *Proc. of AAAI-96 Conference*, 54–61.
31. Ray S., Craven M. (2001). Representing sentence structure in hidden Markov models for information extraction, *Proc. of IJCAI-01 Conference*.
32. Rennie J., McCallum A. (1999). Using reinforcement learning to spider the Web efficiently, *Proc. of ICML-99 Conference*.

33. Riloff E. (1998). *The Sundance Sentence Analyzer*, http://www.cs.utah.edu/projects/nlp/.
34. Rumelhart D., Hinton G., Williams R. (1986). Learning internal representations by error propagation. In: D. Rumelhart and J. McClelland (eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. 1. MIT Press, 318–363.
35. Russell S., Norvig P. (1995). *Artificial Intelligence*: *A Modern Approach*, Prentice Hall.
36. Schapire R., Singer Y. (1998). Improved boosting algorithms using confidence-rated predictions, *Proc. COLT-98 Conference*.
37. Selman B., Kautz H., Cohen B. (1996). Local Search Strategies for Satisfiability Testing. *DIMACS Series in Discrete Mathematics and Theoretical CS*, **26**, 521–531.
38. Seymore K., McCallum A., Rosenfeld R. (1999). Learning hidden Markov model structure for information extraction *Workshop Notes of AAAI-99 Conference on Machine Learning for Information Extraction*, 37–42.
39. Shakes J., Langheinrich M., Etzioni O. (1997). Dynamic reference sifting: A case stury in the homepage domain, *Proc. of WWW-97 Conference*, 189–200.
40. Shavlik J., Eliassi-Rad T. (1998). Intelligent agents for web-based tasks: An advice-taking approach, *Workshop Notes of AAAI-98 Conference on Learning for Text Categorization*, Madison, WI, 63–70.
41. Shavlik J., Calcari S., Eliassi-Rad T., Solock J. (1999). An instructable, adaptive interface for discovering and monitoring information on the World-Wide Web, *Proc. of IUI-99 Conference*, 157–160.
42. Soderland S. (1997). Learning to extract text-based information from the World Wide Web, *Proc. of KDD-97 Conference*, 251–254.
43. Soderland S. (1999). Learning Information Extraction Rules for Semi-Structured and Free Text, *Machine Learning: Special Issue on Natural Language Learning*, **34**, 233–272.
44. Sutton R.S., Barto A.G. (1998). *Reinforcement Learning*, MIT Press.
45. Towell G.G., Shavlik J.W. (1994). Knowledge-Based Artificial Neural Networks. *Artificial Intelligence*, **70**, 119–165.
46. van Rijsbergen C.J. (1979). *Information Retrieval*, Buttersworths. 2nd edition.
47. Yang Y. (1999). An Evaluation of Statistical Approaches to Text Categorization, *Journal of Information Retrieval*, **1**, 67–88.