To appear in the *International Journal on User Modeling and User-Adapted Interaction, Special Issue on User Modeling and Intelligent Agents.*

A System for Building Intelligent Agents that Learn to Retrieve and Extract Information

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

Jude Shavlik
Computer Sciences Department, University of Wisconsin-Madison, 1210 West Dayton Street, Madison, WI 53706 USA. Email: shavlik@cs.wisc.edu

May 2, 2002

Abstract.

We present a system for rapidly and easily building instructable and self-adaptive software agents that retrieve and extract information. Our *Wisconsin Adaptive Web Assistant* (WAWA) constructs intelligent agents by accepting user preferences in the form of instructions. These user-provided instructions are compiled into neural networks that are responsible for the adaptive capabilities of an intelligent agent. The agent's neural networks are modified via user-provided and system-constructed training examples. Users can create training examples by rating Web pages (or documents), but more importantly WAWA's agents uses techniques from reinforcement learning to internally create their own examples. Users can also provide additional instruction throughout the life of an agent. Our experimental evaluations on a "home-page finder" agent and a "seminar-announcement extractor" agent illustrate the value of using instructable and adaptive agents for retrieving and extracting information.

Keywords: instructable and adaptive software agents, Web mining, machine learning, neural networks, information retrieval, information extraction

1. Introduction

The popularity of the World Wide Web has created a surge of interest in tools that are able to retrieve and extract information from online documents. In a perfect world, you would be able to get on the Internet and instantaneously retrieve precisely the information you want (whether it is a whole document or fragments of it). What is the next best option? Consider having an assistant, which rapidly and easily builds instructable and self-adaptive software agents for both the information retrieval (IR) and the information extraction (IE) tasks. These intelligent software agents would process your interests (with respect to the information you would like to receive) and automatically

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

refine its model of your preferences over time. Their mission would be to spend 24 hours a day looking for documents of interest to you and answering specific questions that you might have. Our goal is to build such an assistant.

We call our assistant WAWA (short for *Wisconsin Adaptive Web Assistant*). WAWA interacts with the user and an on-line (textual) environment (*e.g.*, the Web) to build an intelligent agent for retrieving and/or extracting information. Figure 1 illustrates an overview of WAWA. WAWA 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/or 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.

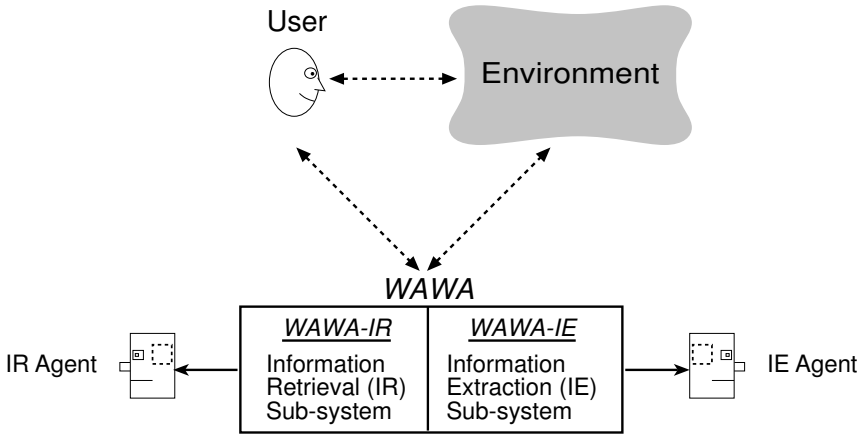

Figure 1. An Overview of WAWA

WAWA builds its agents based on ideas from the theory-refinement community within machine learning (Pazzani and Kibler, 1992; Ourston and Mooney, 1994; Towell and Shavlik, 1994). Users specify their prior knowledge about the desired task. This knowledge is then "compiled" into "knowledge based" neural networks (Towell and Shavlik, 1994), thereby allowing subsequent refinement whenever training examples are available. The advantages of using a theory-refinement approach to build intelligent agents are as follows:

- WAWA's agents are able to perform reasonably well initially because they are able to utilize users' prior knowledge.
- Users' prior knowledge does not have to be correct since it is refined through learning.

– The use of prior knowledge, plus the continual dialog between the user and an agent, decreases the need for a large number of training examples because human-machine communication is not limited to a binary representation of positive and negative examples.

– WAWA provides an appealing middle ground between non-adaptive agent programming languages and systems that solely learn user preferences from training examples.

WAWA's agents are intelligent because they can adapt their behavior according to the users' instructions and the feedback they get from their environments. In other words, they are learning agents that use neural networks to store and modify their knowledge. Figure 2 illustrates the interaction between the user, an intelligent (WAWA) agent, and the agent's environment. The user[1] observes the agent's behavior (*e.g.*, the quality of the pages retrieved) and provides helpful instructions to the agent. Following Maclin and Shavlik (1996), we refer to users' instructions as *advice*, since this name emphasizes that the agent does not blindly follow the user-provided instructions, but instead refines the advice based on its experiences. The user inputs his/her advice into a user-friendly *advice interface*. The given advice is then processed and mapped into the agent's knowledge base (*i.e.*, its two neural networks), where it gets refined based on the agent's experiences. Hence, the agent is able to represent the user model in its neural networks, which have representations for which effective learning algorithms are known (Mitchell, 1997).

This article is organized as follows. We present WAWA's fundamental operations in Section 2. WAWA's information-retrieval (IR) system is discussed in Section 3. Section 4 describes a case study on WAWA's information-retrieval system, namely the rapid creation of an effective "home-page finder" agent. In Section 5, we discuss WAWA's information-extraction (IE) system. Section 6 presents a case study on WAWA's information-extraction system, namely, the "seminar announcements extractor" agent. Related work and future directions are discussed in Sections 7 and 8, respectively. Section 9 summarizes the material in this paper.

[1] We envision that there are two types of potential users of our system: (1) *application developers*, who build an intelligent agent on top of WAWA and (2) *application users*, who use the resulting agent. (When we use the phrase *user* in this article, we mean the former.) Both types of users can provide advice to the underlying neural networks, but we envision that usually the application users will indirectly do this through some specialized interface that the application developers create. A scenario like this is discussed in Section 4.

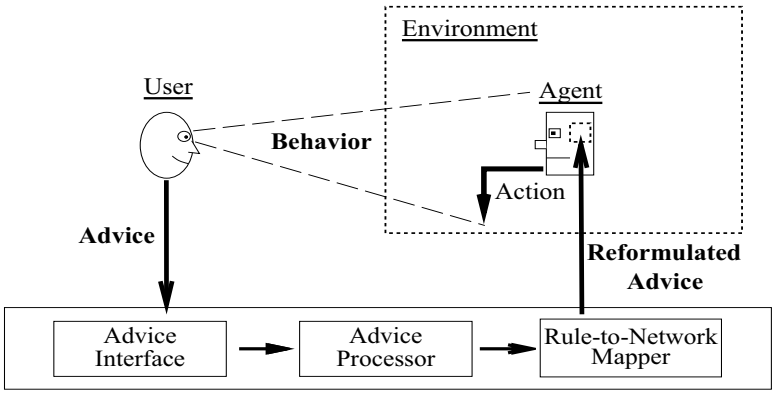

Figure 2. The Interaction between a User, an Intelligent Agent, and the Agent's Environment

2. WAWA's Core

This section presents WAWA's fundamental operations, which are used in both the IR and the IE subsystems of WAWA (see Sections 3 and 5 for further details). These operations include handling of WAWA's advice language, representing Web pages[2] internally for an agent's use, and scoring arbitrary long pages with neural networks. Figure 3 illustrates how an agent uses these operations to score a page. The *page processor* gets a page from the environment (*e.g.*, the Web) and produces an internal representation of the page. This new representation of the page is then given to the agent's knowledge-base (*i.e.*, the agent's neural network), which produces a score for the page by doing forward-propagation (Rumelhart et al., 1986). Finally, the agent's neural network incorporates the user's advice and the environment's feedback, both of which effect the score of a page.

The knowledge base of a WAWA agent is centered around two basic functions: SCORELINK and SCOREPAGE (see Fig. 4). 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.

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 (Towell and Shavlik, 1994) implementing the functions SCORELINK and SCOREPAGE (see Figure 5). 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 re-

[2] In this article, we use the terms "page" and "document" interchangeably.

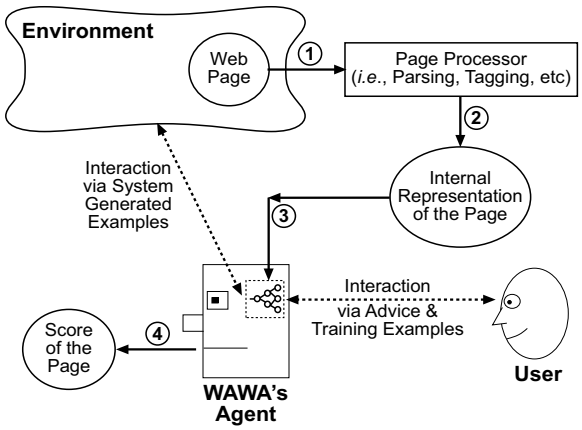

Figure 3. Scoring a Page with a WAWA Agent

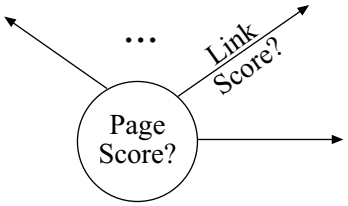

Figure 4. Central Functions of WAWA's Agents Score Web Pages and Hyperlinks

trieved pages that the user wishes to provide are, respectively, used to refine the link- and page-scoring functions.

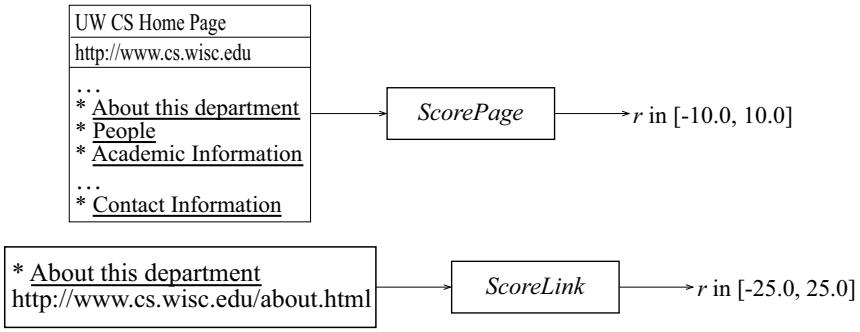

Figure 5. WAWA's Central Functions Score Web Pages and Hyperlinks Respectively

A WAWA agent's SCOREPAGE network is a supervised learner (Mitchell, 1997). That is, it learns through user-provided training examples and advice. A WAWA agent's SCORELINK network is a reinforcement learner (Sutton and Barto, 1998). This network automatically creates its own training examples, 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 (see Section 3.1.2 for details).

2.1. Wawa's Advice Language

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 following basic form:

`when` *precondition* `then` *action*

The preconditions represent aspects of the contents and structure of Web pages. Table I lists the actions of our advice language in *Backus-Naur Form* (BNF) (Aho et al., 1986) notation. The `strength` levels in actions represent the degree to which the user wants to increase or decrease the score of a page or a link.

Table I. Permissible Actions in an Advice Statement

actions	→ *strength* show page
	| *strength* avoid showing page
	| *strength* follow link
	| *strength* avoid following link
	| *strength* show page & follow link
	| *strength* avoid showing page & following link
strength	→ weakly | moderately | strongly | definitely

Wawa extracts features from either HTML or plain-text Web pages. These input features constitute the primitives in our advice language. These primitive constructs can be combined to create more complicated constructs. This section presents Wawa's feature-extraction method and its advice language.

2.1.1. *Extracting Features from Web Pages.*

A standard representation of text used in IR is the *bag-of-words* representation (Salton, 1991). In the bag-of-words representation, word order is lost and all that is used is a vector that records the words present on the page, usually scaled according to the number of occurrences and other properties. The top-right part of Figure 6 illustrates this representation.

Generally, IR systems (Belew, 2000) reduce the dimensionality (*i.e.*, number of possible features) in the problem by discarding common

("stop") words and "stem" all words to their root form (*e.g.*, "walked" becomes "walk"). WAWA typically performs these two preprocessing steps.

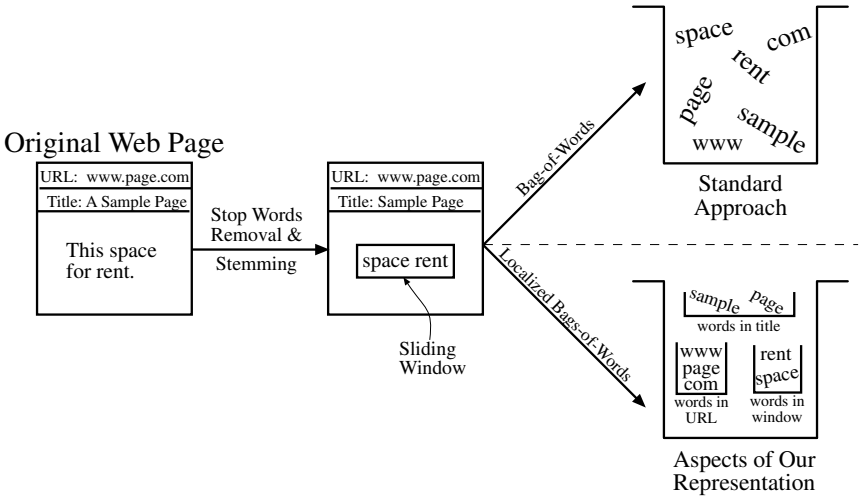

Figure 6. Internally Representing Web Pages

We believe the information provided by word order is important. For example, without word-order information, we cannot express instructions such as *when the phrase "Green Bay" is on the page then show page*. Given that we are using neural networks to score pages and links, one approach to capturing word-order information would be to use recurrent networks, but instead we borrow an idea from NETTALK (Sejnowski and Rosenberg, 1987), though our basic unit is a word rather than an (alphabetic) letter as in NETTALK. Namely, WAWA "reads" a page by sliding a fixed-size[3] window across a page one word at a time. Figure 7 provides an example of a 3-word sliding window going across a page.

Most of the features we use to represent a page are defined with respect to the current center of the sliding window. The sliding window itself allows us to capture word order on a page; however, we also have two bags of words of size 10 around the sliding window which allow us to capture instructions such as *when "Green Bay" is near Packers then show page*.

In addition to preserving some word-order information, we also take advantage of the structure of HTML documents (when a fetched page is so formatted). First, we augment the bag-of-words model, by using several localized bags, some of which are illustrated on the bottom-

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

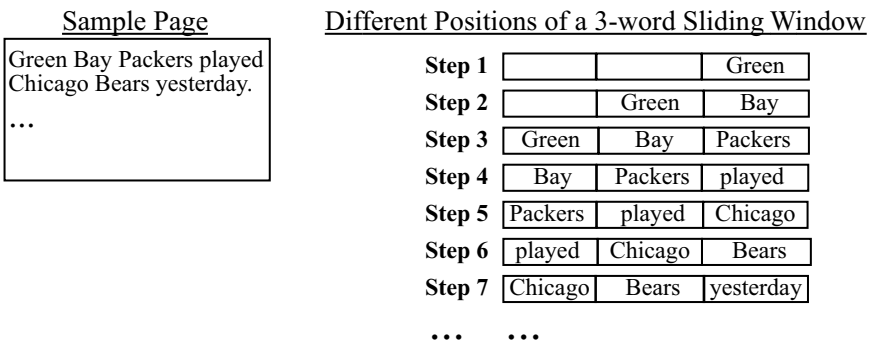

Figure 7. Using a 3-Word Sliding Window to Capture Word-Order Information on a Page

right part of Figure 6. Besides a bag for all the words on the page, we have word bags for: the title, the page's URL, the sliding window, the left and right sides of the sliding window, the current hyperlink[4] (should the window be inside hypertext), and the current section's title. WAWA's parser of Web pages records the "parent" section title of each word; parent's of words are indicated by the standard ⟨H1⟩ through ⟨H6⟩ section-header constructs of HTML, as well as other indicators such as table captions and table-column headings. Moreover, bags for the words in the grandparent and great-grandparent sections are kept, should the current window be nested that deeply. This knowledge of the "context" of words does not limit advice to only describing relations between nearby words.

WAWA uses Brill's tagger (Brill, 1994) to annotate each word on a page with a part-of-speech (POS) tag (*i.e.*, noun, proper noun, verb, etc). This information is represented in the agent's neural networks as input features for the words in the sliding window. By adding POS tags, we are able to distinguish between different grammatical uses of a word. For example, this allows a user to give advice of the form *when "fly" and "bug" are on the page and they are both nouns then show the page*. This rule will not match pages that contain the words "fly" and "bug" as verbs. We also take advantage of the inherent hierarchy in the POS tags (*e.g.*, a proper noun is also a noun, or a present participle verb is also a verb). For example, if the user indicates interest in the word "fly" as a noun, we look for the presence of "fly" as a noun and as a proper noun. However, if the user indicates interest in the word

[4] A Web page has its own URL, while there are also URLs within the page's contents. We refer to the former as *url* and the later cases as *hyperlinks*, in an attempt to reduce confusion.

"Bill" as a proper noun, then we *only* look for the presence of "Bill" as a proper noun and not as a noun.

Table II lists some of WAWA's extracted input features.[5] The features *anywhereOnPage*($\langle word \rangle$) and *anywhereInTitle*($\langle word \rangle$) take a word as input and return true if the word was on the page or inside the title of the page, respectively. These two features act as word bags for the page and the title.

In addition to the feature representing bag-of-words and word order, we also represent several fixed positions. Besides the obvious case of the positions in the sliding window, we represent the first and last N words (for some fixed N) in the title, the URL, the section titles, etc. Due to its important role in the Web, we also specially represent the last N fields (i.e., delimited by dots) in the *server* portion of URLs and hyperlinks, e.g. `www wisc edu` in `http://www.wisc.edu/news/Welcome/`.

Table II. Sample Extracted Input Features

`anywhereOnPage(`$\langle word \rangle$`)`
`anywhereInTitle(`$\langle word \rangle$`)`
...
`isNthWordInTitle(`$\langle N \rangle, \langle word \rangle$`)`
...
`isNthWordFromENDofTitle(`$\langle N \rangle, \langle word \rangle$`)`
...
`NthFromENDofURLhostname(`$\langle N \rangle, \langle word \rangle$`)`
...
`leftNwordInWindow(`$\langle N \rangle, \langle word \rangle$`)`
`centerWordInWindow(`$\langle word \rangle$`)`
...
`numberOfWordsInTitle()`
`numberOfAdviceWordsInTitle()`
...
`insideEmphasizedText()`
`timePageWasLastModified()`
...
`POSatRightSpotInWindow(`$\langle N \rangle, \langle POS\ tag \rangle$`)`
`POSatCenterOfWindow(`$\langle POS\ tag \rangle$`)`
`POSatLeftSpotInWindow(`$\langle N \rangle, \langle POS\ tag \rangle$`)`

Besides the input features related to words and their positions on the page, a WAWA agent's input vector also includes various other features,

[5] See Eliassi-Rad (2001) for a full description of WAWA's input features.

such as the length of the page, the date the page was created/modified (should the page's server provide that information), whether the window is inside emphasized HTML text, the sizes of the various word bags, how many words mentioned in advice are present in the various bags, etc.

Part-of-speech tags for the words in the sliding window are represented by the last three features in Table II. For example, the advice *when* (*consecutive*("*yellow fly*") *and POSatCenterOfWindow*(*noun*)) *then show page* expresses our interest in pages that include the phrase "yellow fly" with the word "fly" as a noun. The features *POSatRightSpotInWindow* and *POSatLeftSpotInWindow* specify the desired POS tag for the Nth position to the right or left of the center of the sliding window, respectively.

We use many boolean-valued features[6] to represent a Web page, ranging from *anywhereOnPage(aardvark)* to *anywhereOnPage(zebra)* to *rightNwordInWindow(3, AAAI)* to *NthFromENDofURLhostname(1, edu)*.

Our design leads to a large number of input features which allows us to have an expressive advice language. However, assuming a typical vocabulary of tens of thousands of words, the number of features is on the order of a million! One might ask how a learning system can hope to do well in such a large space of input features. Dealing with this many input features would indeed be infeasible if a WAWA agent solely learned from labeled examples (Valiant, 1984). Fortunately, our use of advice means that users indirectly select a subset of this huge set of implicit input features. Namely, they indirectly select only those features that involve the words appearing in their advice. The full set of input features is still there, but the weights out of input features used in advice have high values, while all other weights (*i.e.*, absent words) have values near zero. Thus, there is the potential for words not mentioned in advice to impact a network's output, after lots of training.

We also deal with the enormous input space by only explicitly representing what *is* on a page. That is, all zero-valued features, such as *anywhereOnPage(aardvark)* = *false*, are only *implicitly* represented. Fortunately, the nature of weighted sums in both the forward and backward propagation phases of neural networks (Rumelhart et al., 1986) means that zero-valued nodes have no impact and, hence, can be ignored.

[6] The current version of WAWA does not use any TFIDF methods (Salton and Buckley, 1988), due to the manner in which we compile advice into networks (see Section 2.2).

2.1.2. *Complex Advice Constructs and Predicates*

All features extracted from a page or a link constitute the basic constructs and predicates of the advice language.[7] These basic constructs and predicates can be combined via boolean operators (*i.e.*, `AND, OR, NOT`) to create complex predicates. *Phrases* (Croft et al., 1991), which specify desired properties of consecutive words, play a central role in creating more complex predicates out of the primitive features we extract from Web pages. Table III contains some of the more complicated predicates that WAWA defines in terms of the basic input features. The advice rules in this table correspond to instructions a user might provide if he/she is interested in finding *Joe Smith*'s home-page.[8]

Table III. Sample Advice

(1) `WHEN consecutiveInTitle(`
`anyOf(`*Joseph Joe J.*`)`
Smith's home page`)`
`STRONGLY SUGGEST SHOWING PAGE`

(2) `WHEN hyperlinkEndsWith(`
`anyOf(`*Joseph Joe Smith jsmith*`) /`
`anyOf(`*Joseph Joe Smith jsmith*
index home homepage my me`)`
`anyOf(`*htm html /*`))`
`STRONGLY SUGGEST FOLLOWING LINK`

(3) `WHEN (titleStartsWith(`*Joseph Joe J.*`)`
`and titleEndsWith(`*Smith*`))`
`SUGGEST SHOWING PAGE`

(4) `WHEN NOT(anywhereOnPage(`*Smith*`))` `STRONGLY SUGGEST AVOID SHOWING PAGE`

Rule 1 indicates that when the system is sliding the window across the page's title, it should look for any of the plausible variants of *Joe Smith*'s first name, followed by his last name, followed by apostrophe *s*, and then the phrase "home page."

Rule 2 demonstrates another useful piece of advice for home-page finding. This one gets compiled into the *NthFromENDofHyperlink()* input features, which are true when the specified word is the *Nth* one

[7] A predicate is a function that returns either true or false. We define a construct as a function that returns numerical values.

[8] The *`anyOf`*`()` construct used in the table is satisfied when any of the listed words is present.

from the *end* of the current hyperlink. (Note that WAWA treats the '/' in URLs as a separate word.)

Rule 3 depicts our interest in pages that have titles starting with any of the plausible variants of *Joe Smith*'s first name and end with his last name.

Rule 4 shows that advice can also specify when *not* to follow a link or show a page; negations and AVOID instructions become negative weights in the neural networks.

2.1.3. *Advice Variables*

WAWA's advice language contains variables. These variables can range over various kinds of things, like names, places, etc. Advice variables are of particular relevance to WAWA's IE system (see Section 5 for details).

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.

2.2. COMPILATION OF ADVICE INTO NEURAL NETWORKS

Advice is compiled into the SCOREPAGE and SCORELINK networks using a variant of the KBANN algorithm (Towell and Shavlik, 1994). The mapping process (see Table IV) 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 Web. Namely, we can apply the backpropagation algorithm (Rumelhart et al., 1986) to learn from the training set.

We will illustrate the mapping of an advice rule with variables through an example. 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 8). 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).

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

Table IV. General Algorithm for Mapping Advice into Neural Networks

(1) Construct an AND-OR dependency graph based on the advice rules. (1.1) Each node in the AND-OR dependency graph becomes a unit in the neural network. (1.2) Insert additional units for OR nodes. (3) Set the biases of each AND unit and the weights coming into the AND unit such that the unit will get activated only when *all* of its inputs are true. (4) Set the biases of each OR unit and the weights coming into the OR unit such that the unit will get activated only when *at least one* of its inputs is true. (5) Add links with low weights between otherwise unconnected nodes in adjacent layers of the network to allow learning over the long run.

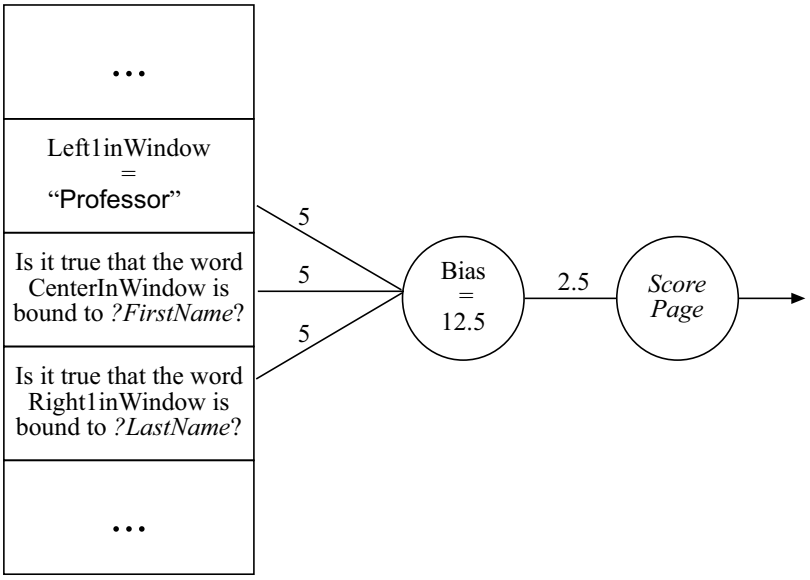

Figure 8. Mapping Advice into ScorePage Network

$$?FirstName \leftarrow \text{“Joe”}$$
$$?LastName \leftarrow \text{“Smith”}$$

The input unit “*Is it true that the word 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”. Similarly the input unit “*Is it true that the Right1inWindow is bound to ?LastName*?” will be set to 1 only if the current word immediately to the right of the center of the window is “Smith”.

WAWA then connects the referenced input units to a newly created hidden unit, using weights of value 5.[9] 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 sigmoidal 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 strength given in the rule's action. WAWA interprets the phrase "suggest showing page" as "moderately increase the page's score."

The mapping of advice rules without variables follows the same process except that there is no variable-binding step.

2.3. Scoring Arbitrary Long Pages with Fixed-Sized Neural Networks

WAWA's use of neural networks means that we need a mechanism for processing arbitrarily long Web pages with fixed-sized input vectors. Our sliding window assists us in this problem. Recall that the sliding window moves across a page one word at a time. There are, however, some HTML tags like ⟨P⟩, ⟨/P⟩, ⟨BR⟩, and ⟨HR⟩ that act as "window breakers." Window breakers do not allow the sliding window to cross over them. When a window breaker is encountered, the unused positions in the sliding window are left unfilled.

The score of a page is computed in two stages. In stage one, we set the input units that represent global features of the page, such as the number of words on the page. Then, we slide our window (hence, the name *sliding window*) across the page. For each window position, we first set the values for the input units representing positions in the window (*e.g.*, word at center of window) and then calculate the values for all hidden units (HUs) that are directly connected to input units. We call these HUs "level-one" HUs. In other words, we perform forward-propagation from the input units to all level-one HUs. This process gives us a list of values for all level-one HUs at each position of the sliding window. For each level-one HU, we pick the best (*i.e.* highest) value to represent its activation.

In stage two, the best values of level-one HUs and the values of input units for global features are used to compute the values for all other HUs and the output unit. That is, we perform forward-propagation

[9] The initial value of the weights and the bias (*i.e.*, "threshold") on an AND unit must satisfy the following equation: $bias = initial\_weight(\#unnegated\_antecedents - 0.5)$.

from the level-one HUs and the "global" input units to the output unit (which obviously will evaluate the values of all other HUs in the process). The value produced by the ScorePage network in the second stage is returned as the page's score.

Note that we scan the sliding window only once across the page, which occurs in stage one. By forward-propagating only to level-one HUs in the first stage, we are effectively trying to get the values of our complex features. In stage two, we use these values and the global input units' values to find the score of the page. For example, the two-stage process allows us to capture advice such as *when the phrase "Milwaukee Brewers" and the phrase "Chicago Cubs" are on the page then show page*. If we only had a one-stage process, we would not be able to correctly capture this advice rule because both phrases cannot be in the sliding window simultaneously. Figure 9 illustrates this point.

The value of a hyperlink is computed similarly, except that the ScoreLink network is used and the sliding window is slid over the *hypertext* associated with that hyperlink and the 15 words surrounding the hypertext on both sides.

3. Using Wawa to Retrieve Information

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 *only* the documents that are relevant to the given query.

This section describes our design for creating specialized/personalized intelligent agents for retrieving information from the Web.

3.1. IR System Description

Table V provides a high-level description of Wawa's IR system, *a.k.a.* Wawa-IR. Wawa-IR is a general search engine agent that through training can be specialized/personalized. First, its two neural networks need to be created using Section 2's techniques (or read from disk should this be a resumption of a previous session).

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: either by specifying a set of starting URLs or by providing a simple query that Wawa-IR converts into "query" URLs that are sent to a user-chosen subset of selectable search engine sites (currently AltaVista, Excite, InfoSeek, Lycos, and Yahoo).

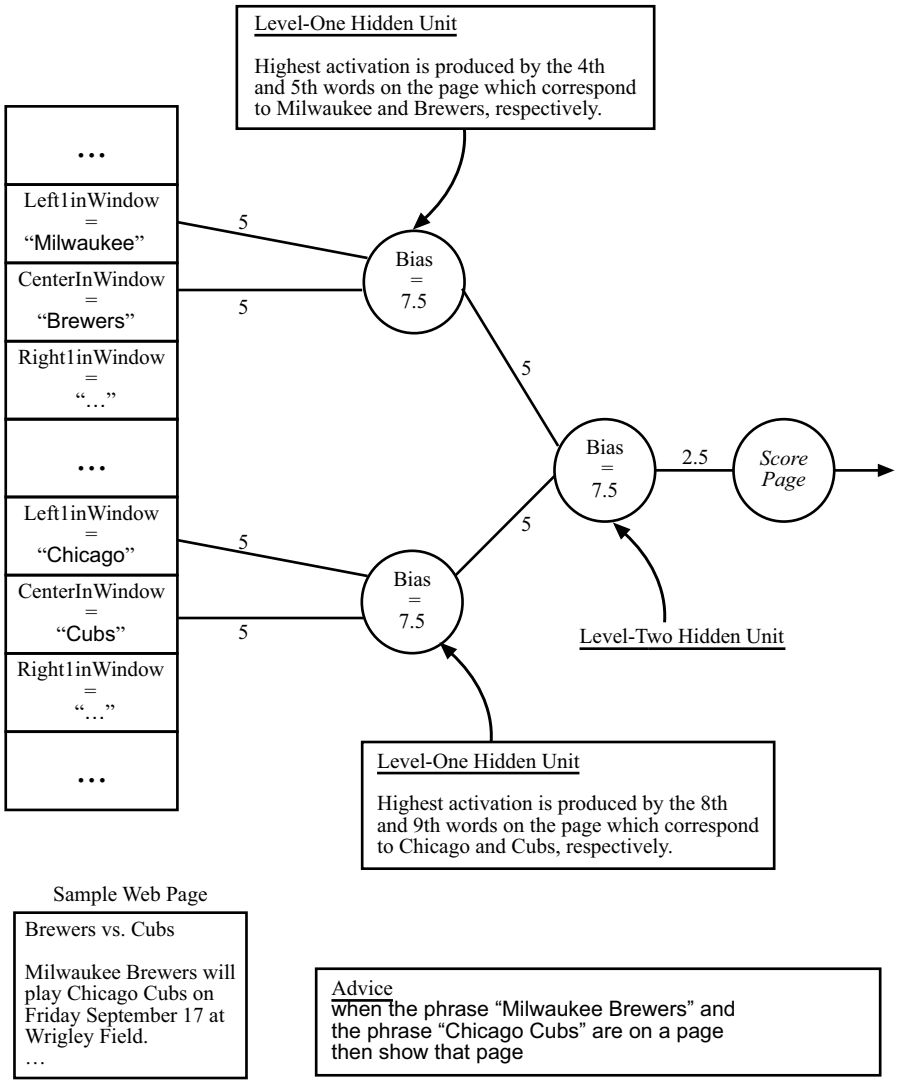

Figure 9. Scoring a Page with a Sliding Window. In stage one, the sliding window is scanned through the page to determine the highest values of the level-one hidden units. In stage two, the highest activations of the level-one hidden units are used to score the level-two hidden unit and the output unit, which produces the overall score of the page.

Although not mentioned in Table V, the *user* may also specify values for the following parameters:

– an upper bound on the distance the agent can wander from the initial URLs (default value is 10)

Table V. 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 *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
      (see Section 3.1.2 for details).

    Evaluate the hyperlinks in $URLtoVisit$
      using *ScoreLink* network (however, 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 person rates a fetched page, use this rating to
create a training example for the *ScorePage* neural network.

– minimum score a hyperlink must receive in order to be put in the search queue (default value is 0.6 on a scale of [0,1])

– maximum number of hyperlinks to add from one page (default value is 50)

– maximum kilobytes to read from each page (default value is 100 kilobytes)

– maximum retrieval time per page (default value is 90 seconds).

3.1.1. *Training* WAWA*'s Two Neural Networks for IR*

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 (Rumelhart et al., 1986) to improve the SCORELINK neural network.[10] The details of this process are further described in Section 3.1.2.

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 (as has repeatedly happened to us in our case studies). 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 (Maclin and Shavlik, 1996). Providing additional hints can rapidly and drastically improve the performance of a WAWA-IR agent, provided the advice is relevant. Maclin and Shavlik (1996) 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

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

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. (1996), and we discuss this aspect of WAWA-IR in Section 4. 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".

3.1.2. *Deriving Training Examples for* SCORELINK

We use *temporal difference* methods (Sutton, 1988) to automatically train SCORELINK. WAWA-IR employs a form of Q-learning (Watkins, 1989), which is a type of reinforcement learning (Sutton and Barto, 1998). Recall that the difference between WAWA-IR's prediction of the link's value before fetching a URL and its new estimate serves as an error that backpropagation tries to reduce. Whenever WAWA-IR has collected all the necessary information to re-estimate a link's value, it invokes backpropagation. In addition, it periodically reuses these training examples several times to refine the network. Notice that WAWA-IR automatically constructs these training examples without direct user intervention, as is typical in reinforcement learning.

As is also typical in reinforcement learning, the value of an action (following a link in this case) is *not* solely determined by the immediate result of the action (the value of the page retrieved minus any retrieval-time penalty). Rather, we wish to also reward links that *lead to* pages with additional good links on them. Figure 10 and Equation 1 illustrate this point.

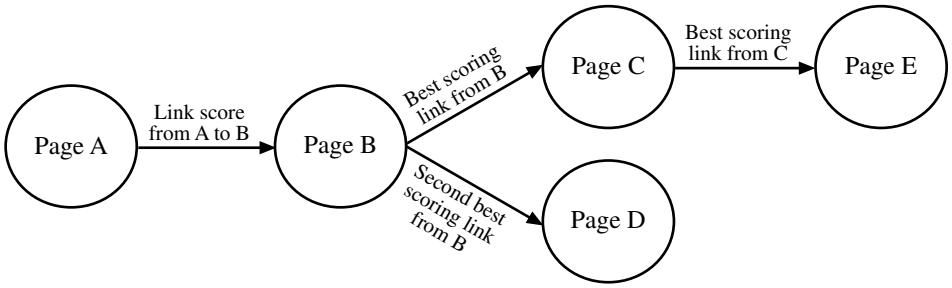

Figure 10. Reestimating the Value of a Link

Equation 1: New Estimate of the Link $A \rightarrow B$ under Best-First Search

```
if ScoreLink(B → C) > 0 then
    new estimate of ScoreLink(A → B)
      = fetchPenalty(B) + ScorePage(B)
        + γ(fetchPenalty(C) + ScorePage(C))
        + γ²MAX(0, ScoreLink(B → D),
                   ScoreLink(C → E))
else
    new estimate of ScoreLink(A → B)
      = fetchPenalty(B) + ScorePage(B)
```

We define the task of the ScoreLink function to be estimating the discounted[11] sum of the scores of the pages fetched; *assuming that the system started its best-first search at the page referred to by the hyperlink* (plus the cost of fetching pages). In other words, if in Figure 10, Page B were the root of a best-first search, Wawa-IR would next visit C and then either D or E, depending on which referring hyperlink scored higher. Hence, the first few terms of the sum would be the value of root page B, plus the value of C discounted by one time step. We then recursively estimate the remainder of this sum by using the higher score of the two URLs that would be at the front of the search queue (discounting their predicted value for being two time steps in the future).

Of course, since we are using best-first search, rather than visiting C after moving from A to B, Wawa-IR may have a more promising URL in its search queue. In order to keep our calculation of the re-estimated ScoreLink function localized, we largely ignore this aspect of the system's behavior. We only partially capture this phenomenon by adjusting the calculation described above by assuming that links with negative predicted value are not followed.[12]

The above scenario (of localizing the re-estimation of ScoreLink function) does not apply when an URL cannot be fetched (*i.e.*, a "dead link"). Upon such an occurrence, ScoreLink receives a large penalty.

The definition of our "Q function" (see Equation 1) represent a best-first, beam-search strategy which is different than the traditional definition (see Equation 2), which essentially assumes hill climbing.

[11] γ=0.95 is the default discounted value.

[12] Wawa-IR's networks are able to produce negative values because their output units simply output their weighted sum of inputs, *i.e.*, they are linear units. Note that the hidden units in Wawa-IR's networks use sigmoidal activation functions.

Equation 2: New Estimate of the Link $A \rightarrow B$ under Hill-Climbing Search

```
if ScoreLink(B → C) > 0 then
    new estimate of ScoreLink(A → B)
      = fetchPenalty(B) + ScorePage(B)
        + γ(fetchPenalty(C) + ScorePage(C))
        + γ²MAX(0, ScoreLink(C → E))
else
    new estimate of ScoreLink(A → B)
      = fetchPenalty(B) + ScorePage(B)
```

We will use Figure 10 to illustrate the difference between our approach and the traditional way of defining the Q function. In the traditional (hill climbing) approach, since $B \rightarrow D$ was the *second* best-scoring link from B, its value is not reconsidered in the calculation of the score of $A \rightarrow B$. This search strategy does not seem optimal for finding the most relevant pages on the Web. Instead, we should always traverse the link with the highest-score from our set of encountered links. For example, if we have encountered the links $B \rightarrow D$ and $C \rightarrow E$, we should follow the link that has the highest score and not discard $B \rightarrow D$ because it was seen at a previous step and it did not have the highest value at that step.

3.2. Discussion of Wawa for Information Retrieval

The main advantage of Wawa's IR system is its use of theory refinement. That is, we utilize the user's prior knowledge, which need not be perfectly correct. Wawa-IR is a learning system, so it is able to correct user's instructions. In this manner, we are able to rapidly transform Wawa-IR, which is a general search engine, into a specialized/personalized IR agent. Section 4 describes the rapid creation of an effective "home-page finder" agent from the generic Wawa-IR system.

We also allow the user to continually provide advice to the agent. This characteristic of Wawa-IR enables the user to observe an agent and guide its behavior (whenever the user feels that Wawa-IR agent's user model is incorrect). Finally, by learning the ScoreLink function, a Wawa-IR agent is able to more effectively search the Web (by learning about relevant links) *and* automatically create its own training examples via reinforcement learning (which in turn improves the accuracy of the agent with respect to the relevancy of the pages returned).

One cost of using our approach is that we fetch and analyze many Web pages. We have not focused on speed in our case study, ignoring such questions as how well we can do fetching only the first N characters of Web pages, only using the capsule summaries search engines return, etc. Making WAWA-IR faster is part of our future work.

Due to our use of artificial neural networks, it is difficult to understand what was learned (Craven and Shavlik, 1996). It would be nice if a WAWA-IR agent could explain its reasoning to the user. In an attempt to alleviate this problem, we have built a "visualizer" for each neural network in WAWA-IR. The visualizer draws the neural network and graphically displays information on all nodes and links in the network.

4. An Experimental Study of WAWA's IR System

This section describes a case study done to evaluate WAWA's IR system. We built a home-page finder agent by using WAWA's advice language. Appendix A presents the complete advice used for the home-page finder agent. The results of our empirical study illustrate that we can build an effective agent for a web-based task quickly.

4.1. An Instructable and Adaptive Home-Page Finder

We chose the task of building a home-page finder because of an existing system named AHOY! (Shakes et al., 1997), 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.

We wrote a simple interface layered on top of WAWA-IR (see Figure 11) that asks for whatever relevant information is known about the person whose home page is being sought: first name, possible nicknames, middle name or initial, last name, miscellaneous phrases, and a partial URL (e.g., `edu` or `ibm.com`). We then wrote a small program that reads these fields and creates advice that is sent to WAWA-IR. We also 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). 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 learn a general concept about home-page finding, we use our advice language's *variable binding* mechanism. Our home-page finder

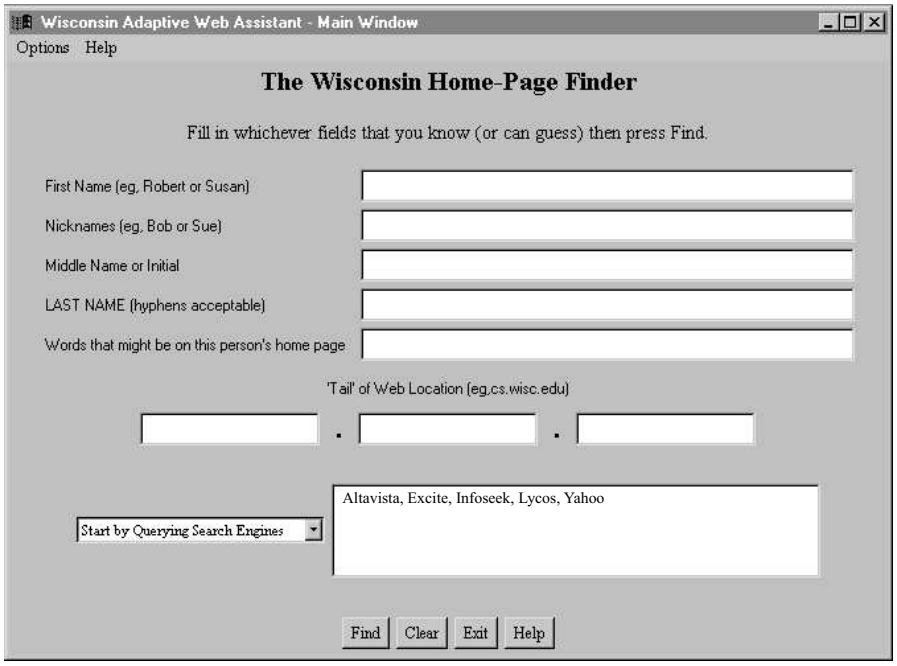

Figure 11. Interface of WAWA-IR's Home-Page Finder

accepts instructions that certain words should be bound to variables associated with first names, last names, etc. We wrote general-purpose advice about home-page finding that uses these variables. Hence, rule 1 in Table III is actually written using advice variables (as illustrated in Figure 8) and not the names of specific people.

Currently, WAWA-IR can only refer to advice variables when they appear in the sliding window. This is a technical limitation which we will address in the future. Advice that refers to other aspects of a Web page needs to be specially created and subsequently retracted for each request to find a specific person's home page.[13] The number of these specific-person rules that our home-page finder creates depends on how much information is provided about the target person. For the experiments below, we only provided information about people's names. This leads to the generation of one to two dozen rules, depending on whether or not middle names or initials are provided.

[13] Users can retract advice from WAWA-IR's neural networks. To retract an advice rule, WAWA-IR removes the network nodes and links associated with that rule.

4.2. Motivation and Methodology

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`) to run experiments that evaluate Wawa-IR. To reduce the computational load of our experiments, we limited this to people in the United States. Out of the 215 people selected, we randomly picked 115 of them to train Wawa-IR and used the remaining 100 as our test set.[14] The "training" phase has two steps. In step (1), we manually run the system on 100 people randomly picked from the training set (we will refer to this set as the *advice-training set*), refining our advice by hand before "freezing" the advice-giving phase. In step (2), we split the remaining 15 people into a set of 10 people for backpropagation training (we will refer to this set as the *training set*) and a set consisting of 5 people for tuning (we will refer to this set as the *tuning set*).

We do not perform backpropagation-based training during the first step of the advice-training phase, since we want to see how accurate we can make our advice without any machine learning. The ScorePage function is trained via backpropagation using the training set. The tuning set is used to avoid overfitting the training examples.[15] For refining the ScoreLink function, we let Wawa-IR automatically generate training examples for each person via temporal-difference learning (see Section 3.1.2). Finally, we evaluate our "trained" home-page finder on the test set. During the testing phase, no learning takes place.

We consider one person as one training example, even though for each person we rate several pages. Hence, the actual number of different examples processed by backpropagation is quite larger than the size of our training set (*i.e.*, by a factor of 10). Table VI describes our technique.

To do neural-network learning, we need to associate a desired score to each page we encounter. We will then be able to compare this score to the output of the ScorePage network for this page and finally perform error backpropagation (Rumelhart et al., 1986). We use a simple heuristic for getting the desired score of a page. We define a *target page* to be the actual home page of a person. Also, recall that the score of a page is a real number in the interval [-10.0, 10.0]. Our heuristic is as follows:

– If the page encountered is the target page, its desired score is 9.5.

[14] We follow standard machine learning methodology in dividing the data set into two subsets, where one subset is used for training and the other for testing purposes.

[15] When a network is overfit, it performs very well on training data but poorly on new data.

Table VI. The Supervised-Learning Technique Used in Training ScorePage Network. See text for explanation of desired output values used during training.

```
While the error on the tuning set is not increasing do the following:
    For each person in the training set do the following 10 times:
        If the person's home page was found,
            then train the ScorePage network on the pages that
            scored higher than the home page, the actual home page, and
            the immediate five pages that scored below the actual home page.
        Otherwise, train the network on the 5 highest scoring pages.
    Calculate the error on the tuning set.
```

– If the page encountered has the same host as the target page, its desired score is 7.0.

– Otherwise, the desired score of the page encountered is -1.0.

For example, suppose the target person is "Alan Turing" and the target page is `http://www.turing.org.uk/turing/` (*i.e.*, his home-page). Upon encountering a page at `http://www.turing.org.uk/`, we will set its desired score to 7.0 since that page has the same host as Alan Turing's home page.

To judge Wawa-IR's performance in the task of finding home-pages, we provide it with the advice discussed above and presented in Appendix A. It is important to note that *for this experiment* we intentionally do *not* provide advice that is specific to ML, CBR, AI research, etc. By doing this, we are able to build a generalized home-page finder and not one that specializes in finding ML, CBR, and AI researchers. Wawa-IR has several options which effect its performance, both in the amount of execution time and the accuracy of its results. We choose small numbers for our parameters, using 100 for the maximum number of pages fetched, and 3 as the maximum distance to travel away from the pages returned by the search engines.

We start Wawa-IR by providing it the person's name as given on Aha's Web page, though we partially standardize our examples by using all common variants of first names. (e.g., "Joseph" and "Joe"). Wawa-IR then converts the name into an initial query (see the next paragraph,) which is sent to the five search engines mentioned earlier (*i.e.*, AltaVista, Excite, InfoSeek, Lycos, and Yahoo).

We compare the performance of Wawa-IR with the performances of Ahoy! and HotBot, a search engine not used by Wawa-IR and the one that performed best in the home-page experiments of Shakes et al.

(1997).[16] We provide the names in our test set to AHOY! via its Web interface. Ahoy! uses *MetaCrawler* as its search engine, which queries nine search engines as opposed to WAWA-IR, which queries only five search engines. We run HOTBOT under two different conditions. The first setting performs a specialized HOTBOT search for people; we use the name given on Aha's page for these queries. In the second variant, we provide HOTBOT 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 its five search engines. 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.

Since people often have different links to their home pages, rather than comparing URLs to those provided on Aha's page, we instead do an exact comparison on the contents of fetched pages to the contents of the page linked to Aha's site. Also, when running WAWA-IR, we never fetch any URLs whose server matched that of Aha's page, thereby preventing visiting Aha's site.

4.3. RESULTS AND DISCUSSION

Table VII lists the best performance of WAWA-IR's home-page finder and the results from AHOY! and HOTBOT. *SL* and *RL* are used to refer to supervised and reinforcement learning, respectively. 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.

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 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. Recall that we specialize WAWA-IR's generic IR system for this task in only a few days.

We next investigate the home-page finder's performance without supervised and/or reinforcement learning. The motivation is to see if we gain performance through learning. We also remove 28 of our initial

[16] Google was not used in this study because it did not yet exist when we ran our experiments in 1998. See Eliassi-Rad (2001) for details on experiments with Google and other aspects of WAWA-IR.

Table VII. Empirical Results: WAWA-IR *vs* AHOY! and HOTBOT (SL = Supervised Learning and RL = Reinforcement Learning)

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

76 rules to see how much our performance degrades with less advice. The 28 rules removed refer to words one might find in a home page that are not the person's name (such as "resume", "cv", "phone", "address", "email", etc). Table VIII reports the performance of WAWA-IR when its home-page finder is trained with less than 76 advice rules and/or is not trained with supervised or reinforcement learning.

Table VIII. Empirical Results on Different Versions of WAWA-IR's Home-Page Finder (SL = Supervised Learning and RL = Reinforcement Learning)

SL	**RL**	**# of Advice Rules**	**% Found**	**Mean Rank Given Page Found**
✓	✓	76	92%	1.3
	✓	76	91%	1.2
		76	90%	1.6
✓	✓	48	89%	1.2
	✓	48	85%	1.4
		48	83%	1.3

The differences between the WAWA-IR runs containing 76 advice rules with learning and without learning are not statistically significant. When we reduce the number of advice rules, WAWA-IR's performance deteriorates. The results show that WAWA-IR is able to learn and increase its accuracy by 6 percentage points; however, the difference is not statistically significant at the 90% confidence level. It is not surprising

that WAWA-IR is not able to reach its best performance, since we do not increase the size of our training data to compensate for the reduction in advice rules. Nonetheless, even with 48 rules, the difference between WAWA-IR and HOTBOT (in this experiment) is statistically significant at the 95% confidence level.

In the cases where the target page for a specific person is found, the mean rank of the target page are similar in all runs. Recall that the mean rank of the target page refers to its ordinal position in the list of pages returned to the user. The mean rank is typically lower with the runs that included some training since without training the target page might not get as high of a score as it would with a trained network.

Assuming that WAWA-IR finds a home page, Table IX lists the average number of pages fetched before the actual home page.

Table IX. Average Number of Pages Fetched by WAWA-IR Before the Target Home Page (SL = Supervised Learning and RL = Reinforcement Learning)

SL	**RL**	**# of Advice Rules**	**Avg Pages Fetched Before Home Page**
✓	✓	76	22
	✓	76	23
		76	31
✓	✓	48	15
	✓	48	17
		48	24

Learning reduces the number of pages fetched before the target page is found. This is quite intuitive. With more learning, WAWA-IR is able to classify pages better and find the target page quicker. However, in Table IX, the average number of pages fetched (before the target page is found) is lower with 48 advice rules than with 76 advice rules. For example, with SL, RL, and 76 rules, the average is 22. With SL, RL, and 48 rules, the average is 15. At first glance, this might not seem intuitive. The reason for this discrepancy can be found in the 28 advice rules that we take out. Recall that we remove advice rules which refer to words one *might* find in a home page that are *not* the person's name (such as "resume", "cv", "phone", "address", "email", etc). With these rules, the SCOREPAGE and SCORELINK networks rate more pages and links as "promising" even though they are not home pages. Hence, more pages are fetched and processed before the target page is found.

5. Using WAWA to Extract Information

Information extraction (IE) is the process of pulling desired pieces of information out of a document, such as the price of a product or the author of an article. Unfortunately, building an IE system requires either a large number of annotated examples[17] 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. WAWA-IE also takes advantage of the intuition that IR and IE are nearly inverse problems of each other. An IR system is given a set of keywords and is asked to rate the relevance of documents. An IE system is given a set of documents and is asked to fill in the slots in a given template. We explore how what is essentially an IR system can be used to address the IE task.

Building an IR agent for the IE task is straightforward in WAWA. The user provides a set of advice rules to WAWA-IE, which describe how the system should score possible bindings to the slots being filled during the IE process. We will call the names of the slots to be filled *variables*, and use "binding a variable" as a synonym for "filling a slot." These initial advice are then "compiled" into the SCOREPAGE network, which rates the goodness of a document in the context of the given variable bindings. Recall that SCOREPAGE is a supervised learner. It learns by being trained on user-provided instructions and user-labeled pages. The SCORELINK network is not used in WAWA-IE since we are only interested in extracting pieces of text from pages and not how they are linked together.

Like its WAWA-IR agents, WAWA-IE agents do not blindly follow user's advice, but instead the agents refine the advice based on the training examples. The use of user-provided advice typically leads to higher accuracy from fewer user-provided training examples (Eliassi-Rad and Shavlik, 2001b; Eliassi-Rad, 2001).

WAWA-IE uses a *generate-and-test* approach to extract information. In the *generation* 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 candidates from the document. In the *test* step, WAWA-IE scores each possible candidate. The candidates that

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

produce scores that are greater than a system-defined threshold are returned as the extracted information.

As already mentioned in Section 2.1.3, of particular relevance to our approach is the fact that WAWA-IE's advice language contains *variables*. To understand how WAWA-IE uses variables, assume that we want to extract speaker names from a collection of seminar announcements. We might wish to give such a system some (good) advice like: *If the page contains the phrase "Speaker . ?FirstName/NNP ?LastName/NNP", then score this page highly.* The leading question marks indicate slots to be filled, and '.' matches any single word. Also, recall that the advice language allows the user to specify the required part of speech for a slot (e.g., *NNP* denotes a proper noun). The precondition of our example rule matches phrases like "*Speaker is Joe Smith*" or "*Speaker : is Jane Doe*".

Figure 12 illustrates an example of extracting speaker names from a seminar announcement using WAWA-IE. The announcement is fed to the candidate generator and selector, which produces a list of speaker candidates. Each entry in the candidates list is then bound to the variables in advice.[18] The output of the (trained) network is a real number (in the interval of -10.0 to 10.0) that represents our confidence in the speaker candidate being a correct slot filler for the given document.

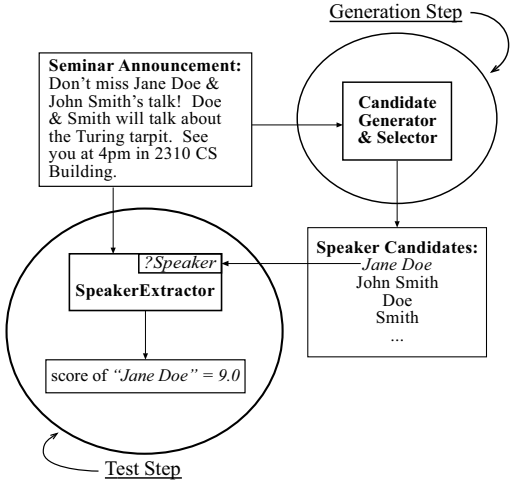

Figure 12. Extraction of speaker names with WAWA-IE

[18] The variable *?Speaker* is a record that holds multiple variables such as *?FirstName* and *?LastName*.

5.1. IE System Description

Wawa-IE uses a candidate generator and selector algorithm along with the ScorePage network to build IE agents. We require the user to provide the following information to Wawa-IE:

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, proper noun, verb, etc) or parse structures (*e.g.*, noun phrase, verb phrase, etc) for each extraction slot.
4. A set of advice rules which refer to the extraction slots as variables.
5. A set of annotated examples, *i.e.*, training documents in which extraction slots have been marked.

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. For example, in our case study, we want to extract speaker names from a set of seminar announcements. So, we give Wawa-IE the following:

1. The collection of CMU's seminar-announcements.
2. The slots to extract are the speaker names.
3. A speaker name is a phrase containing at least one word and at most four words. Each word in the phrase must be tagged as either a noun or a proper noun.
4. A set of advice rules which refer to the speaker name as a variable. For example, one of our advice rules is as follows:

   `when the phrase "`*`?FirstName`*`/NNP `*`?LastName`*`/NNP 's talk" appears in the document then strongly suggest showing page.`

   The variables *?FirstName* and *?LastName* represent the speaker name. The "/NNP" trailing the variables indicates the required POS tag of the variables ("NNP" refers to a proper noun). The precondition of this rule matches phrases such as "John Smith's talk". The action indicates that a document is very likely to be a seminar announcement if it satisfies the precondition. The complete set of rules used in the seminar-announcement domain are listed in Appendix B.

5. The set of word(s) marked as speaker names in the training set of the seminar-announcement domain. For example, the phrases "Jane Doe" and "John Smith" would be marked as speaker names in the seminar announcement depicted in Figure 12.

During training, WAWA-IE first compiles the user's advice into the SCOREPAGE network. WAWA-IE next uses what we call an *individual-slot candidate generator* and a *combination-slots candidate selector* to create training examples for the SCOREPAGE network. The same candidate generation and selection process is used after training to generate the possible extractions that the trained network[19] scores.

During testing, given a document from which we wish to extract information, we generate a large number of candidate bindings, and then in turn we provide each set of bindings to the trained network. The neural network produces a numeric output for each set of bindings. Finally, our extraction process returns the bindings that are greater than a system-defined threshold.

For example, in Figure 12, we showed the following seminar announcement:

> *Don't miss Jane Doe and John Smith's talk! Doe and Smith will talk about Turing tarpit. See you at 4pm in 2310 CS Building.*

The extracted list of candidates for this announcement is "Jane", "Doe", "Jane Doe", "John", "Smith", "John Smith", "talk", "tarpit", "CS", "Building", and "CS Building". These words are tagged as nouns or proper nouns. The word "Turing" is not included in this list since it gets incorrectly tagged as a verb (in present participle form).

Suppose we had "compiled" the following rule (described earlier in this section) to the SCOREPAGE network:

```
when the phrase "?FirstName/NNP ?LastName/NNP 's talk"
appears in the document then strongly suggest showing page
```

Also assume that we had trained the network (the training process is described in Section 5.1.2). We then go through each candidate slot filler and bind them to the appropriate variable. If a candidate filler only has one word, we bound it to the variable *?LastName*. For example, "Jane" gets bound to *?LastName*. If a candidate filler has two words, we bound them to the variables *?FirstName* and *?LastName*, respectively. For example, in the phrase "Jane Doe", the word "Jane" gets bound

[19] We use the terms "trained network" and "trained agent" interchangeably throughout Sections 5 and 6, since the network represents the agent's knowledge-base (*i.e.*, its brain).

to *?FirstName* and the word "Doe" gets bound to *?LastName*. Finally, for each candidate binding, we calculate the score of the ScorePage network (which resides in the agent's knowledge-base) on our seminar announcement. In this case, the network will produce a high score for the candidate "John Smith" since it satisfies the precondition of our rule (assuming, of course, that the training of the network did not significantly alter our rule).

In Section 6, we used four variables to learn about speaker names and four variables to learn about location names. The four variables for speaker names refer to first names, nicknames, middle names (or initials), and last names, respectively. The four variables for location refer to a cardinal number and three other variables representing the non-numerical portion of a location. See Section 6 for some sample rules used in extracting speaker names and location names. The complete sets of rules for both extraction slots are in Appendix B. In the next few sections, we describe the candidate generator and selector, the training phase, and the testing phase in more detail.

5.1.1. *Candidate Generation and Selection*

The first step Wawa-IE takes (both during training and after) is to generate all possible *individual* fillers for each slot on a given document. These candidate fillers can be individual words or phrases. Recall that in Wawa-IE an extraction slot is represented by user-provided variables in the initial advice. Moreover, the user can tag all or any one of the variables representing parts of an extraction slot. Wawa-IE uses the slot's tag information along with either a part-of-speech (POS) tagger (Brill, 1994) or a sentence analyzer (Riloff, 1998) to collect the slot's candidate fillers.

For cases where the user-specified POS tags[20] for a slot (*i.e.*, noun, proper noun, verb, etc), we first annotate each word in a document with its POS using Brill's tagger (1994). Then, for each slot, we collect every word in the document that has the same POS tag as the tag assigned to this variable somewhere in the IE task's advice.

If the user indicated a parse structure for a slot (*i.e.*, noun phrase, verb phrase, etc), then we use *Sundance* (Riloff, 1998), which builds a shallow parse tree by segmenting sentences into noun, verb, or prepositional phrases. We then collect those phrases that match the parse structure for the extraction slot and also generate all possible subphrases of consecutive words (since Sundance only does shallow parsing).

[20] The POS tags provided by the user for an extraction slot can be any POS tag defined in Brill's tagger.

For example, the user specifies that the extraction slot should contain either a word tagged as a proper noun or two consecutive words both tagged as proper nouns. After using the Brill's tagger on the user-provided document, we then collect all the words that were tagged as proper nouns, in addition to every sequence of two words that were both tagged as proper nouns. So, if the phrase "Jane Doe" appeared on the document and the tagger marked both words as proper nouns, we would collect "Jane", "Doe", and "Jane Doe".

At this point we typically have lengthy lists of candidate fillers for each slot, and we need to focus on selecting good *combinations* that fill *all* the slots. Obviously, this process can be combinatorially demanding. In Eliassi-Rad and Shavlik (2001), we present and evaluate several heuristic methods for choosing good combinations.

We do not need to generate combinations of fillers when the IE task contains a template with only one slot (as is the case for the experiments presented in Section 6). However, it is desirable to trim the list of candidate fillers during the training process because training is done iteratively. Therefore, we heuristically select from a slot's list of *training* candidate fillers[21] by scoring each candidate filler using the untrained ScorePage network[22] and returning the highest scoring candidates plus some randomly sampled candidates. This process of picking *informative* candidate fillers from the training data has some beneficial side effects which is described in more detail in the next section.

5.1.2. *Training an IE Agent*

Figure 13 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 use the candidate generator and selector described above. The list of negative training examples collected by the selector contains informative negative examples (*i.e.*, near misses) because the heuristic search used in the selector scores the training documents on the untrained ScorePage network. That is, the (user-provided) prior knowledge scored these "near miss" extractions highly (as if they were true extractions).

After the N highest-scoring negative examples are collected, we train the ScorePage neural network using these negative examples, as well as a small number of randomly selected negative examples, and all the provided positive examples. By training the network to recognize (i.e.,

[21] The candidate fillers associated with the training set.

[22] By untrained, we mean a network containing only compiled (initial) advice and without any further training via backpropagation and labeled examples.

produce a high output score for) a correct extraction in the context of the document as a whole (see Section 2.3), we are able to take advantage of the global layout of the information available in the documents of interest.

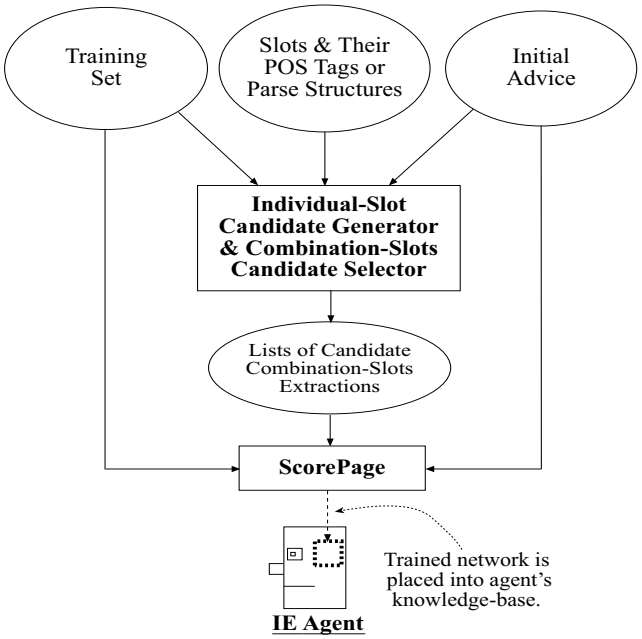

Figure 13. Building a Trained IE agent

Since the ScorePage network outputs a real number, Wawa-IE needs to define a threshold on this output such that the bindings for the scores above the threshold are returned to the user as extractions and the rest are discarded. Note that the value of the threshold can be used to manipulate the performance of the IE agent. For example, if the threshold is set to a high number (*e.g.*, 8.5), then the agent might miss a lot of the correct fillers for a slot (*i.e.*, have low *recall*), but the number of extracted fillers that are correct should be higher (*i.e.*, high *precision*). *Recall* (van Rijsbergen, 1979) is the ratio of the number of correct fillers extracted to the total number of fillers in correct extraction slots. *Precision* (van Rijsbergen, 1979) is the ratio of the number of correct fillers extracted to the total number of fillers extracted. Finally, the F_1-*measure* combines precision and recall using the following formula: $F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$. The F_1-measure (van Rijsbergen, 1979) is used regularly to compare the performances of IR and IE systems because it weights precision and recall equally and produces one single number.

To avoid overfitting[23] the ScorePage network and to find the best threshold on its output after training is done, we actually divide the training set into two disjoint sets. One of the sets is used to train the ScorePage network. The other set, the *tuning* set, is first used to "stop" the training of the ScorePage network. Specifically, we cycle through the training examples 100 times. After each iteration over the training examples, we use the lists of candidate fillers associated with the tuning set to evaluate the F_1-*measure* produced by the network for various settings of the threshold. We pick the network that produced the highest F_1-measure on our tuning set as our final trained network.

We utilize the tuning set (a second time) to find the optimal threshold on the output of the trained ScorePage network. Specifically, we perform the following:

- For each *threshold* value, t, from -8.0 to 9.0 with increments of 0.25, do
  - Run the tuning set through the trained ScorePage network to find the F_1-measure (for the threshold t).
- Set the optimal threshold to the threshold associated with the maximum F_1-measure.

5.1.3. *Testing a Trained IE Agent*

Figure 14 depicts the steps a trained IE agent takes to produce extractions.

For each entry in the list of 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 greater than the threshold defined during the tuning step, we record the bindings as an extraction. Otherwise, these bindings are discarded.

5.2. Discussion of Wawa for Information Extraction

A novel aspect of Wawa-IE is its exploitation of the relationship between IR and IE. That is, we build IR agents that treat possible extractions as keywords, which are in turn judged within the context of the entire document.

[23] When a network is overfit, it performs very well on training data but poorly on new data.

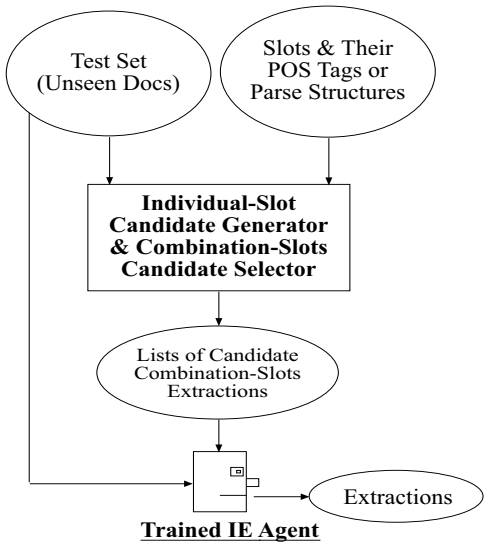

Figure 14. Testing a Trained IE Agent

The use of theory refinement allows us to take advantage of user's prior knowledge, which need not be perfectly correct since WAWA-IE is a learning system. This, in turn, reduces the need for labeled examples, which are very expensive to get in the IE task. Also, compiling users' prior knowledge into the SCOREPAGE network provides a good method for finding informative negative training examples (*i.e.*, near misses).

One cost of using our approach is that we require the user to provide us with the POS tags or parse structures of the extraction slots. We currently assume that the Brill's tagger and Sundance are perfect (*i.e.*, they tag words and parse sentences with 100% accuracy). Brill's tagger annotates the words on a document with 97.2% accuracy (Brill, 1994), so 2.8% error rate propagates into our results. We were not able to find accuracy estimates for Sundance, though recall that we also consider all subphrases of the phrases Sundance produces.

Our approach is computationally demanding, due to its use of a generate-and-test approach. But, CPU cycles are abundant; and, our experiments (Eliassi-Rad and Shavlik, 2001b; Eliassi-Rad, 2001; Eliassi-Rad and Shavlik, 2001a) have shown that WAWA-IE still performs well when only using a subset of all possible combinations of slot fillers.

6. An Experimental Study of WAWA's IE System

In this section, we test WAWA-IE on the CMU seminar-announcements domain (Freitag, 1997). This domain consists of 485 pages. The task[24] is to extract the start time, end time, speaker, and location from an announcement. We report on results for extracting speaker and location. We omit start and end times from our experiments since almost all systems perform very well on these slots. In addition, we follow existing methodology and independently extract speaker and location names, because each document is assumed to contain only one announcement. That is, for each announcement, we do not try to pair up speakers and locations, instead we return a list of speakers and a separate list of locations.

Seminar announcements are tagged using Brill's part-of-speech tagger (Brill, 1994) and common ("stop") words are discarded. We did not stem the words in this study since converting words to their base forms removes information that would be useful in the extraction process.

6.1. Experimental Methodology

We compare our system to seven other information extraction systems using the CMU seminar announcements domain (Freitag, 1998). These systems are HMM (Freitag and McCallum, 1999), BWI (Freitag and Kushmerick, 2000), SRV (Freitag, 1998), Naive Bayes (Freitag, 1998), WHISK (Soderland, 1999), RAPIER (Califf, 1998), and RAPIER-WT (Califf, 1998). None of these systems exploits prior knowledge. Except for Naive Bayes, HMM, and BWI, the rest of the systems use relational learning algorithms. RAPIER-WT is a variant of RAPIER where information about semantic classes is not utilized. HMM (Freitag and McCallum, 1999) employs a hidden Markov model to learn about extraction slots. BWI (Freitag and Kushmerick, 2000) combines wrapper induction techniques with AdaBoost to solve the IE task.

Freitag (1998) first randomly divided the 485 documents in the seminar announcements domain into ten splits, and then randomly divided each of the ten splits into approximately 240 training examples and 240 testing examples. Except for WHISK, the results of the other systems are all based on the same 10 data splits. The results for WHISK are from a single trial with 285 documents in the training set and 200 documents in the testing set.

We give WAWA-IE 9 and 10 advice rules in *Backus-Naur Form* (BNF) (Aho et al., 1986) notation about speakers and locations, re-

[24] We chose this task because it has been widely used in the literature and allows us to directly compare the performance of WAWA-IE to several existing systems.

spectively (see Appendix B). We wrote none of these advice rules with the specifics of the CMU seminar announcements in mind. The rules describe our prior knowledge about what might be a speaker or a location in a *general* seminar announcement. It took us about half a day to write these rules and we did not *manually* refine these rules over time.

For this domain, we create the same number of negative training examples (for speaker and location independently) as the number of positive examples. We choose 95% of the negatives, from the complete list of possibilities, by collecting those that score the highest on the untrained ScorePage network; the remaining 5% are chosen randomly from the complete list.

Table X describes four rules used in the domain theories of speaker and location slots. Rule SR1 matches phrases of length three that start with the word "professor" and have two proper nouns for the remaining words. In rule SR2, we are looking for phrases of length four where the first word is "speaker," followed by another word which we do not care about, and trailed by two proper nouns. SR2 matches phrases like "*Speaker : Joe Smith*" or "*speaker is Jane Doe*". Rules LR1 and LR2 match phrases such as "*Room 2310 CS*". LR2 differs from LR1 in that it requires the two words following "room" to be a cardinal number and a proper noun, respectively (i.e., LR2 is a subset of LR1). Since we are more confident that phrases matching LR2 describe locations, LR2 sends a higher weight to the output unit of the ScorePage network than LR1.

Table X. Sample rules used in the domain theories of speaker and location slots

SR1	When "Professor *?FirstName*/NNP *?LastName*/NNP" then strongly suggest showing page
SR2	When "Speaker . *?FirstName*/NNP *?LastName*/NNP" then strongly suggest showing page
LR1	When "Room *?LocNumber ?LocName*" then suggest showing page
LR2	When "Room *?LocNumber*/CD *?LocName*/NNP" then strongly suggest showing page

6.2. Results

Tables XI and XII show the results of our Wawa-IE agent and the other seven systems for the speaker and location slots, respectively. The results reported are the averaged precision, recall, and F_1 values across the ten splits. The precision, recall, and F_1-measure for a split is determined by the optimal threshold found for that split using the tuning set (see Section 5.1.2 for further details). For all ten splits, the optimal thresholds on Wawa-IE's untrained agent are 5.0 for the speaker slot and 9.0 for the location slot. The optimal threshold on Wawa-IE's trained agent vary from one split to the next in both the speaker slot and the location slot. For the speaker slot, the optimal thresholds on Wawa-IE's trained agent vary from 0.25 to 2.25. For the location slot, the optimal thresholds on Wawa-IE's trained agent range from -6.25 to 0.75.

Since the speaker's name and the location of the seminar may appear in multiple forms in an announcement, an extraction is considered correct as long as any one of the possible correct forms is extracted. For example, if the speaker is "John Doe Smith", the words "Smith", "Joe Smith", "John Doe Smith", "J. Smith", and "J. D. Smith" might appear in a document. Any one of these extractions is considered correct. This method of marking correct extractions is also used in the other IE systems against which we compare our approach.

We use precision, recall, and the F_1-measure to compare the different systems (see Section 5.1.2 for definitions of these terms). An ideal system has a precision and recall of 100%.

6.3. Discussion

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. For both speaker and location slots, the F_1-measure of our system is considerably higher than Naive Bayes and all the relational learners. Our trained IE agent performs competitively with the BWI and HMM learner. Our F_1-measures are high because we generate many extraction candidates in our generate-and-test model. Hence, we are able to extract a lot of the correct fillers from the data set, which in turn leads to higher recall than the other systems. After training, we are able to reject enough candidates so that we obtain reasonable precision. Figure 15 illustrates this fact for the speaker slot. A point in this graph represents the averaged precision and recall values at a specific network output across the ten splits. There are 38 points on each curve in Figure 15 representing the network outputs from 0.0 to

Table XI. Results on the speaker slot for seminar announcements task

System	Precision	Recall	F_1
HMM	77.9	75.2	76.6
Wawa-IE's Trained Agent	61.5	86.6	71.8
BWI	79.1	59.2	67.7
SRV	54.4	58.4	56.3
RAPIER-WT	79.0	40.0	53.1
RAPIER	80.9	39.4	53.0
Wawa-IE's Untrained Agent	29.5	96.8	45.2
Naive Bayes	36.1	25.6	30.0
WHISK	71.0	15.0	24.8

Table XII. Results on the location slot for seminar announcements task

System	Precision	Recall	F_1
Wawa-IE's Trained Agent	73.9	84.4	78.8
HMM	83.0	74.6	78.6
BWI	85.4	69.6	76.7
RAPIER-WT	91.0	61.5	73.4
SRV	74.5	70.1	72.3
RAPIER	91.0	60.5	72.7
WHISK	93.0	59.0	72.2
RAPIER-W	90.0	54.8	68.1
Naive Bayes	59.6	58.8	59.2
Wawa-IE's Untrained Agent	29.2	98.2	45.0

9.0 with increments of 0.25. The trained agent generates much better precision scores than the untrained agent.

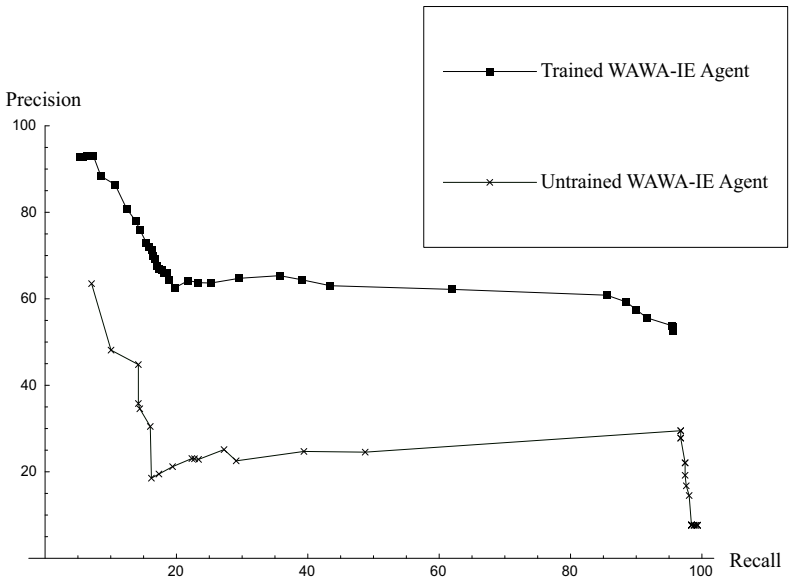

Figure 15. Precision/recall curves for WAWA-IE's untrained and trained agents extracting speaker slots

The increase in performance from WAWA-IE's untrained agent to WAWA-IE's trained agent shows that our agent is not "hard-wired" to perform well on this domain and that training helped our performance.

Finally, we should note that several of the other systems have higher precision, so depending on the user's tradeoff between recall and precision, different systems would be preferred on this testbed.[25].

7. Related Work

Like WAWA-IR, *Syskill and Webert* (Pazzani et al., 1996), and *WebWatcher* (Joachims et al., 1997) are Web agents that use machine learning techniques. They, respectively, use a Bayesian classifier and an RL-TFIDF hybrid to learn. Drummond *et al.* (Drummond et al., 1995) have created a similar system which assists users browsing software libraries; it learns unobtrusively by observing users' actions. *Letizia* (Lieberman, 1995) is a system similar to Drummond et al.'s that uses look-ahead search from the current location in the user's Web browser. Unlike WAWA-IR, these systems are unable to accept (and refine) advice, which usually is simple to provide and can lead to better learning than manually either rating or visiting many Web pages.

[25] See Eliassi-Rad (2001 and Eliassi-Rad and Shavlik (2001a) for details on other experiments using WAWA-IE

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 (Feldman et al., 2000) 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 (Califf, 1998; Freitag, 1998; Soderland, 1999). The second group learns parameters of hidden Markov models (HMMs) and uses the HMMs to extract information (Bikel et al., 1999; Freitag and McCallum, 1999; Leek, 1997; Ray and Craven, 2001; Seymore et al., 1999). Recently, Freitag and Kushmerick (2000) combined wrapper induction techniques (Kushmerick, 2000) with the AdaBoost algorithm (Schapire and Singer, 1998) 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 (1997) 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 (1999) 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. (1999) 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 (1999) 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).

8. Current Work and Future Directions

In order to better understand what people would like to say to an instructable Web agent such as WAWA, we have started work with several research groups in the Medical School of our university to build personalized and easily customized intelligent agents which are capable of returning relevant information from the Web. Based on these interactions, we hope to improve our advice language.

Moreover, we have started running additional experiments to better study how well WAWA-IR is able to learn general concepts like finding research-group pages. This new domain is a much harder problem than finding home pages because the syntactical clues used in finding home pages might not (and usually do not) exist anymore. For example, a person's name is always on his/her home page. However, when searching for pages about a research group like pages about research on "operating systems", the actual phrase "operating systems" might not appear on an actual page about research on operating systems.

We are also currently looking at ways 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 are developing methods whereby WAWA can automatically infer plausible training examples by observing users' normal use of their browsers (Goecks and Shavlik, 2000).

In the IE experiments reported, we trained a network for finding the speaker slots and another one to find the location slots. We needed to train two separate networks because testing the set of the cross-products of all speaker and location candidates was computationally too expensive. However, this posed no serious problem in this domain since each document contained only one announcement. That is, we did not have to match multiple speakers with multiple locations in one document. The problem of having more than once instance of the template in a document is known as the *multi-slot* problem. We have extended our catalog of candidate-selector algorithms to include heuristic methods such as a modified versions of WalkSAT (Selman et al., 1996), GSAT, random local search, and a hill climber with mul-

tiple random starts (Eliassi-Rad and Shavlik, 2001b; Eliassi-Rad, 2001). With this functionality, we have been able to reduce the computational problem of having too many candidate bindings and solve the multi-slot extraction problem for WAWA-IE. Eliassi-Rad and Shavlik (2001) provides empirical results on a multi-slot problem of extracting proteins and their locations on the cell from a set of biological abstracts. Our WAWA-IE agent was able to out-perform the state-of-the-art method of Ray and Craven (2001).

In our IE domains, we have started measuring performance as a function of the number of positive training examples and a function of the number of advice rules. In addition, we are starting experiments on another IE domain, namely the WebKB domain (Freitag, 1998). Finally, we are looking into incorporating 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.

9. Conclusions

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.

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 the efficacy of using system-generated training examples to improve the evaluation of potential hyperlinks to traverse.

We also describe and evaluate a system for using theory refinement to perform information extraction. WAWA's information extraction sys-

tem 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.

In the CMU seminar-announcements domain, the F_1-measures of WAWA-IE's agent are not only significantly higher than many other approaches, but also competitive with a new state-of-the-art system (which uses a boosted wrapper induction technique). In recent work (Eliassi-Rad and Shavlik, 2001b; Eliassi-Rad, 2001), we empirically show the benefits of using an intelligent algorithm for selecting possible candidates for multiple slots and provide additional evidence that our approach improves on the state of the art.

WAWA utilizes the user's knowledge 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. Our empirical studies in both the information retrieval and information extraction tasks illustrate the advantages of building instructable and adaptive agents.

Acknowledgements

The research reported in this article was supported in part by National Science Foundation (NSF) grant IRI-9502990, University of Wisconsin Vilas Trust, and National Library of Medicine (NLM) grant 1 R01 LM07050-01.

Material presented in Sections 2, 3, and 4 was partially reported in Shavlik and Eliassi-Rad (1998a, 1998b) and Shavlik *et al.* (1999). Material covered in Sections 5 and 6 was partially reported in Eliassi-Rad and Shavlik (2001b, 2001a).

Appendix

A. Advice Rules for the Home-Page Finder

This section of the appendix presents the rules given to the home-page finder (see Section 4 for details on this case study). These rules contain the following 13 variables:

1. *?FirstName* ← First name of a person (*e.g.*, Robert)
2. *?FirstInitial* ← First initial of a person (*e.g.*, R)
3. *?NickNameA* ← First nickname of a person (*e.g.*, Rob)
4. *?NickNameB* ← Second nickname of a person (*e.g.*, Bob)
5. *?MiddleName* ← Middle name of a person (*e.g.*, Eric)
6. *?MiddleInitial* ← Middle initial of a person (*e.g.*, E)
7. *?LastName* ← Last name of a person (*e.g.*, Smith)
8. *?MiscWord1* ← First miscellaneous word
9. *?MiscWord2* ← Second miscellaneous word
10. *?MiscWord3* ← Third miscellaneous word
11. *?UrlHostWord1* ← Third word from the end of a host url (*e.g.*, `cs` in `http://www.cs.wisc.edu`)
12. *?UrlHostWord2* ← Second word from the end of a host url (*e.g.*, `wisc` in `http://www.cs.wisc.edu`)
13. *?UrlHostWord3* ← Last word in a host (*e.g.*, `edu` in `http://www.cs.wisc.edu`)

In our experiments, we only used variables numbered 1 through 7 since we wanted to fairly compare Wawa's home-page finder to existing alternative approaches. That is, we did not provide any values for variables numbered 8 through 13. We introduced these variables and even wrote some rules about them only to illustrate that there is other information besides a person's name that might be helpful in finding his/her home-page.

The actions used in the home-page finder rules are as follows:

A1. *suggest doing both*
This action adds a moderately weighted link from the rule's new hidden unit in both the ScorePage and the ScoreLink networks into the output units of these networks.

A2. *suggest showing page*
This action adds a moderately weighted link from the rule's new hidden unit in the ScorePage network into the network's output unit.

A3. *suggest following link*
This action adds a moderately weighted link from the rule's new hidden unit in the ScoreLink network into the network's output unit.

A4. *avoid both*
This action adds a link with a moderately negative weight from the rule's new hidden unit in both the ScorePage and the ScoreLink networks into the output units of these networks.

A5. *avoid showing page*
This action adds a link with a moderately negative weight from the rule's new hidden unit in the ScorePage network into the network's output unit.

A6. *avoid following link*
This action adds a link with a moderately negative weight from the rule's new hidden unit in the ScoreLink network into the network's output unit.

Actions can be prefixed by the following four modifiers:

– *Definitely*: Assuming the preconditions of a 'definite' rule are fully met, the link out of the sigmoidal hidden unit representing the rule will have a weight of

  - 11.25, for actions A1 through A3, and
  - -11.25, for actions A4 through A6.

– *Strongly*: Assuming the preconditions of a 'strong' rule are fully met, the link out of the sigmoidal hidden unit representing the rule will have a weight of

  - 7.5, for actions A1 through A3, and
  - -7.5, for actions A4 through A6.

– *Moderately*: Assuming the preconditions of a 'moderate' rule are fully met, the link out of the sigmoidal hidden unit representing the rule will have a weight of

  - 2.5, for actions A1 through A3, and
  - -2.5, for actions A4 through A6.

  When an action does not have modifier, then "moderately" is used as the default modifier.

– *Weakly*: Assuming the preconditions of a 'weak' rule are fully met, the link out of the sigmoidal hidden unit representing the rule will have a weight of

  - 0.75, for actions A1 through A3, and
  - -0.75, for actions A4 through A6.

Before we present the home-page finder rules, we will define some non-terminal tokens in Backus-Naur form (BNF) (Aho et al., 1986) used in our rules. These non-terminal tokens are:

first_names	→	*?FirstName* | *?FirstInitial* | *?NicknameA* | *?NicknameB*
middle_names	→	*?MiddleName* | *?MiddleInitial*
full_name	→	**first_names middle_names** *?LastName*
regular_name	→	**first_names** *?LastName*
home_page_words	→	home | homepage | home-page
person	→	**full_name** | **regular_name**

The following rules look for *pages* with the person's name and either the phrase "home page of" or the words "home", "homepage", or "home-page" in the title. The function *consecutiveInTitle* takes a sequences of words and returns true if they form a phrase inside the title of a page. The symbol "." is WAWA's "wild card" symbol. It is a placeholder that matches any single word or punctuation.

home_page_rules_A →

WHEN consecutiveInTitle("home page of" **person**)
THEN definitely suggest showing page |

WHEN consecutiveInTitle(**person** "'s" **home_page_words**)
THEN definitely suggest showing page |

WHEN consecutiveInTitle(**person home_page_words**)
THEN strongly suggest showing page |

WHEN consecutiveInTitle(
  first_names . *?LastName* "'s" **home_page_words**)
THEN suggest showing page |

WHEN consecutiveInTitle(
  "home page of" **first_names .** *?LastName*)
THEN suggest showing page |

WHEN consecutiveInTitle("home page of" **.** *?LastName*)
THEN suggest showing page |

WHEN consecutiveInTitle("home page of" **. .** *?LastName*)
THEN weakly suggest showing page |

WHEN consecutiveInTitle(**person . home_page_words**)
THEN suggest showing page

The next set of rules look for *links* to the person's home-page. The function *consecutive* takes a sequences of words and returns true if the words appear as a phrase on the page.

home_page_rules_B →
WHEN consecutive(**person** "'s" **home_page_words**)
THEN definitely suggest following link |

WHEN consecutive("home page of" **person**)
THEN definitely suggest following link |

WHEN consecutive(**first_names .** *?LastName* "'s" **home_page_words**)
THEN strongly suggest following link |

WHEN consecutive("home page of" **first_names .** *?LastName*)
THEN strongly suggest following link |

WHEN consecutive("home page of" **.** *?LastName*)
THEN suggest following link |

WHEN consecutive(**person**)
THEN strongly suggest following link |

WHEN consecutive(*?LastName*)
THEN suggest following link

The following rules look for *pages* and *links leading to pages* with the person's name in the title but not in a question format. We don't want both the person's name and a question mark in a page's title because it could represent a query on that person and not the person's home-page.

home_page_rules_C →
WHEN (NOT(anywhereInTitle("?"))
AND consecutiveInTitle(**regular_name**
noneOf(**home_page_words**)))
THEN strongly suggest doing both |

WHEN (NOT(anywhereInTitle("?"))
AND consecutiveInTitle(**first_names** . *?LastName*
noneOf(**home_page_words**)))
THEN strongly suggest doing both |

WHEN (NOT(anywhereInTitle("?"))
AND consecutiveInTitle(**first_names**)
AND anywhereInTitle(*?LastName*))
THEN suggest doing both |

WHEN (NOT(anywhereInTitle("?"))
AND consecutiveInTitle(*?LastName* "," **first_names**))

THEN suggest doing both |

WHEN consecutive(**first_names** "'s" **home_page_words**)
THEN suggest doing both |

WHEN consecutive(*?LastName* **home_page_words**)
THEN suggest doing both |

WHEN consecutive("my" **home_page_words**)
THEN suggest doing both

This rule looks for *home-pages* that might lead to other home-pages.

home_page_rules_D →
WHEN (NOT(anywhereInTitle("?"))
AND (anywhereInTitle("home page")
OR anywhereInTitle("homepage")
OR anywhereInTitle("home-page")))
THEN suggest following link

This rule seeks *pages* that have the person's last name near an image. We conjecture that the image might be that person's picture.

home_page_rules_E →
WHEN (insideImageCaption() AND consecutive(*?LastName*))
THEN suggest doing both

The next set of rules look for *pages* and *links* that include some of the query words given by the user (*i.e.*, bindings for some of the variables). These rules use functions like *numberOfQueryWordsOnPage* which obviously returns the number of query words on the page.

home_page_rules_F →
WHEN ((insideEmailAddress() OR insideAddress())
AND (numberOfQueryWordsInWindow() ≥ 1))
THEN weakly suggest doing both |

WHEN (numberOfQueryWordsOnPage() < 1)
THEN avoid following link AND definitely avoid showing page |

WHEN anywhereInURL("?")
THEN strongly avoid following link & definitely avoid showing page |

WHEN anywhereInCurrentHyperlink("?")
THEN strongly avoid following link |

WHEN (anywhereInURL($\sim$) AND NOT(anywhereInURL("?"))
AND (numberOfQueryWordsInURL() $\geq$ 1))
THEN weakly suggest both

The next set of rules look for *pages* and *links* that include some of the query words given by the user. They use a function called *ScaleLinearlyBy* which takes a set of conditions and returns a linear sum depending on the number of conditions that were satisfied.

home_page_rules_G $\rightarrow$
ScaleLinearlyBy(numberOfQueryWordsInWindow())
THEN suggest both |

ScaleLinearlyBy(numberOfQueryWordsInTitle())
THEN suggest showing page AND weakly suggest following link |

The following rules look for *pages* and *links* that include a combination of the words "home page", "home-page", "homepage", "home", "page", "directory", and "people" either on the page itself or in its URL.

home_page_rules_H $\rightarrow$
WHEN consecutive(**home_page_words** "directory")
THEN strongly suggest following link |

WHEN consecutiveInaSection(**home_page_words** "directory")
THEN suggest following link |

WHEN consecutiveInHyperText(**home_page_words**)
THEN suggest following link |

WHEN consecutiveInaSection(**home_page_words**)
THEN weakly suggest doing both |

WHEN (consecutive("home page")
OR consecutive(anyOf("homepage" "home-page")))
THEN weakly suggest doing both |

WHEN (anywhereInURL("home") OR anywhereInURL("page")
OR anywhereInURL("people")
OR anywhereInURL("homepage")
OR anywhereInURL("home-page"))
THEN suggest doing both

The subsequent two rules attempt to find *when a page was last modified*. The function *pageLastModifiedAt*() determines the number of days

from today since the page was modified. If the page does not specify when it was last modified, the value for the function *pageLastModifiedAt*() is zero. If the page reports the last time it was modified, we convert the time into the interval [0, 1], where 0 means never and 1 means the page was changed today. Then, the value of *pageLastModifiedAt*() is 1 minus the scaled value.

home_page_rules_I →
ScaleLinearlyBy(pageLastModifiedAt())
THEN weakly suggest showing page AND very weakly suggest following link

The next three rules use the function *consecutiveInURL*. This function takes a sequences of words and returns true if the words form a phrase in the URL of the page. These rules look for *pages* that have a URL containing the person's name and possibly the words "htm" or "html".

home_page_rules_J →
WHEN consecutiveInURL("∼" anyOf(**first_names** *?LastName*))
THEN weakly suggest doing both |

WHEN consecutiveInURL(**person** anyOf("htm" "html"))
THEN definitely suggest showing page |

The following five rules look for *pages* and *links* that have the phrase "*?FirstName ?LastName*" anywhere on them or in their title, their URL, one of their hypertexts, or one of their hyperlinks.

home_page_rules_K →
WHEN consecutive(*?FirstName ?LastName*)
THEN strongly do both |

WHEN consecutiveInURL(*?FirstName ?LastName*)
THEN strongly do both |

WHEN consecutiveInTitle(*?FirstName ?LastName*)
THEN strongly do both |

WHEN consecutiveInHypertext(*?FirstName ?LastName*)
THEN strongly do both |

WHEN consecutiveInHyperlink(*?FirstName ?LastName*)
THEN strongly do both

The next three rules avoid *pages* that have the phrase "404 not found" in their title.

home_page_rules_L →
WHEN titleStartsWith(anyOf("404" "file") "not found")
THEN strongly avoid both |

WHEN titleEndsWith(anyOf("404" "file") "not found")
THEN strongly avoid both |

WHEN anywhereInTitle("404 not found")
THEN avoid both

The following rules contain advice about commonly used words on a person's homepage like "cv", "resume", etc. The function *NofM* used in this set of rules takes an integer, N and a list of conditions of size M. It returns true if at least N of the M conditions are true. The function *anywhereOnPage*("555 − 1234") is true if a telephone number is on the page. Otherwise, it returns false. These rules were removed from the original set of 76 advice rules to create a new set of initial advice containing only 48 rules (see Section 4.3 for more details).[26]

†**home_page_rules_M** →
WHEN (insideMetaWords()
AND (consecutive("home page")
OR consecutive(anyOf("homepage" "home-page"))
OR consecutive("personal"
anyOf("info" "information"))))
THEN suggest showing page |

WHEN consecutive("curriculum" anyOf("vitae" "vita"))
THEN weakly suggest doing both |

WHEN consecutiveInHypertext("curriculum" anyOf("vitae" "vita"))
THEN suggest doing both |

WHEN consecutiveInHypertext("my" anyOf("vitae" "vita"))
THEN suggest following link |

WHEN consecutiveInHypertext("my" anyOf("cv" "resume"))
THEN suggest following link |

WHEN consecutive("my" anyOf("resume" "cv" "vita" "vitae"))
THEN suggest both |

[26] We marked the non-terminal representing these rules with the † symbol to distinguish them from the other rules.

```
WHEN consecutive( “my” anyOf( “homepage” “home” ) )
THEN suggest both |

WHEN consecutiveInaSection( personal anyOf( “info” “information” ) )
THEN weakly suggest doing both |

WHEN NofM( 2, anywhereInSections( “personal” )
              anywhereInSections( “information” )
              anywhereInSections( “info” )
              anywhereInSections( “projects” )
              anywhereInSections( “interests” ) )
THEN weakly suggest doing both |

ScaleLinearlyBy(
  consecutive(anyOf(?LastName ?MiscWord1 ?MiscWord2 ?MiscWord3) )
  ( anywhereInWindow(“email”) OR anywhereInWindow(“e-mail”)
    OR anywhereInWindow(“mailto”) )
  ( anywhereInWindow(“phone”) OR anywhereInWindow(“555-1234”)
    OR anywhereInWindow(“fax”) OR anywhereInWindow(“telephone”) )
  ( anywhereInWindow(“department”) OR anywhereInWindow(“work”)
    OR anywhereInWindow(“office”) OR anywhereInWindow(“dept”) )
  ( anywhereInWindow(“address”) OR anywhereInWindow(“mailing”) ) ) )
THEN strongly suggest doing both |

ScaleLinearlyBy(
  consecutive(anyOf(?LastName ?MiscWord1 ?MiscWord2 ?MiscWord3) )
  ( anywhereOnPage(“email”) OR anywhereOnPage(“e-mail”)
    OR anywhereOnPage(“mailto”) )
  ( anywhereOnPage(“phone”) OR anywhereOnPage(“fax”)
    OR anywhereOnPage(“555-1234”) OR anywhereOnPage(“telephone”) )
  ( anywhereOnPage(“department”) OR anywhereOnPage(“work”)
    OR anywhereOnPage(“office”) OR anywhereOnPage(“dept”) )
  ( anywhereOnPage(“address”) OR anywhereOnPage(“mailing”) ) ) )
THEN suggest doing both

WHEN consecutive(anyOf(“research” “recent”)
                 anyOf(“summary” “publications”) )
THEN weakly suggest doing both |

WHEN consecutive( “recent publications” )
THEN weakly suggest doing both |
```

```
WHEN ( insideEmailAddress()
        AND consecutive( ?UrlHostWord1 ?UrlHostWord2
                         ?UrlHostWord3 ) )
THEN suggest both |

WHEN ( insideEmailAddress()
        AND consecutive( ?UrlHostWord2 ?UrlHostWord3 ) )
THEN suggest both |

WHEN consecutiveInURL(?UrlHostWord1 ?UrlHostWord2
                      ?UrlHostWord3 )
THEN suggest showing page

WHEN consecutiveInUrl( ?UrlHostWord2 ?UrlHostWord3 )
THEN suggest showing page |

WHEN consecutiveInHyperlink(?UrlHostWord1 ?UrlHostWord2
                            ?UrlHostWord3 )
THEN suggest following link |

WHEN consecutiveInHyperlink( ?UrlHostWord2 ?UrlHostWord3 )
THEN suggest following link |

ScaleLinearlyBy(
    anywhereOnPage("bio") anywhereOnPage("interests")
    anywhereOnPage("hobbies") anywhereOnPage("resume")
    anywhereOnPage("cv") anywhereOnPage("vita")
    anywhereOnPage("vitae") anywhereOnPage("degrees")
    anywhereOnPage("employment") anywhereOnPage( "office")
    anywhereOnPage("courses") anywhereOnPage("classes")
    anywhereOnPage("education") anywhereOnPage("dept") )
THEN strongly suggest showing page |

ScaleLinearlyBy(
   anywhereInWindow("bio") anywhereInWindow("interests")
   anywhereInWindow("hobbies") anywhereInWindow("resume")
   anywhereInWindow("cv") anywhereInWindow("vita")
   anywhereInWindow("vitae") anywhereInWindow("degrees")
   anywhereInWindow("employment") anywhereInWindow("office")
   anywhereInWindow("courses") anywhereInWindow("classes")
   anywhereInWindow("education") anywhereInWindow("dept") )
THEN strongly suggest following link |

WHEN ( anywhereInWindow("links")
        AND consecutive(anyOf("interests" "interesting" "cool")) )
THEN suggest showing page |
```

```
WHEN ( anywhereInWindow("links")
       AND consecutive(anyOf("recommended" "stuff")) )
THEN suggest showing page |

WHEN consecutiveInaSection(anyOf("topics" "areas") "of interest")
THEN suggest doing both |

WHEN consecutiveInTitle( "main page" )
THEN weakly suggest doing both |

WHEN ( consecutive( "contact information" )
       AND anywhereOnPage( ?LastName ) )
THEN strongly do both |

WHEN consecutive( "check your spelling" )
THEN strongly avoid both |

WHEN consecutive( "search tips" )
THEN strongly avoid both
```

The set of advice rules given to the home-page finder is a union of the rules in the non-terminal tokens **home_page_rules_A** through **home_page_rules_M**.

B. Advice Rules for the Seminar-Announcement Extractor

In this section of the appendix, we present the advice rules for the speaker and location slots of the seminar-announcement extractor agent. Recall that the function named *consecutive* takes a sequence of words and returns true if they appear as a phrase on the page. Otherwise, it returns false.

To extract the speaker name, we used the following four variables:

1. *?FirstName* ← First name or initial of a person
2. *?NickName* ← Nickname of a person
3. *?MiddleName* ← Middle name or initial of a person
4. *?LastName* ← Last name of a person

The advice rules used for the speaker slot in Backus-Naur form (Aho et al., 1986) are listed below. The non-terminal tokens **talk_VB** and **talk_VBG** used in the rules refer to verbs in base form and present participle form, respectively. Recall that we choose not to do stemming of words in this study.

spk_rules →
WHEN consecutive(**title spk_name**)
THEN strongly suggest showing page |

WHEN consecutive(**spk_name** , **degree**)
THEN strongly suggest showing page |

WHEN consecutive(**spk_intro . spk_name**)
THEN strongly suggest showing page |

WHEN consecutive(**spk_name** "'s" **talk_noun**)
THEN strongly suggest showing page |

WHEN consecutive(**spk_name** "will" **talk_VB**)
THEN strongly suggest showing page |

WHEN consecutive(**spk_name** "will be" **talk_VBG**)
THEN strongly suggest showing page |

WHEN consecutive("presented by" **spk_name**)
THEN strongly suggest showing page |

WHEN consecutive("talk by" **spk_name**)
THEN strongly suggest showing page |

WHEN **spk_name** THEN weakly suggest showing page

spk_name →
?LastName/NNP |
?FirstName/NNP *?LastName*/NNP |
?NickName/NNP *?LastName*/NNP |
?FirstName/NNP *?MiddleName*/NNP *?LastName*/NNP |
?NickName/NNP *?MiddleName*/NNP *?LastName*/NNP

title → "mr" | "ms" | "mrs" | "dr" | "prof" | "professor" | "mr." | "ms." | "mrs." | "dr." | "prof."

degree → "ba" | "bs" | "ms" | "ma" | "jd" | "md" | "phd" | "b.a." | "b.s." | "m.s." | "m.a." | "j.d." | "m.d." | "ph.d."

spk_intro → "visitor" | "who" | "seminar" | "lecturer" | "colloquium" | "speaker" | "talk"

talk_noun → "talk" | "presentation" | "lecture" | "speech"

talk_VB → "talk" | "lecture" | "speak" | "present"

talk_VBG → "talking" | "lecturing" | "speaking" | "presenting"

To extract the location name, we used the following four variables:

1. *?LocNumber* ← A cardinal number representing a room number, a building number, etc
2. *?LocNameA* ← First word in the name of a building, a street, etc
3. *?LocNameB* ← Second word in the name of a building, a street, etc
4. *?LocNameC* ← Third word in the name of a building, a street, etc

The advice rules used for the location slot in Backus-Naur form (Aho et al., 1986) are:

loc_rules →
WHEN consecutive(**loc_name_tagged**)
THEN strongly suggest showing page |

WHEN consecutive(**loc_name**)
THEN weakly suggest showing page |

WHEN consecutive(**loc_name_tagged loc_tokens**)
THEN strongly suggest showing page |

WHEN consecutive("in" **loc_name loc_tokens**)
THEN strongly suggest showing page |

WHEN consecutive(**loc_tokens loc_name_tagged**)
THEN strongly suggest showing page |

WHEN consecutive(**loc_tokens loc_name**)
THEN suggest showing page |

WHEN consecutive(**loc_tokens . loc_name_tagged**)
THEN strongly suggest showing page |

WHEN consecutive(**loc_tokens . loc_name**)
THEN suggest showing page |

WHEN consecutive(**loc_intro . loc_name_tagged**)
THEN strongly suggest showing page |

WHEN consecutive(**loc_intro . loc_name**)
THEN suggest showing page

loc_name_tagged →
?LocNumber/CD |
?LocNameA/NNP |
?LocNumber/CD *?LocNameA*/NNP |
?LocNameA/NNP *?LocNumber*/CD |
?LocNameA/NNP *?LocNameB*/NNP *?LocNumber*/CD |
?LocNumber/CD *?LocNameA*/NNP *?LocNameB*/NNP *?LocNameC*/NNP|
?LocNameA/NNP *?LocNameB*/NNP *?LocNameC*/NNP *?LocNumber*/CD

loc_name →
?LocNumber |
?LocNameA |
?LocNumber ?LocNameA |
?LocNameA ?LocNumber |
?LocNumber ?LocNameA ?LocNameB |
?LocNameA ?LocNameB ?LocNumber |
?LocNumber ?LocNameA ?LocNameB ?LocNameC |
?LocNameA ?LocNameB ?LocNameC ?LocNumber

loc_tokens → "hall" | "auditorium" | "building" | "bldg" | "center" |
"campus" | "school" | "university" | "conference" | "conf" |
"room" | "rm" | "floor" | "inst" | "institute" | "wing" |
"union" | "college" | "office" | "lounge" | "lab" | "laboratory" |
"library"| "classroom" | "tower" | "street" | "avenue" |
"alley" | "road" | "drive"| "circle" | "trail" | "st" | "ave" |
"rd" | "dr" | "cr" | "tr"

loc_intro → "place" | "where" | "location"

References

Aho, A., R. Sethi, and J. Ullman: 1986, *Compilers, Principles, Techniques and Tools*. Reading, MA: Addison Wesley.

Belew, R. K.: 2000, *Finding Out About: A Cognitive Perspective on Search Engine Technology and the WWW*. New York, NY: Cambridge University Press.

Bikel, D., R. Schwartz, and R. Weischedel: 1999, 'An algorithm that learns what's in a name'. *Machine Learning: Special Issue on Natural Language Learning* **34**(1/3), 211–231.

Brill, E.: 1994, 'Some advances in rule-based part of speech tagging'. In: *Proceedings of the Twelfth National Conference on Artificial Intelligence*. Seattle, WA, pp. 722–727.

Califf, M. E.: 1998, 'Relational Learning Techniques for Natural Language Information Extraction'. Ph.D. thesis, Department of Computer Sciences, University of Texas, Austin, TX.

Craven, M. and J. Kumlien: 1999, 'Constructing biological knowledge-bases by extracting information from text sources'. In: *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*. Heidelberg, Germany, pp. 77–86.

Craven, M. W. and J. W. Shavlik: 1996, 'Extracting Tree-Structured Representations of Trained Networks'. In: *Advances in Neural Information Processing Systems*, Vol. 8. Denver, CO, pp. 24–30.

Croft, W., H. Turtle, and D. Lewis: 1991, 'The use of phrases and structured queries in information retrieval'. In: *Proceedings of the Fourteenth International ACM SIGIR Conference on R & D in Information Retrieval*. Chicago, IL, pp. 32–45.

Drummond, C., D. Ionescu, and R. Holte: 1995, 'A Learning Agent that Assists the Browsing of Software Libraries'. Technical Report TR-95-12, University of Ottawa, Ottawa, Canada.

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. (Also appears as UW Technical Report CS-TR-01-1431).

Eliassi-Rad, T. and J. Shavlik: 2001a, 'Intelligent Web Agents that Learn to Retrieve and Extract Information'. In: P. Szczepaniak, F. Segovia, J. Kacprzyk, and L. Zadeh (eds.): *Intelligent Exploration of the Web*. Springer-Verlag.

Eliassi-Rad, T. and J. Shavlik: 2001b, 'A Theory-Refinement Approach to Information Extraction'. In: *Proceedings of the Eighteenth International Conference on Machine Learning*. Williamstown, MA.

Feldman, R., Y. Liberzon, B. Rosenfeld, J. Schler, and J. Stoppi: 2000, 'A framework for specifying explicit bias for revision of approximate information extraction rules'. In: *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Boston, MA, pp. 189–197.

Freitag, D.: 1997, 'Using grammatical inference to improve precision in information extraction'. In: *Proceedings of the Fourteenth International Conference on Machine Learning: Workshop on Automata Induction, Grammatical Inference, and Language Acquisition*.

Freitag, D.: 1998, 'Machine Learning for Information Extraction in Informal Domains'. Ph.D. thesis, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA.

Freitag, D. and N. Kushmerick: 2000, 'Boosted Wrapper Induction'. In: *Proceedings of the Seventeenth National Conference on Artificial Intelligence*. Austin, TX, pp. 577–583.

Freitag, D. and A. McCallum: 1999, 'Information Extraction with HMMs and Shrinkage'. In: *Proceedings of the Sixteenth National Conference on Artificial Intelligence Workshop on Machine Learning for Information Extraction*. Orlando, FL, pp. 31–36.

Goecks, J. and J. Shavlik: 2000, 'Learning Users' Interests by Unobtrusively Observing Their Normal Behavior'. In: *Proceedings of the 2000 International Conference on Intelligent User Interfaces*. New Orleans, LA.

Joachims, T., D. Freitag, and T. Mitchell: 1997, 'WebWatcher: A Tour Guide for the World Wide Web'. In: *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*. Nagoya, Japan, pp. 770–775.

Kushmerick, N.: 2000, 'Wrapper Induction: Efficiency and expressiveness'. *Artificial Intelligence* **118**, 15–68.

Leek, T.: 1997, 'Information extraction using hidden Markov models'. Master's Thesis, Department of Computer Science & Engineering, University of California, San Diego.

Lieberman, H.: 1995, 'Letzia: An Agent that Assists Web Browsing'. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*. Montreal, Canada, pp. 924–929.

Maclin, R. and J. Shavlik: 1996, 'Creating advice-taking reinforcement learners'. *Machine Learning* **22**, 251–281.

Mitchell, T.: 1997, *Machine Learning*. McGraw-Hill.

Ourston, D. and R. Mooney: 1994, 'Theory refinement: Combining analytical and empirical methods'. *Artificial Intelligence* **66**, 273–309.

Pazzani, M. and D. Kibler: 1992, 'The utility of knowledge in inductive learning'. *Machine Learning* **9**, 57–94.

Pazzani, M., J. Muramatsu, and D. Billsus: 1996, 'Syskill & Webert: Identifying interesting web sites'. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*. Portland, OR, pp. 54–61.

Ray, S. and M. Craven: 2001, 'Representing Sentence Structure in Hidden Markov Models for Information Extraction'. In: *Proc. of IJCAI01*. Seattle, WA.

Riloff, E.: 1998, '*The Sundance Sentence Analyzer*'. http://www.cs.utah.edu/projects/nlp/.

Rumelhart, D., G. Hinton, and R. Williams: 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, pp. 318–363.

Salton, G.: 1991, 'Developments in automatic text retrieval'. *Science* **253**, 974–979.

Salton, G. and C. Buckley: 1988, 'Term-weighting approaches in automatic text retrieval'. *Information Processing and Management* **24**(5), 513–523.

Schapire, R. and Y. Singer: 1998, 'Improved boosting algorithms using confidence-rated predictions'. In: *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*.

Sejnowski, T. and C. Rosenberg: 1987, 'Parallel networks that learn to pronounce English text'. *Complex Systems* **1**, 145–168.

Selman, B., H. Kautz, and B. Cohen: 1996, 'Local search strategies for satisfiability testing'. *DIMACS Series in Discrete Mathematics and Theoretical CS* **26**, 521–531.

Seymore, K., A. McCallum, and R. Rosenfeld: 1999, 'Learning Hidden Markov Model Structure for Information Extraction'. In: *Proceedings of the Sixteenth National Conference on Artificial Intelligence Workshop on Machine Learning for Information Extraction*. Orlando, FL, pp. 37–42.

Shakes, J., M. Langheinrich, and O. Etzioni: 1997, 'Dynamic Reference Sifting: A Case Study in the Homepage Domain'. In: *Proceedings of the Sixth International World Wide Web Conference*. Santa Clara, CA, pp. 189–200.

Shavlik, J., S. Calcari, T. Eliassi-Rad, and J. Solock: 1999, 'An Instructable, Adaptive Interface for Discovering and Monitoring Information on the World-Wide Web'. In: *Proceedings of the 1999 International Conference on Intelligent User Interfaces*. Redondo Beach, CA, pp. 157–160.

Shavlik, J. and T. Eliassi-Rad: 1998a, 'Building intelligent agents for web-based tasks: A theory-refinement approach'. In: *Proceedings of the Conference on Automated Learning and Discovery Workshop on Learning from Text and the Web*. Pittsburgh, PA.

Shavlik, J. and T. Eliassi-Rad: 1998b, 'Intelligent agents for web-based tasks: An advice-taking approach'. In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence: Workshop on Learning for Text Categorization*. Madison, WI, pp. 63–70.

Soderland, S.: 1999, 'Learning information extraction rules for semi-structured and free text'. *Machine Learning: Special Issue on Natural Language Learning* **34**(1/3), 233–272.

Sutton, R.: 1988, 'Learning to predict by the methods of temporal differences'. *Machine Learning* **3**, 9–44.

Sutton, R. S. and A. G. Barto: 1998, *Reinforcement Learning*. MIT Press.

Towell, G. G. and J. W. Shavlik: 1994, 'Knowledge-based artificial neural networks'. *Artificial Intelligence* **70**(1/2), 119–165.

Valiant, L.: 1984, 'A theory of the learnable'. *Communications of the ACM* **27**, 1134–1142.

van Rijsbergen, C. J.: 1979, *Information Retrieval*. London: Buttersworths, second edition.

Watkins, C.: 1989, 'Learning from delayed rewards'. Ph.D. thesis, King's College, Cambridge.

Vitae

1. Dr. Tina Eliassi-Rad:

   Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Box 808, L-560, Livermore, CA 94551, USA

   Dr. Tina Eliassi-Rad is a computer scientist in the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory. Dr. Eliassi-Rad earned her Ph.D. in Computer Sciences (with a minor in Mathematical Statistics) from the University of Wisconsin-Madison in 2001. She received her M.S. in Computer Science from the University of Illinois at Urbana-Champaign in 1995, and her B.S. in Computer Sciences from the University of Wisconsin-Madison in 1993. Dr. Eliassi-Rad's research interests include machine learning, knowledge discovery and data mining, artificial intelligence, text and web mining, information extraction, information retrieval, intelligent software agents.

2. Dr. Jude Shavlik:

   Computer Sciences Department, University of Wisconsin-Madison, 1210 W. Dayton Street, Madison, WI 53706, USA

   Dr. Jude Shavlik is a Professor of Computer Sciences and of Biostatistics and Medical Informatics at the University of Wisconsin - Madison. Dr. Shavlik has been at Wisconsin since 1988, following the receipt of his PhD from the University of Illinois for his work on Explanation-Based Learning. His current research interests include machine learning, computational biology, and intrusion detection. Dr. Shavlik recently completed a three-year term as editor-in-chief of the AI Magazine and serves on the editorial board of a half-dozen journals. He chaired the 1998 International Conference on Machine Learning Dr. Shavlik co-edited, with Tom Dietterich, "Readings in Machine Learning."