WebWatcher: A Tour Guide for the World Wide Web

Thorsten Joachims, Dayne Freitag, Tom Mitchell

September 1996
CMU-CS-96-xxx

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

This research is sponsored by the Wright Laboratory, Aeronautical Systems Center, Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant F33615-93-1-1330. The US Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation thereon. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of Wright Laboratory or the United States Government.

Keywords: Machine Learning, Intelligent Agents, Text Classification, World Wide Web

Abstract

We explore the notion of a tour guide software agent for assisting users browsing the world wide web. A web tour guide agent provides assistance similar to that provided by a human tour guide in a museum – it guides the user along an appropriate path through the collection, based on its knowledge of the user's interests, of the location and relevance of various items in the collection, and of the way in which others have interacted with the collection in the past. This paper describes a simple but operational tour guide, called WebWatcher, which has given over 5000 tours to people browsing CMU's School of Computer Science web pages. WebWatcher accompanies users from page to page, suggests appropriate hyperlinks, and learns from experience to improve its advice-giving skills.

1 Introduction

Browsing the World Wide Web is much like visiting a museum. In a museum the visitor has general areas of interest and wants to see relevant artifacts. But visitors find it difficult to locate relevant material given that they do not initially know the contents of the museum. And in many cases their initial interests are poorly defined, and become clear only after they begin to explore. In a museum the user might rely on a tour guide who is familiar with the museum and how people interact with it. The visitor could describe their general initial interests to the tour guide, who could then accompany the user, point out items of interest, and suggest which directions to turn next. During the tour the user could communicate with the guide, express interest in certain artifacts, ask and answer questions, as they explore and refine their interests.

A collection of interconnected (hyperlinked) web pages is analogous to a museum, and people browsing such collections often behave like museum goers. For example, a visitor to CMU's Computer Science web home page might have a general interest in "experimental research on intelligent agents." However, with no specific knowledge about the contents of the collection, the user may find it difficult to locate relevant information. Until they become aware that CMU conducts significant research on robotics, for example, they may not think to mention "learning robots" when describing their general interest in intelligent agents.

We report here research toward software agents that act as tour guides for the web. In its most general form, the metaphor of web agent as tour guide is very broad, suggesting systems that carry on general natural language dialogs with their users, possess detailed knowledge about the semantic content of the web pages they cover, and learn with experience. Here we describe a first experiment with a more restricted, but operational, tour guide. In particular, we describe WebWatcher [AFJM95], a system that accompanies the user as he or she browses the web. Like a museum tour guide, WebWatcher interactively suggests where to go next. The user can communicate with the system and give feedback. WebWatcher acts as a *learning apprentice* [MMS85, MCF+94], observing and learning from its users' actions. Over time WebWatcher learns to become more expert for the parts of the World Wide Web that it has visited in the past, and for the types of topics in which previous visitors have had an interest.

WebWatcher differs in several key respects from keyword-based search engines such as Lycos and AltaVista. First, such search engines require that the user describe their interest in terms of specific words that match those in the target web page. In contrast, WebWatcher can learn that a term such as "machine learning" matches a hyperlink such as "neural networks," even though these

phrases share no words in common. Furthermore, search engines do not take into account that documents are designed as hypertext and that links between documents and the structure of documents contain important information. In many cases only a sequence of pages and the knowledge about how they relate to each other can satisfy the user's information need.

In the following we present the design of WebWatcher, as well as experimental results obtained from over 5000 tours given by WebWatcher to various visitors on the web. We describe how WebWatcher learns from the experience gained from each tour it gives. WebWatcher uses this learned knowledge to improve the quality of advice it gives to subsequent users. We examine a variety of algorithms for this learning task, and present experimental results comparing their effectiveness. Finally, we summarize the lessons and perspectives gained from these first experiment with a tour guide agent for the web.

2 WebWatcher

The following trace illustrates a typical tour given by WebWatcher, from the front door web page of CMU's School of Computer Science.

2.1 Typical Scenario

Imagine the following scenario. We enter the School of Computer Science Web at the Front Door page (figure 1). On this page an instance of WebWatcher is installed which can be invoked by clicking on the hyperlink "The WebWatcher Tour Guide" in the bottom part of the page. Clicking on this hyperlink leads us to the "Welcome to WebWatcher" page (figure 2) where we are asked for a short description of our current interest. In this example, we enter "intelligent agents" as our interest, then click on the start button. WebWatcher now returns us to our initial page, prepared to guide our tour (figure 3). We are no longer browsing alone, but will be accompanied by WebWatcher on any sequence of hyperlinks we follow from this point forward.

We can see that WebWatcher accompanies us by the additions it makes to the original page. The original page augmented with WebWatcher's additions can be seen in figure 3. Notice the changes made by WebWatcher include:

- A menubar is inserted above the original page. Through the menubar we can communicate with WebWatcher by issuing commands or giving feedback.
- A list of new hyperlinks has been added to the page, just below the menu bar. These hyperlinks lead to pages that contain the keywords "intelligent agents" which we typed in. This list is generated using a variant of the

Welcome from the Dean

Index of people and projects

Departments

Computer Science Department
The Robotics Institute
Human–Computer Interaction Institute
Information Technology Center
Language Technology Institute
(and Center for Machine Translation)

Academic Programs

PhD in Computer Science
The 1996 Immigration Course schedule.
Master of Software Engineering
PhD in Robotics
Master of Human Computer Interaction
Ph.D. in Language and Information Technology
M.S. in Language Technology
Neural Processes in Cognition Program
BS in Computer Science
PhD in Pure & Applied Logic
Continuing Education for Professionals
PhD in Algorithms, Combinatorics, & Optimization

SCS Resources

Employment Opportunities
SCS Alumni
Technical Papers and Library
SCS Summer School
Web resources from CMU
The SCS local home page
What's new in the SCS Web
Research Computing Facility
The WebWatcher tour guide.
SCS seminars (cboard) and events
Map of the CMU campus

Figure 1: The original Front-Door page.

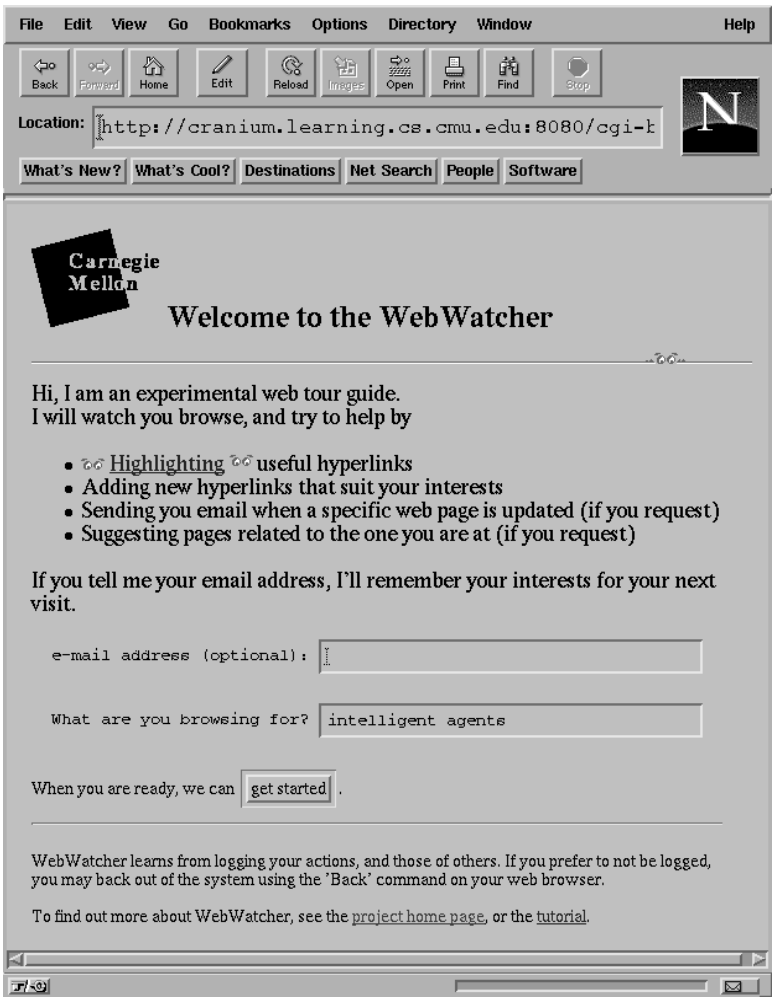

Figure 2: Entering WebWatcher.

Figure 3: The Front-Door page with WebWatcher's additions.

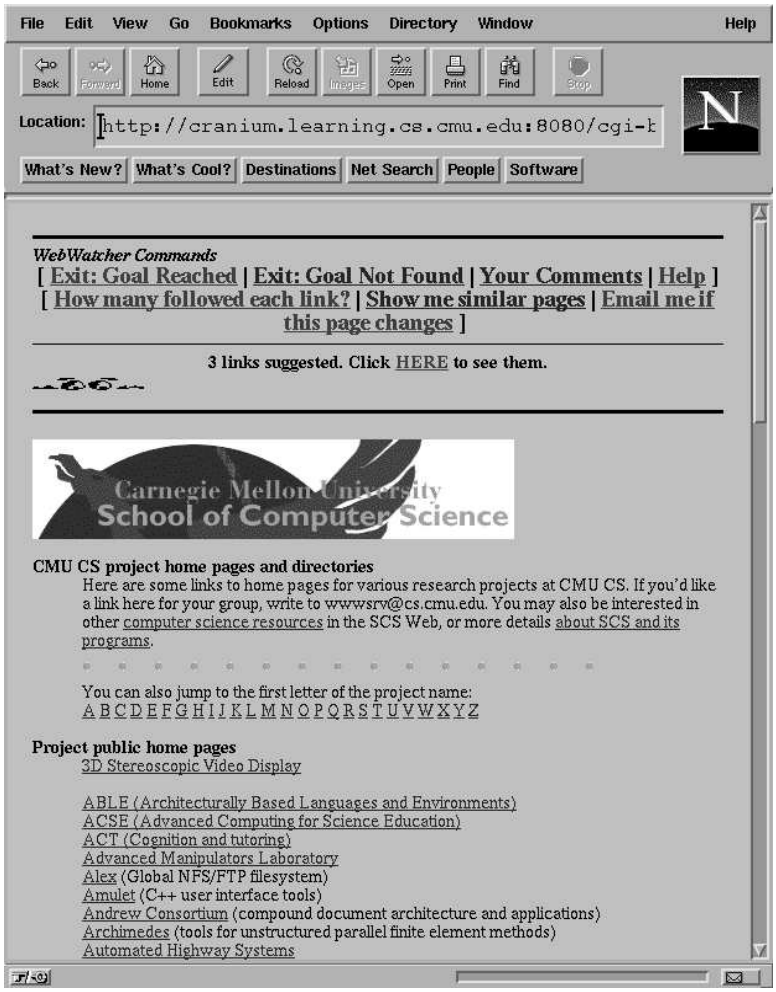

Figure 4: The list of project home pages.

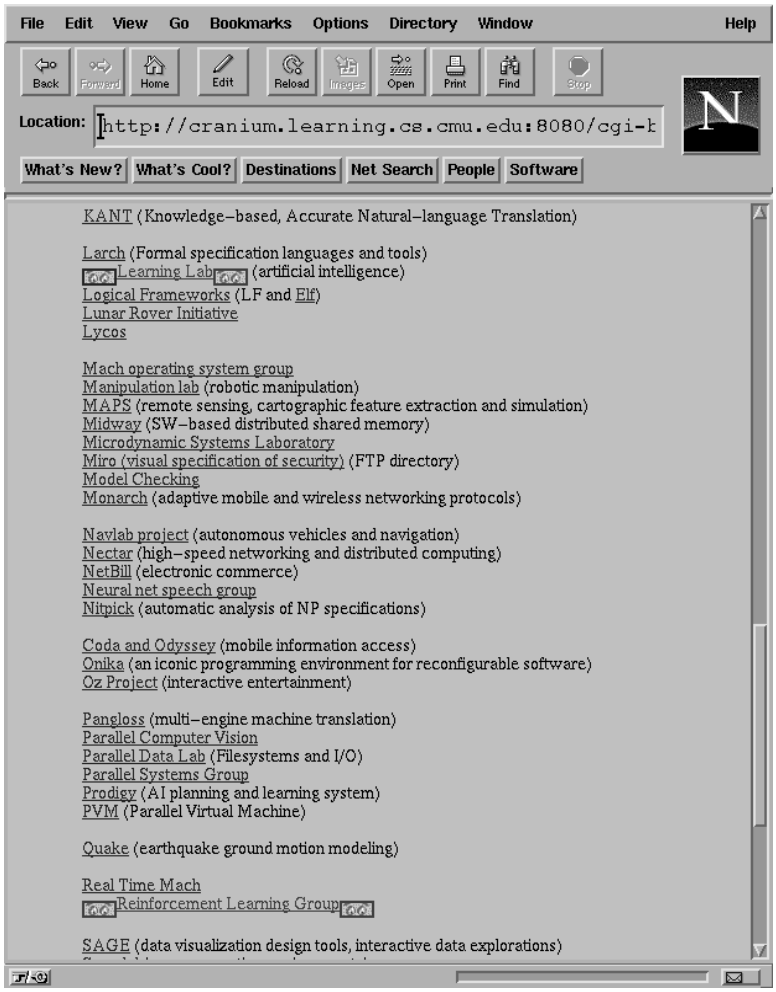

Figure 5: Further down in the list of project home pages.

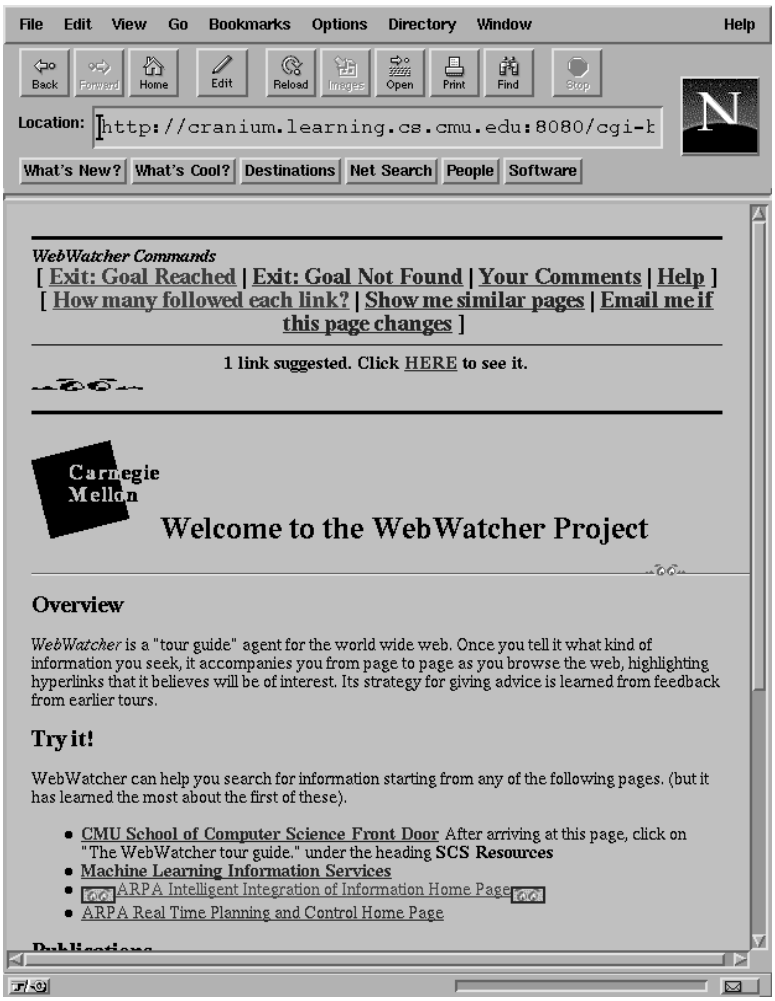

Figure 6: The user arrives at the home page of the WebWatcher project.

Lycos search engine, applied to the set of pages previously visited by WebWatcher.

- Selected hyperlinks from the original page have now been highlighted by WebWatcher, to suggest directions relevant to our browsing interests. WebWatcher highlights these hyperlinks by inserting eyeball icons ( ) around the hyperlink, as shown in the last hyperlink visible in figure 3. This is the most novel feature of WebWatcher. The advice it provides in this fashion is based on knowledge learned from previous tours.

Let us follow WebWatcher's suggestion (the highlighted hyperlink at the bottom of figure 3). In general, the user may click on any hyperlink, recommended or not. Each time the user selects a hyperlink, WebWatcher accompanies the user to the next page, and logs this hyperlink selection as a training example for learning to improve future advice.

When we follow the hyperlink suggested by WebWatcher in this case, we reach a page containing a long list of research project pages (figure 4). WebWatcher again inserts the menubar on top of the page indicating that the system is still accompanying us. On this page WebWatcher suggests three hyperlinks it judges relevant to our interest in "intelligent agents." Clicking on the "HERE" button on top of the page gets us directly to the first of WebWatcher's suggestions (figure 5). By clicking on the eyeball icons next to the suggested hyperlinks, we can jump from one suggestion to the next.

In this scenario we follow WebWatcher's third suggestion, which takes us to a page describing the WebWatcher research project. Again, WebWatcher gives new suggestions there.

If we were particularly interested in a page, we could tell WebWatcher so by clicking on "Show me similar pages" in the menubar. WebWatcher would then display a list of pages which are similar based on a metric derived from hypertext structure [JMFA95].

Two additional commands are available in WebWatcher's menubar on any page. Clicking on the command "how many followed each link?" asks WebWatcher to display for each hyperlink the number of previous visitors who took that link. Clicking on the command "Email me if this page changes" asks WebWatcher to periodically monitor the page and send the user email if it should change.

WebWatcher accompanies the user in this fashion along any hyperlink anywhere on the world wide web. To end the tour, the user clicks on one of two commands in the menu bar: "Exit: Goal reached" or "Exit: Goal not reached."

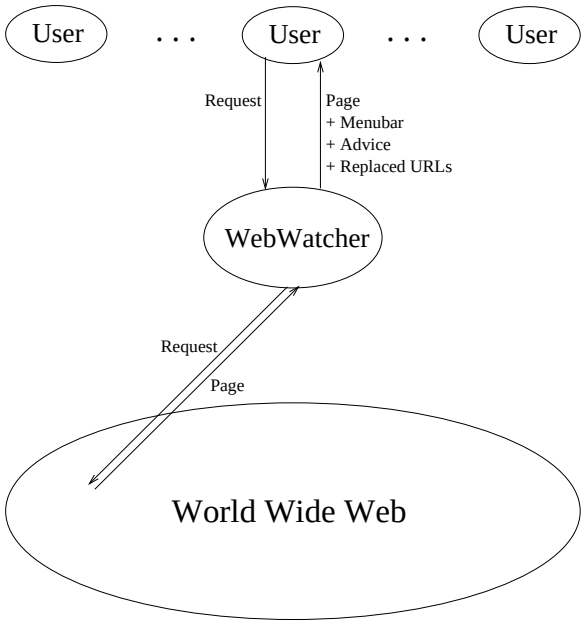

Figure 7: WebWatcher is an interface agent between the user and the World Wide Web.

This exit provides the user with a way of giving final feedback to WebWatcher about the success of the tour in locating information relevant to the stated interests.

2.2 How Does WebWatcher Accompany the User?

From the perspective of WebWatcher, which is implemented as a server on a separate workstation on the network, the above scenario looks somewhat different. When first invoked (i.e., when the user clicks on the WebWatcher hyperlink in figure 1), the WebWatcher server accepts an argument, encoded in the URL that accesses it, which contains a "return address." This return address is the URL of the web page from which the user came. Once the user fills out the form specifying his or her interests, WebWatcher sends the user back to an modified copy of this page. It makes three modifications:

1. First, the WebWatcher menubar is added to the top of the page.
2. Second, each hyperlink URL in the original page is replaced by a new URL that points back to WebWatcher.

3. Third, if WebWatcher finds that any of the hyperlinks on this page are strongly recommended by its search control knowledge, then it highlights the most promising links in order to suggest them to the user.

It sends this modified copy of the web page to the user, and opens a file to begin logging this user's trace as training data. While it waits for the user's next step, it prefetches any web pages it has just recommended to the user to minimize network delays. When the user clicks on a new hyperlink, WebWatcher updates the log for this search, retrieves the page (unless it has already been prefetched), performs similar substitutions, and returns the copy to the user.

This process continues, with WebWatcher tracking the user's tour across the Web, providing advice at each step, until the user elects to dismiss the agent. At this point, WebWatcher closes the log file for this session (indicating either success or failure in the search, depending on which button the user selected when dismissing WebWatcher), and returns the user to the original, unmodified copy of the web page he is currently at. For each such tour, the information logged by WebWatcher summarizes the sequence of web pages and hyperlinks followed by the user, along with a time stamp indicating exactly when each hyperlink was requested.

2.3 What Gets Learned

What is the form of the knowledge required by WebWatcher? In general, its task is to suggest an appropriate link given the current user, interest, and web page. Hence, one general form of knowledge that would be useful corresponds to knowledge of the function $LinkUtility$:

$$LinkUtility : Page \times Interest \times User \times Link \rightarrow [0, 1]$$

where $Page$ is the current web page, $Interest$ is the set of words describing the user's interest, $User$ is the identity of the user, and $Link$ is one of the hyperlinks found on $Page$. The value of $LinkUtility$ is the probability that following $Link$ from $Page$ leads along a shortest path to a page that satisfies the current $Interest$ for the current $User$.

In the learning experiments reported here, we consider learning a simpler function for which training data is more readily available, and which is still of considerable practical use. This function is $UserChoice?$:

$$UserChoice? : Page \times Interest \times Link \rightarrow [0, 1]$$

where the value of $UserChoice?$ is the probability that an arbitrary user will select $Link$ given the current $Page$ and $Interest$. Notice here the $User$ is not

an explicit input, and the function value predicts only whether users tend to select $Link$ – not whether it leads optimally toward pages which satisfy the user's interest. Notice also that information about the search trajectory by which the user arrived at the current page is not considered in this formulation of the target function to be learned.

One reason for focusing on $UserChoice?$ in our initial experiments is that the data automatically logged by WebWatcher provides training examples of this function. In particular, each time the user selects a new hyperlink, a training example is logged for each hyperlink on the current page, corresponding to the $Page$, $Interest$, $Link$, and whether the user chose this $Link$.

To improve the accuracy of learning the function $UserChoice?$, WebWatcher splits up this function so that it learns a separate function for each web page. In other words, for every page p WebWatcher learns a function $UserChoice?_p$

$$UserChoice?_p : Interest \times Link \rightarrow [0, 1]$$

2.4 Learning Method

Learning is accomplished by annotating each hyperlink with the interests of the users who took this hyperlink on tours with positive feedback (i.e., tours ending with "Exit: Goal reached"). Thus, whenever a user follows a hyperlink on a successful tour the description of this hyperlink is augmented with the keywords the user typed in at the beginning of the tour. The initial description of a hyperlink is the underlined text. Figure 8 illustrates how each hyperlink has a list of interests associated with it. Appendix A shows the description of a particular hyperlink as learned by WebWatcher.

To suggest hyperlinks during a tour WebWatcher compares the current user's interest with the descriptions of all hyperlinks in the current page. WebWatcher suggests those hyperlinks which have a description sufficiently similar to the user's interest. Figure 8 gives an example.

The metric used to compute similarity between an interest of a user and the description of a hyperlink is based on a technique from information retrieval. Interests and hyperlink descriptions are represented by very high-dimensional feature vectors (figure 9), with each dimension representing a particular word in the English language. The weight of a word w for a piece of text d is computed using the TFIDF heuristic as described in [Sal91].

$$TFIDF(w, d) = TF(w, d) * \log(\frac{n}{DF(w)})$$

The *term-frequency* $TF(w, d)$ counts the number of times word w occurs in text d. The *document-frequency* $DF(w)$ is the number of texts that contain word w

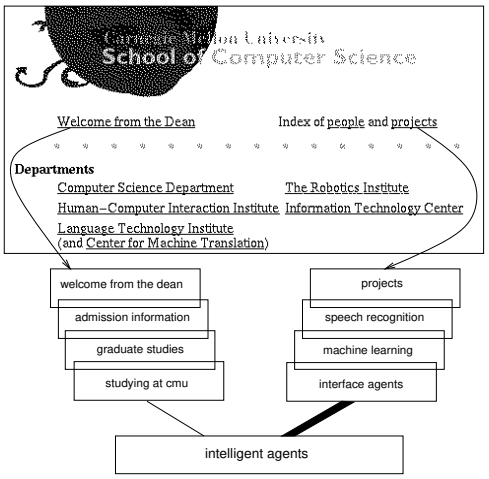

Figure 8: Whenever a user follows a hyperlink, the link is annotated with the user's interest keywords. Here, the interest of a new user in "intelligent agents" matches the "projects" hyperlink better than the "Welcome from the Dean" hyperlink, because of the keywords associated with this hyperlink from previous tours.

and n is the total number of texts. The TFIDF measure assigns a word a higher weight the more often it occurs in a piece of text. In addition, words that occur more frequently throughout all texts receive a lower weight. In particular the weight of a word is zero if it does not occur in a piece of text at all. Based on this vector representation the similarity of two pieces of text can be calculated as the cosine between their vectors [Sal91].

The particular algorithm WebWatcher uses to suggest hyperlinks goes through all the hyperlinks on the current page. For each hyperlink the list of interests (including the original underlined words) associated with it is ranked according to the similarity with the current user's interest. The value of $UserChoice?$ for each hyperlink is estimated as the average of the similarities of the k (usually 5) highest ranked keyword sets. A hyperlink is suggested if its value for $UserChoice?$ is above a threshold. The maximum number of hyperlinks suggested on a page is three.

2.5 Results

We present results based on 5822 tours given by WebWatcher between August 2, 1995, and March 4, 1996, starting at the SCS-FrontDoor page. About 55% of the tours were more than one step long, 18% are more than five steps long. The longest tour consists of 135 steps. In 25% of the searches the user did not provide an interest.

As shown in figure 10, in 20.8% of the pages WebWatcher visited during

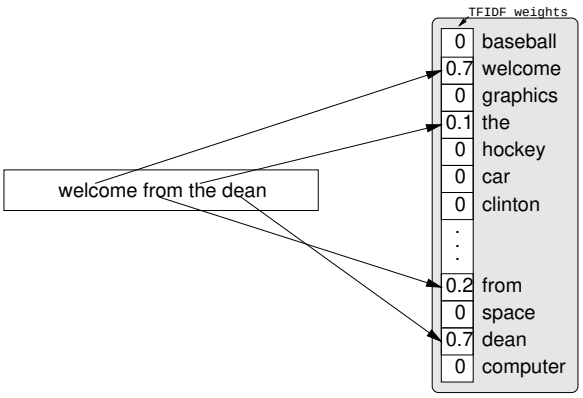

Figure 9: Pieces of text are represented as high dimensional vectors.

	Advice given	Accuracy
Random	20.8%	15.3%
Learning	20.8%	43.9%

Figure 10: WebWatcher's performance over about 5800 tours.

those tours, it highlighted at least one hyperlink. In 43.9% of these cases the user followed the advice given by WebWatcher (excluding cases where the user exited the system or backed out without following any hyperlink). We define this percentage to be the accuracy of WebWatcher's suggestions. To give a lower bound on the accuracy for comparison, WebWatcher would have achieved 15.3% if it highlighted an equivalent number of hyperlinks at random on each of these pages. Note that this accuracy measurement is biased by the fact that WebWatcher's suggestions influenced the decisions of users.

2.6 Comparison with Human Experts

To get a better feel for the nature of the task of suggesting hyperlinks and to evaluate WebWatcher's performance, we conducted an experiment in which we asked humans to do WebWatcher's task. To make the experiment more tractable we focused on one page, namely an earlier version of the SCS-Front-Door page shown in figure 1. From the 408 training examples for this page we took 8 random subsets of 15 examples and presented them to eight people. For each example they were asked to suggest the 3 hyperlinks the user was most

	Advice given	Accuracy
Random	100%	22.4%
Learning	100%	42.9%
Human	100%	47.5%

Figure 11: A comparison to human performance on the SCS-Front-Door page.

likely to follow out of the 18 hyperlinks on this page. The test subjects were all familiar with the page and its locale.

As figure 11 summarizes, the human performance was at 47.5%. The learning algorithm achieved an accuracy of 42.9% under the same conditions. The 22.4% for random suggestion again provides a baseline.

3 Additional Learning Methods

Can WebWatcher's suggestion accuracy be increased? Are there other sources of information that can help WebWatcher improve at its task? This section explores two ideas for improving the accuracy of WebWatcher's advice. The first idea is to take the hypertext web structure into account in order to create more refined descriptions of hyperlinks. We adopt an approach based on reinforcement learning to create these descriptions. The second idea is to combine the results of several learning methods to improve accuracy.

3.1 Reinforcement Learning on the Web

As described above the initial description of a hyperlink is only the underlined anchor text. The approach described here elaborates these sparse initial descriptions. The basic idea is to associate with the hyperlink a weighted word vector estimating which words occur on pages downstream of the hyperlink. The weight given to a downstream word is its TFIDF value discounted exponentially with its distance from the page under consideration (i.e., the number of hyperlinks that must be traversed to reach it). This approach is based on reinforcement learning, as described below.

3.1.1 Reinforcement Learning

Reinforcement learning is a framework allowing agents to learn control strategies that select optimal actions in certain settings. Consider an agent navigating

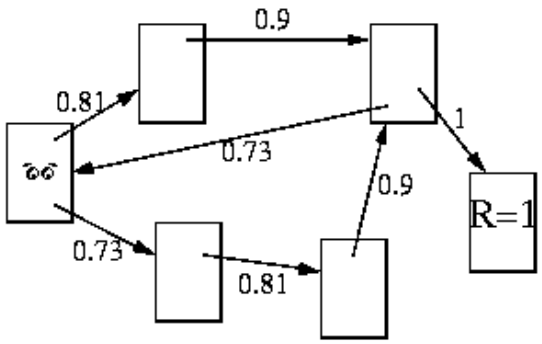

Figure 12: Example state space.

from state to state by performing actions. At each state s the agent receives a certain reward $R(s)$. The goodness of an action a can be expressed in terms of an evaluation function $Q(s, a)$, defined for all possibe state-action pairs. The value of $Q(s, a)$ is the discounted sum of future rewards that will be obtained if the agent performs action a in state s and subsequently chooses optimal actions. If the agent can learn this function, then it will know how to act in any state. More precisely,

$$Q(s_t, a) = \sum_{i=0}^{\infty} \gamma^i \cdot R(s_{t+1+i})$$

where s_t is the state the agent is in at time t, and where γ is a discount factor $0 < \gamma < 1$ that determines how severely to discount the value of rewards received further into the future. Under certain conditions, the Q function can be iteratively approximated by updating the estimate for $Q(s, a)$ each time action a is performed by the agent in state s, as follows (see [Bel57]):

$$Q_{n+1}(s, a) = R(s') + \gamma \max_{a' \in actions_in_s'} [Q_n(s', a')]$$

where s' is the state resulting from performing action a in state s. Once $Q(s, a)$ is known, the optimal control strategy for the agent is simply to pick the action that maximizes Q for whatever state it finds itself in.

Figure 12 gives an example. Boxes represent possible states of the agent. The edges represent actions that bring the agent from one state to another. The edges are annotated with values of the function $Q(s, a)$. In the rightmost state the agent receives a reward of 1. The reward is 0 in all other states. If the agent always follows the action with the highest Q value, it will get to the reward

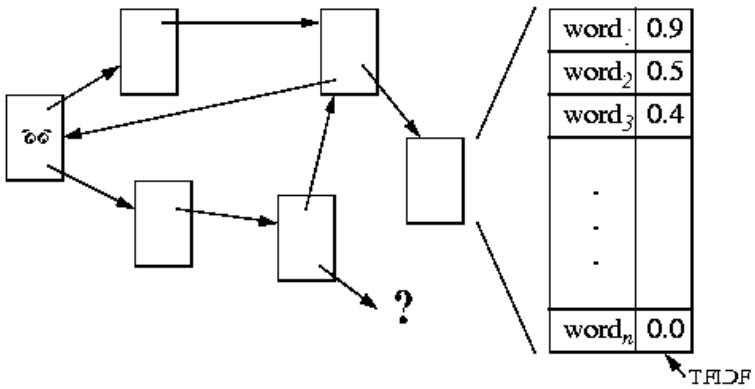

Figure 13: Having a value function for every word. The reward for a word on a page is its TFIDF value.

state in the smallest number of steps and thus maximize the discounted reward it receives.

3.1.2 Reinforcement Learning for Navigating Hypertext

An analogous navigation problem for an agent on the web is to navigate from page to page by taking hyperlinks to get to good sources of information. Pages correspond to states and hyperlinks correspond to actions. The reward a web agent receives for a page is the degree to which this page fits the user's interests. An agent using Q-learning will therefore seek tours through the web that lead as directly as possible to pages of greatest interest to the user.

How shall we measure whether a page fits the user's interest? The particular reward function used here is again based on the vector space retrieval model and the TFIDF heuristic [Sal91] already introduced in section 2.4. Pages are represented as vectors of words and the vector length is normalized to one. The values of the vector give an estimate of how important the corresponding word is on the page. For each word w that is encountered on any of the pages in the training data a separate Q-function $Q_w(s, a)$ is learned. Given the reward function as described above, the Q-function for a particular word is trained to maximize the discounted TFIDF values for that word when following a sequence of hyperlinks (figure 13).

To suggest hyperlinks at runtime, the system can use the Q-functions it learned for each word. If the user's interest consists only of a single word w, the

corresponding Q-function $Q_w(s,a)$ is used and those hyperlinks are suggested for which the Q-function plus an immediate reward, given w occurs in the anchor text, is maximum. If the user's interest consists of multiple words $w_1, \ldots, w_n$, the suggested hyperlinks are those that maximize the sum of the corresponding Q-functions plus the immediate rewards.

Because WebWatcher cannot expect that users will always stick to pages it has already seen, a core question in implementing this approach is how to learn a general approximation for each of the Q-functions $Q_w(s,a)$. The system must be able to estimate the significance of novel states and actions in terms of states and actions (pages and hyperlinks) it already knows. We chose to use a similarity-weighted 3-nearest neighbor function approximator for this purpose, because of the many features needed to describe pages and hyperlinks, and because of certain theoretical advantages [Gor95]. Each hyperlink a is described by the TFIDF vector representation of the underlined anchor text. Each page s is represented in an analogous way using the text in its title. For purposes of determining similarity, the distance between the hyperlink a_1 on page s_1 and the hyperlink a_2 on page s_2 is defined to be the distance between a_1 and a_2, plus twice the distance between s_1 and s_2. The distance between two vectors is defined to be the cosine of the angle between the vectors, in keeping with the standard similarity measure used in TFIDF.

3.1.3 Experimental Setup

The experiments presented in this section were conducted using a subset of the traces starting at the SCS-FrontDoor page WebWatcher collected from August 2, 1995, to March 4, 1996. We used only tours where the user typed in an interest and where the tour was at least four steps long. There were 1777 such tours. Five different test-training splits were used with 2/3rd of the traces being used for training. Of the remaining example traces, 90% were used for testing and 10% were used for purposes that are described in section 3.2.

In addition to the Q-learning approach, the results for four other methods are presented.

- RANDOM. This strategy suggests hyperlinks at random from the current page.
- POPULARITY. This method simply counts the frequency with which each hyperlink on a page has been followed in the past. The most frequently followed hyperlinks are suggested in the future. The information about the user's interest is ignored.
- ANNOTATE. This is the strategy introduced in section 2.4. In the experiments described in this section all tours were used independent of whether they were marked successful or not.

	Accuracy (on all pages)	Accuracy (on known pages)	Accuracy (on unknown pages)
Random	31.6%	23.8%	39.7%
Popularity	41.8%	43.7%	39.7%
Match	40.3%	33.8%	46.0%
Annotate	42.0%	37.0%	46.0%
Q	44.5%	41.5%	47.8%

Figure 14: Results for Q-Learning and several other learning methods (all results are averages over five different test training splits).

- MATCH. This method is similar to "ANNOTATE", with the exception that links are not annotated with user interests. Instead, only the underlined anchor text is used to describe the hyperlink.

3.1.4 Experimental Results

The results in figure 14 provide a comparison of several learning methods. Each example corresponds to a page, from which each learner was allowed to choose three hyperlinks. Accuracy measures the percentage of times in which the user followed one of the chosen hyperlinks. We required each learning algorithm to make a prediction regardless of how confident it was.

The first column of figure 14 shows the accuracy on all pages in the test set. Q-learning performs significantly better than the other methods. The second and third columns show the accuracy over known and unknown pages (pages that do or do not appear in tours in the training set). Surprisingly, the accuracy on unknown pages is higher than on known pages. This is due to the fact that those pages happened to contain on average fewer hyperlinks, making it easier for random guesses to be correct.

3.2 Combining Multiple Learning Algorithms

Four methods were presented which were able to suggest hyperlinks with reasonable accuracy. Can those learning methods be combined to achieve better results?

3.2.1 Method

In section 2.3 we introduced the function $UserChoice?$. This is the target function WebWatcher is designed to learn. The value of $UserChoice?$ is ideally the probability $\Pr(choice|h, i, p)$ that a user with interest i will follow hyperlink h

	Accuracy	Accuracy (known)	Accuracy (unknown)
Random	31.6% (.7)	23.8%	39.7%
Popularity	41.8% (.7)	43.7%	39.7%
Match	40.3% (.7)	33.8%	46.0%
Annotate	42.0% (.9)	37.0%	46.0%
Q	44.5% (.4)	41.5%	47.8%
Combine	49.0% (.6)	50.0%	47.9%

Figure 15: Results for combining learning methods. Reported accuracies are aaverages over five train/test splits. Standard errors are given in parentheses.

on page p. Can we estimate this probability and use it for decision making?

For each hyperlink h we know the predictions of the learning methods $Annotate(h,i)$, $Match(h,i)$, $Q(h,i)$, and $Popularity(h,p)$. These predictions can be used to estimate the probability

$$\Pr(choice|Annotate(h,i), Match(h,i), Q(h,i), Popularity(h,p))$$

The output values of $Annotate(h,i)$ and $Match(h,i)$ for hyperlink h and interest i are each cosine distances. For $Q(h,i)$ it is the sum of the Q-values divided by the number of words in i. The output of $Popularity(h,p)$ is an estimate of the probability $\Pr(choice|h,p)$ that a user follows hyperlink h on page p independent of a particular interest. To estimate $\Pr(choice|h,p)$ a Bayesian estimator is used with uniform prior [Vap82] (often called Laplace estimator). Applied to our problem this estimator can be written as:

$$\hat{\Pr}(choice|Popularity(h)) = \frac{\#of_times_h_was_chosen + 1}{\#of_examples + \#of_hyperlinks_on_page}$$

Note that before observing any training examples this estimator assigns equal probability to any hyperlink on the page.

To estimate $\Pr(choice|Annotate(h,i), Match(h,i), Q(h,i), Popularity(h,p))$ logistic regression [Mit96] is used. An additional small amount of data is used as training data for regression (see section 3.1.3).

3.2.2 Experimental Results

The table in figure 15 shows the results of combining the predictions of all four learning algorithms. The combined method achieves an accuracy of 49.0%, outperforming each of the individual methods. Our conjecture is that this is due to the diversity of the methods and sources of information that are combined.

Popularity uses frequency information derived from user behavior an a particular page, Match uses the underlined text in hyperlinks, Annotate uses the interest descriptions from the user traces and Q-Learning makes use of the hypertext structure.

4 Related Work

Letizia [Lie95] is similar to WebWatcher in the sense that the system accompanies the user while browsing. One difference is that the system serves only one particular user. Letizia is located on the users machine and learns his or her current interest. By doing lookahead search Letizia can recommend pages in the neighborhood of where the user is currently browsing.

Syskill and Webert [PMB96] offers a more restricted way of browsing than WebWatcher and Letizia. Starting from a manually constructed index page for a particular topic, the user can rate hyperlinks off this page. The system uses the ratings to learn a user specific topic profile that can be used to suggest unexplored hyperlinks on the page. Syskill and Webert can also use search engines like LYCOS to retrieve pages by turning the topic profile into a query.

Lira [BS95] works in an offline setting. A general model of one user's interest is learned by asking the user to rate pages. Lira uses the model to browse the web offline and returns a the set of pages that match the user's interest.

5 Summary and Future Research

WebWatcher provides one case study of a tour guide agent for the world wide web. By "tour guide agent" we mean any agent that accompanies users from page to page, providing assistance based on a partial understanding of that user's interests and of the content of the web pages. WebWatcher provides one operational example of a tour guide agent that has served several thousand people browsing CMU's School of Computer Science web pages. WebWatcher is characterized by the following properties:

- WebWatcher provides several types of assistance. Most importantly, it highlights interesting hyperlinks as it accompanies the user, pointing out those hyperlinks it expects will be most appropriate given the user's stated interest. In addition, it provides a keyword search over all pages it has previously visited and offers user commands such as "how many previous users followed each hyperlink on this page?" "show me pages similar to the one I am at," and "email me if this page is updated."
- WebWatcher learns from experience. Each tour given by WebWatcher provides training data consisting of some topic of interest to some user,

a trajectory taken by that user through the graph of web pages, and a final statement by the user indicating whether the tour succeeded in locating appropriate information. From each such trace, WebWatcher can derive multiple training examples. WebWatcher generalizes from such training examples to improve its ability to suggest appropriate hyperlinks for subsequent users. The most effective learning method we found in our experiments was a multi-strategy approach that rates the significance of a hyperlink based on a combination of factors: the frequency with which the hyperlink was taken in the past, the stated interests of previous visitors who took this hyperlink, the words occuring on pages downstream from this hyperlink, and the underlined words associated directly with the hyperlink.

- WebWatcher is implemented as a Web server on a workstation distinct from the user's workstation, and is capable of assisting any web user running any type of browsing software, tracking them to any page anywhere on the world wide web. Because it is a centralized system (as opposed to software running on the workstations of each individual user), it can collect and combine training data from thousands of different users to learn to improve the advice it provides to subsequent users.

Our experience with WebWatcher leads us to believe that self-improving tour guide agents will play an important role on the web in the future. WebWatcher demonstrates that it is possible for such an agent to provide helpful advice to many users, and that it is possible to automatically learn from the thousands of users with whom it interacts. Given that popular web sites are typically visited by many thousands of users, and that the content of the web changes frequently, it appears that machine learning will play a crucial role in future tour guide agents.

Our experience with WebWatcher has also shown that despite its ability to help some users, its highlighted hyperlinks are followed by users in only 48% of all cases. Interestingly, when we assigned expert humans the same task, they could do no better. Examining many specific tours given by WebWatcher, we find the following partial explanation for this disappointing level of accuracy:

- Users tend to have a fairly short attention span. The traces of many users show that even though they initially state an interest in some technical subject such as "algorithmic complexity," once they notice that there is an online coke machine their path veers in a new direction.
- There is great diversity in the interests of users browsing CMU's SCS Front Door, ranging from technical interests such as "machine learning" to non-technical interests such as "windsurfing." Even after thousands of tours it is not uncommon for the next user to express an interest that WebWatcher has not yet encountered.

Together, these two factors suggest that WebWatcher might achieve higher accuracy if the web locale it had to cover had a narrower, more focused scope. For example, WebWatcher could be installed for a collection of web pages describing online documentation for some software library, or describing the products of some particular vendor. We are currently exploring the use of WebWatcher for giving tours begining at a web page describing research on machine learning[1]. More generally, our experience with WebWatcher suggests a number of topics for future research:

- Personalized WebWatcher. Whereas WebWatcher learns to specialize to a specific web locale, one could instead develop tour guide agents that learn to specialize to a particular user. Such an agent could learn a model of the longer-term interests of users by observing which pages they do and do not visit, then use this learned interest model to provide future advice. We have recently begun experiments with such a personalized WebWatcher. One system that demonstrates the feasibility of learning such long-term user interest models is the NewsWeeder system [Lang95], which learns user interests in reading net news. One example of a person-specific web agent is Letizia [Lie95].

- Combining user-specific and web locale-specific learning. Note that one way to implement a personalized WebWatcher is to simply adapt the current system to maintain a separate set of training data for each user. Because it serves multiple users, it could in principle learn user-specific interests at the same time that it retains the capability to annotate hyperlinks based on tours given to many users. In this context, one could explore a variety of methods for combining the benefits of single-user modeling and learning simultaneously from/about multiple users.

- Richer dialogs with users. One major shortcoming of the current WebWatcher is that it allows the user only to express a few keywords, and only at the beginning of the tour. A more flexible approach would involve an ongoing dialog with the user, much more like that a museum visitor might have with a human guide.

- New machine learning algorithms for characterizing hyperlinks. The learning methods used by WebWatcher succeed in improving its performance over time. However, a large space of possible learning methods for this problem remains unexplored. For example, all the methods reported in this paper are based on "bag of words" representations of pages and hyperlinks in which no syntactic or semantic analysis of the text is performed. One interesting research direction would be to explore learning methods that combine linguistic analysis with the statistical techniques described here.

[1] `http://www.ai.univie.ac.at/oefai/ml/ml-resources.html`

- Intelligent distributed hyperlinks. WebWatcher learns by associating new information with hyperlinks based on its experience. Note this learning could be performed in a much more distributed fashion, with each hyperlink separately building up its own model of itself and making recommendations to the user. Due to its distributed nature, this design would scale well even to very large sets of pages. The idea of intelligent hyperlinks learning in a distributed fashion could also be implemented directly by a web server. Currently hyperlinks consist only of an anchor text and the URL they point to. They could be extended by more semantic information. This information could be assigned manually when designing the web page. More interestingly, as WebWatcher demonstrates part of this knowledge could be learned automatically by observing user behavior. Knowing more about the semantics of hyperlinks, an intelligent server could then adapt to particular users. Given information about the general interests of a user, which could be sent along with the request for a page, the server might then give a personalized view of the information.

6 Acknowledgements

This research is supported by a Rotary International fellowship grant, an NSF graduate fellowship, and by Arpa under grant number F33615-93-1-1330.

References

[AFJM95] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. In *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, March 1995.

[Bel57] Richard Bellman. *Dynamic Programming.* Princeton University Press, 1957.

[BS95] M. Balabanovic and Y. Shoham. Learning information retrieval agents: Experiments with automated web browsing. In *AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments, Working Notes.* AAAI-Press, 1995.

[Gor95] G. Gordon. Stable function approximation in dynamic programming. In *International Conference on Machine Learning*, 1995.

[JMFA95] T. Joachims, T. Mitchell, D. Freitag, and R. Armstrong. Webwatcher: Machine learning and hypertext. In K. Morik and J. Herrmann, editors, *GI Fachgruppentreffen Maschinelles Lernen.* University of Dortmund, August 1995.

[Lie95] H. Lieberman. Letizia: An agent that assists web browsing. In *International Joint Conference on Artificial Intelligence, Montreal*, August 1995.

[MCF+94] T. Mitchell, R. Caruana, D. Freitag, J. McDermott, and D. Zabowski. Experience with a learning personal assistent. *Communications of the ACM*, 37(7):81–91, July 1994.

[Mit96] T. Mitchell. *Machine Learning.* McGraw-Hill, 1996.

[MMS85] T. Mitchell, S. Mahadevan, and L. Steinberg. Leap: A learning apprentice for vlsi design. In *Ninth International Joint Conference on Artificial Intelligence*, August 1985.

[PMB96] M. Pazzani, J. Muramatsu, and D. Billsus. Syskill & webert: Identifying interesting web sites. In *AAAI Conference, Portland*, August 1996.

[Sal91] G. Salton. Developments in automatic text retrieval. *Science*, 253:974–979, 1991.

[Vap82] V. Vapnik. *Estimation of Dependencies Based on Empirical Data.* Springer-Verlag, 1982.

A Annotations for link *projects* on SCS-FrontDoor page

Annotations for link *projects* on SCS-FrontDoor page in figure 1. All the interests of users going through this link are shown independent of feedback given.

projects, natural language systems, reinforcement learning, study compiler computer theory, warez, intelligent tutoring systems, speech recognition, machine learning, information about real time mach, machine learning at cmu, apple iigs stuff, admission information intelligent agents, intelligent tutoring, robotics, machine learning, geo, newsweeder, ai, information on a life or ai, andrew, electronic documents, tom mitchell, informedia multimedia, architecture operating systems distributed systems, browser, reinforcement learning, project listen, interesting computer graphics, text learning, ai, browsers, reinforcement learning, reinforcement learning, fast program, human computer interfaces, mathematica, graduate school, network management, projects, robotics, inex, robotics, australia, robotics, robot, punch recipes, 70703 161 compuserve com, 72457 1460 compuserve com, credit based, callback, lamborghini, xavier, information retrieval knowledge representation, information retrieval, artificial intelligence and music, technology info, recipes