WebWatcher: A Tour Guide for the World Wide Web

Thorsten Joachims
Universität Dortmund
Informatik-LS8
Baroper Str. 301
44221 Dortmund, Germany

Dayne Freitag
Carnegie Mellon University
School of Computer Science
5000 Forbes Avenue
Pittsburgh, PA 15213, USA

Tom Mitchell
Carnegie Mellon University
School of Computer Science
5000 Forbes Avenue
Pittsburgh, PA 15213, USA

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. We describe the learning algorithms used by WebWatcher, experimental results showing their effectiveness, and lessons learned from this case study in Web tour guide agents.

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. In many cases their initial interests are poorly defined, becoming 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 his or her 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 visitor could communicate with the guide, express interest in certain artifacts, ask and answer questions as they explore, and refine their interests.

People browsing collections of Web pages 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.

Here, we report research into 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 [Armstrong *et al.*, 1995], 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* [Mitchell *et al.*, 1994], observing and learning from its users' actions. Over time WebWatcher learns to acquire greater expertise 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" or "Avrim Blum," even though these phrases share no words in common. Furthermore, current search engines do not take into account that documents are designed as hypertext. 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

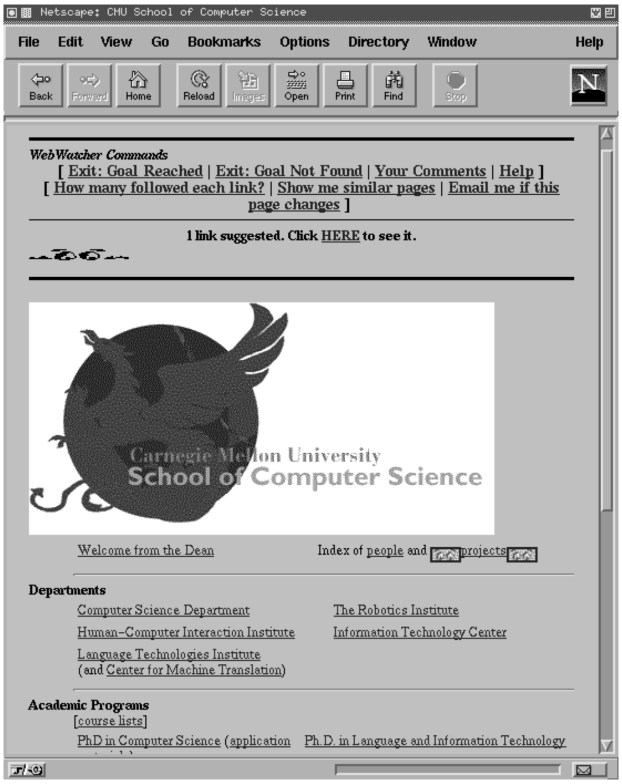

Figure 1: The Front-Door page with WebWatcher's additions.

Web. We describe how WebWatcher learns to improve the quality of its advice, and present experimental results comparing different learning methods. Finally, we summarize the lessons and perspectives gained from these first experiments with a tour guide agent for the Web.

2 Trace of WebWatcher

WebWatcher was in operation from August, 1995, to February, 1997. A typical WebWatcher tour proceeded as follows. We enter the CMU School of Computer Science Web at the front door page. On this page an instance of WebWatcher can be invoked by clicking on the hyperlink "The WebWatcher Tour Guide". Clicking on this hyperlink leads us to a page where we are asked for a short description of our current interest. In this example, we enter the phrase "intelligent agents" as our interest. WebWatcher now returns us to the initial page, prepared to guide our tour (figure 1). 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 tell that WebWatcher accompanies us from the additions it makes to the original page. A page augmented with WebWatcher's additions is shown in figure 1. Among others they include:

- A list of *WebWatcher Commands* is inserted above the original page. The user can invoke these commands to communicate with WebWatcher as he browses from page to page.
- 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 "projects" link in figure 1 (recall in this example the user expressed an interest in "intelligent agents"). The advice WebWatcher provides in this fashion is based on knowledge learned from previous tours.

If we follow WebWatcher's suggestion in figure 1, we reach a page containing a long list of research project pages. WebWatcher again inserts the command list on top of the page and suggests three hyperlinks it judges relevant to our interest in "intelligent agents."

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.

In addition to highlighting hyperlinks WebWatcher also provides other forms of assistance. In particular, it provides a keyword search using a variant of the Lycos search engine applied to the set of pages previously visited by WebWatcher. It also provides a user command "Show me similar pages" in the command list, which causes WebWatcher to display a list of pages which are similar based on a metric derived from hypertext structure [Joachims *et al.*, 1995]. 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. The command "Email me if this page changes" tells WebWatcher to periodically monitor the current page and send the user email if it changes.

WebWatcher accompanies the user along any hyperlink anywhere on the World Wide Web. To end the tour, the user clicks on one of two options in the command list: "Exit: Goal reached" or "Exit: Goal not found." This exit provides the user with a way of giving final feedback to WebWatcher.

3 Accompanying the User

WebWatcher is implemented as a server on a separate workstation on the network and acts much like a proxy. Before returning a page to the user it makes three modifications:

1. The WebWatcher command list is added to the top of the page.
2. Each hyperlink URL in the original page is replaced by a new URL that points back to the WebWatcher server.
3. 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.

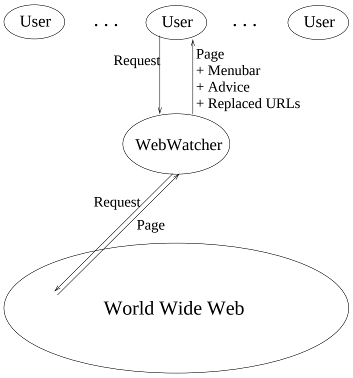

Figure 2: WebWatcher is an interface agent between the user and the World Wide Web.

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. Figure 2 depicts one cycle of user interaction. 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 until the user elects to dismiss the agent.

4 Learning in WebWatcher

What is the form of the knowledge required by WebWatcher? In general, its task is to suggest an appropriate link given an interest and Web page. In other words, it requires knowledge of the following target function:

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

The value of $LinkQuality$ is interpreted as the probability that a user will select $Link$ given the current $Page$ and $Interest$. In the following we present three approaches to learning this target function from experience. The first approach uses previously given tours as a source of information to augment the internal representation of each selected hyperlink. The second approach is based on reinforcement learning. The idea is to find tours through the Web so that the amount of relevant information encountered over the trajectory is maximized. The third approach is the combined method that includes both of the first two approaches.

4.1 Learning from Previous Tours

In the first approach, learning is accomplished by annotating each hyperlink with the interests of the users who took this hyperlink on previous tours. Thus, whenever a user follows a hyperlink the description of this hyperlink is augmented by adding the keywords the user typed in at the beginning of the tour. The initial description of a hyperlink is the underlined text. Figure 3 illustrates the keywords accumulated by hyperlinks, and the way in which these are used to influence subsequent advice.

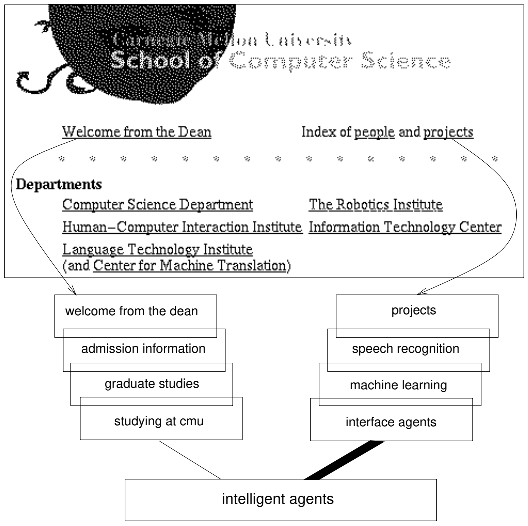

Figure 3: Here the interest of a new user in "intelligent agents" matches the "projects" hyperlink better than the "Welcome from the Dean" hyperlink, because of the keywords accumulated by this hyperlink during previous tours.

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

The metric used to compute similarity between a user's stated interest and a hyperlink description is based on a technique from information retrieval [Salton, 1991]. Interests and hyperlink descriptions are represented by very high-dimensional feature vectors, each dimension representing a particular word in the English language. The elements (called word-weights) of a vector are calculated using the TFIDF heuristic [Salton, 1991]. Based on this vector representation similarity is calculated as the cosine between vectors.

The algorithm WebWatcher uses to suggest hyperlinks considers all hyperlinks on the current page. For each hyperlink, the list of associated keywords (including the original underlined words) is used to calculate its similarity to the current user's interest. The value of $LinkQuality$ for each hyperlink is estimated to be the average similarity of the k (usually 5) highest ranked keyword sets associated with the hyperlink. A hyperlink is suggested if its value for $LinkQuality$ is above a threshold. The maximum number of hyperlinks suggested on a page is three.

4.2 Learning from Hypertext Structure

The previous section describes a learning method that augments a given hyperlink with the stated interests of earlier users who selected it. In this section we describe a second learning method that augments a given hyperlink using words encountered in pages downstream of it. This approach is based on reinforcement learning. The

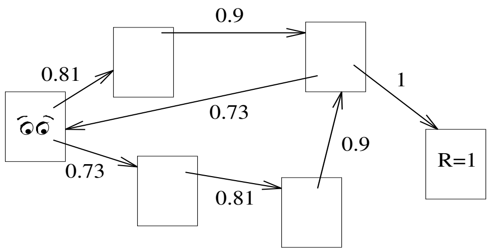

Figure 4: Example state space.

objective is to find paths through the Web which maximize the amount of relevant information encountered.

Reinforcement Learning

Reinforcement learning allows agents to learn control strategies that select optimal actions in certain settings. Consider an agent navigating 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 possible 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 $0 \leq \gamma < 1$ 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 $Q(s,a)$ repeatedly as follows (see [Watkins, 1989]):

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

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 to repeatedly pick the action a that maximizes $Q(s,a)$ for its current state s.

Figure 4 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 state in the smallest number of steps and thus maximize the discounted reward it receives.

Reinforcement Learning and Hypertext

Imagine a Web agent looking for pages on which the word "intelligent" occurs. For this agent, states correspond to Web pages, and actions correspond to hyperlinks. In this case, we define the reward $R_{intelligent}(s)$ for a particular page s to be the TFIDF value of "intelligent" for s. The agent will then learn a $Q_{intelligent}(s,a)$ function whose value for page s and hyperlink a is the sum of discounted TFIDF values of "intelligent" over the optimal tour beginning with a. It can then use this Q function to choose the best link at each step in the tour.

WebWatcher uses a separate reward function $R_w(s)$ and learns a distinct $Q_w(s,a)$ function for every word w. At runtime the system recommends hyperlinks for which the sum of the $Q_w(s,a)$ for the words in the user's interest description is highest. An additional reward is given if the interest matches the underlined text of the hyperlink.

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 $Q_w(s,a)$ that applies even to unseen states (pages) and actions (hyperlinks). We chose a distance-weighted 3-nearest neighbor function approximator [Mitchell, 1997] for this purpose, because of the many features needed to describe pages and hyperlinks, and because of certain theoretical advantages [Gordon, 1995]. Each hyperlink a is described by the TFIDF vector representation of the underlined anchor text, each page s analogously by its title. We define the similarity between the hyperlink a_1 on page s_1 and the hyperlink a_2 on page s_2 to be the distance between a_1 and a_2, plus (heuristically) 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 information retrieval.

4.3 Experimental Setup

The experiments presented in this section were conducted using 1777 of the 5822 traces starting at the SCS-FrontDoor page WebWatcher collected between August 2, 1995, and March 4, 1996. We used only those tours where the user typed in an interest and where the tour was at least four steps long. Five different test-training splits were used with the pages and hyperlinks from 70% of the traces used for training.

In addition to the reinforcement learning approach RL and the learning method ANNOTATE from section 4.1, the results for four other methods are presented. RANDOM suggests hyperlinks at random from the current page. POPULARITY suggests those hyperlinks which have been followed most frequently in the past, ignoring information about the current user's interest. MATCH suggests hyperlinks according to the TFIDF-cosine similarity between their underlined text and the user's interest. Finally, the COMBINE method combines the predictions of RL, POPULARITY, ANNOTATE, and

	Accuracy
Random	31.3% (.9)
Popularity	41.9% (.4)
Match	40.5% (.6)
Annotate	42.2% (.7)
RL	44.6% (.5)
Combine	48.9% (.3)

Figure 5: Results averaged over five different test training splits (with one standard error).

MATCH using logistic regression. 10% of the available training data is used for regression.

4.4 Experimental Results

The results in figure 5 provide a comparison of the learning methods. For each example in the test set each learner was allowed to choose three hyperlinks from the corresponding page. Accuracy measures the percentage of examples for which the user followed one of these chosen hyperlinks. We required each learning algorithm to make a prediction regardless of how confident it was.

The method WebWatcher used while accessible to the public was Annotate. The accuracy of 42.2% in the offline experiment here approximately matches the fraction of times (43.9%) users followed WebWatcher's advice during actual use of the system.

Reinforcement learning performs significantly better than the other basic methods. Nevertheless the combination of all methods achieves the highest accuracy, 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 on a particular page, Match uses the underlined text in hyperlinks, Annotate uses the interest descriptions from previous user traces, and RL makes use of hypertext structure that is downstream of the hyperlink in question.

4.5 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 perform 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 available for this page we took 8 random subsets of 15 examples and presented these to eight people who were already well-informed about this page and its locale. For each example, they were given the stated interest of the user, then asked to suggest the 3 hyperlinks the user was most likely to follow out of the 18 hyperlinks on this page.

As figure 6 summarizes, humans achieved an accuracy of 47.5% in predicting which hyperlink the user would follow. The learning method Annotate achieved an accuracy of 42.9% under the same conditions. The 22.4% for random prediction again provides a baseline. These results suggest that predicting which hyperlink the user is going to follow is a fairly difficult task in a general setting. In comparison to human performance the learning approach does reasonably well.

	Accuracy
Random	22.4%
Annotate	42.9%
Human	47.5%

Figure 6: A comparison to human performance on the SCS-Front-Door page.

5 Related Work

Letizia [Lieberman, 1995] is similar to WebWatcher in the sense that the system accompanies the user while browsing. One difference is that Letizia is located on a single user's machine and learns only his or her current interest. By doing lookahead search Letizia can recommend nearby pages.

Syskill and Webert [Pazzani *et al.*, 1996] 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 [Balabanovic and Shoham, 1995] 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 set of pages that match the user's interest.

6 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. During its 18 months of operation, WebWatcher served thousands of people browsing CMU's School of Computer Science Web pages. The system has the following properties:

- WebWatcher provides several types of assistance, but most importantly highlights interesting hyperlinks as it accompanies the user.
- WebWatcher learns from experience. We found a multi-strategy approach to be the most effective learning method.
- WebWatcher runs as a centralized server so that it can assist any Web user running any type of Web browser as well as combine training data from thousands of different users.

Our experience with WebWatcher led 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 match those 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 level of accuracy. Users tend to have a fairly short attention span and are distracted from their stated interest. Furthermore there is great diversity in the interests of users browsing CMU's SCS Front Door so that 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.

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. We have recently begun experiments with such a personalized WebWatcher [Mladenic, 1996].

- Combining user-specific and Web locale-specific learning. At the same time WebWatcher learns a model of one particular user, it could retain its 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 classifying 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, i.e. the combination of linguistic analysis with the statistical techniques described here.

- 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.

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

[Armstrong *et al.*, 1995] 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.

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

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

[Joachims *et al.*, 1995] 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.

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

[Mitchell *et al.*, 1994] 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.

[Mitchell, 1997] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[Mladenic, 1996] D. Mladenic. Personal webwatcher: Implementation and design. Technical report, J. Stefan Institute, Ljubljana, Slovenia, 1996.

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

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

[Watkins, 1989] C. Watkins. Learning from delayed rewards. Technical report, King's College, Cambridge, England, 1989.