Probabilistic Online Action Prediction

Brian D. Davison and Haym Hirsh
Department of Computer Science
Hill Center for the Mathematical Sciences
Rutgers, The State University of New Jersey
Piscataway, NJ 08855 USA
{davison,hirsh}@cs.rutgers.edu

Abstract

People display regularities in almost everything they do. This paper proposes characteristics of an idealized algorithm that would allow a user interface to adapt to an individual's pattern of use. We describe a simple predictive method with these characteristics and show its predictive accuracy on a large dataset of UNIX commands to be better than others that have been considered, while using fewer computational and memory resources.

Motivation

How predictable are you? Each of us displays patterns of actions throughout whatever we do. Most occur without conscious thought. Some patterns are widespread among large communities, and are taught, as rules, such as reading from left to right, or driving on the correct side of the road. Other patterns are a function of our lifestyle, such as picking up pizza on the way home from work every Friday, or programming the VCR to record our favorite comedy each week. Many are a result of the way interfaces are designed, like the pattern of movement of your finger on a phone dialing a number you call often, or how you might log into your computer, check mail, read news, and visit your favorite website for the latest sports scores. As computers pervade more and more aspects of our lives, the need for a system to be able to adapt to the user, perhaps in ways not programmed explicitly by the system's designer, will become ever more apparent.

A car that can offer advice on driving routes is useful; one that can also guess your destination (such as a pizza parlor because it is Friday and you are leaving work) is likely to be found even more useful, particularly if you didn't have to program it explicitly with that knowledge. The ability to predict the user's next action allows the system to anticipate the user's needs (perhaps through speculative execution) and to adapt to and improve upon the user's work habits (such as automating repetitive tasks). Additionally, adaptive interfaces have also been shown to help those with disabilities (Greenberg *et al.* 1995; Demasco & McCoy 1992).

This paper considers the more mundane, but present-day activities of user actions within a command line shell. We have concentrated initially on UNIX command prediction because of its continued widespread use; the UNIX shell provides an excellent testbed for experimentation and automatic data collection. However, our interest is in more general action prediction, and so we hypothesize that successful methodologies will also be applicable in other interfaces, including futuristic ones anticipated above as well as present-day menu selection in GUIs and voicemail, or URL selection in web browsers. This paper, therefore, reflects our focus on the underlying technology for action prediction, rather than specifically on how prediction can be effectively used within an interface.

In this paper, we use the data from two user studies to suggest that relatively naive methods can predict a particular user's next command surprisingly well. With the generic task in mind, we will describe the characteristics of an ideal algorithm for action prediction. Finally, we will present and analyze a novel algorithm that satisfies these characteristics and additionally performs better than the previous best-performing system.

Background

Technically, the problem addressed by this paper is that of multi-class sequence prediction in which the sequence is of nominal features. This type of problem is not studied often by machine learning researchers; concept recognition (i.e., the boolean classification task) is more common, as are independent samples from a distribution of examples. UNIX commands, and user actions in general, however, are not independent, and being nominal, don't fall into the domain of traditional time-series analysis techniques from statistics.

Evaluation Criteria

In most machine learning experiments that have a single dataset of independent examples, cross-validation is the standard method of evaluating the performance of an algorithm. When cross-validation is inappropri-

ate, partitioning the data into separate training and test sets is common. For sequential datasets, then, the obvious split would have the training set contain the first portion of the sequence, and the test set contain the latter portion (so that the algorithm is not trained on data occuring after the test data). However, since we are proposing an adaptive method, we will be measuring performance by online evaluation; that is, each algorithm was tested on the current command using the preceding commands for training. This maximizes the number of evaluations of the algorithms on unseen data, and reflects the likely application of such an algorithm.

When considering performance across multiple users with differing amounts of data, we use two methods to compute averages. *Macroaverage* refers to an average computed over all users (as in the performance on an average user's data). Alternately, *microaverage* is an average computed over all data (such as the likelihood of predicting the next command regardless of whose data is being used). The former provides equal weight to all users; the latter emphasizes users with large amounts of data.

People Tend To Repeat Themselves

In order to determine how much repetition and other recognizable regularities were present in the average user's command line work habits, we collected command histories of 77 users, totaling over 168,000 commands executed during a period of 2-6 months (see Figure 1 for an example of the data that was collected). The bulk of these users (70) were undergraduate computer science students in an Internet programming course and the rest were graduate students or faculty. All users had the option to disable logging and had access to systems on which logging was not being performed.

The average user had over 2000 command instances in his or her history, using 77 distinct commands on average during that time. On average over all users (macroaverage), 8.4% of the commands were new (had not been logged previously). The microaverage of new commands, however, was only 3.6%, reflecting the fact that smaller samples had larger numbers of unique commands. Surprisingly, approximately one out of five commands were the same as as the previous command executed (that is, the user repeated the last command 20% of the time).

Related Work

Greenberg (1993) and Lee (1992) have studied patterns of usage in the UNIX domain. They found that the recurrence rate (the likelihood of repeating something) was high for command lines as well as for commands themselves, but that individual usage patterns varied. Recently, Tauscher and Greenberg (1997) extended Greenberg's recurrence analysis to URL revisitation in World Wide Web browsers. These efforts consider only some simple methods for offering the top-n possibilities for easy selection (such as the most recent n commands).

```
...
96102513:34:49 cd
96102513:34:49 ls
96102513:34:49 emacs
96102513:34:49 exit
96102513:35:32 BLANK
96102513:35:32 cd
96102513:35:32 cd
96102513:35:32 rlogin
96102513:35:32 exit
96102514:25:46 BLANK
96102514:25:46 cd
96102514:25:46 telnet
96102514:25:46 ps
96102514:25:46 kill
96102514:25:46 emasc
96102514:25:46 emacs
96102514:25:46 cp
96102514:25:46 emacs
...
```

Figure 1: A portion of one user's history, showing the timestamp of the start of the session and the command typed. (The token BLANK marks the start of a new session.)

Stronger methods were attempted by Andrews in his master's thesis (1997) to predict user commands, but he had only a small sample of users with fairly short histories ($\leq$ 500) and thus unclear implications. In a recent conference paper, Debevc, Meyer, and Svecko (1997) report on the application of a stronger adaptive method for presenting a list of potential URLs (which they state had been shown to be effective in other domains).

Yoshida and Motoda (Motoda & Yoshida 1997; Yoshida & Motoda 1996; Yoshida 1994) also apply machine learning techniques to perform command prediction, using a powerful extension to the OS that lets them record file usage. This lets them implement speculative execution and report fairly high predictive accuracy (albeit on a small amount of real user data).

Other applications of machine learning to action prediction include WebWatcher (Joachims, Freitag, & Mitchel 1997) which predicts URL selection, but it focuses on predicting actions based on what those actions (URL selections) represent, such as the content of the selected web page, rather than logging and analyzing recurring patterns of selection. Automated form completion (Schlimmer & Wells 1996; Hermens & Schlimmer 1993) and predictive typing aids (Darragh, Witten, & James 1990) are similar applications.

Command prediction itself has also been considered from the user modeling community with plan recognition (Bauer 1996; Lesh & Etzioni 1995). Our motiva-

tion, however, is to explore the potential for action prediction in a knowledge-sparse environment (i.e., where user plans are not known or cannot easily be developed). It is obvious, though, that user plans for command prediction are possible, and have been considered in restricted domains such as electronic mail handling.

Our goal is to develop knowledge-free methods that can at least be used as performance benchmarks for domain-specific systems, or even as the basis upon which domain knowledge may be incorporated.

Knowledge-free algorithms

In previous work (Davison & Hirsh 1997a; 1997b), we considered a number of simple and well-studied algorithms. In each of these, the learning problem was to examine the commands executed previously, and to predict the command to be executed next. We found that, without explicit domain knowledge, a method based on C4.5 (Quinlan 1993) was able to predict each command, with an macroaverage accuracy of 38.5% (microaverage was 37.2%).

While the prediction method was a relatively straightforward application of a standard machine learning algorithm, it has a number of drawbacks, including that it returned only the single most likely command. C4.5 also has significant computational overhead. It can only generate new decision-trees; it does not incrementally update or improve the decision tree upon receiving new information. Therefore, the decision tree generation is handled outside of the command prediction loop, and is usually performed only once a day.

Additionally, since C4.5 (like many other machine learning algorithms) is not incremental, it must revisit each past command situation, causing the decision-tree generation to require more time and computational resources as the number of commands in the history grows. Finally, it treats each command instance equally; commands at the beginning of the history are just as important as commands that were recently executed.

These initial experiments dealt with some of these issues by only allowing the learning algorithm to consider the command history within some fixed window. This prevented the model generation time from growing without bound and from exceeding all available system memory. This workaround, however, caused the learning algorithms to forget relatively rare but consistently predictable situations (such as typographical errors), and restricted consideration only to recent commands.

Incremental Probabilistic Action Modeling

Ideal Online Learning Algorithm

With this experience in mind and the intuition that recent actions more strongly affect future actions than older actions, we propose the following description of an Ideal Online Learning Algorithm. In order to have the desirable performance characteristics of the best algorithms, an IOLA would:

1. have predictive accuracy as good as the best resource-unlimited methods (which here is C4.5);
2. operate incrementally (modifying an existing model rather than building a new one as new data are obtained);
3. be affected by all events (remembering uncommon, but useful, events regardless of how much time has passed);
4. not need to retain a copy of the user's full history of actions;
5. output a list of predictions, sorted by confidence;
6. adapt to changes to the target concept; and
7. be fast enough for interactive use.

Such a system would be ideally suited for incorporation into many types of user interfaces.

The Algorithm

In our work, we implicitly assumed that the patterns of use would form multi-command chains of actions, and accordingly built algorithms to recognize such patterns. If, however, we make the simpler Markov assumption that each command depends only on the previous command (i.e., patterns of length two), we can use the history data collected to count the number of times each command followed each other command, and thus calculate the probability of a future command. This could be implemented by a simple structure of an n by n table showing the likelihood of going from one command to the next.

For the anticipated use of action prediction in an adaptive interface, however, an incremental method is desirable. If a table of counts were recorded, this

```
Update(PreviousCommand, CurrentCmd):
  - Call UpdateRow for the default row
  - Call UpdateRow for row that corresponds
    to PreviousCommand

UpdateRow(ThisRow, CurrentCmd):
  - If initial update for ThisRow, copy
    distribution from default row
  - Multiply probability in each column by
    alpha and add (1-alpha) to column that
    corresponds to CurrentCmd
```

Figure 2: The update function.

```
Predict(NumCmds,PreviousCmd):
  - Call SelectTopN with NumCmds, the row for
    PreviousCmd, and the empty list
  - Let P be the number of commands returned
  - If P < NumCmds, call SelectTopN again, but
    ask for the top (P - NumCmds) commands from
    the default row and to exclude the set of
    commands already returned
  - Return the combined set of commands

SelectTopN(NumCmds,Row,ExcludeCmds):
  - Sort the probabilities in Row
  - Eliminate commands that are in ExcludeCmds
  - Return the top NumCmds from this sorted list
```

Figure 3: The predict function

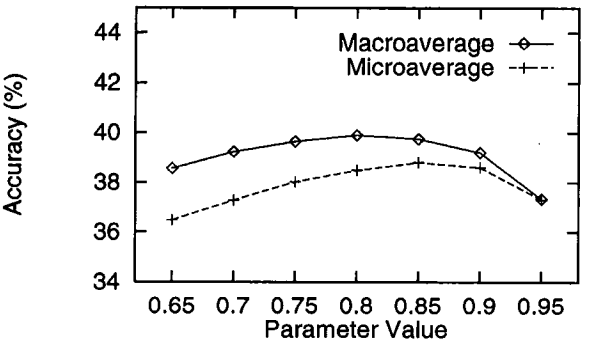

Figure 4: For a range of *alpha* values, the predictive accuracy of the Incremental Probabilistic Action Modeling algorithm is shown.

could be updated periodically and probabilities easily computed. As mentioned in the previous section, we believe it is useful to weigh recent events more highly when calculating a predictive model. This can be accomplished in this probabilistic model by the use of an update function with an exponential decay (in which the most recent occurrence has full impact; older occurrences have ever-decreasing contributions). Given the previous table of probabilities and another table containing probabilities from new data points, a combined new table may be computed by the weighted average of the two, where the weights sum to 1. So, for example, if the weights were both .5, the new probabilities would have equal contributions from the old table and from the new. Assuming that the table updates were performed periodically, the data points making up the first table would be contributing only $\frac{1}{2^n}$ percent of the final weights (where n is the number of table updates so far).

We can extend this model further, to an algorithm that starts with an empty table and updates after every command. An empty table is one in which all commands are equally likely (initially a uniform probability distribution). After seeing the first command, c_i, a new row is added for that command, and has a uniform distribution. When the second command, c_{i+1}, is seen, it too gets a new row with a uniform distribution, but we update the first row (since we saw c_i followed by c_{i+1}) by multiplying all elements of that row by a constant $0 \leq alpha \leq 1$, and the probability of seeing c_{i+1} is increased by adding $(1 - alpha)$. In this way, we emphasize more recent commands, at the expense of older actions, but the sum of the probabilities in each row is always 1.

Note that an *alpha* of 0 equates to a learner that always predicts what it most recently saw for that command, and an *alpha* of 1 corresponds to an algorithm that never changes its probabilities (in this case, keeping a uniform distribution).

For prediction, the command probabilities for the appropriate row can be sorted, and the one with the highest value would be output as the most likely next command. Instead of making no prediction for a command with an empty row, we can track probabilities in an additional *default* row, which would use the same mechanism for updating but would apply to all commands seen so far (without consideration of the preceding command). Finally, since we are keeping track of overall likelihoods in this default row, we can use it to initialize rows for new commands (making the assumption that these default statistics are better than a uniform distribution).

See Figures 2 and 3 for pseudocode for the `Update` and `Predict` functions that implement this Incremental Probabilistic Action Modeling (IPAM) algorithm.

Determining *Alpha*

We empirically determined the best average *alpha* by computing the performance for this algorithm on the dataset with each of seven values of *alpha* (from .65 to .95 in increments of .05). While the best value of *alpha* varied, depending on how performance was calculated over the dataset, our subjective choice for the the best overall was .80. (See Figure 4 for a graph of the parameter study of *alpha* showing the average user's performance as well as the average performance over all commands.) Since *alpha* controls the amount of influence recent commands have over earlier commands, we expect that this value will vary by problem domain.

Evaluation

This algorithm, when applied to the data set discussed earlier, performs better than C4.5 (given an appropriate *alpha*). It achieves a 39.9% macroaverage predictive accuracy (38.5% microaverage) versus C4.5's 38.5% and 37.2% (macroaverage and microaverage, respectively) for best guess predictions (see the bars labeled C4.5 and IPAM in Figure 5). For comparison, we also show our method without the specialized update,

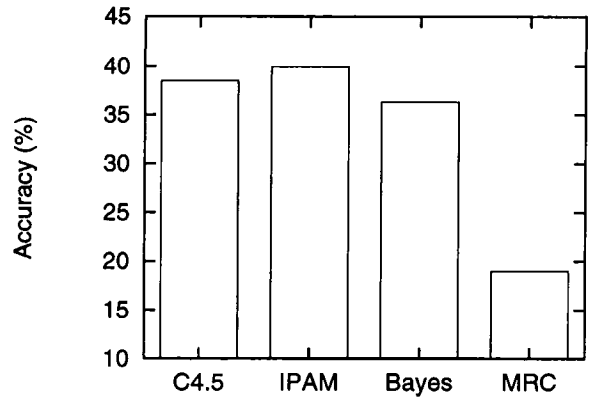

Figure 5: Macroaverage (per user) predictive accuracy for a variety of algorithms.

which corresponds to naive Bayes (that is, a predictor that strictly uses the frequency of pairs of commands to select the most likely command), as well as a straightforward most recent command predictor (labeled MRC).

To be precise, over the 77 users, IPAM beat the C4.5-based system sixty times, tied once, and lost sixteen times on the task of predicting the next command. At the 99% confidence level, the average difference between their scores was 1.42 ± 1.08 percentage points, showing an improvement in predictive accuracy for IPAM over C4.5 that is significant.

IPAM keeps a table in memory of size $O(k^2)$, where k is the number of distinct commands. Predictions can be performed in constant time, with updates requiring

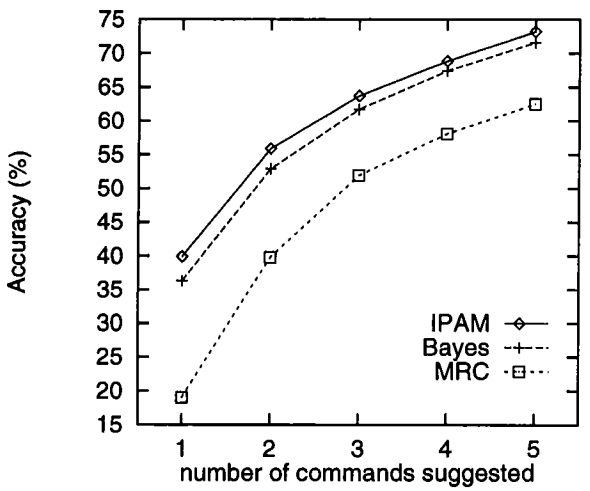

Figure 6: Average per user accuracies of the top-n predictions. The likelihood of including the correct command goes up as the number of suggested commands increases.

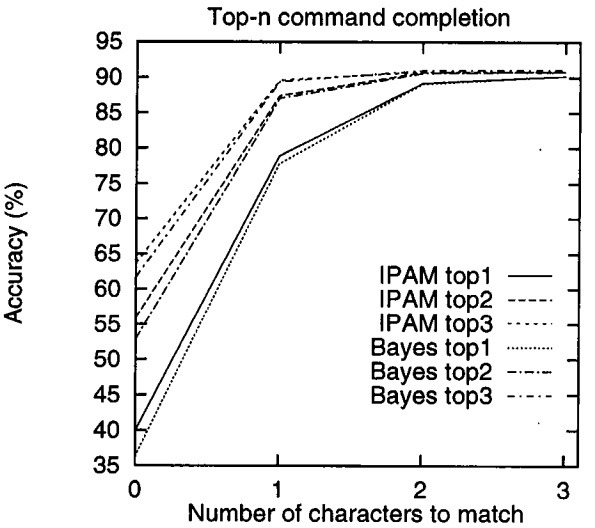

Figure 7: Command completion accuracies.

$O(k)$ time.

Since some applications of this method may be able to take advantage of a top-n predictive system, and this method generates a list of commands with associated probabilities for prediction, we can also compute the average accuracy of the top-n commands for varying values of n (as compared to only the single most likely command). Figure 6 shows that we do get increased performance (as expected) and that for $n = 5$, the correct command will be listed almost 75% of the time. This is valuable to an interface designer making it possible to consider the tradeoff of increased likelihood of listing the correct command versus the increased cognitive load of an interface showing multiple suggestions.

In UNIX command prediction, it is also helpful to be able to perform command completion (that is, taking the first k characters typed and produce the most likely command that is prefixed by those characters). Such a mechanism would enable shells that perform completion when there is a unique command with that prefix to also be able to perform completion when there are multiple possibilities. Figure 7 measures the predictive accuracy when given 0-3 initial characters to match when applied to all of the data. (Note that command completion when given 0 initial characters is just command prediction.)

Similar overall performance of IPAM can be seen in Figures 8 and 9 which show command prediction and completion accuracy over a larger `csh` dataset (Greenberg 1988; 1993), containing more than twice as many users, and approximately twice as many commands overall. Again, IPAM outperforms the simpler Bayes and MRC algorithms.

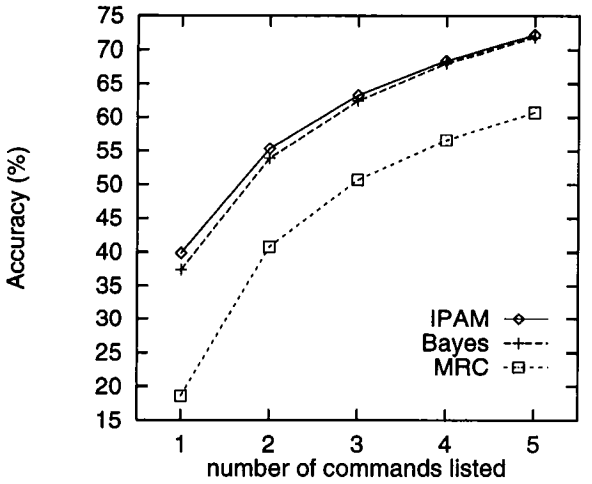

Figure 8: Average per user accuracies of the top-n predictions for the Greenberg dataset.

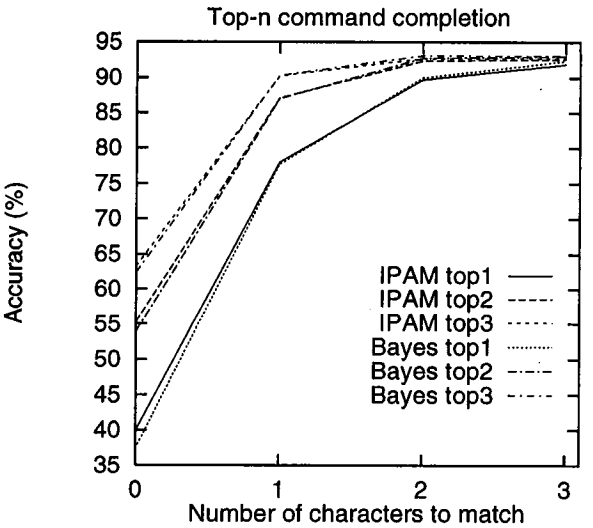

Figure 9: Command completion accuracies for the Greenberg dataset.

Discussion

While not shown, the results described apply to both macroaverage performance (shown in most figures) and microaverage performance, although the former is almost always slightly higher.

The two datasets used in this paper were collected almost ten years apart. The earlier one by Greenberg studied users of `csh` while the other studied users of a later version called `tcsh`. However, the similar performance results imply that they are representative samples of the same population of user actions in a UNIX command line environment.

We might consider initializing the table in IPAM with useful values rather than starting from scratch. For example, is the performance improved if we start with a table averaged over all other users? This lets us examine cross-user training to leverage the experience of others. Unfortunately, preliminary experiments indicate that, at least for this dataset, starting with the average of all other users' command prediction tables does not improve predictive accuracy. This result matches with those of Greenberg (Greenberg 1993) and Lee (Lee 1992), who found that individual users were not as well served by systems tuned for best average performance over a group of users.

Some scholars might criticize the use of knowledge-free 'weak methods' to perform learning. We agree, and suggest that any implementation of an adaptive interface might use IPAM as a base upon which a system with domain-specific knowledge might be built. In any case, it is useful, and often surprising, to discover the performance possible without domain knowledge, which can, if nothing else, be used for comparison against 'strong methods'. IPAM's implementation, in addition, was guided by the characteristics of an IOLA, and thus has other benefits in using limited resources in addition to performance.

This research has many possible extensions that would be desirable to investigate, such as predicting an entire command line (that is, commands plus parameters), and extending IPAM to recognize patterns longer than 2. Finally, incorporation of IPAM into a real-world interface would be useful to get user feedback on its performance (this is underway, as an extension to `tcsh`).

Summary

We have presented a method that fulfills the requirements of an Ideal Online Learning Algorithm. Incremental Probabilistic Action Modeling has an average predictive accuracy better than previously reported with C4.5. It operates incrementally, will remember rare events such as typos, and does not retain a copy of the user's action history. IPAM can generate top-n predictions, and by weighing recent events more heavily than older events it is able to react to 'concept-drift'. Finally, its speed and simplicity make it a strong candidate for incorporation into the next adaptive interface.

Acknowledgments

Our use of a probabilistic bigram model is based on an idea suggested by Corinna Cortes. This research is partially supported by a Rutgers University special allocation to strategic initiatives in the Information Sciences.

References

Andrews, T. 1997. Computer command prediction. Master's thesis, University of Nevada, Reno.

Bauer, M. 1996. Acquisition of user preferences for plan recognition. In Chin, D., ed., *Proceedings of the Fifth International Conference on User Modeling (UM96).*

Darragh, J. J.; Witten, I. H.; and James, M. L. 1990. The reactive keyboard: A predictive typing aid. *IEEE Computer* 23(11):41–49.

Davison, B. D., and Hirsh, H. 1997a. Experiments in UNIX command prediction. Technical Report ML-TR-41, Department of Computer Science, Rutgers University.

Davison, B. D., and Hirsh, H. 1997b. Toward an adaptive command line interface. In *Advances in Human Factors/Ergonomics: Design of Computing Systems: Social and Ergonomic Considerations*, 505–508. San Francisco, CA: Elsevier Science Publishers. Proceedings of the Seventh International Conference on Human-Computer Interaction.

Debevc, M.; Meyer, B.; and Svecko, R. 1997. An adaptive short list for documents on the world wide web. In *Proceedings of the 1997 International Conference on Intelligent User Interfaces*, 209–211. Orlando, FL: ACM Press.

Demasco, P. W., and McCoy, K. F. 1992. Generating text from compressed input: An intelligent interface for people with severe motor impairments. *Communications of the ACM* 35(5):68–78.

Greenberg, S.; Darragh, J. J.; Maulsby, D.; and Witten, I. H. 1995. Predictive interfaces: What will they think of next? In Edwards, A. D. N., ed., *Extraordinary human-computer interaction: interfaces for users with disabilities.* Cambridge University Press. chapter 6, 103–140.

Greenberg, S. 1988. Using unix: collected traces of 168 users. Research Report 88/333/45, Department of Computer Science, University of Calgary, Alberta. Includes tar-format cartridge tape.

Greenberg, S. 1993. *The Computer User as Toolsmith: The Use, Reuse, and Organization of Computer-based Tools.* New York, NY: Cambridge University Press.

Hermens, L. A., and Schlimmer, J. C. 1993. A machine-learning apprentice for the completion of repetitive forms. In *Proceedings of the Ninth IEEE Conference on Artificial Intelligence Applications*, 164–170. Los Alamitos, CA: IEEE Computer Society Press.

Joachims, T.; Freitag, D.; and Mitchel, T. 1997. Webwatcher: A tour guide for the world wide web. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 770–775. Morgan Kaufmann.

Lee, A. 1992. *Investigations into History Tools for User Support.* Ph.D. Dissertation, University of Toronto. Available as Technical Report CSRI–271.

Lesh, N., and Etzioni, O. 1995. A sound and fast goal recognizer. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, 1704–1710. Morgan Kaufmann.

Motoda, H., and Yoshida, K. 1997. Machine learning techniques to make computers easier to use. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, 1622–1631. Morgan Kaufmann.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning.* San Mateo, CA: Morgan Kaufmann.

Schlimmer, J. C., and Wells, P. C. 1996. Quantitative results comparing three intelligent interfaces for information capture: A case study adding name information into an electronic personal organizer. *Journal of Artificial Intelligence Research* 5:329–349.

Tauscher, L., and Greenberg, S. 1997. How people revisit web pages: Empirical findings and implications for the design of history systems. *International Journal of Human Computer Studies* 47(1):97–138.

Yoshida, K., and Motoda, H. 1996. Automated user modeling for intelligent interface. *International Journal of Human-Computer Interaction* 8(3):237–258.

Yoshida, K. 1994. User command prediction by graph-based induction. In *Proceedings of the Sixth International Conference on Tools with Artificial Intelligence*, 732–735. Los Alamitos, CA: IEEE Computer Society Press.