In Proceedings of the 1998 AAAI/ICML Workshop "Predicting the Future: AI Approaches to Time-Series Analysis"

Predicting Sequences of User Actions

Brian D. Davison and Haym Hirsh
Department of Computer Science
Rutgers, The State University of New Jersey
New Brunswick, NJ 08903 USA
{davison,hirsh}@cs.rutgers.edu

Abstract

People display regularities in almost everything they do. This paper proposes characteristics of an idealized algorithm that, when applied to sequences of user actions, would allow a user interface to adapt over time 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 at least as good as 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, 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 or intelligent defaults) 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[1] 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 voice-mail, or URL selection in web browsers. This paper, therefore, reflects our focus on the underlying technology for action prediction, rather than 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

This paper addresses the task of predicting the next element in a sequence, where the sequence is made up of nominal (unordered as well as non-numeric) elements. This type of problem (series prediction) is not studied often by machine learning researchers; concept recognition (i.e., a boolean classification task such as sequence recognition) is more common, as is the use of 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 statistical time-series analysis techniques.

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 inappropriate, 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

[1]We are currently ignoring command arguments and switches.

```
...
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.)

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 evaluating performance online — each algorithm is 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 expected application of such an algorithm.

When considering performance across multiple users with differing amounts of data, we use two methods to compute averages. *Macroaveraged* results compute statistics separately for each user, and then averages these statistics over all users. Alternately, *microaveraged* results compute an average over all data, determining the number of correct predictions made across all users divided by the total number of commands for all users combined. The former provides equal weight to all users, since it averages across the average performance of each user; 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 (Davison & Hirsh 1997a; 1997b) (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 during that time. On average over all users (macroaverage), 8.4% of the commands were new and 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. Approximately one out of five commands were the same as the previous command executed (that is, the user repeated the last command 20% of the time).

Earlier Results

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 naive method based on C4.5 (Quinlan 1993) was able to predict each command with a macroaverage accuracy of 38.5% (microaverage was 37.2%). For each prediction, C4.5 was trained on the series of examples of the form $(\text{Command}_{i-2}, \text{Command}_{i-1}) \Rightarrow \text{Command}_i$; for $1 \leq i \leq k$, where k is the number of examples seen so far. Command_0 and Command_{-1} are both defined to have the value BLANK to allow prediction of the first and second commands using the same form.

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. (While there are other decision-tree systems that can perform incremental updates (Utgoff 1989), they have not achieved the same levels of performance as C4.5.) Therefore, C4.5 decision tree generation must be performed outside of the command prediction loop.

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. Note that C4.5 was selected as a common, well-studied decision-tree learner with excellent performance over a variety of problems, but not with any claim of superiority over other algorithms applicable to this domain.

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 (IOLA). In order to have the desirable characteristics of the best algorithms, an IOLA would:

1. have predictive accuracy at least as good as the best known 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;
7. be fast enough for interactive use;
8. learn by passively watching the user (Mitchell, Mahadevan, & Steinberg 1985); and
9. apply even in the absence of domain knowledge.

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, so that the previous command is the state), 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 the 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 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

```
Update(PreviousCommand, CurrentCmd):
  - Call UpdateRow for the default row
  - Call UpdateRow for row corresponding
    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.

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 ini-

```
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 those commands already returned
  - Return the combined set of commands

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

Figure 3: The predict function

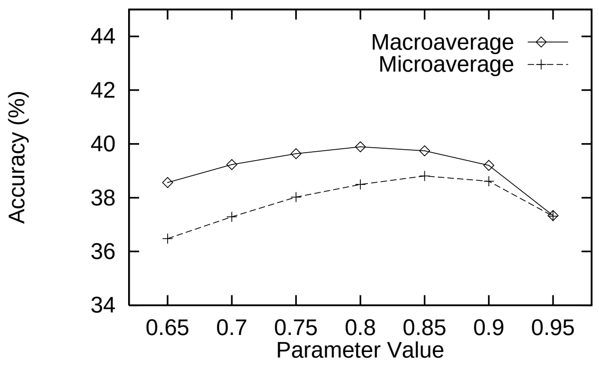

Figure 4: For a range of $alpha$ values, the predictive accuracy of the Incremental Probabilistic Action Modeling algorithm is shown.

tialize 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.) We will use this value of $alpha$ for the rest of the experiments in this paper. 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 $alpha$ of .80). 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, which corresponds to naive Bayes (that is, a predictor in which the conditional probabilities to select the most likely next command are based strictly on the frequency of pairs of commands), 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 that the improvement in predictive accuracy for IPAM over C4.5 is statistically significant, given the ideal value for $alpha$.

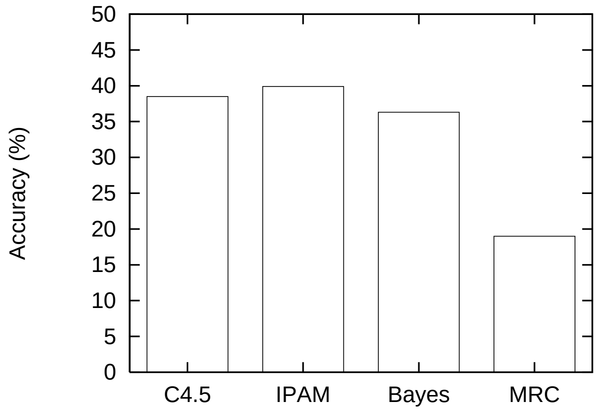

Figure 5: Macroaverage (per user) predictive accuracy for a variety of algorithms.

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 (when a list of next command is kept sorted by probability), with updates requiring $O(k)$ 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 and that for $n = 5$, the correct command will be listed almost 75% of the time. This makes it possible for an interface designer 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

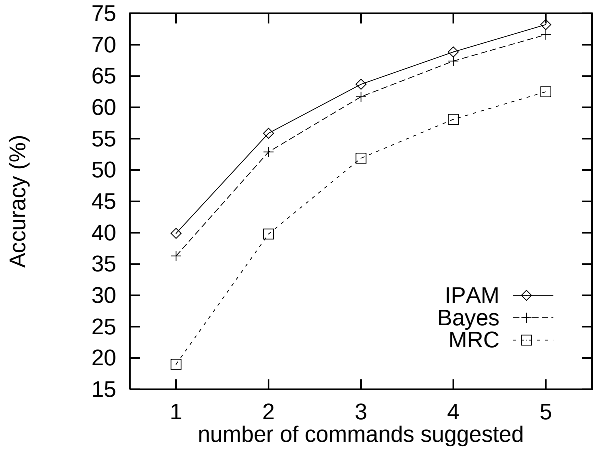

Figure 6: Average per user accuracies of the top-n predictions. The likelihood of including the correct command goes up as the number of suggested commands increases.

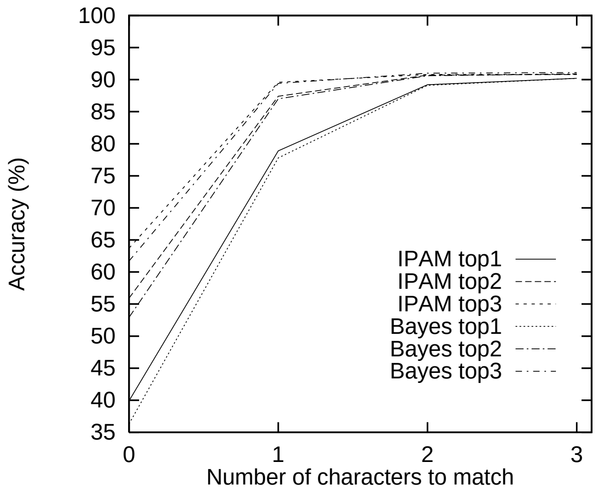

Figure 7: Command completion accuracies.

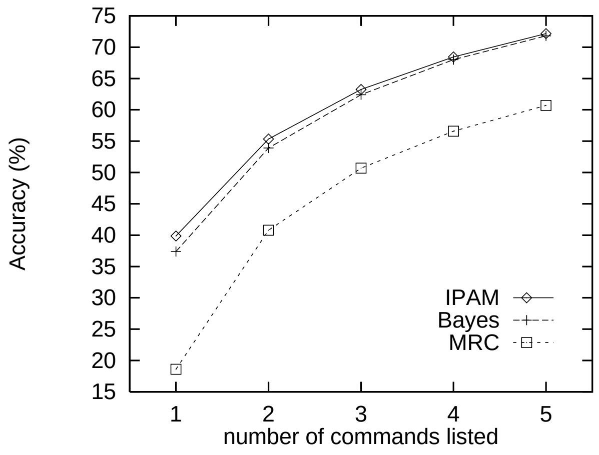

Figure 8: Average per user accuracies of the top-n predictions for the Greenberg dataset.

is prefixed by those characters). Such a mechanism would enable shells that perform completion when there is a unique command with that prefix (such as `tcsh`) 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 shows command prediction accuracy and completion 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, even when using the setting for $alpha$ determined by the parameter study over the first 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. While the results on the first dataset (collected on users of `tcsh`) can be argued as showing the *potential* for this method (since the selection of $alpha$ was based on the same set), the performance on the larger, and arguably more representative, Greenberg dataset (collected almost ten years earlier on users of *csh*) demonstrates a more believable performance.

Although learning may be performed throughout the history of a user's actions, the cumulative accuracy of a user does not vary much after an initial training period. Figures 10 and 11 show the performance of IPAM and Bayes, respectively over the history of one user. The solid line depicts the current overall average predictive accuracy (from the first command to the current command), while the dashed line shows the variations in predictive accuracy when measured over the most recent 30 commands.

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 (1993) and Lee (1992), who found that individual users were not as well served by systems tuned for best average performance over a group of users.

The goal of our work has been to discover the performance possible without domain knowledge. This can then be used as a benchmark for comparison against 'strong methods', or as a base upon which a system with domain-specific knowledge might be built. IPAM's implementation, in addition, was guided by the characteristics of an IOLA, and thus has

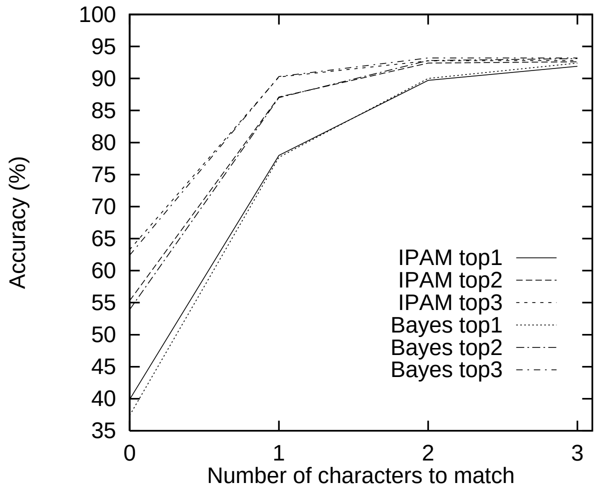

Figure 9: Command completion accuracies for the Greenberg dataset.

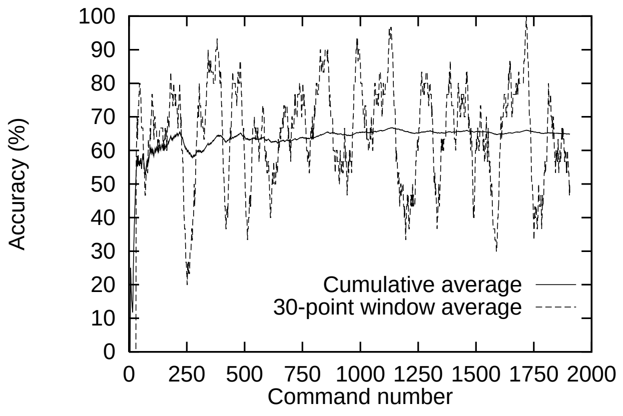

Figure 10: Cumulative performance of IPAM (when n=3) over time for a typical user.

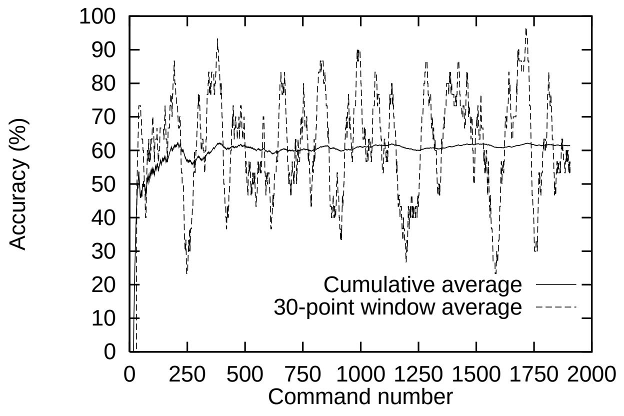

Figure 11: Cumulative performance of Bayes (when n=3) over time for a typical user.

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`).

Related Work

The problem of learning to predict a user's next action is related to work in a number of areas. There are of course many similarities in the problems studied in plan and goal recognition in the user modeling community (e.g., (Bauer 1996; Lesh & Etzioni 1995; Lesh 1997)). Such work attempts to model users in terms of plans and goals specific to the given task domain. Most efforts in this area thus require that the modeling system know the set of goals and plans in advance. This usually requires a significant human investment in acquiring and representing this domain knowledge. In contrast, our goal 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).

A smaller number of researchers have, instead, studied methods that have similar goals of generality. Yoshida and Motoda (Motoda & Yoshida 1997; Yoshida & Motoda 1996; Yoshida 1994) apply specially developed machine learning techniques to perform command prediction. This lets them implement speculative execution, and they report fairly high predictive accuracy (albeit on a small amount of real user data). However, much of the success of their work comes from knowing a fair amount about each action a user takes, by using a powerful extension to the operating system that lets them record the I/O accesses (e.g., reading files) that each command performs.

While on the surface the command prediction problem (and especially the IPAM approach) may bear some similarities to Markov decision processes, reinforcement learning approaches (Sutton & Barto 1998) are not likely to be a good choice for the following reasons: 1) while we have made the Markov assumption, it is only a rationalization that we used to motivate the IPAM approach; 2) every 'state' has equal value – the task is to correctly predict the next state, not to optimize the sequence to a goal state; 3) similarly, the reward function is transparent, with no credit assignment difficulty; and 4) not only is the domain noisy, but we believe the target concept changes over time.

Greenberg (1993) and Lee (1992) have studied patterns of usage in the UNIX domain, focusing on simple patterns of repetitions. 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. More 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). The Reactive Keyboard (Darragh, Witten, & James 1990) also uses simple history-matching methods for prediction, but at the lower level of keypresses.

Stronger methods (including a genetic algorithm-based classifier system) 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) in a batch framework and thus unclear implications. In a recent conference paper, Debevc, Meyer, and Svecko (1997) report on an application for presenting a list of potential URLs as an addition to a web browser. This method goes beyond recency-based selection methods, and instead tracks a priority for each URL which is updated after each visitation. The priority for a particular URL is computed essentially as the normalized averages of: a count of document usage, relative frequency of document usage, and the count of highest sequential document usage. This contrasts with our approach most significantly in that it does not consider relationships between documents and thus patterns of usage. Therefore, the Adaptive Short List computes a simplistic 'most likely' set of documents without regard to context.

A number of researchers have studied the use of machine learning in developing intelligent interfaces incorporating action prediction. For example, WebWatcher (Joachims, Freitag, & Mitchel 1997) predicts which links on a page on World Wide Web a user will select, and Hirashima *et al.* (1998) present a method for context-sensitive filtering, but both systems rely on the precise nature of the artifacts being manipulated (namely decomposable pages of text). Predictive methods in automated form completion (Schlimmer & Wells 1996; Hermens & Schlimmer 1993) are similarly tailored to the specifics on the application.

Maes (Maes 1994; Maes & Kozierok 1993; Sheth & Maes 1993) considers the broader task of building intelligent agents to serve as information filters and personal assistants. Such agents learn to "program themselves" by learning appropriate behavior from the user (and potentially from other agents). These agents are designed to be helpful, making suggestions and only gradually taking on more responsibility as their users' trust grows. Maes uses a variety of learning methods in designing these agents, including learning by watching the user. Thus, a mechanism that can learn to better model a user would be quite valuable in this context.

Programming by demonstration (Cypher 1993b; Nevill-Manning 1993) also has some similarities to our work. For example, Cypher's Eager (Cypher 1993a) can automate explicitly marked loops in user actions in a graphical interface. They, too, are concerned with performance when integrated into a user interface. While our approach is not designed to notice arithmetic progressions in loops, we can find and use the patterns in usage that do not recur as explicit loops and do not require special training by the user. Masui (1994) also learns repeated user patterns, requiring the user to hit the 'repeat' button when the system should learn or execute a macro.

Sequence prediction is also strongly related to data compression (Bell, Cleary, & Witten 1990) since an algorithm that can predict the next item in a sequence well can also be used to compress the data stream. Indeed, many of the approaches we describe could indeed be used in this fashion, precisely because they apply when only the user's history is available. However, we differ in that success in compression would only be an interesting phenomenon but not one that we explicitly target for our methods. Perhaps even more importantly, we target methods in which additional information sources can be easily injected. Our methods also are designed to be responsive to *concept drift*, since we make no assumptions about the stability of a user's actions over time — something that tends to reduce the usefulness of dictionary or grammar-based compression schemes (Nevill-Manning 1996). Laird and Saul (1994) present the TDAG algorithm for discrete sequence prediction, and apply it to a number of problems, including text compression, dynamic program optimization, and predictive caching. TDAG is based on Markov-trees, but limits the growth of storage by discarding the less likely prediction contexts. It is a fast online algorithm, but it, too, does not explicitly consider the problem of concept drift — each point in a sequence has essentially the same weight as any other point.

Finally, more distantly related, work in anomaly detection for computer systems (Kumar 1995; Lunt 1990) develops ways to quantify a user's normal behavior so that unusual activity can be flagged as a potential intrusion. Most intrusion detection systems can be categorized as using either statistical-anomaly detection, or rule-based detection. While rule-based expert systems monitor for known attack patterns and thus trigger few false alarms, they are also criticized as encouraging the development of ad hoc rules (Esmaili, Safavi-Naini, & Pieprzyk 1996) and require significant human engineering effort to develop. In contrast, statistical systems traditionally build profiles of normal user behavior and then search for the unusual sequences of events for consideration. Unlike most systems that perform anomaly detection by audit-trail processing off-line, our method works online, incrementally updating users' profiles as additional data arrives and could be augmented to provide user recognition.

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 at least as good as that 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)*.

Bell, T. C.; Cleary, J. G.; and Witten, I. H. 1990. *Text Compression*. Englewood Cliffs, NJ: Prentice Hall.

Cypher, A. 1993a. Eager: Programming repetitive tasks by demonstration. In Cypher (1993b). 204–217.

Cypher, A., ed. 1993b. *Watch What I Do: Programming by Demonstration*. Cambridge, MA: MIT Press.

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.

Esmaili, M.; Safavi-Naini, R.; and Pieprzyk, J. 1996. Evidential reasoning in network intrusion detection systems. In Pieprzyk, J., and Seberry, J., eds., *Information Security and Privacy: First Australasian Conference, ACISP'96*, 253–265. Wollongong, NSW, Australia: Springer-Verlag. Lecture Notes in Computer Science 1172.

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., *Extra-ordinary 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.

Hirashima, T.; Matsuda, N.; and Toyoda, J. 1998. Context-sensitive filtering for hypertext browsing. In *Proceedings of the 1998 International Conference on Intelligent User Interfaces*. ACM 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.

Kumar, S. 1995. *Classification and detection of computer intrusions*. Ph.D. Dissertation, Purdue University, West Lafayette, IN.

Laird, P., and Saul, R. 1994. Discrete sequence prediction and its applications. *Machine Learning* 15(1):43–68.

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.

Lesh, N. 1997. Adaptive goal recognition. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*. Morgan Kaufmann.

Lunt, T. F. 1990. IDES: An intelligent system for detecting intruders. In *Proceedings of the Symposium: Computer Security, Threat and Countermeasures*.

Maes, P., and Kozierok, R. 1993. Learning interface agents. In *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 459–465. Menlo Park, CA: AAAI Press.

Maes, P. 1994. Agents that reduce work and information overload. *Communications of the ACM* 37(7):31–40.

Masui, T., and Nakayama, K. 1994. Repeat and predict — two keys to efficient text editing. In *Proceedings of the Conference on Human Factors in Computing Systems*, 118–123. New York: ACM Press.

Mitchell, T. M.; Mahadevan, S.; and Steinberg, L. I. 1985. LEAP: A learning apprentice for VLSI design. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*.

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.

Nevill-Manning, C. G. 1993. Programming by demonstration. *New Zealand Journal of Computing* 4(2):15–24.

Nevill-Manning, C. G. 1996. *Inferring Sequential Structure*. Ph.D. Dissertation, University of Waikato, New Zealand.

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.

Sheth, B., and Maes, P. 1993. Evolving agents for personalized information filtering. In *Proceedings of the Ninth IEEE Conference on Artificial Intelligence Applications*, 345–352. Los Alamitos, CA: IEEE Computer Society Press.

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

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.

Utgoff, P. E. 1989. Incremental induction of decision trees. *Machine Learning* 4(2):161–186.

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.