Toward An Adaptive Command Line Interface

Brian D. Davison and Haym Hirsh

Department of Computer Science, Rutgers, The State University of New Jersey
New Brunswick, New Jersey 08903, USA

This paper explores different mechanisms for predicting the next command to be used for the UNIX command-line shell. We have collected command histories from 77 people, and have calculated the predictive accuracy for each of five methods over this dataset. The algorithm with the best performance has an average online predictive accuracy of up to 45%.

1. Introduction

This paper describes research concerned with improving an interface by making it adaptive — changing over time as it learns more about the user. We hypothesize that general machine-learning methods can recognize regularities in human-computer activities without requiring specialized knowledge about users or applications. To test this hypothesis, we have collected and analyzed a sizable set of UNIX command histories. This paper[1] discusses how the data was collected and how a sampling of algorithms performs on it.

There has been a range of work developing systems that recognize regularities in the usage of a computer. Yoshida and Motoda's [2] investigation of the use of machine learning to predict a user's next command is the most similar to the work reported here, but uses a different estimation of performance, different algorithms, and has a very small dataset. Also similar is work in programming by demonstration [3], such as Cypher's Eager [4], which recognizes and automates simple repetitions in user actions in a graphical interface. Finally, Lesh and Etzioni [5] also consider UNIX commands in plan recognition.

2. Approach and Methodology

We captured command histories from 77 people at Rutgers University. Two of these were faculty (including the second author), five were graduate students (including the first author), and the rest were undergraduates in an Internet programming course. Collection periods ranged from two months to over six months. These histories represent the primary work performed online by the authors and the participating graduate students. The undergraduate data was collected over two months on computer systems dedicated to their programming projects, and not their primary systems, and so the patterns of commands selected may reflect the orientation of their use (i.e., editing and compilation rather than email).

[1] An expanded version of this paper is available [1].

We collected data unobtrusively, by causing the UNIX shell at its closing to record the command history in a time-stamped file. This method was used to minimize potential interference with the user's activities.

This paper describes the application of a total of five methods to this data:

- C4.5 [6] is a well-used decision-tree learner developed in the machine-learning community, with demonstrated excellent performance over a wide variety of problems. Each command formed the basis for a single training example containing as its two attributes the two previous commands, and whose label was the command itself.

- The hypothetical "Omniscient" predictor correctly predicts all commands, providing that the desired command was used previously (under the assumption that, with no prior knowledge, no learning method would be able to predict the command). We use this as upper-bound on the potential predictive accuracy of any learner.

- In contrast, "Most Recent Command" (MRC) predicts the same command that was just executed. This method provides a useful baseline method to calibrate results.

- Slightly more sophisticated is "Most Frequent Command" (MFC), which predicts the command that previously occurred most frequently in the user's interactions.

- The last algorithm, Prefix, looks at the sequence of immediately preceding commands in the current shell session, and attempts to find the longest such sequence in the user's history, and predicts the command that followed that sequence.

With the exception of C4.5 (for which we used the distributed "off the shelf" software), each of the algorithms was implemented in Perl. Each algorithm was constrained to use a maximum-sized window of past commands in predicting the next command, where our experiments used window sizes n=100, 500, and 1000. Thus, for example, to predict the next command MFC predicts the command that occurred most frequently in the last 100 (or 500 or 1000) commands.

The data for this domain is inherently sequential, and thus it is inappropriate to use cross-validation and similar evaluation methods that effectively take a random sample of commands from a user's history and use them to predict the remaining commands. Yoshida and Motoda [2] evaluate their ability to predict the last third of commands using the first two-thirds,[2] but this could worsen predictive accuracy on the last third if users' behaviors are a "moving target", changing over time. Further, this eliminates two-thirds of the data from use in computing predictive accuracy. We therefore compute predictive accuracies for each method in an "online" fashion, by predicting each command in sequence using the preceding n commands in the user's history. We computed individual user accuracies and then averaged these results (the "macroaverage"), as well as the average overall prediction accuracy — number of correct predictions divided by total commands for all users (the "microaverage").

[2]For comparison purposes, we note that on average C4.5 achieved 32±12% (macroaverage) accuracy on the last third of each user's data using this evaluation method.

3. Experimental Results

The 77 subjects had history sizes ranging from 15 commands to a maximum of 34,940 commands, and had an average of 2,184 commands. The average number of distinct commands per user was 77.1, and the average command length was 3.77 characters.

Figure 1 demonstrates the macroaverage performance of each algorithm across the range of training-window sizes. Table 1 shows the performance for each algorithm with a training window size of 1000.

The results given so far have all been in terms of predictive accuracy. While this is a common and useful metric, it has some limitations. It does not, for example, measure the extent to which user productivity is improved (or harmed) by a shell with a prediction component. For example, if a user has defined many single-character aliases, then even high accuracy in prediction need not result in a significant improvement in a user's efficiency if it is measured in number of keystrokes, since both the user's command and the command-inserting may require only a single keystroke. Additionally, the overhead of managing a history and computing predictions from it may require too much time for an online system.

In the real world, user productivity is the primary concern. Since the data collected was from unmodified, non-predictive command shells, we can only speculate as to the improvements in productivity afforded by the methods presented here. However, in the best case C4.5's predictions were correct up to 45% of the time. Recall that the lengths of the histories on average were 2184 commands, and the average command was 3.77 characters long. Assuming that a correct prediction could be inserted with a single character, and that commands recorded in the study were all typed explicitly (not using any historical shortcuts or command completion), this method would have saved just over 31% of the keystrokes typed, which is very close to the 33% expected if predictability were independent of command length.

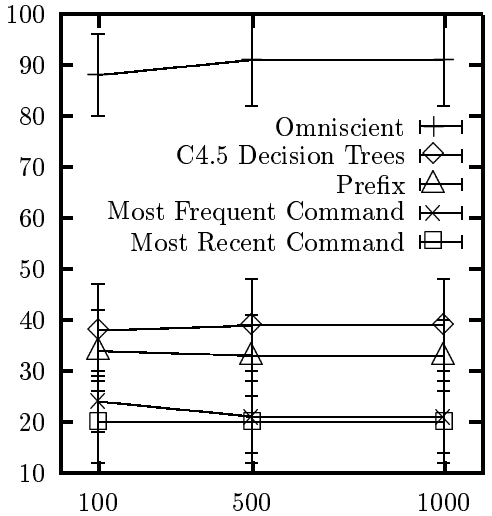

Figure 1. Macroaverage performance (in percent) vs. training-window size.

Table 1
Online predictive accuracy (for training-window size 1000).

algorithm	macroaverage	microaverage
Omniscient	91±9%	96±21%
C4.5	39±9%	45±50%
Prefix	33±7%	36±48%
MFC	21±7%	18±39%
MRC	20±8%	30±46%

4. A Prototype Shell: `ilash`

We have implemented a simple prototype that incorporates some of the ideas described in this paper. Called `ilash` (Inductive Learning Apprentice SHell), it is an extension to the UNIX shell `tcsh` that allows the user to present the predicted command in the prompt string, and to insert the predicted command with a single keystroke.[3] It uses only two attributes — the two previous commands issued. Learning is performed by the decision-tree learner C4.5. This prototype is described further elsewhere [7].

While this initial version of `ilash` incorporates a strong learner, C4.5, it does make other tradeoffs. Decision trees are not created or updated online — a new decision tree must be built explicitly by the user, commonly just once a day. This choice allows the shell to keep its natural responsiveness, but reduces prediction accuracy.

Our user studies did not include use of this prototype, for a number of reasons. First, doing so would have required a commitment to a single prediction method (at the very least, for each user), and we wanted to explore a range of methods. Second, we wanted to be able to separate as much as possible effects of the interface from the quality of competing prediction methods. On the other hand, the system's predictions, when provided to the user through the interface, could influence the user's next action, potentially changing the sequence of commands executed.

5. Summary

This paper has described a number of methods for predicting a user's next command in a UNIX command-line shell. We found that relatively straightforward, knowledge-free methods were able to correctly predict the next command that the user would execute up to 45% of the time, potentially reducing the keystrokes needed by close to one third. The best performance was achieved using C4.5 to learn prediction rules based on two previous commands, and a training window of either 500 or 1000 past commands.

REFERENCES

1. Brian D. Davison and Haym Hirsh. Experiments in UNIX command prediction. Technical Report ML-TR-41, Department of Computer Science, Rutgers University, 1997.
2. Kenichi Yoshida and Hiroshi Motoda. Automated user modeling for intelligent interface. *International Journal of Human-Computer Interaction*, 8(3):237–258, 1996.
3. Allen Cypher, editor. *Watch What I Do: Programming by Demonstration.* MIT Press, Cambridge, MA, 1993.
4. Allen Cypher. Eager: Programming repetitive tasks by demonstration. In Cypher [3], pages 204–217.
5. Neal Lesh and Oren Etzioni. A sound and fast goal recognizer. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence.* Morgan Kaufmann, 1995.
6. J. Ross Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1993.
7. Haym Hirsh and Brian D. Davison. An adaptive UNIX command-line assistant. In *Proceedings of the First International Conference on Autonomous Agents.* ACM Press, 1997.

[3]The modifications to `tcsh` that formed our prototype were implemented by Mark Limotte.