An Adaptive UNIX Command-Line Assistant

Haym Hirsh and Brian D. Davison
Department of Computer Science
Rutgers, the State University of New Jersey
Piscataway, NJ 08855
Email: {hirsh,davison}@cs.rutgers.edu

Introduction

A typical user exhibits many regularities in interacting with a computer system. Although each user's mode of interaction is different, a single user's interactions often demonstrate systematic patterns of use. The goal of the work reported here is to develop software agents that recognize such patterns so as to learn to act on the user's behalf.

There has been a range of work developing agents that recognize regularities in computer usage (Armstrong *et al.* 1995; Mitchell *et al.* 1994; Hermens & Schlimmer 1993; Schlimmer & Hermens 1993; Darragh, Witten, & James 1990; Masui & Nakayama 1994; Yoshida 1994). This paper describes our ongoing efforts on building such pattern-recognizing agents into a UNIX shell, in particular, through the addition of a personalized software agent — a "personal assistant" (Mitchell *et al.* 1994) — to the UNIX shell `tcsh`. Before displaying a user's command-line prompt, the agent extrapolates from the user's past interactions with the shell and makes a prediction concerning what the user's next command will be.

The Shell

We have implemented our command-prediction mechanism as an extension to the existing UNIX `tcsh` shell. Our extensions manifest themselves in two ways to the shell user, through the addition of a special symbol for use in defining prompts, and the addition of a keystroke that is interpreted specially by the command-line processor.

Most UNIX shells provide a way for the user to specify how the command-line prompt should appear. For example, in `tcsh` the symbols "`%m`" are replaced with the name of the user's machine whenever it appears in the prompt string. Our first modification was to add one more special sequence, "`%g`", that is expanded into the system's prediction for what the next command will be.

As the user types to the shell, each character is received and processed by the shell code. In most cases the character is just passed on, as the next character in the command string that the user is entering. However, not all characters are treated in such a fashion; some have special interpretations within the shell. We have added an additional character that is interpreted specially by the shell, "C-g". Whenever the user types "C-g" the shell inserts its guess for the next command into wherever the cursor is on the command line.

Command Prediction

In order for a shell to predict a user's next command the shell must have some function that inspects the current state of the user's session and produces a prediction of what the next command will be. The approach taken here is to learn such functions by passively watching the user interact with the shell, and from such observations learn to predict the next command. This "learning apprentice" approach (Mitchell, Mahadevan, & Steinberg 1985) gathers data by recording, after executing each command, the state of the user's session with the shell and what command was executed. This forms the basis for applying an off-the-shelf inductive-learning method that extrapolates from such data a procedure for inspecting the current state of the shell and generating a prediction of the user's next command.

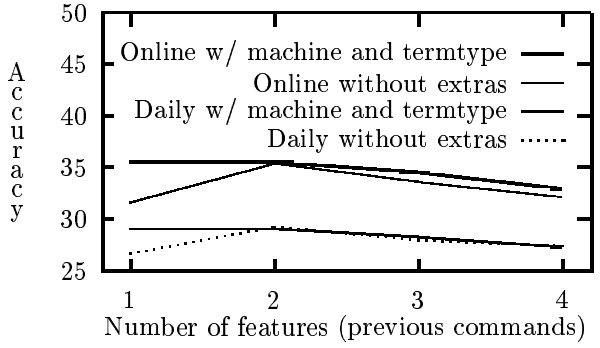
The state of a user's session with a shell can include many things — time of day, files in the current directory, previous commands, etc. Our current implementation only predicts the command that will be executed, without arguments, and bases its prediction solely on the user's previous two commands. Data for most learning systems require that items be described by a fixed set of attributes, each labeled with one item out of some set of possible labels. For command prediction we create datapoints for learning described by two attributes — the previous two commands (without arguments) — with each datapoint labeled by the command name that was executed. Learning is performed by the decision-tree learner C4.5 (Quinlan 1993).

Evaluation

An important question that must be addressed by developers of any personal software assistant is evaluation. When relevant, one common technique is to measure the predictor's success rate — how often its guess matches the user's actual command. When this is measured by testing in an online fashion (train on all preceeding commands, test on current command; for these tests we evaluated every tenth command), the shell had an estimated accuracy of 35.4% for the first author, and a 67.8% accuracy for the second author, based on roughly 6 months of data (7.8k and 30k commands respectively).

Although this accuracy-assessment method is not without its shortcomings, the whole predictive-accuracy approach to evaluation has serious limitations. It does not measure the extent to which user productivity is improved (or harmed). 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, say, number of keystrokes, since both the user's command and the command-inserting "C-g" require a single keystroke. Although accuracy measures do provide an easy way to compare competing command-prediction methods (as in the results reported in the next section), continued work in this area requires the development of adequate performance measures. We therefore conclude this section only

Figure 1: Prediction accuracy (%) on first author's data using Decision Trees



with two anecdotal assessments of our enhanced `tcsh`.

First, the shell has already exhibited properties not planned nor predicted by its developers. For example, it provides a simple form of spelling correction. If a user regularly mistypes "telnet" as "telenet", the shell will observe this in its data, and in the future predict that the next command will be "telnet" whenever the user had just attempted to execute a "telenet" command.

Second, although there clearly must be some element of a user's cognitive attention that focuses on the prediction displayed in the prompt, it does not appear to be substantial. In practice, once the user is comfortable with the shell we find that when the prediction is wrong the user doesn't really notice the presence of the guess, but when it is correct the prediction is noticed and the user can type "C-g".

Improving Command Prediction

There are many pieces of information that could be relevant in predicting a user's next command. For example, including terminal types or machine names as an attribute for learning could prove helpful. Even just restricting oneself to past commands, as was done here, leaves open the question of how many past commands to use. We designed our implementation to consult only two past commands simply as the basis for building a tractable initial prototype of a shell with command prediction. The main liability in adding more attributes is that the more that are included, the more likely it is that baseless correlations will be found in the data, yielding inferior command predictions. Figure 1 investigates this question using data from the first author, and shows how accuracy is affected by the number of previous commands that are used for learning and prediction. From the results of online testing every tenth command, as well as generating a decision tree only once daily, it would appear that using two previous commands was a good choice.

Our system used the decision-tree learner C4.5 simply because it is a fast and well-used learning method. Other methods can also be considered for learning command-prediction procedures. For example, one simple approach is to make predictions concerning the next command by finding the largest suffix of the sequence of commands up to the current point that previously occurred in the command history, and predict whatever command followed them. Table 1 reports the accuracy of this method as the maximum saved history size is varied, and the consequences of restricting searching to histories of sessions with the same machine name and terminal type as the current session. It also shows the simpler

Table 1: Accuracies (%) for MFC and Suffix Matching

author	algorithm	extras features	max. history considered infinity	1000	500	100
HH	Suffix	with	27.0	27.3	27.3	27.1
		w/out	25.3	25.5	25.5	26.6
	MFC	with	16.1	16.2	16.2	17.2
		w/out	8.8	9.2	10.4	14.7
BD	Suffix	with	52.9	53.8	59.0	67.4
		w/out	53.1	48.3	48.4	60.3
	MFC	with	15.3	16.3	16.4	20.5
		w/out	12.8	12.8	12.7	13.8

base line of predicting the most frequently used command (MFC) within the specified maximum history size. Since performance generally improved as the size of the history used for training was reduced, we believe this shows the importance of recent history over older history for command prediction. We also note that suffix matching appears to do about as well as decision-tree prediction for the first author, but not as well for the second author (results not shown here).

Final Remarks

This paper has described the addition to the UNIX shell `tcsh` of a personal software assistant that records a user's history of interactions with the shell and learns procedures for predicting users' future commands as a function of past commands. As the agent predicts the user's next command, the user can have the prediction displayed in the command prompt and may insert the prediction into the current command line with a single keystroke. We are continuing to explore a range of prediction methods and approaches to integrating them into the shell in ongoing work.

Acknowledgements

Mark Limotte implemented the modifications to `tcsh` described here. We thanks our colleagues at Rutgers and CMU for helpful discussions concerning this work.

References

Armstrong, R.; Freitag, D.; Joachims, T.; and Mitchell, T. 1995. WebWatcher: A learning apprentice for the world wide web. In *Proceedings of the AAAI Spring Symposium on Information Gathering from Distributed, Heterogeneous Environments.*

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

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.* Los Alamitos, CA: IEEE Computer Society Press.

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.; Caruana, R.; Freitag, D.; McDermott, J.; and Zabowski, D. 1994. Experience with a learning personal assistant. *Communications of the ACM* 37(7):81–91.

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.*

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

Schlimmer, J. C., and Hermens, L. A. 1993. Software agents: Completing patterns and constructing user interfaces. *Journal of Artificial Intelligence Research* 1:61–89.

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.