# Predicting UNIX Command Lines

**Benjamin Korvemaker** and **Russell Greiner**
{benjamin,greiner}@cs.ualberta.ca
Department of Computing Science
University of Alberta
Edmonton, Canada

## Problem

UNIX command line prediction appears at first glance to be a trivial task: after all, how many commands can one person possibly use with regularity? Although this applies to novice users[1], novice users become advanced users with experience. The problem of predicting UNIX command lines has been targeted for the last decade by a number of people in the Human-Computer Interaction (HCI) and Machine Learning (ML) communities. Greenberg collected a large amount of data(Greenberg 1988), providing the community with the usage patterns of 168 users of varying degrees of skill (non-programmers, novice programmers, experienced programmers, and computer scientists). Despite being ten years old, this data is still quite usable, and has become a de facto benchmark.

Davison and Hirsh more recently provided a hand-crafted algorithm that adapts to user activity over time,(Davison & Hirsh 1997; )generating a simple probabilistic model to predict command stubs[2]. Given a command at time $t$, they keep track of which command occurs at time $t+1$. Over time, a conditional probability table can be built. However, instead of building a table based on stub frequencies, a pseudo-probability is calculated by assigning each stub within the table an initial probability (during insertion into the table), decaying all probabilities within the table (by multiplication by $\alpha|0 \leq \alpha \leq 1$), and increasing the current probability when a previously encountered stub is observed. This method obtains nearly 75% accuracy when predicting the 5 most likely commands[3]. This was a substantial improvement over applying C4.5 to the task, which obtained only 38.5% accuracy.

[1] Novice users are typically characterized by a lack of knowledge about existing commands, as well as a lack of knowledge about how to effectively use (or abuse) system resources.

[2] A **stub** is simply the name of the command, rather than the complete command line (i.e. if the command line is "latex foo.tex", the stub is "latex").

[3] Predicting the top $n$ commands allows a number of commands to follow a single command without a great penalty.

## Solution

Predicting a user's next command line is a moving target: user goals can change, external events (mail arriving, deadlines approaching) occur, what happened in the past affects the future. Dividing data between training and testing is problematic, and off-line machine learning methods do not adapt easily to short-term patterns. Updating a table of estimated probabilities of the next command as each command is executed addresses these issues somewhat:

- Short-term patterns can be recognized quickly.
- Testing can be done by comparing the predictions to the actual command typed.

By constructing a Bayesian network (BN) that dynamically updates the local structure of conditional probability tables(Friedman & Goldszmidt 1996), we are able to model user actions and habits over time. Constructing this model as an Interpolated Markov Model (IMM)(Salzberg *et al.* 1998) becomes fairly straightforward:

1. If command $C_t$ occurs after command $C_{t-1}$, increase the probability that $C_t$ will occur after $C_{t-1}$. For all other commands $C'$ which have been observed to follow $C_{t-1}$, reduce the probability they will occur $C_{t-1}$.
2. If $C_t$ has been observed to frequently follow the sequence $C_{t-2}\,C_{t-1}$, apply step 1 similarly. Repeat for the sequence $C_{t-3}\,C_{t-2}\,C_{t-1}$, and so on.

By constructing multiple experts (see Figure˜**??**

*fig : experts), eachfocusingonaparticularaspectofprediction(e.g. oneexpertspecializesonpredictionsforMondaymorning, onespecializesonpredictionsbasedoncommandsequence, anotherspecializesonpredictionsforworkingonLaTeX files), andstatisticallycombiningtheirpredictions, strongerresultsareobtained.Furtherimprovementscanbemadebycreatinga“parsingexpert”whichparsesthecommandlineintotokensreflectingthepreviouscommandlines.Forinstance, ifthefollowingcommandsaretyped :*

```
vi abstract.tex
latex abstract.tex
xdvi abstract.dvi
dvips abstract.dvi -o abstract.ps
gv abstract.ps
```

they could be represented as:

```
vi abstract.tex
latex <T-1 ARG1>
xdvi <T-1 ARG1 STEM>.dvi
dvips <T-1 ARG1> -o <T-1 ARG1 STEM>.ps
gv <T-1 ARG3>
```

We have developed a number of operators to improve the chances of finding common, generic patterns. The advantages of this are two-fold:

1. Improved prediction accuracy.
2. Computer-generated shell-aliases can be generated.

## Results

While duplicating the work of Davison and Hirsh, we obtained 72.7% accuracy when predicting stubs (as opposed to their "almost 75%"), and 46.9% accuracy when predicting complete command lines. Further algorithm enhancements and a stronger statistical basis provide even better results:

- Parsing command lines
- Skipping the first 100 commands as purely training data (empirically determined to be a reasonable value)
- Looking for longer command line sequences (e.g. the LaTeX sequence tends to have at least 3 or 4 commands)
- Using multiple "experts" predictions and combining their prediction by standard techniques

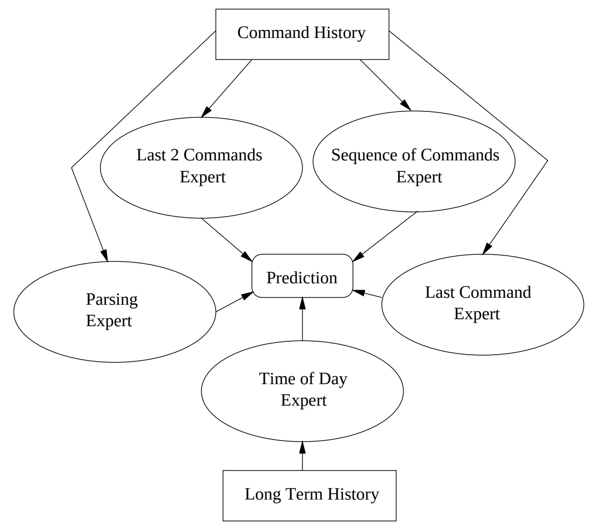


Figure 1: Example combination of multiple expert predictors.

| Source of Prediction | Cmd | Stub |
|---|---|---|
| 5 most frequent lines | 33.9% | 62.3% |
| command$_{t-1}$[5] | 47.4% | 73.3% |
| command history[6] | **48.9%** | **74.6%** |
| command history[7] | 46.9% | 73.4% |

Where

Cmd is the accuracy of the predicted command lines, not counting the first 100 as training data.

Stub is the accuracy of the predicted command stubs, not counting the first 100 as training data.

Our first implementation required an extension to ZSH, labourously entrenched into the various inner workings of ZSH. Although we had a functional shell which could predict complete command lines, there were usability issues. Our most recent implementation is tied far less to any particular UNIX shell, and potentially is far more usable.

See `http://www.cs.ualberta.ca/~benjamin/AUI/` for more information and updates.

## References

Davison, B. D., and Hirsh, H. In *Predicting the Future: AI Appreaches to Time-Series Analysis.*

Davison, B. D., and Hirsh, H. 1997. Toward an adaptive command line interface. In *Advances in Human Factors/Ergonomics: Design of Computing Sytems: Social and Ergonomic Considarations*, 505–508. Elsevier.

Friedman, N., and Goldszmidt, M. 1996. Learning bayesian networks with local structure. In *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence.*

Greenberg, S. 1988. Using unix: Collected traces of 168 users. Research Report 88/333/45, Department of Computer Science, University of Calgary, Calgary, Alberta.

Salzberg, S.; Delcher, A.; Kasif, S.; and White, O. 1998. Microbial gene identification using interpolated markov models. *Nucleic Acids Research* 26(2):544–548.

[5] Duplication of Davison and Hirsh's work.
[6] New Algorithm
[7] New Algorithm + Improved Statistics