

Predicting UNIX commands using decision tables and decision trees

K. T. Durant, M. D. Smith

Department of Engineering and Applied Sciences, Harvard University, USA

Abstract

This paper presents promising results from a series of experiments of command prediction for the UNIX command-line interface. The results provide insight into the effectiveness of decision tables and decision trees for predicting the commands of 77 different users comprising faculty members, graduate students, and undergraduate students of Rutgers University. We use the Weka system developed at the University of Waikato in New Zealand to explore the impact of these different learning techniques on command prediction. In general, our work addresses the important problem of predicting the next element in a sequence, where the sequence is made up of nominal elements, when the concept being learned changes over time. This feature is known as *concept shift*, and it occurs in our command line history session since the task each user is performing often changes during a session. As the concept shifts, predictions that were accurate previously are no longer correct. We measure success based on a micro-average and macro-average of prediction. A macro-averaged result computes statistics separately for each user and then averages these statistics over all users. The micro-average is the average predictive average over all the commands in the study independent of the user. The algorithms reported in this paper produce on average a predictive accuracy between 40% and 42%. This is a slight improvement of predictive accuracy from previous studies.

1. Introduction

The UNIX system and its user interface are ubiquitous in research and academic settings. There have been many variations of the interface over the years, yet few have taken advantage of a user's pattern of use. Some

variations of the UNIX shell allowed the user to recall commands from his/her history, but the burden was on the user to track when the command was issued in the user's history. While these small improvements introduced have been useful, the interface could still be drastically improved.

This paper investigates the feasibility of improving the UNIX user interface by making it adaptive, specifically with a machine learning technique. An adaptive system changes over time as it learns more about the user. To determine the feasibility of this goal, the questions that need to be answered are:

- 1) Can a UNIX shell command be predicted for a given user given a series of n previous commands?
- 2) Can a machine learning technique be used for prediction given the real-time constraints associated with a user's interaction with the UNIX shell?
- 3) Can a predicted command reduce the work performed by a user?

This study provides an initial set of answers to these questions by conducting a series of experiments. These experiments extend previous work done by Davison and Hirsh [3]. Our experiments use the Weka system, an exploratory tool developed at the University of Waikato in New Zealand. Weka provides implementations of state-of-the-art learning algorithms. It is an acronym for the Waikato Environment for Knowledge Analysis.

2. Description of the domain

The problem addressed in this paper is the task of predicting the next element in a sequence, where the sequence is made up of nominal elements. A nominal element is an unordered, non-numeric element. This type of problem is not studied often by machine learning researchers. Concept recognition is more common as is the use of independent samples from a distribution of examples. UNIX commands and user actions are not independent, and being nominal, do not fall into the domain of traditional statistical time-series analysis techniques.

Another unusual feature of this domain set is known as concept shift. The concept that is being learned changes over time, since the task the user is performing changes. As the concept shifts, predictions that were accurate previously are not correct. This means more recent sequences of commands have a higher probability of being more applicable than older ones.

3. Previous work

An *adaptive* user interface system usually classifies the current user and dependent on the expertise of the user offers a different level of assistance in performing a task. The system created by Takada ET. AL [14] is an example of a UNIX shell that provides this capability. Our paper proposes to adapt the UNIX shell so that it predicts a user's next command and allows the user to take advantage of this prediction. The UNIX shell created by Eidie [6] and the UNIX shell created by Davison and Hirsh [4] provided this functionality.

Eidie's [6] system, Valet, uses an augmented transition network to determine the action a user is trying to perform. It has an extensive knowledge base of UNIX shell commands. Once it determines the action, it can then determine the commands associated with the action. It uses its knowledge base to determine the next command the user will perform. This is different than the approach taken by Davison and Hirsh [4] and the authors of this paper.

The study performed at Rutgers University by Brian Davison and Haym Hirsh [4] took a different approach for predicting user commands. Their approach was to use a user's command history to determine the next command the user wants to perform. It created a UNIX dataset by modifying the `tcsh` command line interface in such a fashion that it collects the commands issued by a user and other pertinent information about the state of the shell. This approach allowed the collection of the data to be done passively, during a user's regular interactions with the interface.

Davison and Hirsh considered five different algorithms for prediction:

- 1) C4.5 a decision-tree learner
- 2) An Omniscient predictor capable of predicting every command, as long as it was present in the current training set. This provided an upper bound on the potential predictive accuracy of any learner.
- 3) The Most Recent Command just issued
- 4) The Most Frequently Used Command of the training set
- 5) The Longest Matching Prefix to the current command

The measurement of success is based on the micro-average and macro-average of prediction. The macro-average is the average per person predictive accuracy. A Macro-averaged result computes statistics separately for each user and then averages these statistics over all users. The micro-average is the average over all the commands in the study. A micro-averaged result computes an average over all data, by determining the number of correct predictions made across all users and dividing by the total number of commands for all users combined. The former provides equal weight to all users; since it averages performance for each user; the latter emphasizes users with large amounts of data. The most successful learning algorithm was C4.5, reaching a macro-average predictive accuracy of 38%. We chose to use the same measurement of success as used by the Davison and Hirsh [3] study. We achieve a slightly higher predictive accuracy than their study.

Our work differs from Davison and Hirsh [4] in the prediction techniques considered and tools used. The only learning technique considered by Davison and Hirsh [3] was decision trees, we consider decision trees, boosted decision trees and variations of decision tables. We chose to use the Weka system's implementation of these machine-learning techniques; Davison and Hirsh chose to use C4.5's implementation of decision trees.

We use the dataset created by Davison and Hirsh [3]; its characteristics are described in the following section. They chose to define the attributes of a command as the previously issued commands performed by the user. They ran experiments varying the length of the number of commands used as attributes to the C4.5 learner. The length varied from 1 to 4. They determined that 2

previous commands give a slightly better prediction than 1, 3, or 4. We chose to use the same number of attributes for our experiments.

4. Methodology

We performed our experiments on the dataset collected at Rutgers University. The dataset is a collection of history sessions, one history session for each person participating in the study. We converted the history sessions to be in ARFF format, ARFF format is the format accepted by all the learning techniques of the Weka system.

4.1. The Dataset

The 77 people in the study are all from Rutgers University. There are three different categories of users:

- 2 faculty members
- 5 graduate students
- 70 undergraduate students

All subjects were aware that their commands were being logged; they were given the option of explicitly turning off the history collection. Collection periods ranged from two to six months. Most subjects had access to systems on which the history collection was not taking place. The undergraduate data was collected over two months on computer systems dedicated to their programming projects. These systems were not their primary systems.

An entry in a history session consists of a timestamp, terminal emulation type and a UNIX command that was stripped of its parameters. There is one history session for each user. An example of a history session is in Figure 1.

```
96100720:13:31 green-486 vs100 BLANK
96100720:13:31 green-486 vs100 vi
96100720:13:31 green-486 vs100 ls
96100720:13:47 green-486 vs100 lpr
```

Figure 1: An Example of a history session. The command value of *BLANK* represents the start of a new session for the user. Pipes and variability in the passed arguments were not collected.

The 77 history sessions ranged from a minimum of 15 commands to a maximum of 34,940 commands. The average user had 2184(± 4389) command instances in his/her history session, using 77 (± 98) distinct commands during that time. The average length of a command is 3.77 characters.

4.2. Filters

The first step in our experiments was to create a filter, which converted the 77 different history sessions into ARFF format. ARFF format is a requirement of the Weka system. The structure of an ARFF file is the following: lines beginning with % are comments. The name of the relation is the first symbol to be defined; it is specified after the token @relation. A block defining the attributes follows the relation's definition. Each attribute's definition follows the token @attribute. Nominal attributes are followed by the set of values they can take on, enclosed by braces. Numeric attributes are followed by the keyword numeric. The following is an excerpt from an example ARFF file.

```
@relation user10
@attribute ct-2 {BLANK,exit,rcp,example,fdigner,finger,groups,uptime}
@attribute ct-1 {BLANK,exit,rcp,example,fdigner,finger,groups,uptime}
@attribute ct0 {exit,rcp,example,fdigner,finger,group,gcc,uptime}
@data
BLANK,BLANK,rcp
BLANK,rcp,example
rcp,example,fdigner
example,fdigner,finger
fdigner,finger,exit
BLANK,BLANK,uptime
BLANK,uptime,groups
uptime,groups,rcp
```

Figure 2: An example of an ARFF file. The encoding tells us that user10 had two different sessions. The first session consisted of commands: rcp, example, fdigner, finger, exit. The second session consisted of commands: uptime, groups, rcp.

5. Learning techniques and experiments

The original goal of this research was to duplicate the results of the C4.5 experiments performed by Davison and Hirsh. Then, apply boosting to the decision trees; hence achieving a higher predictive accuracy than reported in previous results. However, in working with the Weka system, the limitations of decision trees became apparent. Because of the lack of interesting results generated by the decision tree experiments, we investigated other techniques and ultimately chose to pursue an approach based on decision tables.

5.1. Decision trees experiment

The decision tree algorithm provided by Weka, J48, is a variation of C4.5 revision 8. Like all decision tree algorithms J48 uses the divide-and-conquer technique to address the classification problem. The algorithm works top-down, seeking a splitting attribute at each stage that best separates the classes, it then recursively processes the sub problems that result from that split. Like most decision tree algorithms, J48 builds the complete tree and prunes it after all data has been processed. This post-pruning approach allows the discovery of the correct combinations of the affecting attributes. The algorithm uses a pruning confidence value to determine the elimination of sub-trees. We use a value of 25% for all experiments; this is the recommended default value.

The decision trees were boosted using the AdaBoost algorithm. AdaBoost begins by assigning equal weights to all instances in the training data. It then calls the learning algorithm (in our case the decision tree algorithm) to form a classifier for the data and reweighs each instance according to the classifier's output. The weight of correctly classified instances is decreased and that of misclassified instances is increased. In the following iteration of AdaBoost, a classifier is built for the reweighed data. This classifier focuses on the misclassified instances, since they had a higher weighted value. This reassignment of weights forces the learning algorithm to concentrate on the set of instances in which it misclassified. After this iteration, the generated weights reflect how often the classifiers have misclassified the instances. To form a prediction, the output of the classifiers is combined using a weighted vote.

We found that decision trees could not be generated for history sessions with more than 2000 entries. Breaking larger history sessions into a series of smaller sized sessions and processing multiple sessions in succession addressed this; however, even when a decision tree could be created for a history session, some of the decision trees were not boosted because they only contained one classifier. When boosting was performed, it gave only a slight advantage because the number of iterations performed in boosting was small, between 1-3.

Unfortunately boosting did not give us the increased predictability we expected, this is primarily because our problem is a multi-class classification problem, and in order to achieve a benefit from boosting in this domain our weak learner (the decision trees) needs to achieve at least 50% accuracy on the misclassified instances. This did not occur frequently in our experiments; boosting only improved the accuracy of 12 decision trees in our experiments.

5.2. Decision table experiments

We chose to explore decision tables because it is a simpler, less compute intensive algorithm than the decision-tree-based approach, and hence would lend itself better to an on-line system. Decision tables allow for incremental cross-validation, this is a speedup in cross-validation. Decreasing the amount of time spent of cross-validation is an important improvement for an on-line system. Also, the number of attributes in our dataset is small; this means the attribute space searched during the creation of our decision table is small.

The Weka System provided the decision table algorithms used in our experiments. Like other decision table algorithms, the Weka decision table algorithm determines which attribute or combination of attributes needs to be included for predicting the class best. The attribute space is searched greedily either top to bottom or bottom to top. A top-to-bottom search adds attributes at each stage; this is called *forward selection*. A bottom-to-top search starts with a full set of attributes and deletes attributes one at a time; this is *backward elimination*. The Weka system performs forward selection; it uses the table's cross-validation performance to identify the best performing subset of attributes. Obtaining the cross-validation error for a decision table is a simple algorithm that involves the manipulation of counts associated with each of the table's entries. The attribute space is searched by a best-first search because this strategy is less likely to get stuck in a local maximum. The attribute selection is evaluated using a leave-one-out cross validation.

There were three different variations of decision table experiments. Two variations of the decision tables changed the algorithm for non-matching examples. One of the variations assigned a non-match to its k -nearest neighbors using the IBk algorithm. The value of k was determined automatically using leave-one-out cross-validation. The other variation assigned non-matches to the majority class. Both used 10-fold cross validation. The last variation performed a split on the data, using 66% of the data to train and the other 33% to test. Decision tables using the majority match for non-matching data produced the highest micro-average. Decision tables using the IBk algorithm generated the highest macro-average.

6. Results

Figure 4 presents the results of our experiments. The difference between the best and the worst macro-average prediction accuracy is 2.14%; we observe a similarly small difference between the best and the worst micro-average prediction (1.38%). The predictive accuracy between the most compute intensive and the least compute intensive of these algorithms is miniscule. In fact the predictive accuracy of most variations are very similar. Consider the decision tables created using the majority class for non-matching cases and the IBk variation. Their graphs are very similar, most values only disagreeing by a fraction of a percent.

A disappointing result is the lack of improvement of predictive accuracy when boosting was used with decision trees. This result is because of the small number of boosting iterations performed by the algorithm.

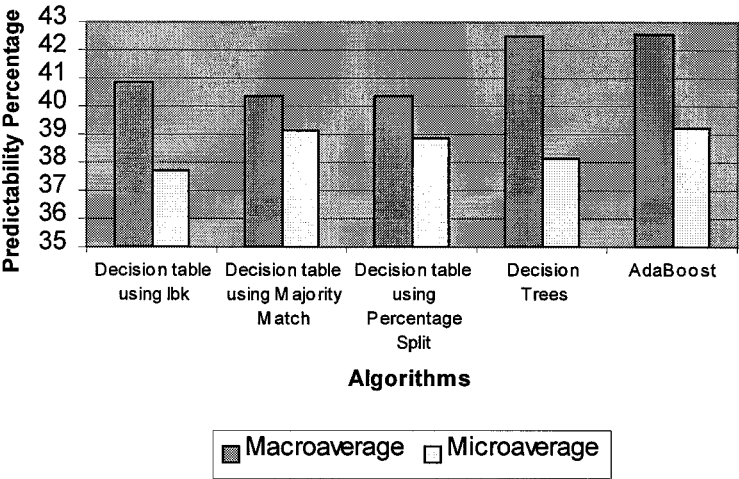


Figure 3: A Visualization of the Experimental Results. The experiments are ordered from left to right with the most compute intensive on the right.

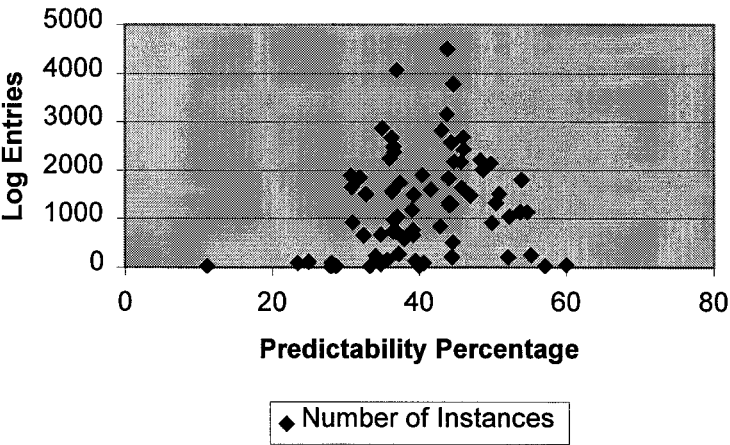


Figure 4: Predictive Accuracy of decision tables for users with history sessions less than 5000 entries; there is no increase in predictability given larger history sessions.

An interesting comparison is between the decision table IBk variation, which used 10-fold cross-validation, and the decision table IBk variation that

tested its results by training on 66% of the data and testing on the other 33%. The graphs are very similar; the only time the graphs differ substantially is when the user had fewer than 100 commands in his/her history session. Both minimums and maximums were realized on small history sessions. This implies there is a minimum startup time associated with this problem; our results imply a minimum startup of 100.

Another interesting question is the impact of the size of the history session on the predictive accuracy. Besides the obvious problem with history sessions smaller than 100 commands, there does not seem to be any benefit of using larger history sessions for predicting commands. Figure 4 shows that there is no correlation between higher predictive accuracy and larger history sessions. This result shows the impact of concept shift; larger history sessions are not more easily predicted because the task the user is trying to achieve changes as time proceeds.

7. Conclusion

The above research concludes that a UNIX command can be predicted with an average predictive accuracy of 40% using decision tables and 42% using boosted decision trees. This accuracy can be achieved on relatively small history sessions, where a small history session is defined to be history sessions smaller than 1000 entries yet larger than 100 entries. Also, for this domain, the search space for the contributing features of a decision table is relatively small. Such a problem with a small search space can be solved quickly enough to be part of an online system. Another nice feature of decision tables is the validation of a decision table is incremental; this is another reason this approach is feasible for an online system.

Predicting 40% of the commands of a user can be beneficial for a user's productivity. If a user is capable of issuing the predicted command in 1 keystroke and the average command-length is 3.77 characters, issuing 40% of the commands with 1 keystroke saves the user on average just over 30% of the keystrokes typed. Another useful feature of this system is the correction of typos. This system would recognize all of your typical typing errors and with high predictive accuracy suggest the correct spelling of the command.

8. Future directions

The results support the feasibility of an online decision table algorithm for predicting UNIX commands. Future research involves the design and implementation of the algorithm. Further experiments that vary the number of attributes should be conducted, since the current value of 3 was determined for decision trees not decision tables. Also, there may be some methods that could help us understand the total impact of concept shift. The mechanisms used by Almeida [1] to measure temporal locality could be applicable.

Acknowledgements

This research was supported in part by research grants from Compaq, IBM, Intel, Hewlett-Packard and Microsoft. We would like to thank Brian Davison for sharing this interesting dataset with us.

References

- [1] Almeida V, Bestavros A, Crovella M, de Oliveira A. *Characterizing reference locality in the WWW*. Technical Report BUCS-TR-1996-011, Boston University, Computer Science Department, June 1996.
- [2] Berry, M.J.A., and G. Linoff. *Data mining techniques for marketing, sales and customer support*. John Wiley & Sons Publications: New York and Toronto, 1997.
- [3] Davison, B.D., and Hirsh, H. *Experiments in UNIX command prediction*. Technical Report ML-TR-41, Department of Computer Science, Rutgers University, 1997.
- [4] Davison, B.D., and Hirsh, H. Predicting Sequences of User Actions, *Predicting the Future: AI Approaches to Time-Series Analysis*. Technical Report WS-98-07 Papers of the AAA/ICML 1998 Workshop, pp. 5-12, 1998.
- [5] Davison, B.D., and Hirsh, H. Probabilistic Online Action Prediction. Proceedings of the 1998 AAAI Spring Symposium on Intelligent Environments, 1998.
- [6] Eidie, E.N. *Valet: An Intelligent UNIX SHELL Interface*, Master's thesis, The University of Utah. 1995.
- [7] Elkan, C. *Boosting and Naïve Bayesian Learning*. Technical Report No CS97-557, Department of Computer Science and Engineering, University of California, San Diego, 1997.
- [8] Freund, Y., Schapire R.E. A Short Introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence* 14(5), pp. 771-780, 1998.
- [9] Holte, R.C. Very Simple Classification rules perform well on most commonly used datasets. *Machine Learning* 11, pp.63-91, 1993.
- [10] Kearns M. J. and Vazirani U.V. *An Introduction to Computational Learning Theory*. MIT Press, 1994.
- [11] Kohavi, R. The Power of Decision Tables. *Proceedings of the 1995 European Conference on Machine Learning*, pp. 174-189, 1995.
- [12] Kohavi, R., and G.H. John. Wrappers for feature subset selection. *Artificial Intelligence* 97(1-2), pp. 273-324, 1997.
- [13] Quinlan, J.R. *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufman, 1993.
- [14] Takado, M., Nishino J., Odaka, T. Ogura, H. Implementation of Adaptive Human Interface on UNIX High Performance Shell and Its Evaluation. *Information Processing Society of Japan (IPSJ) Journal* 38 No.10-005, 1998.
- [15] Witten I.H., and Frank E. *Data Mining, Practical Machine Learning Tools*
- [16] *and Techniques with Java Implementations*. Morgan Kaufmann Publishers, 2000.