The Learning Shell

Nico Jacobs
Dept. of Computer Science, K.U.Leuven
Celestijnenlaan 200A,
B-3001 Heverlee, Belgium
Nico.Jacobs@cs.kuleuven.ac.be

Abstract

In this paper we present preliminary work in the area of automated construction of user profiles. We consider the task of modeling the behavior of a user working with a command shell (such as csh or bash under the Unix operating system). We illustrate how the technique of inductive logic programming can be used to build a user profile and how this profile can be used to predict the future behavior of the user. We describe a first set of experiments to test these hypotheses and present directions for future work.

Introduction

Computers are probably the most flexible tools mankind ever built. However most users are not able to optimally use all the features computers can offer because they lack the knowledge about these features or lack the time to optimally configure their hardware and software. By combining domain knowledge about potential optimizations with a user profile describing how the user (mis-)uses the computer changes can be made to optimize the use. In this paper we focus on the automatic construction of a user profile as a set of rules in first order logic. We implemented this technique to model users of a command shell in a Unix environment and present some preliminary results on using this model to predict the user's future behavior.

Inductive Logic Programming

Inductive logic programming (ILP) (Muggleton & De Raedt 1994) combines machine learning with logic programming. ILP systems take as input a set of examples and produces an hypothesis which generalizes over these examples. Input as well as output are represented in a subset of first order logic. Also background knowledge can be used. This is information provided by a domain expert which can be used in building the output hypothesis.

We can use ILP techniques to automatically build a model of the behavior of the user. The input of the system consists of actions performed by the user together with a description of the state of the environment if relevant. In the learning from interpretations setting (Blockeel *et al.* 1999) we represent input as facts in first order logic. The output is a general description of the behavior of the user represented as a first order logic theory.

In the rest of the paper we will focus on modeling a command shell user. Although this is a very old and basic user interface, it is still widely used. The main reason for selecting this user interface is that it's very easy to log the user's actions and state of the environment when using such shells. It's also very easy to adapt the shell so that it uses the generated model. Modern graphical user interfaces (e.g. the Microsoft Windows 2000 operating system) also keep a log of all actions performed, which is comparable with what a user types in at a shell. As a result, the technique we present here can be used to generate user models from the logs of modern GUIs as well. However, the main difference is how the resulting model will be used.

We use the ILP system WARMR (Dehaspe & Toivonen 1999) to discover first order association rules in the user behavior. As input to the system we use the commands typed at the prompt of a Unix command shell.

```
% cd tex
% emacs -nw -q aaai.tex
```

These commands are parsed and transformed into logical facts. We split up the information in three facts: information about the command used, the switches used[1] and the attributes (such as file names). The first argument of each of these facts is a unique example identifier which indicates the order in which the command was entered, the second argument of switch/3 and attribute/3 predicates indicates the order of switches and attributes within the command because the order may be important (e.g. in the cp (copy)-command). In Unix it's possible to submit multiple commands one input line (using the pipe-symbol '—') in such a way that the result of the first command is used as the input for the second etc. To incorporate that in our knowledge representation scheme we can either introduce a fourth

[1]a switch is a parameter provided to a command to modify it's behavior

fact which takes as argument the identifier of a piped command, or we can add a third boolean argument to the command fact. This feature is not yet included in the prototype system.

```
command(1,'cd').
attribute(1,1,'tex').
command(2,'emacs').
switch(2,1,'-nw').
switch(2,2,'-q').
attribute(2,1,'aaai.tex').
```

Now we can start looking for regularities in these examples using WARMR. However, within one example one can only find relations between a command and it's attributes, a command and it's switches or between switches and attributes. Interesting rules describe relations between a command and the previously used commands. WARMR doesn't automatically take into account these relations, but using background knowledge we can inform WARMR about these relations by defining relations such as **nextcommand** and **recentcommand**. Background information is represented as Prolog programs, see (Bratko 1990) if you are unfamiliar with the used Prolog notation.

```
nextcommand(CommandID,Nextcommand) :-
     NextID is CommandID - 1 ,
     command(NextID,Nextcommand).
recentcommand(CommandID,Recentcommand) :-
     command(RecentCommandID,Recentcommand),
     Recentborder is RecentcommandID + 5 ,
     Recentborder > CommandID .
```

We can also use this background knowledge to 'enrich' the data that's available. One type of background knowledge is defining some sort of taxonomy of commands (emacs is an editor, editors are text tools, ...). Other information can be provided as well (is this new or old software, does it uses a GUI, network connection, or other recourses, ...)

```
command(Id,Command) :-
       isa(Origcommand,Command),
       command(Id,Origcommand).
isa(emacs,editor).
isa(vi,editor).
```

When all input data is converted and all useful background knowledge is defined, WARMR calculates the set of rules which describes all[2] frequent patterns which hold in the dataset. These rules can contain constants as well as variables.

```
IF command(Id,'ls')
   THEN switch(Id,'-l').
IF recentcommand(Id,'cd') AND command(Id,'ls')
   THEN nextcommand(Id,'editor').
```

[2]we have to provide a language bias which defines the space of rules we are interested in

For each rule we know the accuracy of the rule, the deviation (an indication of how 'unusual' the rule is[3]) and the number of examples which obey this rule.

Experiments

To test the quality of the generated user models we performed tests in which we use these models to predict the next command the user will type based on a history of 5 previous commands. This raises the problem of how the user model can be used to predict the next command a user will use, because in most situations more than one rule in the user model will be applicable, often with different conclusions. Therefor we tested 9 strategies for selecting a rule to predict the next command: using (the prediction of) the rule with the highest accuracy, the highest deviation, the highest support, the command which was predicted the most often, the command with the highest accumulated accuracy, accumulated deviation or accumulated support. We also used the command with the highest product of accuracy and deviation, and with the highest accumulated product of these two statistics.

In this experiment we used data from real users. Some of the commands typed by the user don't exist (errors), other are only very rarely used. We instructed the system to only learn rules to predict commands from a set of 30 frequent occurring commands[4]. We logged the shell use of 10 users, each log consisting of approximately 500 commands. We split up each log in a training set consisting of 66% of the examples, the remaining examples were used as a test set. We performed two experiments. In a first experiment one general user model was built from the combined training data of all the users and we tested this general model on the test data of each user individually. In a second experiment we build an individual user model for each user based on her training data and tested this on the test data for that user.

The first experiment ('global'-column in table 1) confirms that the most intuitive rule selection method (namely selecting the rule with the highest accuracy on the training set (maxacc)) leads to the best results: 36.4% of the occurrence of frequent commands was predicted correctly[5]. Using the rule with the highest deviation (maxdev), with the highest product of accuracy and deviation (maxpro) and with the highest accumulated product of accuracy and deviation (accpro) all perform almost as well. A clearly bad rule selection method is selecting the command which is predicted by the highest number of applicable rules regardless of there accuracy or deviation (count), resulting in

[3]the proportional difference between the number of examples which obey this rule and the number of examples which would obey this rule in a uniform dataset

[4]ls, cd, exit, pine, rm, vi, more, pwd, dir, make, cp, ps, finger, telnet, javac, export, pico, tt, lpq, elm, man, java, rlogin, w, mv, grep, logout, less, ssh and mail

[5]all percentages are averages over the 10 users

method	global	specific
maxacc	36.4	44.7
maxdev	34.8	43.3
maxpro	36.0	45.3
maxrul	28.3	34.4
accacc	32.5	41.3
accdev	34.0	40.1
accpro	35.5	42.5
accrul	30.2	40.0
count	20.5	31.2

Table 1: Prediction accuracies for different rule selection methods and for global as well as user specific models

a 20.4% accurate prediction. Rule selection methods based on accumulated accuracy (accacc), accumulated deviation (accdev), predictions from rules which cover most examples (maxrul) and commands which have the highest accumulated number of covered examples in the training set (accrul) all perform in between.

The second experiment — in which user models are built for each user individually — shows the same results concerning the rule selection methods: maxacc, maxdev, maxpro and accpro perform well, count performs bad and the others are in between. The prediction accuracies are significant higher than in the first experiment, and this for all rule selection methods. This shows that constructing user models for each individual separately is beneficial for the resulting user model and confirms previous results in the domain of building user models for the browsing behavior of users (Jacobs 1999).

Future Work

The system and experiments described above are only preliminary work. It illustrates that it's possible to learn user models from logs, and that it's beneficial to learn individual models for each user. However, the system still needs improvements to become practical useful.

In both experiments the best rule selection method doesn't outperform the other methods: for some users other methods perform better than the method with the overall highest accuracy. This information could be used as input for a meta-learning task: which type of rule selecting method is best suited for which type of user model. An other possible improvement of the rule selection method is by defining new methods as combinations of the above methods (the product of accuracy and deviation is already an example of such a method). Voting or weighted voting techniques could be useful.

The rules that the system builds up till now only predict the command the user will use. Often the parameters the user provides (such as filenames) can be predicted as well. Since we use inductive logic programming as a induction technique, it's easy to adapt the system such that rules can be constructed which not only predict the command but also (some of) the parameters, either as constants or as variables (in this way linking them to parameters of previous commands).

We focused on how to generate the user model and how to select rules from this model. A next step is the actual use of these models to aid the user. An obvious use of these models is intelligent command completion in which the computer tries to complete the command the user is typing. With these models this is even possible before the user has typed the first character of the new command! But other possible more useful applications of the models exist. One such application is the automatic construction of useful new commands (macros). The system can do this by selecting a 'good' rule (for example a rule with high accuracy and deviation) `if command(A) then nextcommand(B)` and creating a command C which just executes `A` and then `B`. An other application of this user model is in giving the user information about how he (mis)uses the shell. The system could e.g. detect that the user often removes file A after copying file A to B. The system can then inform the user that it's more efficient to use the `move`-command.

The log file that the system uses to induce a user model lacks information which can be useful. Up till now the file only contains a history of commands, but no information at all about the state of the environment. We can inform the system by including information such as the date, the time, the workload of the machine and the current directory at the moment that a command is invoked in the log file. This information can then be used in the construction of new rules.

Finally we considered the user model as static. In reality a user model changes often, so our system should build user models in a non-monotonic incremental way. This can be achieved with the current system by keeping a window on the log file and use those examples to construct a user model from scratch each time. But this is clearly a very inefficient approach and a truly incremental system would speed up the user model construction/maintenance.

Conclusions

We briefly described how we can use the inductive logic programming system WARMR to build user models and discussed the benefits of this approach: the use of background knowledge to provide all sorts of useful information to the induction process and an expressive hypothesis space (horn clause logic) which allows for finding rules that contain variables. We illustrated this in the domain of command shells. We also presented 9 strategies to select rules from this model for prediction. We constructed a global user model as well as individual models for each user and tested their predictive accuracy on real user data, which indicated that individual models perform significant better than a global model.

In future work we have to extend the system to predict the parameters of the commands as well, improve the rule selection method, incorporate more information about the environment and look for a truly incremental construction of the user model. Finally more attention should be paid to the use of the model to improve the user interface.

Acknowledgements: The author thanks Luc Dehaspe for his help with the WARMR system and Luc De Raedt for the discussions on this subject. Nico Jacobs is financed by a specialization grant of the Flemish Institute for the promotion of scientific and technological research in the industry (IWT).

References

Blockeel, H.; De Raedt, L.; Jacobs, N.; and Demoen, B. 1999. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery* 3(1):59–93.

Bratko, I. 1990. *Prolog Programming for Artificial Intelligence*. Addison-Wesley. 2nd Edition.

Dehaspe, L., and Toivonen, H. 1999. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery* 3(1):7–36.

Jacobs, N. 1999. Adaplix: Towards adaptive websites. In *Proceedings of the Benelearn '99 workshop.*

Muggleton, S., and De Raedt, L. 1994. Inductive logic programming : Theory and methods. *Journal of Logic Programming* 19,20:629–679.