Modeling User Behavior by Integrating AQ Learning with a Database: Initial Results

Guido Cervone and Ryszard S. Michalski*
Machine Learning and Inference Laboratory
School Computational Sciences
George Mason University
Fairfax, VA, 22030
{gcervone, michalski}@gmu.edu

*Also with the Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland

Abstract: The paper describes recent results from developing and testing LUS methodology for user modeling. LUS employs AQ learning for automatically creating user models from datasets representing activities of computer users. The datasets are stored in a relational database and employed in the learning process through an SQL-style command that automatically executes the AQ20 rule learning program and generates user models. The models are in the form of *attributional rulesets* that are more expressive than conventional decision rules, and are easy to interpret and understand. Early experimental results from the testing of the LUS method gave highly encouraging results.

Keywords: User modeling, Computer intrusion detection, Machine learning, AQ learning, Inductive databases

1 Introduction

The rapidly growing global connectivity of computer systems creates a great need for effective methods that are able to detect unauthorized use of computers. Standard methods for assuring computer security, such as passwords, gateways, and firewalls not always provide sufficient protection from unauthorized accesses. Intruders typically exploit holes in the operating system or crack password files to gain access to the computer system and masquerade as legitimate users. As a result, detection of a sophisticated intruder is increasingly difficult, especially when there are many computer users or the intruder is an insider.

The approach discussed in this paper, called *Learning User Style* (*LUS),* applies symbolic learning, specifically AQ learning, to induce typical patterns of interactions between individual users and computers. Given records measuring various characteristics of the interaction between users and computers (in our project process LUS automatically creates models of user behavior (*symbolic user*

signatures) by employing a machine learning program. The user models are in the form of rulesets relating the measured characteristics to the individual users.

The rulesets are expressed in *attributional calculus*, a highly expressive, logic-style language that can concisely represent complex relationships (Michalski, 2001). In the experiments described here, the rules are created by AQ20 learning program, which is the most recent implementation of AQ-type inductive learning. An important characteristic of AQ learning is that the generated rulesets (user signatures) are easy to interpret and understand. This means that they can be inspected and verified by experts, and hand-modified or extended, if desired.

To develop effective user models, large training and testing datasets may be needed for each user. If there are many users, the datasets to be handled may become massive. This creates an issue of how to handle such massive sets effectively both for user model creation and model testing. To address this problem, the learning system was integrated with a relational database and invoked through a *create* command of KQL, a knowledge generation language under development.

3 Basic Concepts and Terminology

To explain this research, we need to introduce some terminology. An *event* is a description of an entity or situation under consideration. In the context of user modeling, an event is vector of attribute-values that characterizes the use of computer by a user at a specific time or during a specific time period.

A *session* is a sequence of events characterizing a user's interaction with the computer from the login to the logoff. An *episode* is a sequence of events extracted from a session; it may contain just a few events, or all of the events in a session. In the training phase, it is generally desirable to use long episodes, or even whole sessions, as this helps to generate better user models. In the testing (or execution) phase, it is desirable to use short episodes, so that a user can be identified from as little information as possible.

The report by Goldring et al. (2000) indicated that one of the most relevant characteristics of the user behavior is the *mode* attribute. Therefore, in initial experiments, we have concentrated on the user model employing sequences of values of this attribute determined from the process table. Specifically, events were *n*-grams of the mode attribute, that is, sequences of *n* consecutive values of the mode attribute extracted from the data stream. The behavior of a user was characterized by a set of consecutive, overlapping *n*-grams (events) spanning a given period of user interaction with the computer.

The sequence of modes recorded in a session was transformed into a set of overlapping *n*-grams (events), each representing a sequence of *n* consecutive modes in the session's log. A set of events selected from one or more sessions of a specific user was used as a training set for learning this user's signature. In addition to a training set, a different set of testing events was created for each user

for the purpose of testing the learned model. Both training and testing sets were stored in a relational database connected to ORACLE DB through Squirrel SQL client (see Section 6).

Training sets for each user were submitted to an AQ-type symbolic learning program, AQ20, described briefly in the next section. The program generated user profiles (symbolic user signatures) in the form of *attributional rulesets*—sets of attributional rules characterizing the behavior of one user.

4 The AQ20 Learning System

In this project, we used learning system AQ20, which is the latest implementation of the AQ learning methodology. Among AQ20 features that are most important for user modeling are:

1) generation of attributional rules that are more expressive than conventional ones, and this produces more compact models

2) ability to cope with noisy data

3) ability to work with continuous data without needing discretization. (This feature has been added specifically for this project)

4) ability to learn hypotheses according to a multi-criterion optimization function

5) scalable implementation that can work efficiently with large numbers of training examples (e.g., in this project AQ20 learned from several million examples).

A discussion of an initial (incomplete) AQ20 implementation, and the results from early experiments can be found in (Cervone, Panait, and Michalski, 2001).

5 The "Create ruleset" Command

In order to seamlessly integrate inductive learning and data mining capabilities with a database, a new language is being developed, called KQL (Knowledge Query Language), which includes SQL as a subset. A major command of KQL is *Create Ruleset* that calls a learning program to create rules from a dataset selected from the database. The general form of this command is:

Create Ruleset <Output-tableset> *from* <Input-tableset> *for* <Consequent> *Using* <Parameter-table>

where

<Output-table> is a relational table that will contain the ruleset to be learned. Individual rules in the ruleset are in the form:

> Consequent <= Premise, where PREMISE is a conjunction of conditions involving one or more attributes (such conditions are called *attributional conditions* (Michalski, 2001))

<Input-table> is a relational table that stores training examples

<Consequent> may specify just one value of the output attribute, in which case it is in the form of a simple condition [output-attribute=v], or all values of the output attribute, in which it is in the form [output-attribute=*], or, simply, output-attribute. In more general form, Consequent can be a product of *attributional conditions*.

<Parameter-table> specifies all control parameters of the learning program (in this case, AQ20).

In this project, the *ruleset create* command has been implemented so far in a somewhat more specialized form. It uses Squirrel/KQL (CREATE RULES, FROM, FOR and USING are terminators).

CREATE RULES [MULTIHEAD] RuleSetFamily FROM <DATA_TABLE>
FOR <TARGET-CONCEPT> USING <PARAM_SPEC>

<DATA_TABLE> : data_table

<TARGET> : Attributional_Complex {[x=1,4,6] & [y > 6]}
| Annotted_Attibute_list
| All_attributes [Except Attribute_list]
| Target_table

Annotated_Attribute_List : Annotated_Attribute_list Annotated_Attribute
| Annotated_attribute

Annotated_Attribute : Attribute [For <attribute_values>]

<PARAMETER_SPEC> : Table of parameters and values
| ID for parameter relational table
IN here Char or Discr mode would be defined

6 Squirrel SQL Client

In this research we employed Squirrel, a complete SQL client. Squirrel is a graphical Java program that allows one to view the structure of a JDBC database (Java Database Connection), browse the data stored in relational tables, issue SQL commands, etc. The distribution of Squirrel is handled by the sourceforge network (www.sourceforge.com). The home page for the Squirrel SQL client is at www.sourceforge.net/projects/squirrel-sql/. The modifications that we have done to the Squirrel code involved a modification of the Squirrel GUI and handling of SQL queries. An option was also added to the Squirrel that allows one to import raw data in form of comma separated format (CSV). Squirrel does not come with such an option, as it was designed primarily to browse and issue SQL commands rather than import data.

The Create-ruleset command was deeply integrated with the Squirrel program. A new Java class (KQL-adapter) was created that first checks if the query is a "create ruleset" command. If it is not, the control is passed back to Squirrel, which checks if the query is a valid SQL command. If it is, KQL-adapter creates SQL queries that retrieve target data and parameters from the database, store them in

the AQ20 input file, and then runs AQ20 to generate rulesets. The resulting rules are displayed on the screen in text format.

In this project we used Oracle 8.1.7 working under the Irix operating system. The modified by us Squirrel client can be used, however, with any database for which a JDBC connection is supported, such as MySQL, mSQL, PostgreSQL, and others.

7. Datasets used in experiments

Datasets used in this experiment included information about 777 user sessions, collected from a Window operating system's process table, and characterized activities of 23 different users. The data were obtained from Dr. Thomas Goldring. Prior work done by Goldring et al. (2000) evaluated several existing methods for user modeling and indicated that an important attribute for user modeling is *mode* that characterizes the type of activity a user is engaged in at a given time, such as reading email, word processing, etc. Therefore, in our studies we also employed the *mode* attribute.

To be able to apply a learning program to a sequential data stream, the data were transformed to collections of n-grams. Given a sequence of items, an n-gram is constructed using a sliding window of size n. In our experiments we chose n=4, based on findings by Goldring et al. (2000).

The raw data were transferred into 4,808,024 4-grams characterizing 24 users, labeled from User0 to User23. A different number of sessions were extracted for each user, and each session had different length, which led to different numbers of n-grams for each users (Figure 1 and 2). Another characteristic feature of this data was that the number of distinct events (n-grams, in this case) was significantly different from the number of total events. This means is that there were many repetitions of the same n-grams in the data streams from different users.

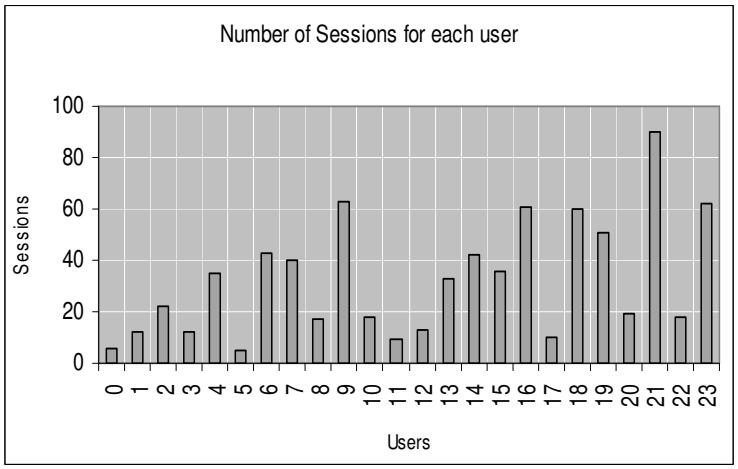


Figure 1: Number of sessions per user in the Windows dataset

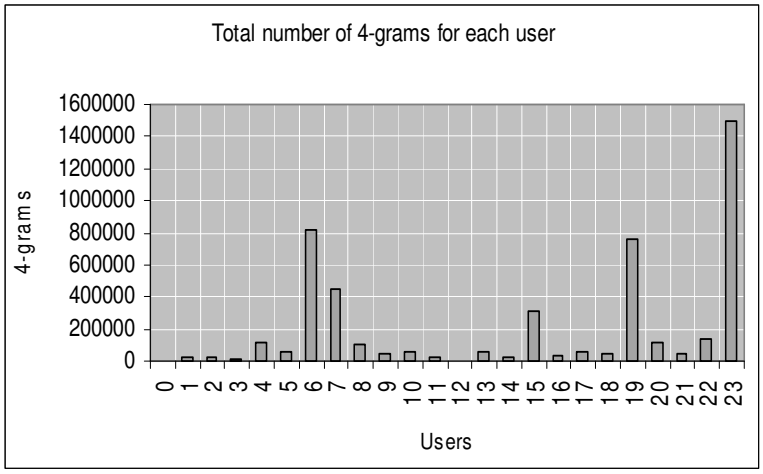


Figure 2: The number of different 4-grams per user in the dataset.

8 Creating user models and matching them against testing episodes

The original datasets were split into training and testing sets. The training set was subsequently split to different portions in order to determine the learning curve. Given a training set for each user, the AQ20 learning system learned rules models from them. These rules were subsequently tested on the testing set. The next section describes one of the experiments and obtained results.

Testing of rulesets typically involves matching single events against the learned rules. Attributional rules created by an AQ-type learning program are matched to events by the ATEST program (Reinke, 1984; Michalski and Kaufman, 2000). In the case of user modeling, to obtain meaningful results, one needs to match user models against a sequence of events (episode). To this end, a special method was developed that matches episodes with attributional rulesets and determines a score. The method was implemented in EPICn (Episode Classifier for *n*-grams).

The EPICn module calls the ATEST module for each of the distinct events in the episode, and for each decision class determines a degree of match between the corresponding ruleset R_i. and the episode. For each rule $R_{i,j}$ in ruleset i, it calculates a degree of match, c_{ijk}, between event k and rule R_{ij}, using the selected ATEST method. The degree of match between an event k and a ruleset R_i, called an *event score for class i and event k*, and denoted $\boldsymbol{EV}_{ik}$, is defined:

$$\boldsymbol{EV}_{ik} = \mathrm{Max}_{j=1\ldots s(i)} (c_{ijk} \times t_{ij}) \quad (1)$$

where t_{ij} is the number of training examples satisfying R_{ij}.

The degree of match between an episode and the ruleset R_i, called the *episode score for class i* and denoted EP_i, is defined:

$$\boldsymbol{EP_i} = \sum_{k=1}^{z} EV_{ik} \qquad (2)$$

EPICn classifies the episode to the class with the highest episode score, if the episode score is above the *score acceptance threshold* (SATH), and the difference between the highest and the next highest episode scores is greater than the *score acceptance tolerance (SATO)*. The score acceptance threshold and score acceptance tolerance allow the program to avoid making definite decisions when the episode score or the difference between the highest and the next highest episode scores are too small. In such cases, the program classifies an episode as "unknown." Up to this point, EPICn has run only with the acceptance threshold and the acceptance tolerance of 0, so that no classifications have been assigned "unknown." Since EPICn calls upon ATEST, both EPICn and ATEST have been integrated within the same program, which leads to a faster execution of the testing process.

EPICn normalizes the scores defined in (2) so that for each episode, the sum total of degrees of match is 1. The definition of the episode score as stated in (2), is one of many possible such definitions.

9 Experiment 1 (7 users):

The AQ20 allows the user to tune the learning process to the problem at hand by specifying program control parameters (Michalski and Kaufman, 2000). In the experiment described below, the control parameters were:

ambiguity = empty
mode = Theory Formation and Pattern Discovery
maxstar = 1 & maxrule = 1
LEF = (MinNumSelectors, 0.3) (MaxNewPositives, 0.1)
LEF1 = (MaxQ)(MaxNewPositives, 0.0) (MinNumSelectors, 0.0)
LEF2 = (MaxTotalQ)(MaxNewPositives, 0.0) (MinNumSelectors, 0.0)

Several experiments were performed using different combinations of parameters. In every results (characterized by predictive accuracy on the testing set) were very similar. This means that AQ20 was not very sensitive to the input parameters in this application.

For this experiment the dataset was divided into two parts, the first 80% (chronologically) of the sessions for training and the last 20% for testing.

User	Distinct +	Distinct -	Total +	Total -
0	345	5236	3573	616828
1	348	5214	20858	671154
2	784	4497	19477	570508
3	226	5253	9351	627480
4	3006	2012	92626	545656
5	142	5537	59524	647063
6	865	4413	506532	84895

Table 1: Distribution of total and distinct events.

Initially, we experimented on a smaller dataset, consisting of the data from users 0-6 only. Testing was done both on the first 50% of the testing set, and then on the full testing set based on rules generated from training sessions that used 4%, 33%, 66% and 100% of the total training data. Table 1 illustrates the large difference between distinct events and total events, and it explains why some of the rules that have rather high rule quality according to the Q(*w*) measure (Kaufman and Michalski, 1999) when this is computed using the total events appear to be quite poor when the Q is compared using distinct events.

To illustrate results obtained in this experiment, below are the first two rules from the set of 17 rules generated as a User 1 model (the rules are represented in the form of *generalized n-grams*, in which each position is occupied not by a single value but by a set):

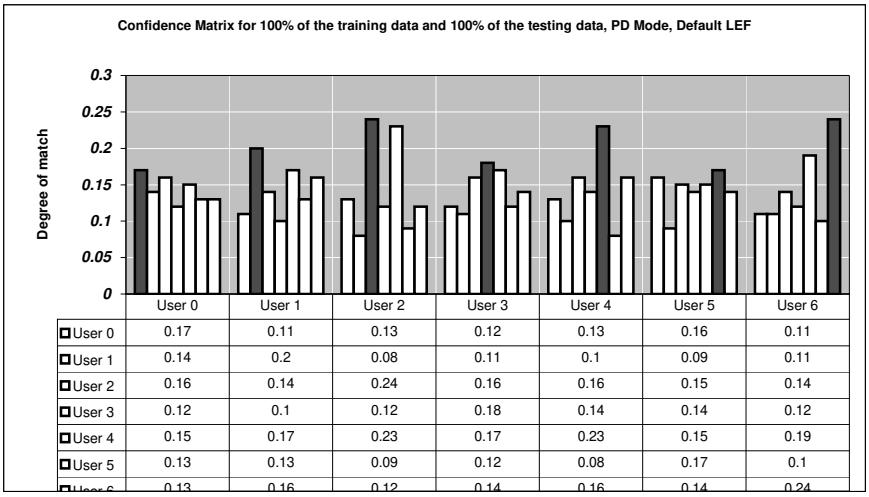
```
 # -- This learning task took: 11.92 seconds of system time
 # -- Number of rules for User 1 = 17
 # -- Number of the distinct events in the target class:          348
 # -- Number of the distinct events in the other class(es):      5214
 # -- Number of the total training events in the target class:  20858
 # -- Number of the total training in the other class(es):     671154

[User = 1]
← <{netscape,msie,telnet,explorer,web,acrobat,logon,rundll32,system,welcome,help},
    {netscape,msie,telnet,explorer,web,acrobat,logon,welcome,help}
    {netscape,msie,telnet,explorer,web,acrobat,logon,printing,welcome,dos,help}
    {netscape,msie,telnet,explorer,web,acrobat,logon,rundll32,welcome,dos,help}>
     : pd=262,nd=58,ud=118,pt=20718,nt=140,ut=3197,qd=0.607308,qt=0.986414

 ← <{netscape,telnet,office,acrobat,rundll32,welcome,help}
    {netscape,msie,telnet,web,acrobat,logon,printing,rundll32,dos,help}
    {netscape,msie,telnet,explorer,logon,rundll32,help}
    {netscape,msie,telnet,network,acrobat,printing}>
    : pd=74,nd=6,ud=6,pt=16565,nt=17,ut=7,qd=0.195631,qt=0.79334
```

The rules were learned by AQ20 from all events in the training set, running in the PD mode using LEF1 rule selection criterion. The first rule states that User 1 behavior is characterized by a set of 4-grams, in which the first position is occupied by any mode from the first set {netscape, msie, telnet, …}, the second

position is occupied by any mode from the second set {netscape, msie, ..help}, etc. This rule thus describes compactly 11979 4-grams.

The lines marked by # provide supplementary information about the experiment. The first line gives information about the system (kernel) time spent on learning the user model (from 20858 training examples). The next line specifies the number of rules learned for User 1. Lines 3-6 specify numbers of different example types used in the experiment.
Each rule is accompanied by annotations that represent various characteristics of the rule. Parameters pd and pt represent the number of distinct positive examples and the total positive examples, respectively, that are covered by the rule. Similarly, nd and nt represent the number of distinct negative and the total negative examples, respectively, covered by the rule. Parameters qd and qt indicate the rule quality measure, which takes into consideration both the number of positives covered out of all positives, and the number of negatives covered out of all negatives in the dataset. The difference between qd and qt is that qd is computed over distinct positives and distinct negatives, whereas qt is computed over the total positives and total negatives.

Figure 2 describes the performance of user models on the testing data. The darkened column indicates the matching score for the correct user model. As figure shows, in every testing case the correct user model was indicated.



	User 0	User 1	User 2	User 3	User 4	User 5	User 6
User 0	0.17	0.11	0.13	0.12	0.13	0.16	0.11
User 1	0.14	0.2	0.08	0.11	0.1	0.09	0.11
User 2	0.16	0.14	0.24	0.16	0.16	0.15	0.14
User 3	0.12	0.1	0.12	0.18	0.14	0.14	0.12
User 4	0.15	0.17	0.23	0.17	0.23	0.15	0.19
User 5	0.13	0.13	0.09	0.12	0.08	0.17	0.1
User 6	0.13	0.16	0.12	0.14	0.16	0.14	0.24

Figure 2: Confidence matrix for rules learned from the complete training set.

In order to determine how sensitive is performance to the size of the training set, we have performed experiments in which the learning set was varied from 4%, 33%, 66% and 100% of the training data. The results are shown in Figure 3. As the figure shows, the perfomance was about .6 (60%) correct when the training set had only 4% of the events (random guessing is about 14% correct).

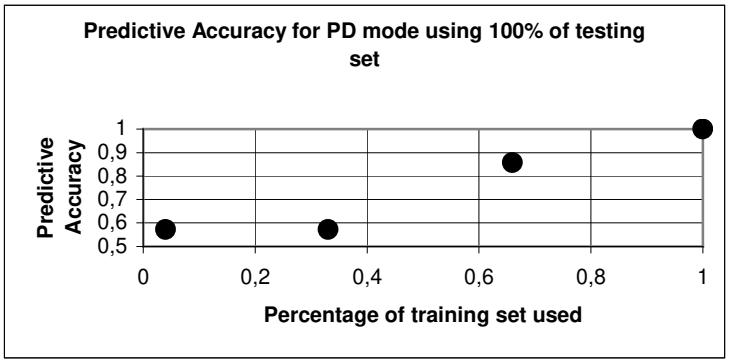
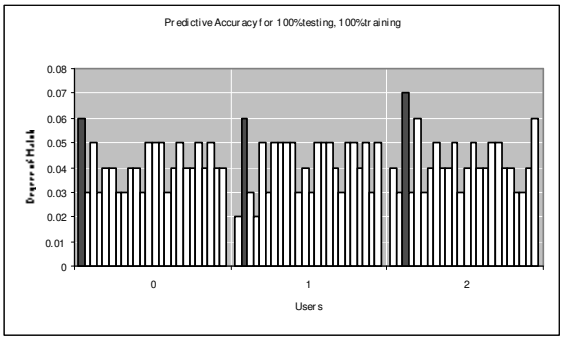
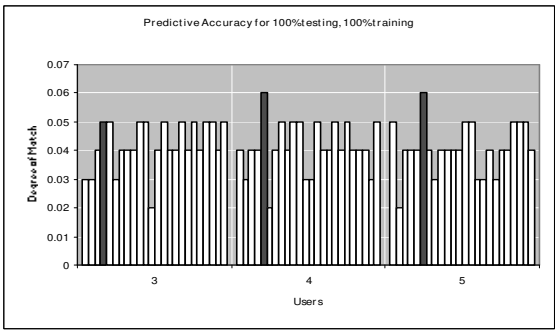


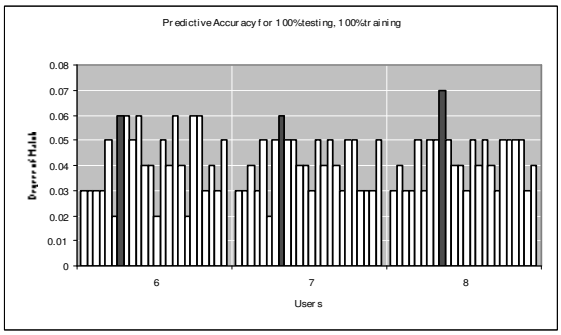
Figure 3: The learning curve for 7 users.

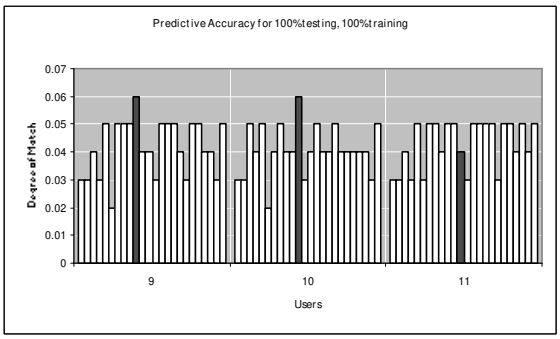
10 Experiment 2 (24 users)

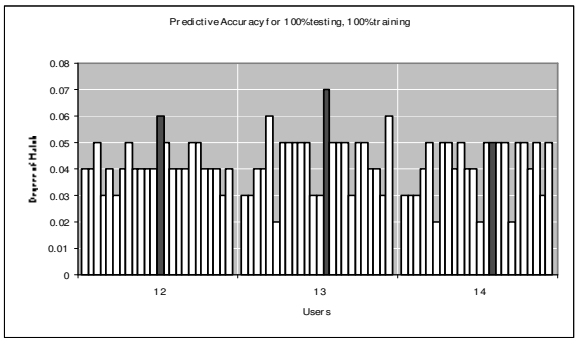
The experiment involved learning user models for 24 users, using nearly 5 million training examples (the complete training set). Results were tested on approximately one million testing examples (the complete testing set). Results are illustrated in Figure 4. As before, darkened columns represent the matching score for the correct user model. As the figure shows, all users were classified correctly except one, User 11. This seems to be due to the fact that the dataset for user 11 had only a small number of sessions and a very small number of events per session (Figure 1 and 2). In some cases, e.g., for users 6 and 14, the matching score was the same for the correct models as for a few other models. This indicates insufficient discrimination.

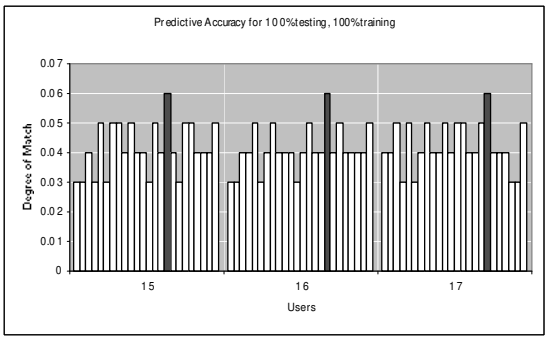

Predictive Accuracy for 100%testing, 100%training
Degree of Match
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0
1
2
Users

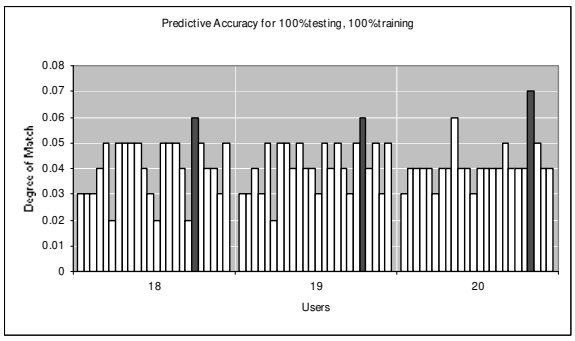

Predictive Accuracy for 100%testing, 100%training
Degree of Match
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
3
4
5
Users

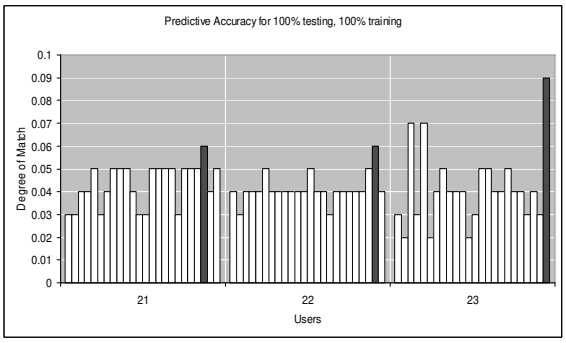

Predictive Accuracy for 100%testing, 100%training
Degree of Match
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
6
7
8
Users


Predictive Accuracy for 100%testing, 100%training
Degree of Match
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
9
10
11
Users









Figures 4-10: Results from testing 24 user models.

To illustrate the rules obtained in this experiment, below a selection of the rules learned for User 0. The learning time was larger in experiment 1, as expected, since have here 24 users rather than 7.

```
  # -- This learning took:
  # -- System time 767.15 sec
  # -- Number of rules for this class = 52

  # -- Number of distinct training events in the target class:
346
  # -- Number of distinct training events in other classes:
71,931
  # -- Total number of training events in the target class:
1,826
  # -- Total number of training events in the other classes:
       3,750,169

[user=0]
    ← <{explorer,install,multimedia,system,time},
        {multimedia,system},
        {explorer,install,system},
        {explorer,install,multimedia,system},
        :pd=64,nd=31,ud=8,pt=916,nt=404,ut=11,qd=0.124322,qt=0.348035

    ← <{explorer,install,office,rundll32,system,time},
        {multimedia,system},
        {install,multimedia,rundll32,system,time},
        {explorer,install,rundll32,system,time}>
        :pd=68,nd=42,ud=9,pt=919,nt=73,ut=11,qd=0.121131,qt=0.466232

    ←
<{explorer,help,install,mail,multimedia,rundll32,system,time,web},
        {help,install,logon,mail,office,rundll32,system,time,web},
        {help,install,mail,office,printing,rundll32,system,time,web},
        {help,install,rundll32,system,time}

:pd=140,nd=343,ud=41,pt=1316,nt=701,ut=66,qd=0.1159,qt=0.470102

    ← <{install,office,printing,system},
        {install,rundll32,time},
        {install,multimedia,office,sql,system,web},
        {explorer,install,multimedia,rundll32,system,web}
        :pd=43,nd=4,ud=2,pt=397,nt=4,ut=2,qd=0.11,qt=0.21
```

10 Conclusion

The presented LUS method employs AQ20 learning program to learn user models from n-grams representing interactions between users and the computer. In view of the large datasets involved in this application, to make the learning and testing processes easier to handle, the learning systems was deeply integrated with a relational database, accessible through Squirrel, an SQL client. The obtained results for a small number of users (7) indicated perfect recognition rate. In the

case of a larger number of users (24), there was one misclassification, which was likely due to a small number of training examples used.

Acknowledgments

Authors thank Dr. Goldring for providing datasets used in this study and for consultation on the n-gram approach to user modeling, and Dr. Kenneth A. Kaufman for his assistance and feedback in conducting this research. They also thank Dr. Menas Kafatos and Dr. Ruxin Yang for providing access to their mighty esip computer system that was used for storing datasets and running experiments.

Valuable help was also given by Colin Bell and all the members of the Squirrel development team. They helped solving many problems, and give valuable information on where to modify the code.

We also wish to thank the School of Computational Sciences for providing other computational equipment and logistic space that was used during the development of this project.

References

Bloedorn, E. and Michalski, R.S., "Data Driven Constructive Induction in AQ17-PRE: A Method and Experiments," *Proceedings of the Third International Conference on Tools for AI,* San Jose, CA, November 9-14, 1991.

Cervone G., Panait L. A., Michalski R.S., "The Development of the AQ20 Learning System and Initial Experiments," *Proceedings of the International Conference on Intelligent Systems (IIS 2000),* Poland, July 2001.

Michalski, R.S. and Kaufman, K., "Building Knowledge Scouts Using KGL Metalanguage," *Fundamenta Informaticae* 40, pp. 433-447, 2000a.

Kaufman, K.A. and Michalski, R.S., "An Adjustable Rule Learner for Pattern Discovery Using the AQ Methodology," *Journal of Intelligent Information Systems*, 14, pp. 199-216, 2000b.

Michalski, R.S, "A Theory and Methodology of Inductive Learning, in *Machine Learning: An Artificial Intelligence Approach*, Michalski, R.S, Carbonell, J.G. and Mitchell, T.M. (Eds.), Tioga Publishing Company, 1983, pp. 83-134.

Michalski, R.S., and Chilausky, R.L., "Learning By Being Told and Learning From Examples: An Experimental Comparison of the Two Methods of Knowledge Acquisition in the Context of Developing an Expert System for Soybean Disease Diagnosis," *Policy Analysis and Information Systems*, Vol. 4, No. 2, 1980.

Wnek, J. and Michalski, R.S., "Hypothesis-Driven Constructive Induction in AQ17: A Method and Experiments," *Reports of the Machine Learning and Inference Laboratory*, MLI 91-4, School of Information Technology and Engineering, George Mason University, Fairfax, VA, May 1991.