User Modeling: Through Statistical Analysis and an Evolving Classifier.

Jose A. Iglesias, Plamen Angelov, Agapito Ledezma and Araceli Sanchis

***Abstract*— Knowledge about computer users is very beneficial for assisting them, predicting their future actions or detecting masqueraders. In this paper, an approach for creating and recognizing automatically the behavior profile of a computer user is combined with an evolving method to keep up to date the created profiles. The behavior of a computer is represented in this research as the sequence of commands s/he types during a period of time. This sequence is treated using statistical methods in order to create the corresponding user profile. However, as a user profile is usually not fixed but rather it changes and evolves, we propose a user profile classifier based on *Evolving Systems*. This paper describes briefly the model creation method and the evolving classifier, which are compared with well established off-line and on-line classifiers.**

I. Introduction

Recognizing the behavior of users in real-time is a significant challenge in different tasks, such as to predict their behavior, to coordinate with them or to assist them. Computer user modeling is the process of learning about ordinary computer users by observing the way the use the computer. The result of a user modeling in computer systems is usually stored in user profiles that contains information that characterizes the usage behavior of a computer user. This information may be used in many areas, such as interaction with users appropriately [1] or computer-aided instruction [2].

Experience has shown that users themselves do not know how to articulate what they do, especially if they are very familiar with the tasks they perform. Computer users, like all of us, leave out activities that they do not even notice they are doing. Thus, only by observing users we can model his/her behavior correctly [3]. However, the construction of effective computer user profiles is a difficult problem because of the following aspects: human behavior is usually erratic, and sometimes humans behave differently because of a change in their goals. The latter problem makes necessary that the user profiles we create evolve.

Most existing techniques for computer user modeling assume the availability of carefully hand-crafted user profiles, which encode the *a-priori* known behavior repertoire of the observed user. Unfortunately, techniques for automatically acquiring user profiles from observations are only beginning to emerge.

A user model may take different forms depending on the purposes for which it is created. As Webb et al. propose [4], user models may seek to describe:

Plamen Angelov is with the Department of Communication Systems, InfoLab21, Lancaster University, UK. E-mail:p.angelov@lancaster.ac.uk

Jose A. Iglesias, Agapito Ledezma and Araceli Sanchis are with the CAOS Group, Carlos III University of Madrid, Spain. E-mails:{jiglesia,ledezma,masm}@inf.uc3m.es.

1) the cognitive processes that underlie the users actions;
2) the difference between the users skills and expert skills;
3) the users behavioral patterns or preferences; or
4) the user characteristics.

Recent research has predominantly pursued the third approach, focusing on users behaviors as proposed by Webb [5], rather than on the cognitive processes that underlie that behavior. This research is also focused on finding relevant users behavioral patterns in order to classify a user behavior. We will use the approach presented in [6] for automatically creating the profile of a user based on the analysis of the sequence of commands s/he typed.

Once many profiles have been created, in order to classify a new user during run-time, most existing algorithms match the observed behavior of a user against a profile-library, and matches are reported as hypotheses. However, as a user profile is not necessarily fixed but rather it evolves/changes, we use in this research the evolving classification method proposed in [7]. This approach based on *Evolving Systems* [8] classifies an observed user and the profiles created are always up to date. In this research, this method is exhaustively analyzed and compared in detail with many other classification methods. A novel study about the ability of this approach to adapt to new data is also presented.

This paper is organized as follows; Section 2 provides a brief overview of the background and related work relevant to this research. How the computer user profiles are created from a sequence of commands is explained in section 3. In section 4, it is detailed the evolving user classifier. Section 5 describes the experimental setting and the experimental results obtained. Finally, Section 6 contains future work and concluding remarks.

II. Background and Related Work

To model, recognize, or classify the behavior of a computer user is very useful in many different computer areas: Discovery of navigation patterns [9], Web recommender systems [10] or Computer security [11]. For this reason, the literature of agent modeling is truly vast. However, in this research we focus on discovering computer user patterns from the sequence of commands a user types. The problem of behavior classification is examined as a problem of learning to characterize the behavior of a user in terms of sequences of commands.

As in this research, there are other areas in which sequential data are analyzed in order to solve a specific problem. In general, the sequence learning problem can be categorized in

978-1-4244-8126-2/10/$26.00

four basic categories: sequence prediction, sequence generation, sequence classification and sequential decision making. In this paper, the sequence classification is the category analyzed and developed.

Considering the sequence classification, the main reason to need to handle sequential data is because of the observed data from some environments are inherently sequential. An example of these data is the DNA sequence. Ma et al. [12] present new techniques for bio-sequence classification. In a very different problem (computer intrusion detection problem), Coull et al. [13] propose an algorithm that uses pairwise sequence alignment to characterize similarity between sequences of commands. The algorithm produces an effective metric for distinguishing a legitimate user from a masquerader. Schonlau et al. [14] investigate a number of statistical approaches for detecting masqueraders.

A very important issue in sequence learning is temporal dependencies. The following aspect is essential in our research: A current situation or the action that an agent performs usually depend on what has happened before. This aspect is taken into account in our research and in models such as HMMs; however, there are some other models which have problems dealing with such dependencies. For example, recurrent neural network models or reinforcement learning can not manage efficiently the long-range dependencies.

III. Creating a computer user profile

In this research we consider that the commands typed by a user are usually influenced by past experiences. This aspect motivates the idea of automated sequence learning for behavior classification; if we do not know the features that influence the behavior of an agent, we can consider a sequence of past actions to incorporate some of the historical context of the agent. Indeed, sequence learning is arguable the most common form of human and animal learning. Sequences are absolutely relevant in human skill learning and in high-level problem solving and reasoning [15]. Taking this aspect into account in this paper, the computer user modeling is transformed into a sequence analysis problem where a sequence of commands represents a specific behavior. This transformation can be done because it is clear that any behavior has a sequential aspect, as actions are performed in a sequence.

The commands typed by a computer user are inherently sequential, and this sequentiality is considered in the modeling process (when a user types a command, it usually depends on the previous typed commands and it is related to the following commands). According to this aspect, in order to get the most representative set of subsequences from a sequence, we propose the use of a *trie* data structure [16]. This *trie* data structure was also used in [17], [18] where a team behavior was learned as well as in [19] to classify the behavior patterns of a *RoboCup* soccer simulation team.

The construction of a user profile from a single sequence of commands is done as it is explained in [6] for any sequence of events. This process consists of three steps: 1. Segmentation of the sequence of commands, 2. Storage of the subsequences in a *trie*, and 3. Creation of the user profile. These steps are detailed in the following 3 subsections.

These steps are detailed in the following 3 subsections. For the sake of simplicity, let us consider the following sequence as an example: $\{ls \rightarrow date \rightarrow ls \rightarrow date \rightarrow cat\}$.

A. Segmentation of the sequence of commands

First, the sequence is segmented into subsequences of equal length from the first to the last element. Thus, the sequence $A=A_1A_2...A_n$ (where n is the number of commands of the sequence) will be segmented in the subsequences described by $A_i...A_{i+length}\ \forall\ i,i=[1,n\text{-}length+1]$, where *length* is the size of the subsequences created and this value determines how many commands are considered as dependent. In the remainder of the paper, we will use the term *subsequence length* to denote the value of this length. In addition, in this case, the value of this term represents how many commands a user usually types consecutively as part of his/her behavior pattern.

In the proposed sample sequence ($\{ ls \rightarrow date \rightarrow ls \rightarrow date \rightarrow cat\}$), let 3 be the subsequence length, then we obtain:

$\{ls \rightarrow date \rightarrow ls\}$, $\{date \rightarrow ls \rightarrow date\}$, $\{ls \rightarrow date \rightarrow cat\}$

B. Storage of the subsequences in a trie

The subsequences of commands are stored in a *trie* in a way that all possible subsequences are accessible and explicitly represented. A node of a *trie* represents a command, and its *children* represent the commands that follow it. Also, each node keeps track of the number of times a command has been inserted into it. When a new subsequence is inserted into a *trie*, the existing nodes are modified and/or new nodes are created. Moreover, as the dependencies of the commands are relevant in the user profile, the subsequence suffixes (subsequences that extend to the end of the given sequence) are also inserted.

Considering the previous example, the first subsequence ($\{ls \rightarrow date \rightarrow ls\}$) is added as the first branch of the empty *trie* (Figure 1 a). Each node is labeled with the number 1 which indicates that the command has been inserted in the node once (in Figure 1, this number is enclosed in square brackets). Then, the suffixes of the subsequence ($\{date \rightarrow ls\}$ and $\{ls\}$) are also inserted (Figure 1 b). Finally, after inserting the three subsequences and its corresponding suffixes, the completed *trie* is obtained (Figure 1 c).

C. Creation of the user profile

Once the *trie* is created, the subsequences that characterize the user profile and its relevance are calculated by traversing the *trie*. For this purpose, frequency-based methods are used. In particular, to evaluate the relevance of a subsequence, its relative frequency or support [20] is calculated. In this case, the support of a subsequence is defined as the ratio of the number of times the subsequence has been inserted into the *trie* to the total number of subsequences of equal size inserted.

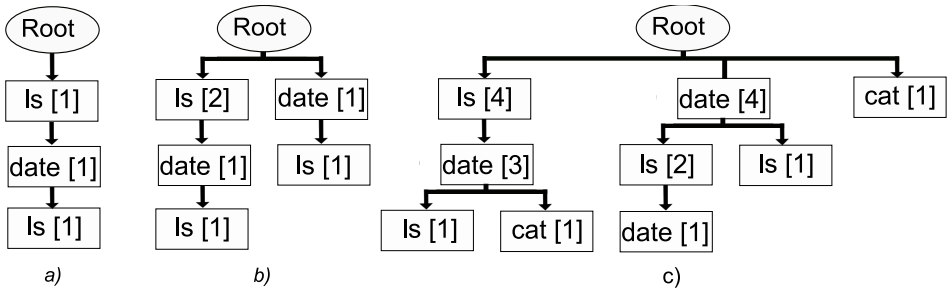

Fig. 1. Steps of creating an example trie.

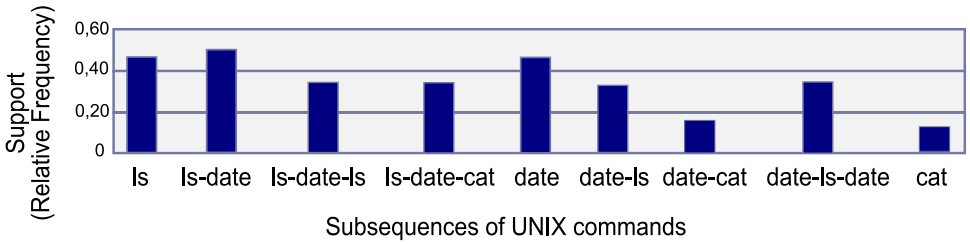

Fig. 2. Distribution of subsequences of commands - Example.

Thus, in this step, the *trie* can be transformed into a set of subsequences labeled by its support value. This set of subsequences is represented as a distribution of relevant subsequences. In the previous example, the *trie* consists of 9 nodes; therefore, the corresponding profile consists of 9 different subsequences which are labeled with its support. Figure 2 shows the distribution of these subsequences.

Once a user behavior profile has been created, it is classified and used to update the *Evolving Profile Library* (*EPLib*), as explained in the next section.

IV. Evolving Computer User Classifier

A classifier is a mapping from the feature space to the class label space. In the proposed evolving classifier, the feature space is defined by distributions of subsequences of commands. On the other hand, the class label space is represented by the most representative distributions. Thus, a distribution in the class label space represents a specific profile which is one of the prototypes of the evolving library *EPLib*. These prototypes are not fixed and evolve taking into account the new information collected on-line from the data stream - this is what makes the classifier *Evolving*. The number of these prototypes is not pre-fixed but it depends on the homogeneity of the observed sequences.

The following subsections describes how a user profile is represented by the proposed classifier, called EvABCD - *Evolving Agent Behavior Classifier Based on Distributions of commands*, and how this classifier is created in an evolving manner.

A. User behavior representation

First, EvABCD converts the sequence of commands into the corresponding distribution of subsequences on-line. In order to classify a user profile, these distributions must be represented in a data space. For this reason, each distribution is considered as a data vector that defines a *point* that can be represented in the data space.

The data space in which these *points* can be represented should consist of n dimensions, where n is the number of the different subsequences observed. It means that we should know all the different subsequences of the environment *a priori*. However, this value is unknown and the creation of this data space from the beginning is not efficient. For this reason, in EvABCD, the dimension of the data space is incrementally growing according to the different subsequences that are represented in it.

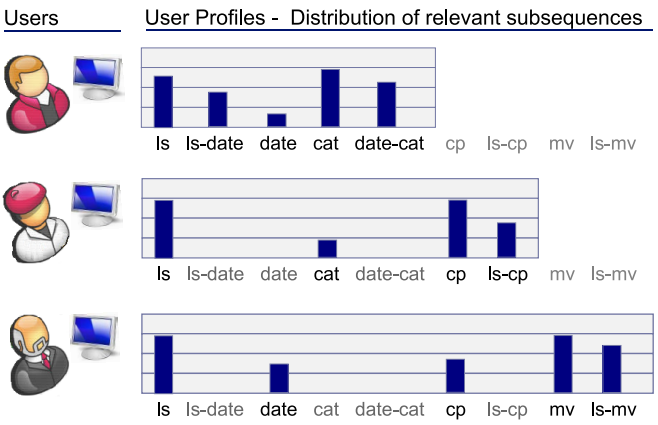

Fig. 3. Distributions of subsequences of commands in an evolving system approach - Example

Figure 3 explains graphically this idea. In this example, the distribution of the first user consists of 5 subsequences of commands (*ls, ls-date, date, cat* and *date-cat*), therefore we need a 5 dimensional data space to represent this distribution because each different subsequence is represented by one dimension. If we consider the second user, we can see that

3 of the 5 previous subsequences have not been typed by this user (*ls-date, date* and *date-cat*), so these values are not available. Also, there are 2 new subsequences (*cp* and *ls-cp*) and the representation of these values in the same data space needs to increase the dimensionality of the data space from 5 to 7. To sum up, the dimensions of the data space represent the different subsequences typed by the users and they will increase according to the different new subsequences obtained.

B. Structure of the classifier EvABCD

Once the corresponding distribution has been created, it is processed by the classifier. This classifier does not need to be configured according to the environment where it is used because it can start '*from scratch*'. Also, the relevant information of the obtained samples is necessary to update the library; but, as we will explain in the next subsection, there is no need to store all the samples in it. The structure of this classifier includes:

1) **Classify the new sample** in a user group represented by a prototype.
2) **Calculate the *potential*** of the new data sample to be a prototype.
3) **Update all the prototypes** considering the new data sample. It is done because the *density* of the data space surrounding certain data sample changes with the insertion of each new data sample. **Insert the new data sample** as a new prototype if needed.
4) **Remove** any prototype if needed.

Next 4 subsections explain each step of this evolving classification method.

*1) **Classify the new sample**:* In order to classify a new data sample, we compare it with all the prototypes stored in *EPLib*. This comparison is done using cosine distance and the smallest distance determines the closest similarity. This aspect is considered in equation (1).

$$\begin{aligned} &Class(x_z) = Class(Prot^*); \\ &Prot^* = MIN_{i=1}^{NumProt}(cosDist(x_{Prototype_i}, x_z)) \end{aligned} \tag{1}$$

The time-consumed for classifying a new sample depends on the number of prototypes and its number of attributes. However, we can consider, in general terms, that both the time-consumed and the computational complexity are reduced and acceptable for real-time applications (in order of milliseconds per data sample).

*2) **Calculate the potential of a new data sample**:* As in [21], a prototype is a data sample (a computer user profile represented by a distribution of subsequences of commands) that groups several samples which represent a certain class. The classifier is initialized with the first data sample, which is stored in *EPLib*. Then, each data sample is classified to one of the prototypes (classes) defined in the classifier. Finally, based on the *potential* of the new data sample to become a prototype [22], it could form a new prototype or replace an existing one.

The potential (P) of the k^{th} data sample (x_k) is calculated by the equation (2) which represents a function of the accumulated distance between a sample and all the other k-1 samples in the data space [21]. The result of this function represents the *density* of the data that surrounds a certain data sample.

$$P(x_k) = \frac{1}{1 + \frac{\sum_{i=1}^{k-1} distance(x_k, x_i)}{k-1}} \tag{2}$$

where *distance* represents the distance between two samples in the data space.

In [23] the potential is calculated using the euclidean distance and in [21] it is calculated using the cosine distance. Cosine distance has the advantage that it tolerates different samples to have different number of attributes (in this case, an attribute is the support value of a subsequence of commands). Also, cosine distance tolerates that the value of several subsequences in a sample can be null (null is different than zero). Therefore, EvABCD uses the cosine distance (*cosDist*) to measure the similarity between two samples; as it is described in equation (3).

$$cosDist(x_k, x_p) = 1 - \frac{\sum_{j=1}^{n} x_{kj} x_{pj}}{\sqrt{\sum_{j=1}^{n} x_{kj}^2 \sum_{j=1}^{n} x_{pj}^2}} \tag{3}$$

where x_k and x_p represent the two samples to measure its distance and *n* represents the number of different attributes in both samples.

Note that the expression in the equation (2) requires all the accumulated data sample available to be calculated, which contradicts to the requirement for real-time and on-line application needed in the proposed approach. For this reason, in [21] it is developed a recursive expression cosine distance. This formula is as follows:

$$\begin{aligned} &P_k(z_k) = \frac{1}{2 - \frac{1}{k-1} \frac{1}{\sqrt{\sum_{j=1}^{n} (z_k^j)^2}} B_k}; k = 2, 3, ...; P_1(z_1) = 1 \\ &where \;\; B_k = \sum_{j=1}^{n} z_k^j b_k^j \;;\; b_k^j = b_{(k-1)}^j + \sqrt{\frac{(z_k^j)^2}{\sum_{l=1}^{n} (z_k^l)^2}} \\ &and \;\; b_1^j = \sqrt{\frac{(z_1^j)^2}{\sum_{l=1}^{n} (z_1^l)^2}} \;;\; j = [1, n+1] \end{aligned} \tag{4}$$

Using this expression, it is only necessary to calculate *(n+1)* values where *n* is the number of different subsequences obtained; this value is represented by b, where $b_k^j, j = [1, n]$ represents the accumulated value for the k^{th} data sample.

*3) **Creating new prototypes**:* The proposed evolving user behavior classifier, EvABCD, can start '*from scratch*' (without prototypes in the library) in a similar manner as *eClass* evolving fuzzy rule-based classifier proposed in [23], used in [24] for robotics and further developed in [21]. The potential of each new data sample is calculated *recursively*

and the potential of the other prototypes is updated. Then, the potential of the new sample (z_k) is compared with the potential of the existing prototypes. A new prototype is created if its value is higher than any other existing prototype, as shown in equation (5).

$$\exists i,\ i = [1, NumPrototypes] :\ P(z_k) > P(Prot_i) \quad (5)$$

Thus, if the new data sample is not relevant, the overall structure of the classifier is not changed. Otherwise, if the new data sample has high descriptive power and generalization potential, the classifier evolves by adding a new prototype which represents a part of the observed data samples.

4) ***Removing existing prototypes****:* After adding a new prototype, we check whether any of the already existing prototypes are described *well* by the newly added prototype [21]. By *well* we mean that the value of the membership function that describes the closeness to the prototype is a Gaussian bell function chosen due to its generalization capabilities:

$$\exists i,\ i = [1, NumPrototypes] :\ \mu_i(z_k) > e^{-1} \quad (6)$$

For this reason, we calculate the membership function between a data sample and a prototype which is defined as:

$$\mu_i(z_k) = e^{-\frac{1}{2}[\frac{cosDist(z_k, Prot_i)}{\sigma_i}]},\ i = [1, NumPrototypes] \quad (7)$$

where cosDist($z_k, Prot_i$) represents the cosine distance between a data sample (z_k) and the i^{th} prototype ($Prot_i$); σ_i represents the spread of the membership function, which also symbolizes the radius of the zone of influence of the prototype. This spread is determined based on the scatter [25] of the data. The equation to get the spread of the k^{th} data sample is defined as:

$$\sigma_i(k) = \sqrt{\frac{1}{k}\sum_{j=1}^{k} cosDist(Prot_i, z_k)};\quad \sigma_i(0) = 1 \quad (8)$$

where *k* is the number of data samples inserted in the data space; *cosDist($Prot_i, z_k$)* is the cosine distance between the new data sample (z_k) and the i^{th} prototype.

However, to calculate the scatter without storing all the received samples, this value can be updated (as shown in [23]) recursively by:

$$\sigma_i(k) = \\ = \sqrt{[\sigma_i(k-1)]^2 + \frac{[cosDist^2(Prot_i, z_k) - [\sigma_i(k-1)]^2]}{k}} \quad (9)$$

V. Experimental Setup and Results

In order to evaluate EvABCD in a command interface, we use a data set with the *UNIX* commands typed by 168 real users and labeled in 4 different groups. In this section we will detail the experimental results obtained and the comparison with other different classification methods.

A. Data Set

For these experiments, we use the command-line data collected by Greenberg [26] using *UNIX csh* command interpreter. In this data, four target groups were identified, representing a total of 168 male and female users with a wide cross-section of computer experience and needs. Salient features of each group are described below, and the sample sizes (the number of people observed) are indicated in table I.

- ***Novice Programmers***: The users of this group had little or no previous exposure to programming, operating systems, or *UNIX*-like command-based interfaces. These users spent most of their time learning how to program and use the basic system facilities.
- ***Experienced Programmers***: These group members were senior Computer Science undergraduates, expected to have a fair knowledge of the *UNIX* environment. These users used the system for coding, word processing, employing more advanced *UNIX* facilities to fulfill course requirements, and social and exploratory purposes.
- ***Computer Scientist***: This group, graduates and researchers from the Department of Computer Science, had varying experience with *UNIX*, although all were experts with computers. Tasks performed were less predictable and more varied than other groups, research investigations, social communication, word-processing, maintaining databases, and so on.
- ***Non-programmers***: Word-processing and document preparation was the dominant activity of the members of this group, made up of office staff and members of the Faculty of Environmental Design. Knowledge of *UNIX* was the minimum necessary to get the job done.

TABLE I
SAMPLE GROUP SIZES AND COMMAND LINES RECORDED.

Group of users name	Sample size	Total number of command lines
Novice Programmers	55	77423
Experienced Programmers	36	74906
Computer Scientists	52	125691
Non-Programmers	25	25608
Total	**168**	**303628**

B. Experimental Design

Although the proposed classifier can be used in real-time, in order to measure its performance using the above data set, the 10-fold cross-validation thecnique is used. all the users (training set) are divided into 10 disjoint subsets with equal size. Each of the 10 subsets is left out in turn for evaluation. It should be emphasized that EvABCD does not need to work in this model. This is done mainly in order to have comparable results with very different techniques.

The number of *UNIX* commands typed by a user and used for creating his/her profile, is very relevant in the classification process. When EvABCD is carried out in the field, the behavior of a user is classified (and the evolving behavior library updated) after s/he types a limited number

of commands. In order to show the relevance of this aspect using the data set already described, we consider sequences of different number of *UNIX* commands for creating the user profile: 100, 500 and 1000 commands per user. Also, if the number of users increases, the number of different subsequences increases, too.

In the phase of user profile creation, the length of the subsequences in which the original sequence is segmented (used for creating the *trie*) is a relevant parameter: using long subsequences, the time consumed for creating the *trie* and the number of relevant subsequences of the corresponding distribution increase drastically. In the experiments presented in this paper, the *subsequence length* varies from 2 to 6.

In addition, we should consider that the number of subsequences obtained using different sequences of data is often very large. To get an idea of how this number increases, table II shows the number of different subsequences obtained using different number of commands for training (100, 500 and 1000 commands per user) and subsequence lengths (3 and 5).

TABLE II
TOTAL NUMBER OF DIFFERENT SUBSEQUENCES OBTAINED.

Number of commands per user	Subsequence Length	Number of different subsequences
100	3	11451
	5	34164
500	3	39428
	5	134133
1000	3	63375
	5	227715

Using EvABCD, the number of prototypes to create per each group is not fixed, it varies automatically depending on the heterogeneity of the data. To get an idea about this aspect, table III tabulates the number of different prototypes per group created in each of the 10 runs using 1000 commands per user as training dataset and a subsequence length of 3.

TABLE III
EVABCD: NUMBER OF PROTOTYPES CREATED PER GROUP USING 10-FOLD CROSS-VALIDATION

	Number of prototypes in each of the 10 runs									
Group	1	2	3	4	5	6	7	8	9	10
Novice Progr.	4	5	3	4	4	3	2	3	4	5
Exp. Progr.	1	1	1	1	2	3	2	1	1	2
Comp. Scientists	1	1	1	1	1	1	2	2	2	2
Non-Progr.	1	1	1	1	1	1	1	1	1	1

C. Results

In order to evaluate the performance of EvABCD, we compare it with incremental and non-incremental classifiers: Naive Bayes [27], k-Nearest Neighbor [28] (incremental and non-incremental), and C5.0 (a commercial version of C4.5 [29], [30]).

For this purpose, these classifiers were trained using a feature vector for each user (168 samples). This vector consists of the support value of all the different subsequences obtained for all the users; thus, there are a lot of subsequences which do not have a value because the corresponding user has not typed those commands. In this case, in order to be able to use this data for training the classifiers, we consider the value 0 (although its real value is *null*).

Table IV shows the percentage of users correctly classified into its corresponding group using different number of commands for training (100, 500 and 1000 commands per user) and subsequences lengths for segmenting the initial sequence (from 2 to 6).

According to this data, we can see that the percentages of users correctly classified by our approach are better than the obtained by *K-NN* but worse than the obtained by *Naive Bayes*. For small subsequences length (2 or 3) the difference between EvABCD and *Naive Bayes* is considerable; but this difference decreases if this length is longer (5 or 6). In general, these results show that the proposed classifier works well in this kind of environments when the subsequence length is around 5. However, EvABCD is suitable in the proposed environment because it does not need to store the entire data stream in the memory and disregards any sample after being used. In addition, EvABCD is one-pass (each sample is proceeded once at the time of its arrival), while non-incremental classifiers are offline algorithms which require a batch set of training data in the memory and make many iterations. For this reason, EvABCD is computationally simple and efficient as it is recursive and one pass. In fact, because the number of attributes is very large in a real environment, the proposed approach EvABCD is the best working alternative.

In addition to the advantage that EvABCD is an online classifier, we want to prove the ability of our approach to adapt quickly to new data (in this case, new users or a different behavior of a user). For this purpose, we design a new experimental scenario in which the number of commands used as training set for each user in a *new class* is incremented to detect how the different classifiers *recognize* the users of that *new* class. Thus, using the same experimental setup explained above, firstly the users of the *Novice Programmers* group will be added incrementally in the training data. In this case, we also use 10-fold cross-validation, hence the process finishes when the training data contains the 90% of the users of that group. The sequence of commands used for creating the profile of a user contains 1000 commands and it is created using a subsequence length of 5.

The first graph of the Figure 4 shows the results of this experiment: *x-axis* represents the number of *Novice Programmers* users that have been considered in the training data, and *y-axis* represents the classification rate. In the graph we can see how quickly EvABCD evolves and adapts to the new class. Only with 3 users of the new class, *our* method is able to create a *new* prototype in which almost 90% of the *test users* are classified correctly. However, the other non-incremental classifiers need a higher number of samples for

TABLE IV

RESULTS OF *UNIX* USER CLASSIFICATION - COMPARATIVE. #1: NUMBER OF COMMANDS IN A TRAINING SEQUENCE. #2: SUBSEQUENCE LENGTH.

		Classifier and classification rate (%)					
			Incremental Classifier		Non Incremental Classifier		
#1	#2	EVABCD	Naive Bayes Incremental	k-NN Incremental (k=1)	C5.0	Naive Bayes	k-NN (k=1)
100	2	65,5	77,3	38,3	73,9	79,1	42,8
	3	64,9	76,1	36,5	69,6	79,7	39,8
	4	64,5	72	34,1	74,6	74,4	39,2
	5	67,9	73,2	32,3	68,6	75	33,3
	6	64,3	73,2	32,3	70,1	77,3	32,7
500	2	58,3	77,9	33,9	73,9	82,1	41,9
	3	59,5	73,8	36,9	74,6	77,3	39,8
	4	59,2	70,8	39,2	73,9	76,7	38,9
	5	66,7	71,4	35,1	73,6	76,1	37,3
	6	70,8	72,6	35,7	75,6	75,5	36,9
1.000	2	60,1	78,5	43,6	73,9	85,1	44,1
	3	65,5	77,9	44,0	74,6	81,5	47,3
	4	61,3	77,3	43,5	73,9	79,1	46,1
	5	70,2	76,7	42,5	73,6	78,5	44,0
	6	72,0	76,1	41,9	74,6	77,9	44,6

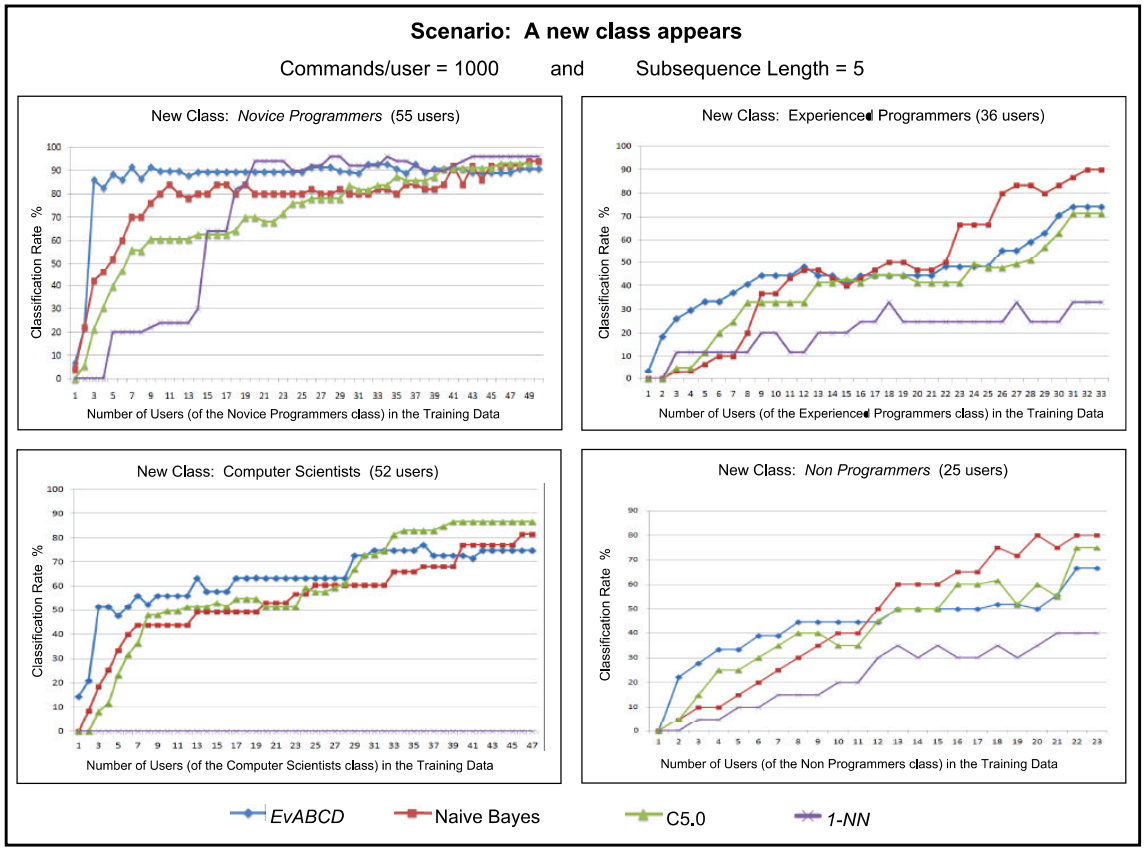

Fig. 4. Evolution of the classification rate during online learning with a subset of users data set. - Comparison with non-incremental classifiers.

recognizing the users of that *new class*. The increase in the classification rate is not perfectly smooth because the new data bring useful information but also noise. The other three graphs of the Figure show similar performance for the other 3 groups.

VI. Conclusions

In this paper we describe a generic approach to model and classify automatically computer users from the sequence of commands s/he types during a period of time. However, as a user profile is usually not fixed but rather it changes and evolves, we have proposed a user profile classifier able to keep up to date the created profiles based on *Evolving Systems*. This evolving classifier is one pass, non-iterative, recursive and it has the potential to be used in an interactive mode; therefore, it is computationally very efficient and fast.

The test results with a data set of 168 real *UNIX* users demonstrates that, using an appropriate subsequence length, EvABCD can perform almost as well as other well established off-line classifiers in terms of correct classification on validation data. However, the proposed classifier is able to adapt extremely quickly to new data.

Although it is not addressed in this paper, the proposed method can be also used to monitor, analyze and detect abnormalities based on a time varying behavior of same users and to detect masqueraders. It can also be applied to other type of users such as users of e-services, digital communications, etc.

References

[1] F. Nasoz and C. L. Lisetti, "Affective user modeling for adaptive intelligent user interfaces." in *HCI (3)*, ser. Lecture Notes in Computer Science, J. A. Jacko, Ed., vol. 4552. Springer, 2007, pp. 421–430.

[2] L. Barrow, L. Markman, and C. E. Rouse, "Technology's edge: The educational benefits of computer-aided instruction," National Bureau of Economic Research, Working Paper 14240, August 2008.

[3] J. T. Hackos and J. C. Redish, *User and Task Analysis for Interface Design*. Wiley, 1998.

[4] G. I. Webb, M. J. Pazzani, and D. Billsus, "Machine learning for user modeling," *User Modeling and User-Adapted Interaction*, vol. 11, no. 1, pp. 19–29, March 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1011117102175

[5] G. I. Webb, "Feature based modelling: A methodology for producing coherent, consistent, dynamically changing models of agents competency," in *Proceedings of the 1993 World Conference on Artificial Intelligence in Education (AI-ED'93)*, P. Brna, S. Ohlsson, and H. Pain, Eds. Charlottesville, VA: AACE, 1993, pp. 497–504.

[6] J. A. Iglesias, A. Ledezma, and A. Sanchis, "Creating user profiles from a command-line interface: A statistical approach," in *UMAP 2009: Proceedings of the 1st and 7th International Conference on User Modeling, Adaptation, and Personalization*, ser. LNCS, vol. 5535. Springer, June 2009, pp. 90–101.

[7] J. A. Iglesias, P. Angelov, A. Ledezma, and A. Sanchis, "Modelling evolving user behaviours," in *IEEE Workshop on Evolving and Self-Developing and Self-Developing Intelligent Systems, ESDIS 2009*, 2009, pp. 16–23.

[8] P. Angelov, *Rule-based Models: A Tool for Design of Flexible Adaptive Systems*. Heidelberg, New York: Springer-Verlag, 2002.

[9] M. Spiliopoulou and L. C. Faulstich, "Wum: A web utilization miner," in *In Proceedings of EDBT Workshop WebDB98*. Springer Verlag, 1998, pp. 109–115.

[10] A. A. Macedo, K. N. Truong, J. A. Camacho-Guerrero, and M. da GraÇa Pimentel, "Automatically sharing web experiences through a hyperdocument recommender system," in *HYPERTEXT 2003*. New York, NY, USA: ACM, 2003, pp. 48–56.

[11] D. L. Pepyne, J. Hu, and W. Gong, "User profiling for computer security," in *Proc. American Control Conference*, 2004, pp. 982–987.

[12] Q. Ma, J. T.-L. Wang, D. Shasha, and C. H. Wu, "Dna sequence classification via an expectation maximization algorithm and neural networks: a case study." *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 31, no. 4, pp. 468–475, 2001.

[13] S. E. Coull, J. W. Branch, B. K. Szymanski, and E. Breimer, "Intrusion detection: A bioinformatics approach," in *ACSAC*, 2003, pp. 24–33.

[14] M. Schonlau, W. DuMouchel, W. Ju, A. Karr, M. Theus, and Y. Vardi, "Computer intrusion: Detecting masquerades," 2001, statistical Science.

[15] J. Anderson, *Learning and Memory: An Integrated Approach*. New York: John Wiley and Sons., 1995.

[16] E. Fredkin, "Trie memory," *Comm. A.C.M.*, vol. 3, no. 9, pp. 490–499, 1960.

[17] J. A. Iglesias, A. Ledezma, and A. Sanchs, "Sequence classification using statistical pattern recognition." in *IDA*, ser. LNCS, vol. 4723. Springer, 2007, pp. 207–218.

[18] G. A. Kaminka, M. Fidanboylu, A. Chang, and M. M. Veloso, "Learning the sequential coordinated behavior of teams from observations," in *RoboCup*, ser. Lecture Notes in Computer Science, vol. 2752. Springer, 2002, pp. 111–125.

[19] J. A. Iglesias, A. Ledezma, and A. Sanchis, "A comparing method of two team behaviours in the simulation coach competition," in *MDAI*, ser. LNCS, vol. 3885. Springer, 2006, pp. 117–128.

[20] R. Agrawal and R. Srikant, "Mining sequential patterns," in *International Conference on Data Engineering*, Taiwan, 1995, pp. 3–14.

[21] P. Angelov and X. Zhou, "Evolving fuzzy rule-based classifiers from data streams," *IEEE Transactions on Fuzzy Systems: Special issue on Evolving Fuzzy Systems*, vol. 16, no. 6, pp. 1462–1475, 2008.

[22] P. Angelov and D. Filev, "An approach to online identification of takagi-sugeno fuzzy models," *Systems, Man, and Cybernetics, Part B, IEEE Transactions on*, vol. 34, no. 1, pp. 484–498, Feb. 2004.

[23] P. Angelov, X. Zhou, and F. Klawonn, "Evolving fuzzy rule-based classifiers," *Computational Intelligence in Image and Signal Processing, 2007. CIISP 2007. IEEE Symposium on*, pp. 220–225, April 2007.

[24] X. Zhou and P. Angelov, "Autonomous visual self-localization in completely unknown environment using evolving fuzzy rule-based classifier," *Computational Intelligence in Security and Defense Applications, 2007. CISDA 2007. IEEE Symp.*, pp. 131–138, April 2007.

[25] P. Angelov and D. Filev, "Simpl_eTS: a simplified method for learning evolving Takagi-Sugeno fuzzy models," *The IEEE International Conference on Fuzzy Systems. FUZZ-IEEE 2005.*, vol. -, pp. 1068–1073, May 2005.

[26] S. Greenberg, "Using unix: Collected traces of 168 users," Master's thesis, Department of Computer Science, University of Calgary, Alberta, Canada, 1988.

[27] I. Rish, "An empirical study of the naive bayes classifier," in *Proceedings of IJCAI-01 Workshop on Empirical Methods in Artificial Intelligence*, 2001.

[28] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[29] J. R. Quinlan, *C4.5: programs for machine learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[30] J. Quinlan, "Data mining tools see5 and c5.0," 2003 [online], available: http://www.rulequest.com/see5-info.html.