

An Evolving Framework for Clustering Computer Users



Jose A. Iglesias, Agapito Ledezma and Araceli Sanchis

Abstract—The idea of clustering computer users is very beneficial for making recommendations to a user based on the histories of other users with similar preferences, detecting changes in the behavior of a user, and so on. However, computer users have different needs as they learn to use a software system or their goals changes. Although there are several approaches for clustering users, most of them do not consider the changes in their behavior. In this paper, we present an approach for clustering automatically the behavior profile of a computer user and an evolving method based on *Evolving Systems* to keep up to date the created profile clusters.

I. INTRODUCTION

The idea of clustering computer users is very beneficial for making recommendations to a user based on the histories of other users with similar preferences, detecting changes in the behavior of a user, and so on. In addition, 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 usually leave out activities that they do not even notice they are doing. Thus, only if we can observe users we can model his/her behavior correctly [1]. However, the construction of an effective computer user profile is normally a difficult problem because a 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 profile clusters we create change and evolve.

Computer users have different needs as they learn to use a software system or their goals changes. Moreover, users needs change as they use a system and become more familiar with the task domain and its capabilities. Recent growing interest in modeling user behavior has generated an increase in research efforts on computer systems which can automatically alter aspects of their functionality to suit the needs of individuals or groups of users. However, these systems are still difficult to develop, and there are many challenges to be overcome. In this paper, we face one of the challenges of the user clustering: the creation of user clusters which can be updated dynamically.

Recent researches on user modeling have predominantly pursued users behavioral patterns or preferences, rather than on the cognitive processes that underlie that behavior. This research is also focused on this aspect in order to cluster a user behavior. We will use the approach presented in [2] for automatically creating the profile of a user based on the analysis of the sequence of commands s/he typed.

Although there are several approaches for clustering users, most of the user profile groups do not change according to the environment and new goals of the users. In this research, it is proposed an adaptive approach for creating behavior profiles in order to cluster these users.

This paper is organized as follows: Section 2 provides a brief overview of the background and related work relevant to this research. Section 3 describes the construction of the user behavior profile. The evolving clustering is explained in Section 4. Finally, section 5 contains future work and concluding remarks.

II. BACKGROUND AND RELATED WORK

Different methods have been used to find out relevant information under the computer user behavior in different computer areas:

Discovery of navigation patterns: Spiliopoulou et al. [3] present the *Web Utilization Miner WUM*, a mining system for discovering navigation patterns in web sites.

Web recommender systems: Macedo et al. [4] propose a system (*WebMemex*) that provides recommended information based on the captured history of navigation from a list of known users. *WebMemex* captures information such as IP addresses, user Ids and URL accessed for future analysis.

Web page filtering: Godoy and Amandi [5] present a technique to generate readable user profiles that accurately capture interests by observing their behavior on the Web. The proposed technique is built on the *Web Document Conceptual Clustering* algorithm, with which profiles without an a priori knowledge of user interest categories can be acquired.

Computer security: Pepyne et al. [6] describe a method using queuing theory and logistic regression modeling methods for profiling computer users based on simple temporal aspects of their behavior.

In this paper, as sequences are very relevant in human skill learning and reasoning [7], the problem of user profile clustering is examined as a problem of sequence clustering. According to this aspect, Horman and Kaminka [8] present a learner with unlabeled sequential data that discover meaningful patterns of sequential behavior from example streams.

III. CREATING A COMPUTER USER PROFILE

In this research, as it was explained in [2], 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 clustering; 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

This work is partially supported by the Spanish Government under project TRA2007-67374-C02-02

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.

learning. Sequences are absolutely relevant in human skill learning and in high-level problem solving and reasoning [9]. 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 [10], [11].

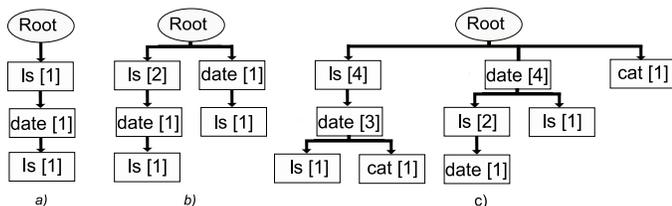


Fig. 1. Steps of creating an example trie.

The construction of a user profile from a single sequence of commands 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 $A=A_1A_2\dots A_n$ (where n is the number of commands of the sequence) will be segmented in the subsequences described by $A_i\dots A_{i+length} \forall i,i=[1,n-length+1]$, 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 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 [12] 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.

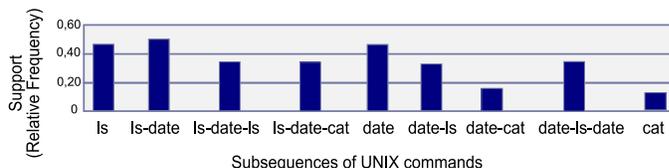


Fig. 2. Distribution of subsequences of commands - Example.

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 in a cluster and used to update the *Evolving Clustering Library (ECLib)*, as explained in the next section.

IV. EVOLVING COMPUTER USER CLUSTERING

Clustering is the process of grouping data objects into clusters so that objects within a cluster have a high similarity in comparison to one another, but are very dissimilar to objects in other clusters. In this case, the clustering method is devised to group computer users by taking into account their distributions similarity in a feature space. The feature space is defined by distributions of subsequences of commands. Thus, a distribution in the feature space represents a specific profile which is one of the clusters (prototypes) of the evolving

clustering library *ECLib*. 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 clustering method *Evolving*. The number of these prototypes is not prefixed but it depends on the homogeneity of the observed sequences. This clustering method is based in the evolving classifier proposed in [13].

The following subsections describes how a user profile is represented by the proposed clustering methods, and how the proposed clustering method work.

A. User behavior representation

First, the sequence of commands is converted into the corresponding distribution of subsequences on-line. In order to cluster 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.

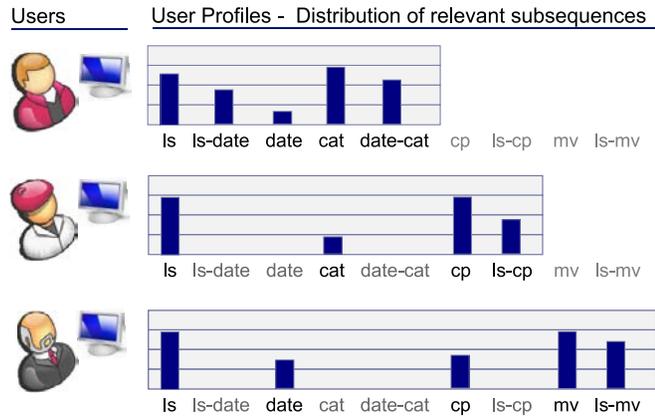


Fig. 3. Distributions of subsequences of commands in an evolving system approach - Example

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 this approach, the dimension of the data space is incrementally growing according to the different subsequences that are represented in it. Figure 3 explains graphically this idea, where 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 clustering method

Once the corresponding distribution has been created, it is processed by the clustering method, which 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 clustering method includes:

- 1) **Calculate the *potential*** of the new data sample to be a prototype (cluster).
- 2) **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.
- 3) **Remove** any prototype if needed.

Next 3 subsections explain each step of this evolving clustering method.

1) **Calculate the *potential of a new data sample***: As in [14], 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 cluster. The cluster method is initialized with the first data sample, which is stored in *ECLib*. Then, based on the *potential* of the new data sample to become a prototype, 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 (1) which represents a function of the accumulated distance between a sample and all the other $k-1$ samples in the data space. The result of this function represents the *density* of the data that surrounds a certain data sample.

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

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

In [14] the potential 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, the proposed method uses the cosine distance (*cosDist*) to measure the similarity between two samples; as it is described in equation (2).

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

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 (1) 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 [14] it is developed a recursive expression cosine distance. This formula is as follows:

$$P_k(z_k) = \frac{1}{2 - \frac{1}{k-1} \frac{1}{\sqrt{\sum_{j=1}^n (z_k^j)^2}} B_k}; k = 2, 3, \dots; P_1(z_1) = 1$$

$$\text{where } B_k = \sum_{j=1}^n z_k^j b_k^j; b_k^j = b_{(k-1)}^j + \sqrt{\frac{(z_k^j)^2}{\sum_{l=1}^n (z_k^l)^2}}$$

$$\text{and } b_1^j = \sqrt{\frac{(z_1^j)^2}{\sum_{l=1}^n (z_1^l)^2}}; j = [1, n+1] \quad (3)$$

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.

2) Creating new prototypes (clusters): The proposed evolving user behavior clustering method can start 'from scratch' (without prototypes in the library) in a similar manner as *eClass* evolving fuzzy rule-based classifier developed in [14]. 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 (4).

$$\exists i, i = [1, NumPrototypes]: P(z_k) > P(Prot_i) \quad (4)$$

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

3) Removing existing prototypes (clusters): After adding a new prototype, we check whether any of the already existing prototypes are described *well* by the newly added prototype [14]. 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:

$$\exists i, i = [1, NumPrototypes]: \mu_i(z_k) > e^{-1} \quad (5)$$

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

$$\mu_i(z_k) = e^{-\frac{1}{2} \left\{ \frac{\cosDist(z_k, Prot_i)}{\sigma_i} \right\}^2}, i = [1, NumPrototypes] \quad (6)$$

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 [15] of the data. The equation to get the spread of the k^{th} data sample is defined as:

$$\sigma_i(k) = \sqrt{\frac{1}{k} \sum_{j=1}^k \cosDist(Prot_i, z_k)}; \sigma_i(0) = 1 \quad (7)$$

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 [16]) recursively by:

$$\sigma_i(k) = \sqrt{[\sigma_i(k-1)]^2 + \frac{[\cosDist^2(Prot_i, z_k) - [\sigma_i(k-1)]^2]}{k}} \quad (8)$$

V. CONCLUSIONS

In this paper we describe a generic approach to model and cluster automatically computer users from the sequence of commands s/he types during a period of time. As a user profile is usually not fixed but rather it changes and evolves, we have proposed a user profile cluster method able to keep up to date the created clusters based on *Evolving Systems*. This evolving cluster method 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. Although the evaluation of this approach is not easy, we have conducted several experiments that can help us to know that we are going in a promising direction.

REFERENCES

- [1] J. T. Hackos and J. C. Redish, *User and Task Analysis for Interface Design*. Wiley, February 1998.
- [2] J. A. Iglesias, A. Ledezma, and A. Sanchis, "Creating user profiles from a command-line interface: A statistical approach," in *Conf. on User Modeling, Adaptation, and Personalization*, 2009, pp. 90–101.
- [3] M. Spiliopoulou and L. C. Faulstich, "Wum: A web utilization miner," in *In Proceedings of EDBT Workshop WebDB98*, 1998, pp. 109–115.
- [4] 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.
- [5] D. Godoy and A. Amandi, "User profiling for web page filtering," *IEEE Internet Computing*, vol. 9, no. 4, pp. 56–64, 2005.
- [6] D. L. Pepyne, J. Hu, and W. Gong, "User profiling for computer security," in *Proc. American Control Conference*, 2004, pp. 982–987.
- [7] J. Anderson, *Learning and Memory: An Integrated Approach*. New York: John Wiley and Sons., 1995.
- [8] Y. Horman and G. A. Kaminka, "Removing biases in unsupervised learning of sequential patterns," *Intelligent Data Analysis*, vol. 11, no. 5, pp. 457–480, 2007.
- [9] R. Sun, E. Merrill, and T. Peterson, "From implicit skills to explicit knowledge: a bottom-up model of skill learning," *Cognitive Science*, vol. 25, no. 2, pp. 203–244, 2001.
- [10] E. Fredkin, "Trie memory," *Comm. A.C.M.*, vol. 3, no. 9, pp. 490–499, Sept. 1960.
- [11] J. A. Iglesias, A. Ledezma, and A. Sanchis, "Sequence classification using statistical pattern recognition," in *Proceedings of Intelligence Data Analysis*. Springer, 2007, pp. 207–218.
- [12] R. Agrawal and R. Srikant, "Mining sequential patterns," in *International Conference on Data Engineering*, 1995, pp. 3–14.
- [13] J. A. Iglesias, P. Angelov, A. Ledezma, and A. Sanchis, "Modelling evolving user behaviours," in *IEEE Workshop on Evolving and Self-Developing Intelligent Systems*, 2009, 2009, pp. 16–23.
- [14] 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, p. to appear, 2008.
- [15] P. Angelov and D. Filev, "Simpl_ets: a simplified method for learning evolving takagi-sugeno fuzzy models," *Fuzzy Systems, FUZZ '05. The 14th IEEE International Conference on*, pp. 1068–1073, 2005.
- [16] 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, 2007.