Evolving Systems (2010) 1:161–171
DOI 10.1007/s12530-010-9008-8

ORIGINAL PAPER

Evolving classification of agents' behaviors: a general approach

Jose Antonio Iglesias · Plamen Angelov ·
Agapito Ledezma · Araceli Sanchis

Received: 12 April 2010 / Accepted: 1 July 2010 / Published online: 17 July 2010
© Springer-Verlag 2010

Abstract By recognizing the behavior of others, many different tasks can be performed, such as to predict their future behavior, to coordinate with them or to assist them. If this behavior recognition can be done automatically, it can be very useful in many applications. However, an agents' behavior is not necessarily fixed but rather it evolves/changes. Thus, it is essential to take into account these changes in any behavior recognition system. In this paper, we present a general approach to the classification of streaming data which represent a specific agent behavior based on evolving systems. The experiment results show that an evolving system based on our approach can efficiently model and recognize different behaviors in very different domains, in particular, UNIX command-line data streams, and intelligent home environments.

Keywords Evolving fuzzy systems · Agent modeling · Behavior classification

J. A. Iglesias (✉) · A. Ledezma · A. Sanchis
Carlos III University of Madrid, Avda. Universidad, 30, Leganés, 28914 Madrid, Spain
e-mail: jiglesia@inf.uc3m.es

A. Ledezma
e-mail: ledezma@inf.uc3m.es

A. Sanchis
e-mail: masm@inf.uc3m.es

P. Angelov
InfoLab21, Lancaster University, South Drive, Lancaster LA1 4WA, UK
e-mail: p.angelov@lancaster.ac.uk

1 Introduction

Recent theories claim that a high percentage of the human brain capacity is used for predicting the future, including the behavior of other humans (Mulcahy and Call 2006). Recognizing the behavior of others in real-time is a significant challenge in different tasks, such as to predict their activity, state or future actions, to coordinate with them or to assist them.

In a multi-agent system, it is important for agents (software agents, robots or humans) to recognize other agents' internal states (selected behaviors, plans, intentions or goals). Specifically, behavior recognition is the task of recognizing the unobservable behavior-based state of an agent, given a stream of observations of its interaction with its environment. The focus here is on recognizing patterns (possibly, multiple patterns) in the stream, that would allow its classification. This is in contrast to other agent modeling tasks, where the entire sequence of observed actions is to be recognized and matched against the plan library [e.g., to predict goals (Hong 2001), or identify the sequence of actions that compose a plan (Tambe and Rosenbloom 1995; Carrbery 2001; Steffens 2002; Ledezma et al. 2004)]. Many existing techniques for behavior recognition assume the availability of carefully hand-crafted libraries, which encode the a priori known behavioral repertoire of the observed agents. During run-time, different algorithms match the observed behavior of the agents against the libraries, and successful matches are reported as hypotheses.

An agent is capable of acting in the environment, and the agent changes the environment with its actions. However, the behavior of the agent in the environment usually changes for different reasons: the agent can learn how to act optimally through experience with the environment, the goals of the agent can be modified, the environment can change and the behavior of the agent can change as appropriate to it, and so on.

Techniques for automatically acquiring behavior models from observations (e.g. by learning or data-mining), are only beginning to emerge, and there are many challenges to be overcome. In this paper, we face one of the challenges of the agent behavior modeling: the creation of user behavior models which can be updated dynamically.

We present an approach to behavior classification based on sequence classification. This approach represents the behavior of an agent as a distribution over sequences of observed atomic events, where such sequences have been identified during training as statistically significant. However, as the behavior of an agent is not necessarily fixed, this approach is also based on *Evolving Systems* that allows for the agent models to be dynamic, to evolve.

2 Background and related work

To model, recognize, or classify the behavior of an agent is very useful in many different areas. We, thus, focus in this section the most relevant work in behavior classification.

Han and Veloso (1999) recognize behaviors of robots using *Hidden Markov Models* and their approach is evaluated in a real world scenario. In this case, states in the *HMMs* correspond to an abstracted decomposition of a robot's behavior. This approach makes a Markovian assumption (the probability of moving from the current state to another is independent of its previous states) in modeling an agent, whereas our proposal takes into account a short sequence of events to incorporate some of the historical context of the agent.

Riley and Veloso (2000) propose a classification of the adversary behavior into predefined adversary classes in the domain of simulated robotic soccer. The behavior of the opponent is modeled by useful features based on the areas in which the soccer events occur. The system accumulates adversary position information in grids and then a decision tree is used for classifying it. In contrast, the approach we present in this paper examines the temporal ordering of events, but for the most part ignores their location. This consideration is a complementary approach.

Instead of describing the complete opponent behavior, Steffens (2002) presents a feature-based declarative opponent-modeling (*FBDOM*) technique which identifies tactical moves of the opponent in multi-agent systems. In this case, the models built need distinct and stable features which describe the behavior of opponents. However, it does not discover sequences.

Kaminka et al. (2002) recognize basic actions based on descriptive predicates, and learn relevant sequences of actions using a statistical approach. Horman and Kaminka (2007) expanded on this approach. A similar process is also used in (Huang et al. 2003) to create frequent patterns in dynamic scenes. However, these previous works focused on unsupervised learning, with no ability to classify behaviors into classes.

As the main goal of this research is to classify an observed behavior, we consider that the actions performed by an agent are usually influenced by past experiences. Indeed, sequence learning is arguably the most common form of human and animal learning. Sequences are absolutely relevant in human skill learning (Sun et al. 2001) and in high-level problem solving and reasoning (Anderson 1995). Taking this aspect into account in this paper, the problem of behavior classification is examined as a problem of learning to characterize the behavior of an agent in terms of sequences of atomic behaviors. Therefore, the behavior classification problem is transformed into a sequence classification problem where a sequence represents a specific behavior, as it is detailed in (Iglesias et al. 2010). This consideration makes possible to provide a *general* approach which can represent and handle different behaviors in a wide range of application domains.

It should be emphasized that the above approaches ignore the fact that agents change and evolve. A very important issue in agent modeling is to evolve the created agent behavior models according to the new observations collected in the corresponding environment. This challenge is related to the need to cope with huge amounts of data, and process streaming data on-line and in real-time (Domingos and Hulten 2001). Taking this into account, in this paper the agent behavior modeling is considered, treated and modeled as a dynamic and evolving phenomenon. This is the most important contribution of this paper.

3 Evolving classifier of agent behaviors

The proposed approach for classifying an agent behavior in an evolving manner is presented in this section. This classifier, called evolving classifier of agent behaviors (EvCAB) , is based on *Evolving Fuzzy Systems* and it takes into account the fact that the behavior of an agent is not fixed, but is rather changing. This approach can be applied to classify any agent whose behavior can be represented by a sequence of events.

To classify an observed behavior, EvCAB, as many other agent modeling methods (Riley and Veloso 2000), creates a library which contains the behavior models extracted by observation. This library, unlike other methods, is not a pre-fixed one, but is evolving, learning from the observations of the agent behaviors and, moreover, it starts to be filled in 'from scratch' by assigning temporarily to it the first observed sequence as a prototype. The library, called *Evolving Behavior Model Library* (EvBMLib), is continuously changing, evolving influenced by the changing agent behaviors observed in the environment.

The proposed approach includes the following three modules (which are described in Fig. 1):

1. *Creating sequence of events module* (*CSEMod*) The aim of this module is to create a sequence of events from the observations of the agent behavior. This module is domain-dependent and it needs a study of the different events which can be obtained. Section 4 details the most relevant characteristics of this module.
2. *Creating agent behavior model module* (*CBMMod*) This module analyzes the sequences of events and creates the corresponding models (agent behavior profiles). This process is detailed in Section 5.
3. *Evolving classifier of behavior models module* (*EvCBMMod*) This module involves in itself two sub-actions:
   (a) *Evolving the classifier* This sub-action includes on-line learning and update of the classifier, including the potential of each behavior to be a prototype, stored in the EvBMLib.
   (b) *Agent behavior classification* The agent behavior model created is associated with one of the prototypes from the EvBMLib and they are classified into one of the classes formed by the prototypes.

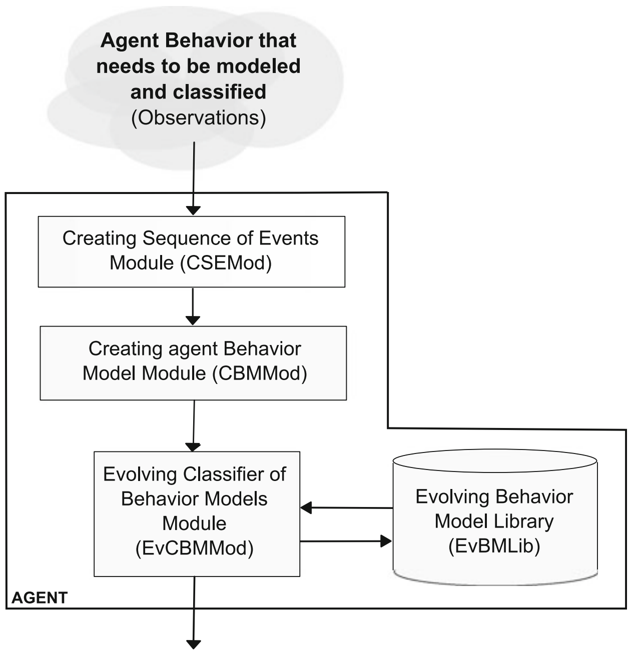
Agent Behavior that needs to be modeled and classified (Observations)
Creating Sequence of Events Module (CSEMod)
Creating agent Behavior Model Module (CBMMod)
Evolving Classifier of Behavior Models Module (EvCBMMod)
Evolving Behavior Model Library (EvBMLib)
AGENT

Fig. 1 Structure of EvCAB

The whole process is explained in Section 6.

4 Creating sequence of events module

This module creates a stream of observed atomic discrete events which describes the behavior of an agent in its environment. Each event is an atomic observation that occurs in a certain place during a particular interval of time and defines a specific act of an agent.

The kind of events and its features have to be determined by the designer taking into account the environment, and is beyond the scope of this paper. We note in passing that, in general, this capability exists even for domains in which observations are of continuous states, rather than discrete actions. For example, in some dynamic environments (e.g. *RoboCup Soccer Simulation*), each observation is a snapshot of the agent that do not offer any information about its actions. In this case, the actions taken by the agent should be estimated by contrasting consecutive snapshots (Kaminka et al. 2002). In other domains, the observations are inherently sequential events which do not need to be processed (e.g. UNIX commands in a command-line interface).

Once a sequence of events—representing the behavior of the agent—has been obtained, the *CBMMod* (explained in the next section), constructs the corresponding agent model.

5 Creating agent behavior model module

In most of the application domains, the actions performed by an agent are inherently sequential, and, thus, their ordering within the sequence should be considered in the modeling process. For example, in a human–computer interaction by commands, the specific ordering of commands within a sequence is essential for the result of the interaction[1]. For this reason, in this research, as it is also detailed in (Iglesias et al. 2010), the proposed process creates behavior models that specifically encode the observed sequences of actions executed by the observed agents.

The first step in the *CBMMod* is to extract the significant pieces of the sequence that can represent a repeating pattern of behavior. In many domains of interest, the temporal (non-Markovian) dependencies are very significant and we consider that a current event might depend on the events that have happened before it, and is, possibly, related to the

[1] For instance, consider the difference between the UNIX command sequence "`rm a.txt ; mv b.txt a.txt`", and the sequence "`mv b.txt a.txt; rm a.txt`".

events that will happen after it is observed. Thus, the event sequence needs to be segmented into several subsequences which will be inserted in the same model separately. This segmentation can be done by using some environment characteristic that can separate efficiently the sequence in several subsequences of uninterrupted events. Otherwise, the sequence can be segmented by defining an appropriate maximum length and obtaining every possible ordered subsequence of that specific length. Thus, the sequence $A = A_1 A_2 \ldots A_n$ (where n is the number of commands of the sequence) will be segmented in the subsequences described by $A_i \ldots A_{i+\text{length}} \ \forall \ i, i = [1, \ n - \text{length} + 1]$, 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. The length of these subsequences is an important aspect (which is analyzed in different environments in Sect. 7) because it modifies both the size of the model and the final results quite significantly.

For the sake of simplicity, let us consider we are observing an agent and its behavior is represented by the following sequence: $\{A \to B \to A \to B \to C\}$ where each different capital letter represents a different atomic event. If we divide this example sequence into subsequences of equal size; let 3 be the subsequence length, then we obtain: $(A \to B \to A)$ and $(B \to A \to B)$ and $(A \to B \to C)$.

Once the sequence has been segmented and based on the work done in (Iglesias et al. 2009), we propose the use of a *trie* data structure (Fredkin 1960) for storing the subsequences. This structure was also used in (Iglesias et al. 2007) to classify different sequences and in (Kaminka et al. 2002; Iglesias et al. 2006) to classify the behavior patterns of a *RoboCup* soccer simulation team. Thus, when a new model needs to be constructed, we create an empty *trie*, and insert each subsequence of events into it, such that all possible subsequences are accessible and explicitly represented. Every *trie*-node represents an event appearing at the end of a subsequence, and the node's children represent the events that have appeared following this event. Also, each node keeps track of the number of times an event has been inserted in to it. When a new subsequence is inserted into the *trie*, existing nodes of the *trie* are modified and/or new nodes are created. As the dependencies of the events are relevant in an agent behavior, the subsequence suffixes (subsequences that extend to the end of the given sequence) are also inserted.

Considering the previous example, the first subsequence $(\{A \to B \to A\})$ is added as the first branch of the empty *trie* (Fig. 2a). Each event is labeled with the number 1 that indicates that the event has been inserted in the node once (in Fig. 2, this number is enclosed in square brackets). Then, the suffixes of the subsequence ($\{B \to A\}$ and $\{A\}$)

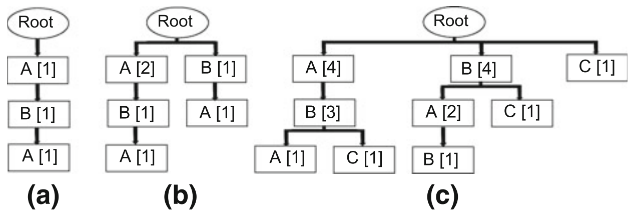

Fig. 2 Steps of creating an example *trie*

are also inserted (Fig. 2b). In this case, as the subsequence $\{A\}$ has already been inserted in the *trie*, we increase its corresponding number (in brackets) from 1 to 2. Finally, after inserting the three subsequences and its remaining suffixes, the completed *trie* is obtained (Fig. 2c). As we can see, the number of times that a node is inserted depends on its position in the sequence.

Once the *trie* is created, the subsequences that characterize the behavior have to be obtained (where a subsequence is a path from the *root* node to any other node of the *trie*). Thus, the *trie* is traversed to calculate the relevance of each subsequence. For this purpose, frequency-based methods (Agrawal and Srikant 1995) are used. In particular, in this approach, to evaluate the relevance of a subsequence, its relative frequency or support (Agrawal and Srikant 1995) is calculated. This value is the number of occurrences of a particular subsequence (of length n) divided by the total number of subsequences of equal length (n). As the subsequences in a *trie* are the different paths from the root to a node, the support value of a subsequence is stored in its last node. Therefore, in this step the *trie* is transformed into a set of subsequences labeled with a value (support). Note that this step does not necessarily have to be carried out separately, after the creation of *trie*. Rather, support counts can be updated during the insertion of every subsequence.

In the previous example, the *trie* consists of nine nodes; therefore, the model consists of nine different subsequences which are labeled with its support. Considering the first subsequence (A), its support is 4/(4+4+1) = 0.44. The distribution of the value of the nine subsequences is represented in Fig. 3.

The model of an agent, encoded by the behavior library, is then the distribution of subsequences within the library

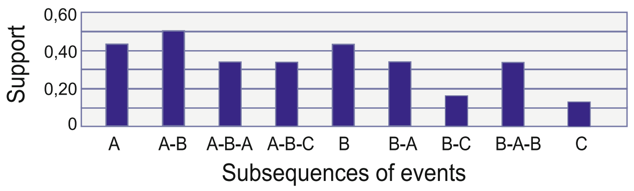

Fig. 3 Distribution of subsequences

(stored in a *trie*). Although, it is not considered in this research, the model could be created with the subsequences with the greatest support value (the more relevant subsequences) in order to reduce its corresponding set of subsequences.

Once a behavior model (distribution of relevant subsequences) has been created, it is classified and the EvBMLib is updated with its relevant information. This evolving process is developed in the EvCBMMod, which is explained in the next section.

6 Evolving classifier of behavior models module

This module proposes a classifier, called EvCBM (*evolving classifier of behavior models*), which is addressing an important challenge in the agent modeling classification: to dynamically adapt the classifier according to the new observations collected in the corresponding environment.

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 and the class label space is represented by the most representative distributions. Thus, a distribution in the class label space represents a specific behavior model which is one of the prototypes of the evolving Library. These prototypes are not fixed and evolve taking into account the new observations 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. In other words, if the behavior of the agents observed is very similar, the number of prototypes is reduced.

Once the corresponding data vector, which represents the distribution of a specific agent behavior, has been created in the previous modules, it is processed by EvCBM. In this case, the distributions need to be represented in a data space; and each distribution is considered as a data vector that defines a *point* in the data space (which is stored in EvBMLib).

The data space in which these *points* (behavior models) can be represented should consist of n dimensions, where n is the number of the different subsequences obtained. 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, the dimension of the data space is incrementally growing according to the different subsequences that are represented in it.

One of the most important characteristics of this general classifier is that it 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 subsections, there is no need to store all the samples in it. This aspect is important because the amount of different sequences collected in an environment could be quite large, and the proposed approach needs to cope with huge amounts of data and process streaming data in real-time and on-line. In most of the environments in which this approach can be applied, storing the complete data set and analyzing the data streams in off-line mode could be impractical.

6.1 Procedure of the classifier EvCBM

The procedure of this classifier includes the following stages/steps:

1. *Classify a new sample* (agent behavior) in a 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.
4. *Insert the new data sample as a new prototype* if needed.
5. *Remove existing prototypes* if needed.

The following five subsections explain each step of this evolving classification method.

6.1.1 Classify the new sample

In order to classify a new data sample, we compare it with all the prototypes stored in EvBMLib. This comparison is done using cosine distance and the smallest distance determines the closest similarity. Since a prototype is related to each class, the classification is done in the corresponding class of the selected prototype. This aspect is considered in Eq. 1.

$$Class(x_z) = Class(Prot^*); \quad Prot^* = MIN_{i=1}^{NumProt}(cosDist(Prototype_i, x_z)) \tag{1}$$

where x_z represents the zth sample to classify, $NumProt$ determines the number of existing prototypes in the EvBMLib, $Prototype_i$ represents the ith prototype, and $cosDist$ represents the cosine distance between two samples in the data space.

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) because the cosine distance is calculated recursively, as it is explained in the next subsection.

6.1.2 Calculate the potential of the new data sample

As in Angelov and Zhou (2008), a prototype is a data sample (the model of an agent behavior represented by a distribution of subsequences of events) that groups several samples which represent a certain behavior. The classifier is initialized with the first data sample, which is stored in the EvBMLib. Then, each data sample is classified into one of the prototypes defined in the classifier. Finally, based on the *potential* of the new data sample to become a prototype (Angelov and Filev 2004), it could form a new prototype or replace an existing one.

The potential (P) of the kth data sample (x_k) is calculated by Eq. 2 which represents a function of the accumulated distance between a sample and all the other $k-1$ samples in the data space (Angelov and Zhou 2008). The result of this function represents the *density* of the data that surrounds a certain data sample.

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

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

In Angelov et al. (2007) the potential is calculated using the Euclidean distance and in (Angelov and Zhou 2008) 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 sensor readings). Also, cosine distance tolerates the case when the value of several subsequences in a sample is null (null is different than zero). Therefore, EvCBM uses the cosine distance (*cosDist*) to measure the similarity between two samples; as it is described in Eq. 3.

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

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 Eq. 2 requires all the accumulated data samples available to be calculated, which contradicts to the requirement for real-time and on-line application needed in the proposed problem. For this reason, in Angelov and Zhou (2008) it is developed a recursive expression for the cosine distance. This formula is as follows:

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

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

$$\text{and } b_1^j = \sqrt{\frac{(z_1^j)^2}{\sum_{l=1}^{n} (z_1^l)^2}}; \quad j = [1, n+1] \qquad (4)$$

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 kth data sample.

6.1.3 Update all the prototypes

Once the potential of the new data sample has been calculated, all the existing prototypes in the EvBMLib are updated taking into account this 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. This operation is done really fast and it requires very little memory space because of the use of recursive equations.

6.1.4 Insert the new data sample as a new prototype

The proposed evolving classifier, EvCBM, can start '*from scratch*' (without prototypes in the library) in a similar manner as eClass evolving fuzzy rule-based classifier proposed in (Angelov et al. 2007), used in (Zhou and Angelov 2007) for robotics and further developed in (Angelov and Zhou 2008). 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 Eq. 5.

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

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 in the EvBMLib which represents a part of the obtained data samples.

6.1.5 Removing existing prototypes

After adding a new prototype, we check whether any of the already existing prototypes in the EvBMLib are described *well* by the newly added prototype (Angelov and Zhou

2008). By *well* we mean that the value of the membership function that describes the closeness to the prototype:

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

The membership function between a data sample and a prototype can be defined e.g. by a Gaussian bell function (chosen due to its generalization capabilities) as:

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

where $cosDist(z_k, Prot_i)$ represents the cosine distance between a data sample (z_k) and the ith 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 (Angelov and Filev 2005) of the data. The equation to get the spread of the kth data sample is defined as:

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

where k is the number of data samples considered so far; $cosDist(Prot_i, z_k)$ is the cosine distance between the new data sample (z_k) and the ith prototype.

However, to calculate the scatter without storing all the received samples, this value can be updated [as shown in Angelov et al. (2007)] recursively by:

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

7 Experiments

In order to evaluate EvCAB, we conducted extensive experiments in two different environments: *UNIX User Data* (Sect. 7.1) and *Intelligent Home Environments* (Sect. 7.2).

7.1 UNIX user data

In this domain, the observed behavior of a user consists of the UNIX commands s/he typed during a period of time. The goal is to classify a given sequence of UNIX commands (user behavior) in one of the behavior models previously created and stored or create a new one. However, the UNIX user behavior models created represent the behavior of several agents and they are updated based on the *new* commands typed by the users. This task is very useful in different application areas such as computer intrusion detection or intelligent tutoring systems.

To evaluate *EvCAB* in this environment, we have used a source of *UNIX* commands typed by 168 real users and labeled in four different groups. These data were collected by Greenberg (1988) using *UNIX csh* command interpreter. Salient features of each group are described below, and the sample sizes (the number of people observed) are indicated in Table 1.

- *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* This 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.

7.1.1 Experimental design

Although the proposed classifier has been designed to be used in real-time, the use of the above data set allows us to have comparable results with the established off-line and incremental techniques. It should be emphasized that EvCAB does not need to work in this mode. This is done solely to have comparable results with very different techniques. For this reason, the tenfold cross-validation technique is used.

The number of *UNIX* commands typed by a user, and used for creating his/her behavior model, is very relevant

Table 1 Sample group sizes and command lines recorded

Group of users name	Sample size	Total number of command lines
Novice Programmers	55	77.423
Experienced Programmers	36	74.906
Computer Scientists	52	125.691
Non-Programmers	25	25.608
Total	168	303.628

in the classification process. When EvCAB 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 1,000 commands per user. Also, if the number of users increases, the number of different subsequences increases, too.

In the phase of user behavior model creation, the length of the subsequences in which the original sequence is segmented (used for creating the *trie*) is an important 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 two to sic.

In order to evaluate the performance of EvCAB, we compare it with different (incremental and non-incremental) classifiers. For this comparison, the different 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*).

The classifiers we used are detailed as follows:

- *Naive Bayes* (NB) classifier (Rish 2001) and its incremental version (Incremental NB), in which it is used a default precision of 0.1 for numeric attributes when it is created with zero training instances. The reason for selecting Naive Bayes is because it performs comparably to C4.5 (Langley and Sage 1994), it requires little training data and it is computationally fast when making decisions.
- *Incremental k-NN* classifies objects based on closest training examples in the feature space. Unlike EvCAB and other incremental algorithms, k-NN stores entire dataset internally.
- *Learning Vector Quantization* classifier (LVQ) is a supervised version of vector quantization. In this case, it is used the enhanced version of LVQ1, the OLVQ1 implementation (Kohonen et al. 2001).
- *Support Vector Machine* Classifier (SVM) relies on the statistical learning theory (Platt 1998).

7.1.2 Results

Figure 4 shows the percentage of users correctly classified into its corresponding group using different number of

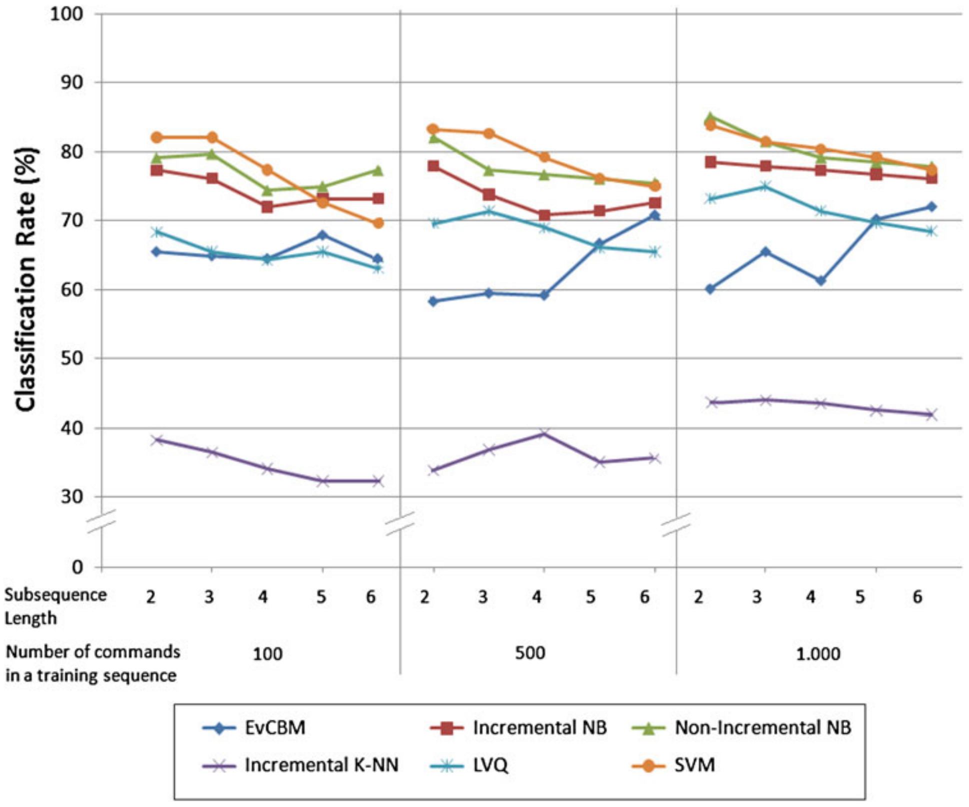

Fig. 4 EvCAB: results of the UNIX user classification

commands for training (100, 500 and 1,000 commands per user) and subsequences lengths for segmenting the initial sequence (from 2 to 6).

According to this data, we can see that for small subsequences length (2 or 3) the difference between EvCAB and the other classifiers (except *k-NN*) 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 environment when the subsequence length is around five.

Taking into account only the obtained results, we could conclude that the proposed classifier is not the most suitable choice. However, due to the characteristics of the environment, we need a classifier able to process streaming data as it arrives, continuously. EvCAB does not need to store the entire data stream in the memory and disregards any sample after being used. In addition, EvCAB is one-pass (each sample is preceded 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, EvCAB is computationally simple and efficient as it is recursive. In fact, because the number of attributes is very large in the proposed environment, EvCAB is the best working alternative.

7.2 Intelligent home environments

The goal of this experiment is to model and classify sequences of sensor readings which represent a certain human activity in an intelligent home environment. Therefore, in this case, instead of creating an agent behavior model, the model of a specific human activity is created using the sequence of sensor readings collecting while a human executes that activity. Although, the sensor readings are usually tagged with the time and date of the event, in this case this information is not used, and the information obtained from the intelligent home is a sequence of sensor readings.

In order to evaluate EvCAB in this environment, we use a dataset with the sensor readings activated by a person while s/he is doing a specific activity. Thus, the sequence of sensor readings is labeled. The dataset used in this research was created by the *CASAS Smart Home project*, which is a multi-disciplinary research project at Washington State University (WSU) focused on the creation of an intelligent home environment (Rashidi and Cook 2009). This dataset represents sensor readings collected in a WSU smart apartment testbed. The apartment is equipped with 38 sensors distributed throughout the space (26 of them are motion sensors). The data set represents 24 participants performing the following five activities:

1. *Make a phone call* The participant moves to the phone in the dining room, looks a specific number in the phone book, dials the number, and listens to the message.
2. *Wash hands* The participant moves into the kitchen sink and washes his/her hands in the sink.
3. *Cook* The participant cooks a pot of oatmeal. S/he measures water, pours the water into a pot and boils it, adds oats, then puts the oatmeal into a bowl with raisins and brown sugar.
4. *Eat* The participant takes the oatmeal and a medicine container to the dining room and eats the food.
5. *Clean* The participant takes all of the dishes to the sink and cleans them with water and dish soap in the kitchen.

Thus, the dataset consists of 120 different samples labeled with the corresponding activity.

7.2.1 *Experiment design*

In these experiments, the length of the subsequences in which the original sequence is segmented also varies from 2 to 6. In addition, in order to have comparable results with other different classifiers using the above dataset, threefold cross validation is used.

As in the *UNIX user data* environment, EvCAB is compared with: Naive Bayes (incremental and non-incremental), Incremental k-NN, LVQ and SVM classifiers, which are trained using a feature vector for each activity done by a resident. The corresponding vector consists of the support value of all the different subsequences of sensor readings obtained for all the activities.

7.2.2 *Results*

Figure 5 shows the percentage of sequences correctly classified into its corresponding activity using different

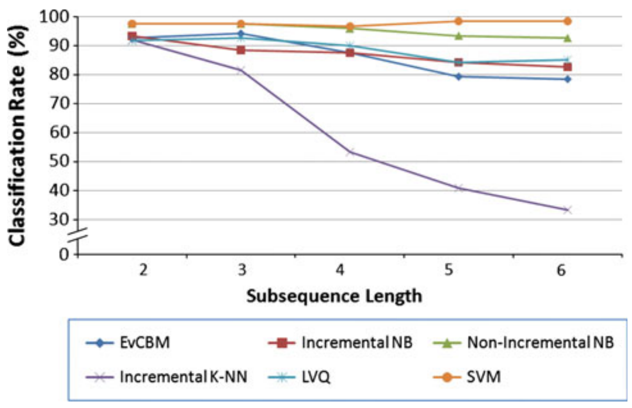

Fig. 5 EvCAB: results of the human activity classification in an intelligent home environment

subsequences lengths for segmenting the initial sequence. According to these data, the percentages of sequences correctly classified by our approach are very similar to the obtained by the other (incremental and non-incremental) classifiers (except k-NN). However, our approach can evolve the created classifier according to the new sequences collected in the intelligent environment. Besides, the data streams (sensor readings) collected in an intelligent home environment can be very large and EvCAB is suitable in this case because it does not need to store the entire data stream in the memory.

8 Conclusions

This paper presents an approach to model and classify automatically user behaviors from the sequence of events executed during a period of time. However, as a user profile is usually not fixed but rather it changes and evolves, we have proposed a classifier able to keep up to date the created models 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.

Also, an important aim in this work is to provide a *general* approach which can represent, handle and evolve different behaviors in a wide range of domains. Therefore, the proposed approach is generalizable to modeling, classifying and updating agent behaviors represented by a sequence of events. To demonstrate this generalization, the proposed approach has been experimentally evaluated in two very different domains: *UNIX User Classification* and *Human Activity Classification in Intelligent Home Environment*. A large set of experiments have been conducted in both domains.

The experimental results show that, using an appropriate subsequence length, EvCAB is very effective in both domains and it 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 suitable in environments which it is necessary to cope with huge amounts of data and process streaming data quickly, because it does not need to store the entire data stream in the memory, and it is computationally simple and efficient as it is recursive and one pass.

Acknowledgments This work has been partially supported by the Spanish Government under project TRA2007-67374-C02-02.

References

Agrawal R, Srikant R (1995) Mining sequential patterns. In: International conference on data engineering, Taipei, Taiwan, pp 3–14

Anderson J (1995) Learning and memory: an integrated approach. Wiley, New York

Angelov P, Filev D (2004) An approach to online identification of Takagi–Sugeno fuzzy models. IEEE Trans Syst Man Cybern Part B 34(1):484–498

Angelov P, Filev D (2005) Simpl_eTS: a simplified method for learning evolving Takagi–Sugeno fuzzy models. In: The IEEE international conference on fuzzy systems (FUZZ-IEEE 2005), pp 1068–1073

Angelov P, Zhou X (2008) Evolving fuzzy rule-based classifiers from data streams. IEEE Trans Fuzzy Syst 16(6):1462–1475

Angelov P, Zhou X, Klawonn F (2007) Evolving fuzzy rule-based classifiers. In: IEEE symposium on computational intelligence in image and signal processing (CIISP 2007), pp 220–225

Carrbery S (2001) Techniques for plan recognition. User Model User Adap Interact 11(1–2):31–48

Domingos P, Hulten G (2001) Catching up with the data: research issues in mining data streams. In: In workshop on research issues in data mining and knowledge discovery, 2001

Fredkin E (1960) Trie memory. Comm ACM 3(9):490–499

Greenberg S (1988) Using UNIX: collected traces of 168 users. Master's thesis, Department of Computer Science, University of Calgary, Alberta, Canada

Han K, Veloso M (1999) Automated robot behavior recognition applied to robotic soccer. In: Proceedings of IJCAI-99 workshop on team behaviors and plan recognition, 1999

Hong J (2001) Goal recognition through goal graph analysis. J Artif Intell Res 15:1–30

Horman Y, Kaminka GA (2007) Removing biases in unsupervised learning of sequential patterns. Intell Data Analysis 11(5): 457–480

Huang Z, Yang Y, Chen X (2003) An approach to plan recognition and retrieval for multi-agent systems. In: Proceedings of AORC, 2003

Iglesias J, Ledezma A, Sanchis A (2006) A comparing method of two team behaviours in the simulation coach competition. In: Proceedings of the international conference on tools with artificial intelligence (MDAI 2006), ser. LNCS, vol 3885. Springer, pp 117–128

Iglesias JA, Ledezma A, Sanchis A (2007) Sequence classification using statistical pattern recognition. In: Proceedings of the international conference on intelligent data analysis (IDA 2007), ser, 2007. LNCS, vol 4723. Springer, pp 207–218

Iglesias JA, Ledezma A, Sanchis A (2009) Creating user profiles from a command-line interface: a statistical approach. In: Proceedings of the international conference on user modeling, adaptation, and personalization (UMAP 2009), ser, June 2009. LNCS, vol 5535. Springer, pp 90–101

Iglesias JA, Ledezma A, Sanchis A, Kaminka G (2010) A plan classifier based on chi-square distribution tests. Intell Data Analysis 15(2) (in press)

Kaminka GA, Fidanboylu M, Chang A, Veloso MM (2002) Learning the sequential coordinated behavior of teams from observations. In: RoboCup, ser. Lecture notes in computer science, vol 2752. Springer, pp 111–125

Kohonen T, Schroeder MR, Huang TS (eds) (2001) Self-organizing maps. Springer-Verlag New York Inc, Secaucus

Langley P, Sage S (1994) Induction of selective bayesian classifiers. In: Proceedings of the conference on uncertainty in artificial intelligence. Morgan Kaufmann, pp 399–406

Ledezma A, Aler R, Sanchis A, Borrajo D (2004) Predicting opponent actions by observation. In: RobuCup, ser. Lecture notes in computer science, vol 3276. Springer, pp 286–296

Mulcahy NJ, Call J (2006) Apes save tools for future use. Science 312(5776):1038–1040. http://dx.doi.org/10.1126/science.1125456

Platt J (1998) Machines using sequential minimal optimization. In: Schoelkopf B, Burges C, Smola A (eds) Advances in Kernel methods-support vector learning,MIT Press, Cambridge

Rashidi P, Cook DJ (2009) Keeping the resident in the loop: adapting the smart home to the user. IEEE Trans Syst Man Cybern Part A 39(5):949–959

Riley P, Veloso MM (2000) On behavior classification in adversarial environments. In: DARS, 2000, pp 371–380

Rish I (2001) An empirical study of the naive Bayes classifier. In: Proceedings of IJCAI-01 workshop on empirical methods in artificial intelligence, 2001

Steffens T (2002) Feature-based declarative opponent-modelling in multi-agent systems. Master's thesis, Institute of Cognitive Science Osnabrnck, 2002

Sun R, Merrill E, Peterson T (2001) From implicit skills to explicit knowledge: a bottom-up model of skill learning. Cogn Sci 25(2):203–244

Tambe M, Rosenbloom PS (1995) Resc: an approach for dynamic, real-time agent tracking. In: International joint conference on artificial intelligence (IJCAI-95), Montreal, Canada, 1995. http://citeseer.ist.psu.edu/tambe95resc.html

Zhou X, Angelov P (2007) Autonomous visual self-localization in completely unknown environment using evolving fuzzy rule-based classifier. In: IEEE symposium on computational intelligence in security and defense applications (CISDA 2007), pp 131–138