Soft Comput (2011) 15:1009–1022
DOI 10.1007/s00500-010-0657-0

ORIGINAL PAPER

# Fuzzy classification in dynamic environments

Abdelhamid Bouchachia

Published online: 5 October 2010

**Abstract** The persistence and evolution of systems essentially depend on their adaptivity to new situations. As an expression of intelligence, adaptivity is a distinguishing quality of any system that is able to learn and to adjust itself in a flexible manner to new environmental conditions and such ability ensures self-correction over time as new events happen, new input becomes available, or new operational conditions occur. This requires self-monitoring of the performance in an ever-changing environment. The relevance of adaptivity is established in numerous domains and by versatile real-world applications. The present paper presents an incremental fuzzy rule-based system for classification purposes. Relying on fuzzy min–max neural networks, the paper explains how fuzzy rules can be continuously online generated to meet the requirements of non-stationary dynamic environments, where data arrives over long periods of time. The approach proposed to deal with an ambient intelligence application. The simulation results show its effectiveness in dealing with dynamic situations and its performance when compared with existing approaches.

## 1 Introduction

Learning of fuzzy rule-based systems in dynamic environments is typical and relevant to various real-world applications. In contrast to the offline rule-based systems, where the process of rule induction is performed at once, incremental learning is evolutionary rather than revolutionary (Bouchachia and Mittermeir 2006). Learning takes place over long periods of time and implicitly does not finish once available data is exhausted. In other terms, the systems are subjected to refinement as long as data arrives. More interestingly online rule-based systems aim at dealing with starving-data applications, as well as with intensive-data applications.

The difficultly of incremental learning is of course the inability to accurately estimate the statistical characteristics of the incoming data in the future. In non-stationary changing environments, the challenge is big, because the rule system may change drastically over the time due to concept drift (Widmer and Kubat 1996).

Moreover, incremental learning relies on several assumptions (Bouchachia et al. 2007) among which online tuning is fundamental. The learning machinery should ensure to store only the learning model (e.g., rules in the rule-based system) and use it as an old experience to bias learning in the future with the overall goal of increasing the system's effectiveness. In this sense, the system becomes self-adaptive and hence its acquired knowledge/model becomes self-corrective. As new data arrives, new rules may be created and existing ones modified allowing the system to evolve over time.

There are several real-world applications, such as user profile learning, computer intrusion, data mining applications, etc. where incremental learning is relevant since data arrives over time. In such applications, it is important to devise learning mechanisms to induce new knowledge without 'catastrophic forgetting' and/or to refine the existing knowledge. The whole problem is then summarized in how to accommodate new data in an incremental way while keeping the system under use.

All incremental learning algorithms are confronted with the plasticity–stability dilemma. This dilemma establishes

A. Bouchachia (✉)
Department of Informatics, University of Klagenfurt,
Klagenfurt, Austria
e-mail: hamid@isys.uni-klu.ac.at

the tradeoff between catastrophic interference (or forgetting) on one hand and the ability to incrementally and continually accommodate new knowledge in the future whenever new data becomes available. The former aspect is referred to as stability, while the latter is referred to as plasticity. In a nutshell, the stability–plasticity dilemma is concerned with learning new knowledge without forgetting the previously learned one. This problem has been thoroughly studied by many researchers (French 1999; Grossberg 1988; McCloskey and Cohen 1999; Ratcliff 1990; Sharkey and Sharkey 1995).

In Bouchachia et al. (2007), we discussed and compared many incremental algorithms from the perspective of their online learning capabilities. These algorithms include adaptive resonance theory (fuzzy ARTMAP) (Grossberg 1988), nearest generalized exemplar (NGE) (Salzberg 1991), generalized fuzzy min–max neural networks (GFMMNN) (Gabrys and Bargiela 2000), growing neural gas (GNG) (Fritzke 1995), and incremental learning based on the function decomposition (ILFD) (Bouchachia 2006). The main outcome was that GFMMNN often performs better than the other algorithms. However, at many occasions, the discrepancies between the accuracy values are very low.

In a further study Sahel et al. (2007), we investigated these incremental algorithms from the perspective of stability when data drifts. In particular, these algorithms were compared against five static classifiers which are updated using retraining. The experiments showed that both classes of approaches (incremental vs. retraining) improve the performance as compared to the non-adaptive mode although a number of outstanding research issues remain open. However, their accuracy deteriorates with time. The main result of this study was that more robust approaches yielding a balance between incremental learning and forgetting are needed to deal with changing environments.

In the present paper, an algorithm for building incremental fuzzy classification systems (IFCS) is proposed. It uses GFMMNN, which is described by the characteristics shown in Table 1, as a routine for updating the prototypes. GFMMNN is used in its original version, but the size of the hyperboxes is controlled from outside GFMMNN. Another aspect that has been included consists of merging adjacent hyperboxes of the same class. This is done for the sake of optimization, that is, a reduced number of hyperboxes which will be transformed into fuzzy rules can be derived from the model learned so far. In a nutshell, the originality of the paper is the proposed algorithm (IFCS) itself. It relies on GFMMNN but involves (1) semi-supervised learning (learning from both labeled and unlabeled data), (2) drift tracking, (3) a pure online scenario and (4) mechanisms for deriving compact fuzzy rules.

**Table 1** Characteristics of the GFMMNN algorithm

| Characteristics | Conformance |
|---|---|
| Online learning | ✓ |
| Type of prototypes | Hyperbox |
| Generation control | ✓ |
| Prototype shrinking | ✓ |
| Prototype deletion | × |
| Prototype overlap | × |
| Prototype growing | ✓ |
| Noise resistance | ✓ |
| Sensitivity to data order | ✓ |
| Normalization | ✓ |

The paper is structured as follows. In addition to the overview of incremental algorithms introduced in Sect. 1, further incremental fuzzy algorithms are outlined in Sect. 2. Section 3 describes the IFCS algorithm, the GFMMNN algorithm, the generation of fuzzy rules and some computational refinement. Section 4 discusses the problem of concept drift and shows some proposed techniques for dealing with it. Section 5 deals with the empirical evaluation of the IFCS algorithm and various aspects before concluding in Sect. 6.

## 2 Incremental fuzzy rule-based systems

Traditional FRSs are designed in batch mode, that is, using the complete training data at once. For stationary processes, this is sufficient, while for time-based and complex non-stationary processes, efficient techniques for updating the induced models are needed. To avoid starting from scratch every time, these techniques must be able to learn online and incrementally by adapting the current model using only the new data without referring to the old one. They have to be equipped with mechanisms capable of reacting to changes (be it gradual changes or abrupt ones). Another issue concerns knowledge and data integration, that is the adaptation and evolution of the knowledge (model/rule base) by accommodating the information brought by the new data and reconciling this with the existing knowledge. Even though initial work in this direction has been made (e.g., Angelov et al. 2008; Bouchachia and Mittermeir 2006; Hagras et al. 2007; Kasabov 2001), this research direction is still in its infancy and more focused efforts are needed. Many approaches do simply perform "adaptive tuning", that is, they permanently re-estimate the parameters of the computed model. Quite often, however, it is necessary to adapt the structure of the rule base. In the sequel, we review the few up-to-date and important references on incremental FRSs.

Probably, the first evolving architecture proposed in the context fuzzy neural network is EFuNN proposed by Kasabov (2001). EFuNN (evolving fuzzy neural network) has a five-layer structure similar to the traditional FuNN. EFuNNs adopt known techniques from resource allocation networks, cell structures and ART networks. The incrementality in EFuNNs takes place at the rule layer whose nodes are defined by two weight vectors. If the input falls in the radius of the rule and its corresponding output falls in the rule's output reactive field, the connection weight vectors and the rule's radius are updated. Otherwise, a new rule node is allocated.

In Angelov (2004), an approach for adaptation of a FRS of Takagi–Sugeno type was proposed. It consists of two tasks: (a) generating focal points that correspond to the rule's antecedents using clustering and (b) estimating the consequents' parameters using the least squares algorithm. The first task is realized by applying an online version of the subtractive clustering algorithm. New rules are added as new clusters are generated. The task (b) is executed using the recursive least squares algorithm to re-estimate the parameters of the rules' consequent. The approach is developed for control rather than for classification. This work by Angelov has several merits. It suggests an online version of the fuzzy subtractive clustering algorithm and mechanisms to update the rule's consequent. Similar approaches relying on subtractive clustering and least squares have been proposed later as in (de Barros and Dexter 2007).

Recently, Angelov et al. (2008) suggested two approaches for online evolving fuzzy classifiers called eClass and FLEXFIS-Class. These methods investigate different architectures to express the type of output (consequent). They adapt their parameters in the antecedent and consequent parts as new data arrives. One new aspect in this investigation is the drift treatment that is explicitly handled by aging mechanisms. This allows to keep the system consistent with the new data.

Hagras et al. (2007) proposed a type-2 fuzzy controller that is able to incrementally update the rules. These rules model the relationship between actuators (output variables) and sensors (input variables) that equip an ambient intelligent environment. The system aims at learning rules modeling the behavior of the user based on its interaction with the appliances. Whenever the actuators are changed, the state of the environment (input) is recorded before it is mapped to the rules' antecedents. The consequents of the firing rules are then replaced by the actual output emanating from the actuators. If no rule is fired, a new one is added.

Bouchachia and Mittermeir (2006) proposed an integrated approach. To accommodate incremental rule learning, appropriate mechanisms are applied in all steps of the FRCS: (1) incremental supervised clustering to generate the rule antecedents in a progressive manner, (2) online and systematic update of fuzzy partitions, (3) incremental feature selection using an incremental version of the Fisher's interclass separability criterion. In a previous study (Bouchachia 2004), the first mechanism was realized using a cascade of clustering algorithms. Here, the feature selection procedure is applied to update the set of features in an online way. Moreover, the optimization of the rules are based on a similarity measure that considers only the geometry of the membership function.

## 3 IFCS: an incremental fuzzy rule-based classification system

In general, fuzzy rule-based systems intend to simulate human knowledge in the form of fuzzy "IF-THEN" rules. These rules are usually extracted from raw data by an inductive learning process. Generically, a fuzzy rule has the form:

$$R_r \equiv \text{If } x_1 \text{ is } A_{r,1} \wedge \cdots \wedge x_n \text{ } is \text{ } A_{r,n} \text{ then } y_r \quad (1)$$

where $x_i$ are fuzzy linguistic input variables, $A_{r,i}$ are linguistic terms in the form of fuzzy sets that characterize $x_i$ and $y_r$ is the output variable/function. In the case of classification systems, a rule looks like:

$$\text{If } x_1 \text{ is } A_{r,1} \wedge \cdots \wedge x_n \text{ is } A_{r,n} \text{ then } y_r \text{ is } C_1[\tau_1], \ldots, C_{\mathscr{K}}[\tau_{\mathscr{K}}] \quad (2)$$

where $C_j$, $\tau_j$ indicate, respectively, the class label and the certainty factor representing the confidence degree of assigning the input to a class, i.e., how good the rule covers the classes space. There exists another type of fuzzy classification systems based on the multidimensional fuzzy sets, where rules are expressed in the form:

$$R_r \equiv \text{ If } x \text{ is } K_i \text{ then } y_r \text{ is } C_j \quad (3)$$

where $K_i$ is a cluster. The rule means that if the sample $x$ is CLOSE to $K_j$ then the label of $x$ should be that of class $C_j$.

The proposed incremental fuzzy rule-based classification system consists of four modules as shown in Fig. 1:

1. Membership function controller (MFC) is responsible for the fuzzification process transforming the crisp input into fuzzy input relying on membership functions (e.g., triangular, trapezoidal, etc.). These functions specify a partitioning of the input and eventually the output space. To meet the requirements of incrementality, the membership functions are dynamic in the sense that their characteristics change over time as new data become available or

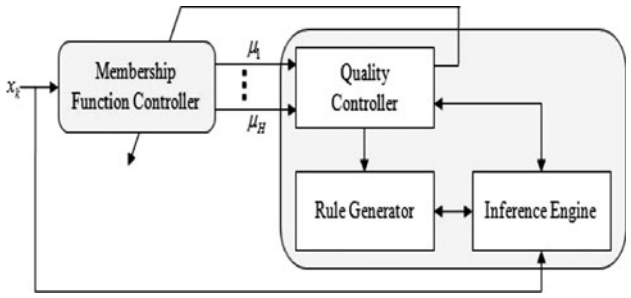

**Fig. 1** Structure of the adaptive fuzzy classifier

some optimization procedures dictate that. In the present work, fuzzy sets generation is achieved by mechanisms that ensure incremental partitioning, novelty detection, rapid adaptation, partially labeled data and even sparse data generalization. The computational model that exhibits such properties is GFMMNN that relies on the hyperbox fuzzy membership function.

2. Quality controller (QC): the classification accuracy alone is not the only comprehensive indicator of the FRS quality. Other major qualitative factors, such as comprehensibility and completeness are required (Liu et al. 2007). In this context, we have applied some reduction techniques (Bouchachia and Mittermeir 2006; Bouchachia 2004): (1) elimination of redundant fuzzy sets, (2) merge of fuzzy sets, (3) removal of weakly firing rules, (4) removal of redundant rules and (5) and'ing and or'ing the antecedents. In addition to this, in the context of IFCS, QC allows to observe the quality of classification over time to deal with crucial phenomena, such as concept drift and novelty detection.
3. Rule generator (RG): the formulation of rules based on the update of the quality controller. Actually, the rules data are stored in the knowledge base of the system.
4. Inference system: consists of two modules, the inference engine and the defuzzifier. The inference system maps the input to the rules and computes an aggregated fuzzy output of the system according to an inference method. In the present system, the inference engine uses the common inference method max–min composition. The defuzzification module which transforms the aggregated fuzzy output generated by the inference engine into a crisp output using the centroid method.

Because the most specific modules in IFCS are MFC and QC, they will henceforth the focus of the discussion. They are basically intertwined as shown explicitly in Algorithm 1.

**Algorithm 1** : Steps of IFCS

```
1: if Initial=true then
2:    $\mathscr{M}$ ← Train_Classifier(<TrainingData, Labels>)
3:    $\mathscr{E}$ ← Test_Classifier(<TestingData, Labels>,$\mathscr{M}$)
          // Just for the sake of observation
4: end if
5: i←0
6: while true do
7:    i← i+1
8:    Read <Input, Label>
9:    if IsLabeled(Label)=true then
10:       if Saturation_Training(i)=false then
11:          $\mathscr{M}$ ← Train_Classifier(<Input, Label>,$\mathscr{M}$)
12:          If Input falls in a hyperbox with Flabel, then Flabel ← Label
13:       else
14:          Err ← Test_Classifier(<Input,Label>,$\mathscr{M}$)
15:          Cumulated_Err ← Cumulated_Err+Err
16:          if Detect_Drift(Cumulated_Err)=true then
17:             $\mathscr{M}$ ← Reset(Cumulated_Err,$\mathscr{M}$)
18:          else
19:             $\mathscr{M}$ ← Update_Classifier(<Input,Label>,$\mathscr{M}$)
20:             If Input falls in a hyperbox with Flabel, then Flabel ← Label
21:          end if
22:       end if
23:    else
24:       Flabel← Predict_Label(Input,$\mathscr{M}$)
25:       $\mathscr{M}$ ← Update_Classifier(<Input,Flabel>,$\mathscr{M}$)
26:    end if
27: end while
```

The classifier IFCS consists of three steps:

(a) Initial one-shot experiment training: available data are used to obtain an initial model of the IFCS.
(b) Training over time before saturation: given a saturation training level, incoming data are used to further adjust the model.
(c) Correction after training saturation: beyond the saturation level, incoming data are used to observe the evolution of classification performance which allow to correct the classifier if necessary.

As mentioned earlier, the proposed algorithm handles partially labeled data whose relevance is particularly of importance for step (c). If data are labeled, the classifier uses the true label to check its adaptation capability and controlling the evolution of the data in terms of drift detection. If the data are not labeled, the classifier estimate the class label of the incoming data and such a label is rather dynamic, because the data distribution may change over time regardless of the closeness criteria (in terms of distance to the existing prototypes). If the unlabeled input causes the generation of new prototype, the label of this later remains undefined until a new label input falls within its influence region. In such a case, the unlabeled prototype and lately covered input are tagged by the class label of the new input. This procedure is similar to that used in Gabrys and Bargiela (2000). In contrast, if a new unlabeled input falls within the influence region of a prototype whose label is known, a dynamic label is assigned to the input. The dynamic label can change over time as evidence is sensed by mechanisms of data density.

Because the IFCS is based on GFMMNN to develop the incremental adaptation, in the following, a brief description of GFMMNN is presented and the way fuzzy hyperboxes are formulated as linguistic variables.

## 3.1 Generalized fuzzy min–max neural network

GFMMNN (Gabrys and Bargiela 2000) is a neural network that consists of three layers: $F_1$, $F_2$, and $F_3$. The input layer, $F_1$, consists of $2 \times n$ processing nodes, twice as many dimensions as of the input.

The hidden layer, $F_2$ consists of nodes in the form of hyperbox fuzzy set. These nodes are created during training. The connections between $F_1$ and $F_2$ represent the min–max points of hyperboxes, while the transfer function of the $F_2$ nodes is the hyperbox membership function. The min and max points are stored in the matrices $V$ and $W$, respectively. The connections are adjusted using a learning algorithm. The layer, $F_3$, is the class layer. Learning consists of three steps: hyperbox expansion, hyperbox overlap test, and hyperbox contraction.

Given a training labeled input $x_k$ defined as $[x_k^l, x_k^u]$ ($l$: lower corner, $u$: upper corner), the algorithm attempts to accommodate it in one of the existing hyperboxes of the same class. The selected hyperbox is the one that allows for the highest degree of membership of the input. In precise terms, let a hyperbox $B_j$ be defined by a min point $V_j = [v_{ij}]_{i=1\ldots d}$ and a max point $W_j = [w_{ij}]_{i=1\ldots d}$, and given an input $x_k$, then the membership value of $x_k$ to $B_j$ is computed as:

$$B_j(x_k) = \underset{i=1\ldots d}{\mathrm{Min}}\Big(\mathrm{Min}\big(\big[1 - f\big(x_{ki}^u - w_{ji}, \gamma_i\big)\big], \big[1 - f\big(v_{ji} - x_{ki}^l, \gamma_i\big)\big]\big)\Big) \tag{4}$$

where

$$f(x,\gamma) = \begin{cases} 1 & \text{if } x\gamma > 1 \\ x\gamma & \text{if } 0 \le x\gamma \le 1 \\ 0 & \text{if } x\gamma < 0 \end{cases} \tag{5}$$

and $\gamma$ is sensitivity parameter regulating how fast the membership values decrease, $v_{ji}$ is the $i$th min point for the $j$th hyperbox and $w_{ji}$ is the $i$th max point for the $j$th hyperbox.

Once the hyperbox with the highest degree of membership, the expansion test is performed. It consists of two conditions. The first should ensure that the expansion of the hyperbox should not lead the hyperbox to exceed a specified maximum limit $\theta$:

$$\underset{i=1\ldots d}{\forall}(\mathrm{Max}(w_{ji}, x_{ki}^u) - \mathrm{Min}(v_{ji}, x_{ki}^l)) \le \theta \tag{6}$$

The second allows to ensure the label compatibility. If the input is not labeled, then the expansion follows immediately. If the input is labeled, three cases can occur: (1) the hyperbox is not labeled, then it is assigned the label of the input and expanded; (2) if the labels of the input and hyperbox are the same, then the hyperbox is expanded; (3) if they have different labels, then select another hyperbox and re-check the expansion conditions. If neither the existing hyperboxes include or can expand to include the input, then a new hyperbox $B_k$ is created.

After expansion, the overlap test is executed. If the hyperbox $B_j$ creates overlap with the hyperboxes of other classes, then its removal is required, that is, the task of the the contraction step which indeed removes the overlap only on the one dimension where the overlap is minimum. The goal of contraction is to avoid ambiguity in the classes.

## 3.2 From hyperboxes to explicit rules

The hyperbooxes defined by the membership function shown in (1) can serve to explicitly formulate the classification rules of the form:

$$R_r \equiv \text{If } x \text{ is in } H_j \text{ then } y_r \text{ is } C_i \tag{7}$$

This is equivalent to say:

$$R_r \equiv \text{If } x \text{ is } B_j \text{ then } y_r \text{ is } C_i \tag{8}$$

As a second representation that provides lends itself more interpretability is the one that expresses the hyperboxes as fuzzy linguistic labels. These linguistic labels are derived by projecting the hyperboxes on each dimension. This allows to represent the rules of the form (9) in a typical fuzzy rules expressed in form (2). Before arriving to this results, the quality controller performs merge of hyperboxes of the same class that are adjacent. Following the method adopted in Gabrys (2002), two hyperboxes can be merged using various similarity measures. The most straightforward measure[1] is that expressed in Eq. 4 and which can be adapted to the case of hyperboxes as follows:

$$\mathrm{Sim}(B_j, B_k) = \underset{i=1\ldots d}{\mathrm{Min}}\Big(\mathrm{Min}([1 - f(b_{kj}^u - w_{ji}, \gamma_i)], [1 - f(v_{ji} - b_{ki}^l, \gamma_i)])\Big) \tag{9}$$

where $b\_$ indicate the coordinates of $B_k$. Other measures include the maximum possible distance between hyperboxes and the minimum gap between hyperboxes.

[1] Actually it is not a similarity measure, since it does not satisfy the symmetry property, but it is referred to so just in the sense of closeness

Using these similarity measures, the two most similar hyperboxes are found and temporarily aggregated, four tests must be carried out: (1) the overlap test: checks if the resulting hyperbox does not overlap with other hyperboxes of other classes; (2) size limit: checks whether the size of the temporarily aggregated hyperbox does not exceed the threshold $\theta$; (3) minimum similarity: checks whether a minimum similarity between the aggregated hyperbox is ensured; (4) class compatibility: checks if the hyperboxes belong to the same class (or at least one is unlabeled). If one of these tests fails, the temporary aggregation is dissolved and a new potential pair is chosen. This process is repeated until there are no more hyperboxes that can be aggregated. Note the difference between aggregation and extension. Although extension is followed by contraction, aggregation is not.

To ensure a systematic projection of the resulting hyperboxes after aggregation on the various dimensions and obtaining an optimal number of linguistic labels, adjacent hyperboxes of the same class are aggregated in a second run. However, in this run, the contraction operation is executed not only in one dimension with the smallest overlap, but also with all dimensions. This results in contiguous and homogenous patches $M^i = \bigcup_{j=1}^{m}\{[l_j^i, u_j^i]\}$ along each dimension $i$.

Such a patching has to satisfy some constraints defined in Pedrycz and Gomide (1998) known as the frame of cognition (FOC). This latter stipulates that a fuzzy set (patch) along a dimension must satisfy: normality, typicality, full membership, convexity, and overlap. To fulfill these constraints, the projection of the hyperboxes is transformed into trapezoidal forms.

Let $A^i$ be the set of breakpoints characterizing the trapezoids $\langle a_j^i, b_j^i, c_j^i, d_j^i\rangle$ :

$$A^i = \{a_1^i, b_1^i, c_1^i, d_1^i, , a_m^i, b_m^i, c_m^i, d_m^i\} \qquad (10)$$

along the $i$th dimension.

$$a_j^i = \begin{cases} l_j^i - \varepsilon & j = 1 \\ \frac{u_{j-1}^i + l_j^i}{2} - \varepsilon & 1 < j \leq m \end{cases} \qquad (11)$$

$$b_j^i = l_j^i \qquad (12)$$

$$c_j^i = u_j^i \qquad (13)$$

$$d_j^i = \begin{cases} \frac{u_j^i + l_{j+1}^i}{2} + \varepsilon & 1 \leq j < m \\ u_j^i + \varepsilon & j = m \end{cases} \qquad (14)$$

## 3.3 Adaptation of the hyperbox size

It is clear that the parameters play a central role in reflecting the performance of any algorithm. This applies also to GFMMNN. The results shown in the paper have been obtained based on an estimated parameter setting. You may notice that IFCS starts with an initial offline training step during which one can estimate reasonable values of the algorithm parameters. However, this does not solve the entire problem, because the data we use during the offline phase may not remain consistent with the data that comes over time in the future according to the spirit of incremental learning; hence, the challenge of adapting the parameters online. In the present study, we started with the best parameters, including $\theta$ that allowed obtaining the best performance in the initial offline phase. Basically, our main motivation behind such an initial training step is to find some estimates of the parameters. Once we have the estimate of $\theta$, we can tune the algorithm during the online phase. Algorithm 2 shows the steps used to adapt incrementally the value of $\theta$. This algorithm is to be merged within Algorithm 1.

**Algorithm 2** : Adaptation of $\theta$

```
1: Start with $\theta_{init}$
2: while training in incoming data do
3:    Compute the prediction if the data example is labeled (Err ← Test_Classifier(<Input,Label>,$\mathscr{M}$))
4:    if prediction=wrong then
5:       count_missed=count_missed +1;
6:    else
7:       count_missed=0;
8:    end if
9:    if count_missed=5 then
10:      $\theta = \theta - 0.0001$;
11:   end if
12:   Cumulated_Err ← Cumulated_Err+Err
13:   if Detect_Drift(Cumulated_Err)=true then
14:      $\theta \leftarrow \theta_{init}$
15:   end if
16: end while
```

The idea is to reduce $\theta$ in order to generate low granular hyperboxes, which will be then optimized in later stage through the aggregation stage. In case drift is detected, it is set back to the initial value so that we try to obtain larger hyperboxes to save space memory.

Regarding the minimum similarity between hyperboxes that can be merged during the aggregation stage, the value can be empirically determined during the offline stage and kept constant. Higher values (i.e., the membership of one hyperbox to the second is high) lead to lower aggregation rate. This may not provide the minimal number of hyperboxes. Recall that the number of hyperboxes corresponds to the number of fuzzy rules. In the present study, we rather test over an interval of values, say [0.6, 0.99] and retain the value that allows in getting the smallest number of hyperboxes at the end of the aggregation process. This is possible because the process is done independently of the data for generating the fuzzy rules at any time point. Another alternative that we have not investigated is to decide ahead of time the maximum number of fuzzy rules that we are interested in and apply our method (iterative search over an interval of values) to find the one leading a number of hyperboxes less than the specified one.

## 4 Drift handling

In dynamic environments, very often, the data distribution drifts over time leading to performance deterioration of the system, that is, the model built using old data becomes inconsistent with the new data. To tackle this problem, the system needs to be equipped with appropriate mechanisms to handle concept drift allowing to monitor the performance of the system efficiently. Although concept drift has been investigated in several papers (Tsymbal 2004), it is far from being solved. In many reported research, the type of change is not well described or it is not even clear whether drift indeed exits.

The current state-of-art techniques in the context of concept drift are rather data driven, meaning that drift is handled only from the perspective of data. There exist several techniques: instance selection, instance weighting, and ensemble learning. Instance selection (Klinkenberg 2004) is the best-known technique and includes two methodologies: fixed window and adaptive window, where the model is regenerated using the last data batches. The instance selection technique suffers the problem of window size in case of fixed size and the pace of drift when adaptive windowing is adopted. Using instance weighting (Widmer and Kubat 1996), it is often hard to decide which instances should be assigned higher weights, although some heuristics using aging and typicality can be devised. Using the ensemble learning (Kuncheva 2004), the idea of instance selection is generalized so that many classifiers vote. Their weight is changed so that the successful classifiers that detect drift are rewarded and/or those which do not detect drift are replaced by new ones trained on newly arriving data.

Although there are several research avenues using data-driven techniques, we prefer to focus on model-driven drift handling techniques. Model-driven drift means using appropriate models that are incremental and able to handle drift. This approach rests on our previous studies in Sahel et al. (2007), Bouchachia (2009) and Bouchachia et al. (2007). We are interested in classification and clustering models that are capable of handling drift in a systematic way without relying on additional techniques, such as time windowing. Robust approaches yielding a balance between incremental learning and forgetting are needed to deal with changing environments.

In this paper, we combine three mechanisms: staleness, penalization and overall accuracy. The first two measures intend to tackle the problem of model complexity, but also gradual drift. If one of the two measure values falls below a very small threshold, called *removal threshold*), the hyperbox is removed. The last one is intended for handling gradual and abrupt drift.

- Staleness: tracks the activity of the hyperboxes in making decisions about the new input and such an activity indicates that the recent incoming new input patterns emanate from the input space covered by the hyperbox. Those stale hyperboxes tend to cover obsolete regions. We propose the following formula to express the staleness mechanism:

$$w_i^{(t)} = \zeta^{(t-a_t)} \tag{15}$$

  where $t$ and $a_t$ indicate, respectively, the current time index and the last time the hyperbox $i$ was a winner. Note that small values of the forgetting factor $\zeta$ accelerates the reduction in the weight. Clearly, if the staleness is long (that is $t - a_t$ is large enough), $w_i$ diminishes and then the hyperbox vanishes.
- Penalization: the accuracy of the decisions made allows also to observe the evolution of the model in terms of consistency with the recent input patterns. The aim is to ensure that the accuracy does not deteriorate (at least significantly). To ensure that we adopt similar forgetting formula as that shown in Eq. 15

$$z_i^{(t)} = \zeta^{(s_i)} \tag{16}$$

  where $s_i$ is number of errors made by the hyperbox since it has been created. Again, the weight decreases exponentially as the number of errors increases. Note that the smaller the value of $\zeta$, the more quickly are the forgetting speed and model update.

- On the other hand, we also tried the approach proposed by Gama et al. (2004) for handling gradual and abrupt changes based on the number of errors produced by the learning model during prediction. It suggests that the performance of the system is measured following the binomial distribution which provides the general form of the probability for the random variable that represents the number of errors in a sample of $n$ examples. For each arriving pattern, the error rate is the probability to misclassify it (denoted as $p_i$) with the standard deviation $s_i = \sqrt{p_i(1 - p_i)/i}$.
- The probability is computed as $p_i \pm \alpha \times s_i$ with the confidence interval $1 - \alpha/2$. Based on this idea, one can control the effect of the newly arriving data, if $p_i + s_i \geq p_{\min} + a \times s_{\min}$ then drift is detected—$a$ is a multiplier. In Gama et al. (2004), $a$ was assigned the value 3. To apply this method all we need is to continuously update the values of $p_{\min}$, $s_{\min}$ every time a new pattern $i$ arrives such that $p_i + s_i < p_{\min} + a \times s_{\min}$. It is however important to note that in the present paper, we do not consider a time window as used in (Gama et al. 2004), but rather once drift is detected, the weight of the hyperboxes is sharply decreased. This mechanism accelerates the production of a new model that fits the new incoming data.

## 5 Simulation results

To evaluate the approach proposed, we use a data set that models an ambient intelligent environment. The aim is to learn to classify the behavior and habits of the inhabitants of a student dormitory (iDorm). Basically, the iDorm test bed will serve to predict the different classes of user activities. The iDorm test bed is provided by the University of Essex, UK[2]

The data are generated by recording the student activities in the iDorm which is equipped with embedded sensors, actuators, processors and heterogeneous networks that are concealed in a way letting the user behaves naturally, unaware of the hidden intelligent and pervasive infrastructure of the room. The data (11 input variables) are provided by the sensors: internal light level, external light level, internal temperature, external temperature, chair pressure, bed pressure, occupancy and time, etc. The output is provided by the actuators and consist of six variables (that define the classes): variable intensity spot lights, the desk and bed side lamps, the window blinds, the heater and PC-based applications comprising a word processing program and a media playing program. The outputs cover the spectrum of physical devices and computer-based applications found in a typical study bedroom environment.

The goal of the study is to use the iDorm data obtained over long periods of time (more than one season so that various aspects, e.g. concept drift, can be conveniently studied) to build a transparent classification model capable of learning the user's behavior (and habits) as the user acts on the actuators given some environmental conditions that are captured by the sensors. Here, we use the data pertaining to two inhabitants $I_1$ and $I_2$: with the first, the observations of 2 months June and September are used, while with the second, the observations of March and June are used.

Among other adaptation aspects worth discussing in the context of incremental learning are the evolution of the accuracy relying on Eq. 17 that expresses the error rate defined over time as:

$$p_i(t) = |\text{misses}(t)|/|\text{Seen\_so\_far}(t)| \quad (17)$$

and the evolution of the number of hyperboxes in response to the event of new data arrival. Further aspects are such as the effect of various parameters and a comparative study against other hyperbox-based algorithms are considered too.

To enable such a discussion, we split each of the data sets into two subsets. The first subset contains 75% samples for the first month that serve to sequentially train the classifier. The second subset contains 25% of the first month and all samples of the second month. Because we do not have explicit classes, we consider any distinct combination of the actuators' values as a class. The details of the experimental data are shown in Table 2.

Setting the parameters is in general a difficult issue. However, in this study, some of the parameters have been fixed based on the initial experiments, others are based on "standard" values and some others are adaptive like the case of $\theta$. Table 3 outlines the parameter setting. Recall that the value of the hyperbox ($\theta$) is online and adaptively computed over time as shown in Algorithm 2. As to the forgetting factor, the smaller its value, the quicker the forgetting speed and model update. Based on the initial experiments, we found that the value of 0.8 is very reasonable, it allows a certain stability in the model being learned in the sense that hyperboxes do not vanish quickly. It ensures that these will disappear if they do not win for a long period of time. Moreover, the value 0.8 is set taking the various types of drift into consideration. For abrupt drift, a small value is appropriate to change the model quickly, for gradual and cyclic change, a larger value should be better. Unfortunately, we cannot know whether the data contain some particular type of drift, given that it comes over time. Hence, a "middle" value, which is 0.8, is

[2] With acknowledgment to Dr. Hagras for furnishing this study with the iDorm data.

**Table 2** Characteristics of the data sets

| Data set | Training (offline) | Online testing (the same season) | Online testing (from the next season) |
|---|---|---|---|
| Inhabitant 1 | 684 | 220 | 561 |
| Inhabitant 2 | 246 | 80 | 995 |

**Table 3** Parameter setting

| Parameter | Value |
|---|---|
| Saturation level | 0.95 |
| Max value of the hyperbox size ($\theta_{\text{init}}$) | 0.3 |
| Min value of the hyperbox size ($\theta_{\min}$) | 0.01 |
| Sensitivity ($\gamma$) | 0.3 |
| Forgetting factor ($\zeta$) | 0.8 |
| Removal threshold | 0.0001 |

considered in this study. It must also be mentioned that having an incremental learning algorithm such as GFMMNN helps in dealing to some extent with the sensitivity of the parameters. The forgetting factor $\zeta$ and the the saturation level in conjunction with the training data for the offline training phase will be discussed in Sects. 5.2 and 5.3, respectively.

### 5.1 Insight into the actual accuracy

When considering the adaptation potential of the algorithm, Fig. 2 shows the adaptive behavior. It is clear from the figure that online adaptation corrects the inaccurate behavior of the classifier. Although the adaptation may not be perfect, the improvement of the accuracy by means of adaptation is strong. It is quite interesting to remark that the adaptation is useful even on data that can look presumably easier to classify as compared to an offline testing. Indeed, considering the 25% of the unseen data from the same season for which the classifier has been trained (see indication on Figs. 2, 3), the results of offline testing are of lesser quality when compared with those produced by the continuous adaptation.

Very similar behavior of the classifier is obtained on the data related to the second inhabitant. Figure 4 shows that the classifier on this data set performs better, since the error always decreases in both cases with and without adaptation. However, the contribution of the adaptation is intelligibly better and such a contribution is observed even with the first part of the testing data that stems from the same period of time (see Fig. 5).

Concerning the evolution of the hyperboxes, the role of the adaptation is to control the quality of the hyperboxes in terms of coverage and echoing the online change. In a dynamic situation, the accuracy of the classifier deteriorates steadily if the change is not efficiently handled. In presence of drift, the problem of ensuring the efficiency is even more crucial. In our case, the seasonal change is of particular interest as we have data emanating from two different months.

Figures 6 and 7 reflect the seasonal change. During training, the number of the hyperboxes increases and could

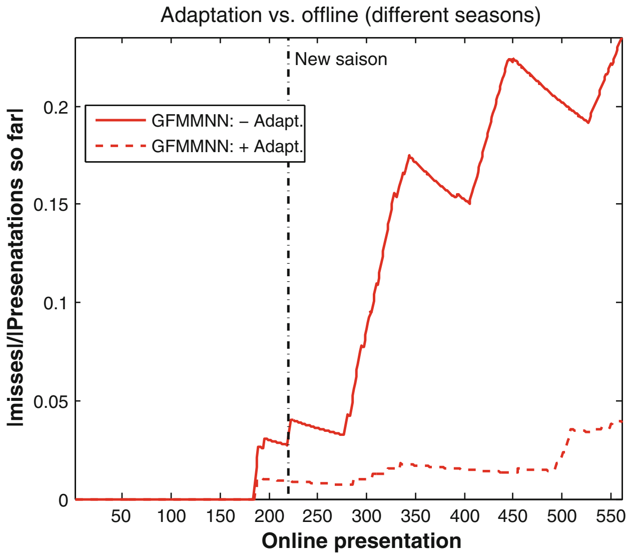

**Fig. 2** Online adaptation: data from both seasons (Inhabitant 1)

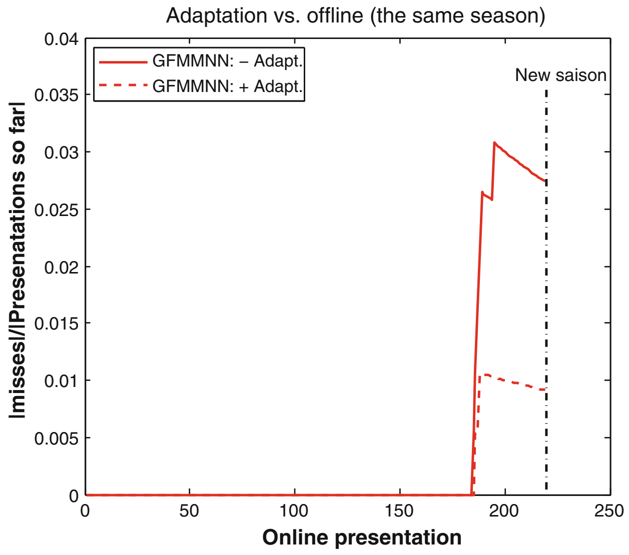

**Fig. 3** Online adaptation: data from the same season (Inhabitant 1)

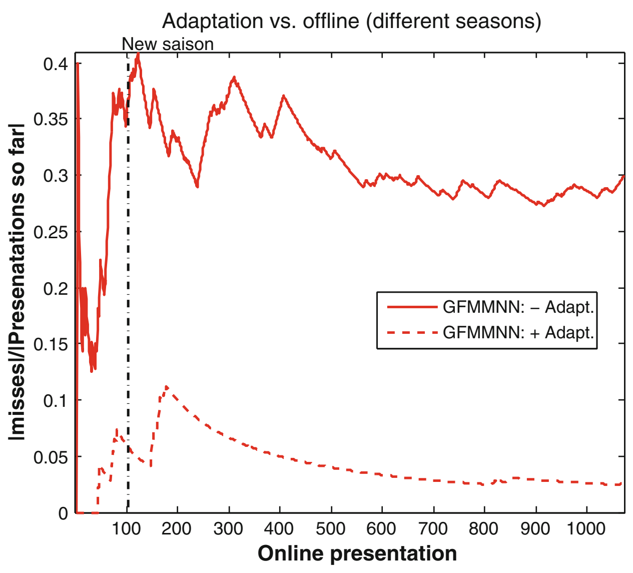

**Fig. 4** Online adaptation: data from both seasons (Inhabitant 2)

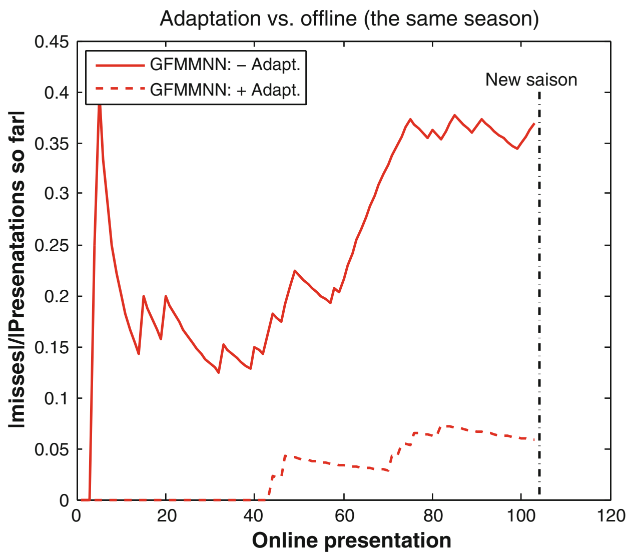

**Fig. 5** Online adaptation: data from the same season (Inhabitant 2)

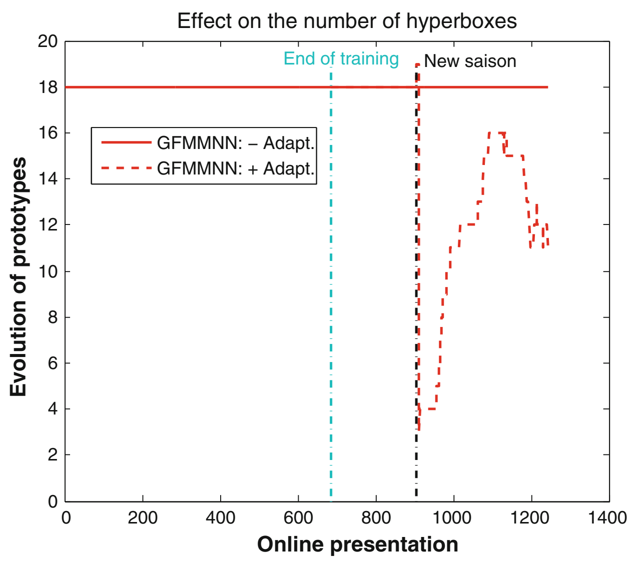

**Fig. 6** Evolution of hyperboxes number (Inhabitant 1)

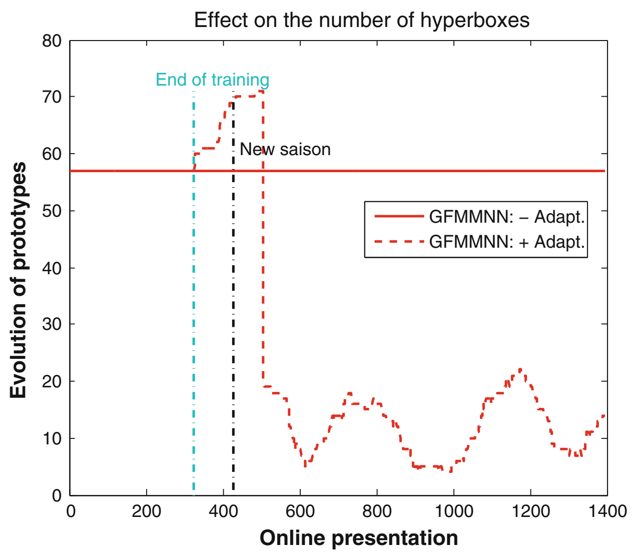

**Fig. 7** Evolution of hyperboxes number (Inhabitant 2)

continue increasing on the first part of the tuning (testing) data (see for instance Fig. 7). Once the transition between the two seasons is encountered, most of the hyperboxes do not win any competition over the new data, thus becoming stale and covering non-active regions of the data. Hence, the three mechanisms described in Sect. 4 react by removing stale and inaccurate hyperboxes. Then, the adaptation mechanisms allow the algorithm to adapt to the new data as shown in both figures.

Note that at any time, explicit fuzzy rules can be generated following the procedure highlighted in Sect. 3.1 in either forms Eqs. 7 or 2 relying on the expressions of Eq. 11.

## 5.2 Insight into the forgetting factor

Another aspect that can be analyzed is the drift mechanism. In the following, we focus on the staleness mechanism (Eq. 15) in particular, because it turned out in our experiments that it is the one that mostly affects the complexity of the model and impacts the online adaptation. Figures 8, 9 and 10 provide insight into the effect of the forgetting factor $\zeta$ shown in Eq. 15. They are obtained after setting $\zeta$ to 0.6, 0.8 and 0.95, respectively.

Clearly, the value of $\zeta$ plays a central role in getting rid of stale hyperboxes. Increasing the value tends to lead to a complex model, but that has a positive effect on the accuracy. The value 0.6 results actually in over-fitting as

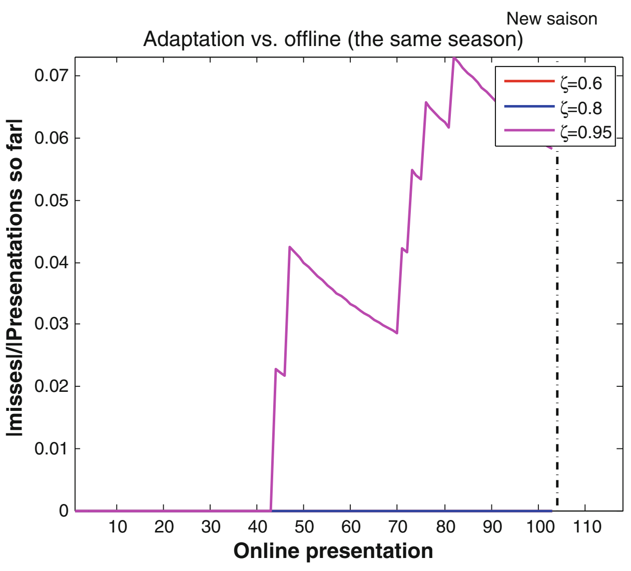

**Fig. 8** Results of testing the classifier on data of the same season by varying $\zeta$—Inhabitant 2

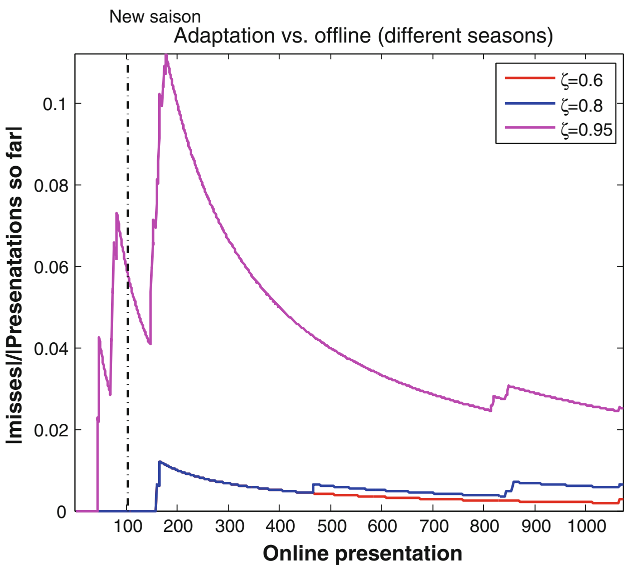

**Fig. 9** Results of testing the classifier on data of a different season by varying $\zeta$—Inhabitant 2

one can see from figs. 8 and 10. The number of hyperboxes is very often less than the number of existing classes (that is 5). This has been checked for several values <0.8. To see such effect from the accuracy perspective, the testing data have been presented to the models obtained by varying $\zeta$. The results are shown in Table 4. The optimal value is a fortiori 0.8.

## 5.3 Insight into the saturation level

Algorithm 1 consists basically of three stages: (1) offline training, (2) tuning through labeled data and (3) online

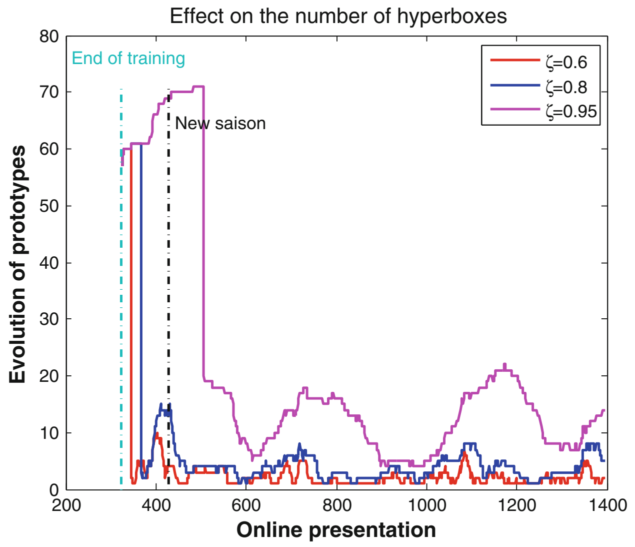

**Fig. 10** Evolution of hyperboxes by varying $\zeta$—Inhabitant 2

**Table 4** Effect of the forgetting factor $\zeta$

| $\zeta$ | Accuracy |
|---|---|
| 0.95 | 0.8396 |
| 0.8 | 0.8834 |
| 0.7 | 0.8144 |
| 0.5 | 0.0616 |

adaptation. In the present section, we will look at the effect of the size of data which is used in the first stage. This is related to the saturation level referred to in step 10 of Algorithm 1. Setting the saturation level at 95% after a certain number $n$ of examples reflects the efficiency of the algorithm. Considering the data related to Inhabitant 1 and changing the size of the offline training data (the learner can see the data more than once). Figures 11 and 12 show the effect of reducing the size of data used in the first phase (offline training) of the algorithm.

The behavior of the algorithm expressed in terms of the actual accuracy changes as the size of the offline training data used in the first stage changes. Unexpectedly, when the size is small, the algorithm performs slightly better as portrayed in Fig. 11. In fact, in the case of 228 examples with adaptation (indicated by a dashed line and denoted by $< + A >$). The accuracy of the algorithm starts to decrease very late compared to the cases of 456 and 684 examples. This can also be noticed in Fig. 12, where the number of hyperboxes in the first case evolves in a particular range. In the other two cases, the number of hyperboxes changes somehow radically at some points. However, in all cases, the online adaptation keeps the error rate low. Quite interesting is the fact that when presenting the testing data to the algorithm according to these three cases, we obtained the accuracy values shown in Tables 4 and 5.

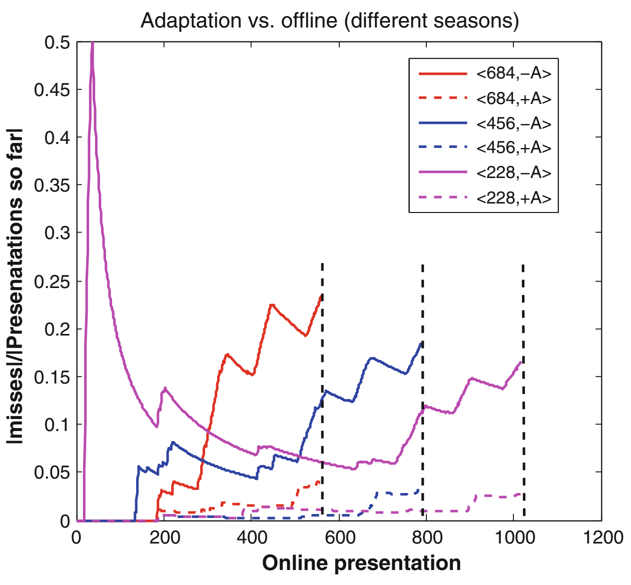

Fig. 11 Effect of the actual accuracy

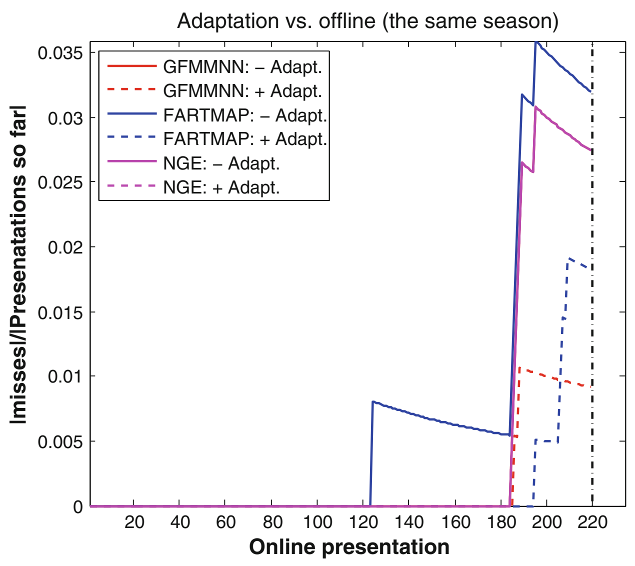

Fig. 13 Results of testing the classifier on data of the same season—Inhabitant 2

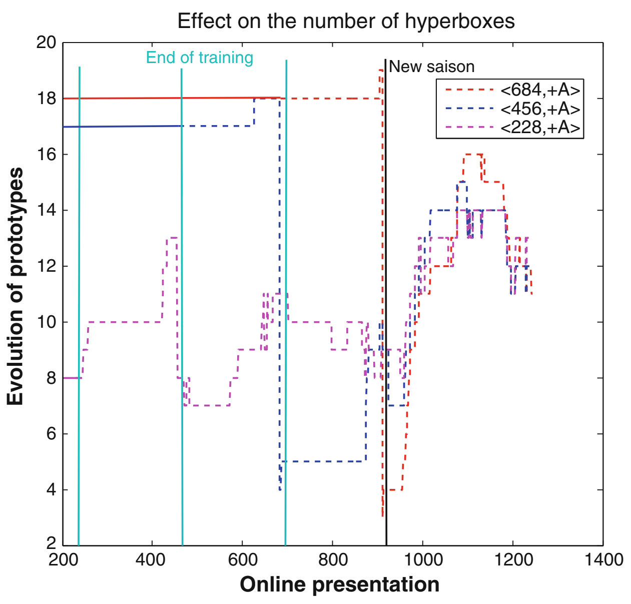

Fig. 12 Effect of the number of hyperboxes

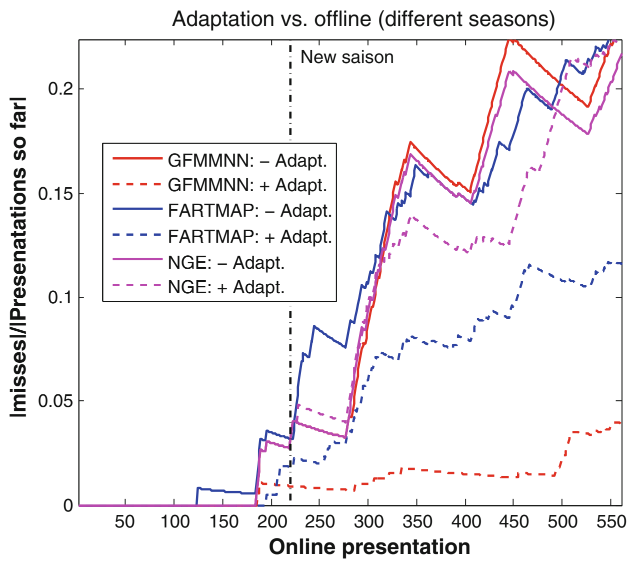

Fig. 14 Results of testing the classifier on data of a different season—Inhabitant 2

Table 5 Effect of the training data size

| Size | Accuracy |
|---|---|
| 228 | 0.8328 |
| 456 | 0.8112 |
| 684 | 0.6239 |

The reason that with less training data the classifier performs better is basically due to the mechanism of adaptation of the number of hyperboxes. The classifier starts earlier to remove redundant hyperboxes and keep the more consistent ones.

## 5.4 Comparative study

To compare the presented algorithm using various 0 incremental learning algorithms, the same experimental setting of IFCS has been adopted. The algorithms of interest are the fuzzy ARTMAP (FARTMAP) (Carpenter et al. 1991) and the nearest generalized exemplar (NGE) (Salzberg 1991). These have been chosen because they provide the same structure. They create hyperboxes in a

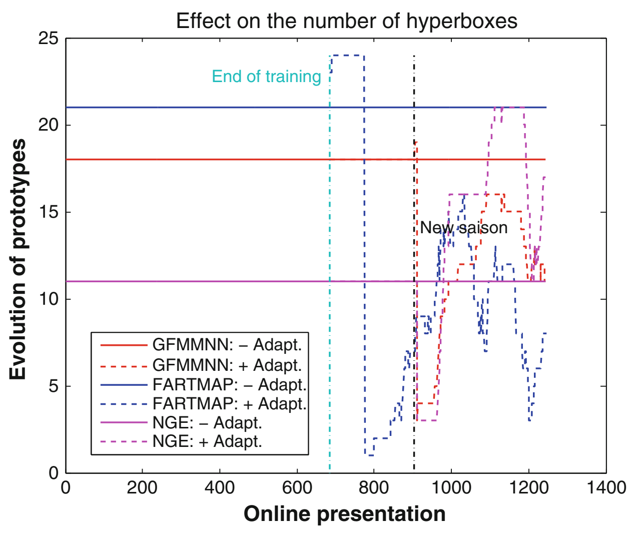

**Fig. 15** Evolution of hyperboxes—Inhabitant 2

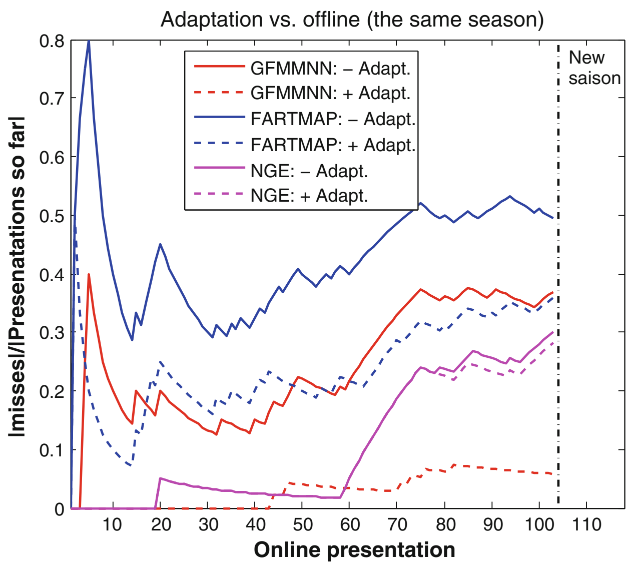

**Fig. 16** Results of testing the classifier on data of the same season—Inhabitant 2

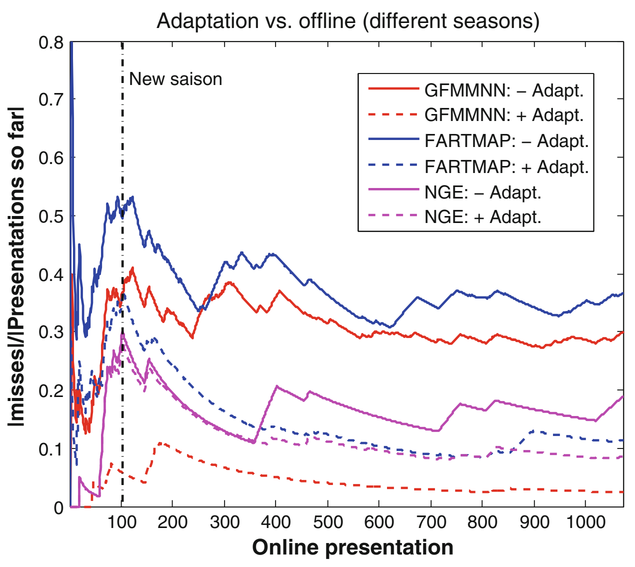

**Fig. 17** Results of testing the classifier on data of a different season—Inhabitant 2

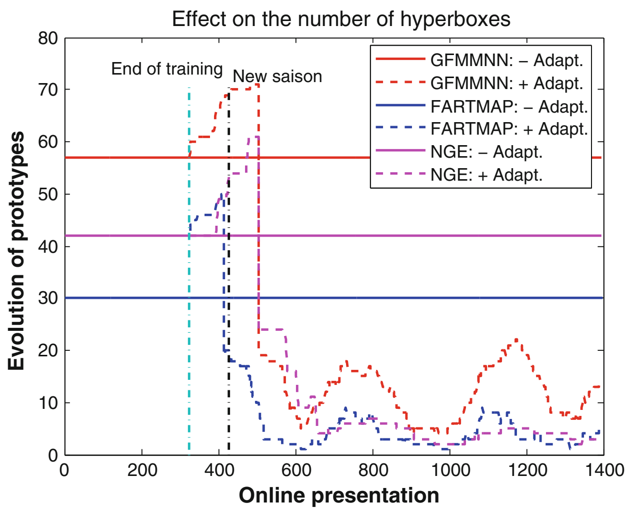

**Fig. 18** Evolution of hyperboxes—Inhabitant 2

similar way as in GFMMNN. They are incremental and they fit the IFCS framework in a straightforward manner. Although NGE does not possess any parameter, FARTMAP requires the setting of four parameters: baseline vigilance ($\rho_a$), map field vigilance ($\rho_{ab}$), choice parameter ($\alpha$) and learning rate ($\beta$). After initial experiments, we found out that the best setting that produces the highest classification accuracy corresponds to: $\rho_a = 0.75$, $\rho_{ab} = 0.3$, $\alpha = 0.1$ and $\beta = 1$ (fast learning mode). The parameter setting for GFMMNN remains the same as in the previous experiments.

The execution of the algorithms on both data sets yields the results portrayed in Figs. 13, 14, 15 for the first inhabitant data and Figs. 16, 17, 18 for the second inhabitant data. These figures show that IFCS based on GFMMNN provides the best results as compared to FARTMAP and NGE. It is worthwhile to notice that all the algorithms behave in the same way, but differ just in terms of accuracy. Of course the difference lies in the reasoning underlying the algorithms, the way the hyperboxes are generated and the matching function used in these algorithms.

## 6 Conclusion

In this paper, a new incremental learning algorithm is proposed to deal with classification in dynamic environments.

It relies on the generalized fuzzy min–max neural networks. Based on the data stemming from an ambient intelligence application, the approach proposed shows good performance due to its adaptation capabilities. As a future work, the deep insight into the evolution on a per day basis is still required since the ultimate goal of the present study is to learn the behavior of the inhabitants so that an automatic tuning of the heater, the light, etc. becomes possible.

## References

Angelov P (2004) An approach for fuzzy rule-base adaptation using on-line clustering. Int J Approx Reason 35(3):275–289

Angelov P, Lughoferb E, Zhou X (2008) Evolving fuzzy classifiers using different model architectures. Fuzzy Sets Syst 159:3160–3182

Bouchachia A (2004) Incremental rule learning using incremental clustering. In: Proceedings of the 10th conference on information processing and management of uncertainty in knowledge-based systems, vol 3, pp 2085–2092

Bouchachia A (2004) Maintaining knowledge bases via incremental rule learning. In: Proceedings of the international workshop on soft computing for information mining, pp 51–63

Bouchachia A (2006) Incremental learning via function decomposition. In: Proceedings of the international conference on machine learning and applications, pp 63–68

Bouchachia A (2009) Encyclopedia of data warehousing and mining, chapter incremental learning, 2nd edn. Idea-Group, pp 1006–1012

Bouchachia A, Gabrys B, Sahel Z (2007) Overview of some incremental learning algorithms. In: Proceedings of the 2007 IEEE international conference on fuzzy systems, pp 1–6

Bouchachia A, Mittermeir R (2006) Towards fuzzy incremental classifiers. Soft Comput 11(2):193–207

Carpenter G, Grossberg S, Rosen D (1991) Fuzzy ART: fast stable learning and categorization of analog patterns by an adaptive resonance system. Neural Netw 4(6):759–771

de Barros J, Dexter L (2007) On-line identification of computationally undemanding evolving fuzzy models. Fuzzy Sets Syst 158(16): 1997–2012

French R (1999) Catastrophic forgetting in connectionist networks: causes, consequences and solutions. Trends Cogn Sci 3(4): 128–135

Fritzke B (1995) A growing neural gas network learns topologies. In: Advances in neural information processing systems, pp 625–632

Gabrys B (2002) Agglomerative learning algorithms for general fuzzy min–max neural network. J VLSI Signal Process Syst 32(1): 67–82

Gabrys B, Bargiela A (2000) General fuzzy min–max neural network for clustering and classification. IEEE Trans Neural Netw 11(3):769–783

Gama J, Medas P, Castillo G, Pereira Rodrigues P (2004) Learning with drift detection. In: Advances in artificial intelligence—17th Brazilian symposium on artificial intelligence, pp 286–295

Grossberg S (1988) Nonlinear neural networks: principles, mechanism, and architectures. Neural Netw 1:17–61

Hagras H, Doctor F, Lopez A, Callaghan V (2007) An incremental adaptive life long learning approach for type-2 fuzzy embedded agents in ambient intelligent environments. IEEE Trans Fuzzy Syst 15(1):41–55

Kasabov N (2001) On-line learning, reasoning, rule extraction and aggregation in locally optimized evolving fuzzy neural networks. Neurocomputing 41:25–45

Klinkenberg R (2004) Learning drifting concepts: example selection vs. example weighting. Intell Data Anal 8(3):281–300

Kuncheva L (2004) Classifier ensembles for changing environments. In: Proceedings of the fifth international workshop on multiple classifier systems, pp 1–15

Liu F, Quek C, Ng G (2007) A novel generic hebbian ordering-based fuzzy rule base reduction approach to mamdani neuro-fuzzy system. Neural Comput 19(6):1656–1680

McCloskey M, Cohen N (1999) Catastrophic interference in connectionist networks: the sequential learning problem. Psychol Learn Motivation 24:109–164

Pedrycz W, Gomide F (1998) Introduction to fuzzy sets: analysis and design. MIT Press, Cambridge

Ratcliff R (1990) Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. Psychol Rev 97:285–308

Sahel Z, Bouchachia A, Gabrys B (2007) Adaptive mechanisms for classification problems with drifting data. In: Proceedings of the 11th international conference on knowledge-based intelligent information and engineering systems (KES'07), LNCS 4693, pp 419–426

Salzberg S (1991) A nearest hyperrectangle learning method. Mach Learn 6:277–309

Sharkey N, Sharkey A (1995) Catastrophic forgetting in connectionist networks: causes, consequences and solutions. Anal Catastrophic Interference 7(3–4):301–329

Tsymbal A (2004) The problem of concept drift: definitions and related work. Technical Report TCD-CS-2004-15, Department of Computer Science Trinity College, Dublin

Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23:69–101