2014 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)
July 6-11, 2014, Beijing, China

Dynamically Evolving Fuzzy Classifier for Real-time Classification of Data Streams

Rashmi Dutta Baruah
Department of Computer Science & Engineering
Sikkim Manipal Institute of Technology
Majitar - 737136, Sikkim, India
rashmi.dbaruah@gmail.com

Plamen Angelov
School of Computing & Communications
Lancaster University
Lancaster LA1 4WA, UK
p.angelov@lancaster.ac.uk

Diganta Baruah
Department of Information Technology
Sikkim Manipal Institute of Technology
Majitar - 737136, Sikkim, India
baruah.diganta@gmail.com

***Abstract*—In this paper, a novel evolving fuzzy rule-based classifier is presented. The proposed classifier addresses the three fundamental issues of data stream learning, viz., computational efficiency in terms of processing time and memory requirements, adaptive to changes, and robustness to noise. Though, there are several online classifiers available, most of them do not take into account all the three issues simultaneously. The newly proposed classifier is inherently adaptive and can attend to any minute changes as it learns the rules in online manner by considering each incoming example. However, it should be emphasized that it can easily distinguish noise from new concepts and automatically handles noise. The performance of the classifier is evaluated using real-life data with evolving characteristic and compared with state-of-the-art adaptive classifiers. The experimental results show that the classifier attains a simple model in terms of number of rules. Further, the memory requirements and processing time per sample does not increase linearly with the progress of the stream. Thus, the classifier is capable of performing both prediction and model update in real-time in a streaming environment.**

Keywords—data streams; evolving fuzzy classifier; online classifier; real-time classification

I. INTRODUCTION

Traditionally, classifiers (or predictive models) are trained using historical input-output data and afterwards the resulting models are applied to predict the class (or output) for new unseen input data. This model learning approach using past data is often referred to as *batch* or *offline* learning. However, applications that generate streams of data pose new challenges to such learning methods due to the reasons that streams arrive continuously in high speed and the data pattern often changes dynamically. The change in the statistical properties of the target concept or target variable over time is referred to as *concept drift* [1]. The term concept drift is more often associated to gradual changes, while the abrupt changes are referred to as *concept shift* [2]. Further, the ever growing amount of data would increase the processing requirements such that there may be situations where the entire data would not fit in the memory or the computation time would become prohibitively long for offline training. In such scenarios, the model that is trained offline may initially perform well but the performance would start to degrade with the progress of the stream due to the evolving nature of the data. Thus, the key challenges to any data stream learning approach can be summarized as [3, 4] :

(i) **Fast and memory efficient**- for on-line and real-time prediction, the processing of each data sample should be done in small constant amount of time to keep up with their speed of arrival, and the memory requirements should not increase appreciably with the progress of the data stream.

(ii) **Adaptive**- Adapts (evolves) the model structure and parameters in the presence of concept drift or concept shift and should be able to present up-to-date model.

(iii) **Robust to noise** - in a streaming environment, it is difficult to detect noise from data shift. Noisy data can interfere with the learning process, for example, a greedy algorithm that adapts itself as soon as it sees a change in the data pattern may overfit noise by mistakenly interpreting it as data from a new concept. On the other hand, if it is too conservative and slow to adapt, it may overlook important changes.

To meet such requirements, the area of *evolving fuzzy systems* emerged that focuses on *online* learning of predictive fuzzy models that are capable of adapting autonomously to concept drift and shift [5-7]. Very often, the learning algorithms that are related with continuously arriving data or data streams are referred interchangeably as *streaming algorithms* or *online algorithms*. Both streaming and online algorithms are very similar as they need decisions to be made before all data are available and can use only limited memory. However, there is a subtle difference; the streaming algorithms can defer action until a group of samples (data chunk) arrive while online algorithms take irrevocable action as soon as each data sample arrives [8].

To build a fuzzy rule-based (FRB) classifier from input-output data (also known as *data-driven fuzzy modelling*), a common approach is to apply clustering to partition the input or input-output data to get the rules and their antecedent parameters. Each of the cluster and its centre defines a rule, and the number of clusters determines the number of rules. For example, in case of a Gaussian membership function, the

978-1-4799-2072-3/14/$31.00 ©2014 IEEE

centre of a cluster represents the centre of the corresponding membership function, and the width of the Gaussian is same as the cluster spread. For first-order Takagi-Sugeno type of rules, consequence parameters of the rules are estimated separately using methods like least square estimates [9, 10]. In this paper, we focus on the problem of online learning of FRB classifier and describe an approach to incrementally learn the rules of the classifier from streaming data. The resulting classifier, named *Dynamically Evolving Fuzzy Classifier* (DEFC), is simple in structure in terms of number of rules. This in turn reduces the memory requirements and computational time. Due to the incremental nature, the classifier is inherently adaptive both in terms of model structure and parameters, and is capable of detecting and reacting to concept drifts and shifts in time. At the same time, it can distinguish data shift from noise and appropriately handles noise. Moreover, the classifier performs both prediction and model update in real-time.

The rest of the paper is organized as follows: section II presents the related work, section III describes the design of DEFC, section IV discusses the experimental results, and finally section V concludes the paper with a direction to future work.

II. Related Work

In recent years, several approaches have been proposed for learning predictive models from streaming data [2, 7]. In this section, we focus on incremental learning approaches for fuzzy classifiers. In [11], the design and development of a family of evolving FRB classifiers is described. The family consists of two classifiers, *eClass0* and *eClass1* which are based on zero-order and first order multi-input multi-output (MIMO) Takagi-Sugeno type of rules, respectively. The incremental learning of rules is based on the *evolving Takagi-Sugeno* approach [10]. A rule is created around a new data sample in two conditions: (i) if the associated class label of the data sample has not been previously seen by the classifier, (ii) if the *potential* of the new data sample is higher compared to all the existing rules (or cluster centres). The potential value is the measure of data density. The second condition makes the classifier robust to noise but slower in adaptation. Along with addition of new rules, the eClass classifiers remove outdated rules based on *Age* measure. However, the age measure is required to be computed after arrival of every sample which increases the computational overhead by O(N) times the number of samples, where N is the number of rules. In [12], simplified versions of eClass classifiers are presented. The two classfiers, viz., *simpl_eClass0* and *simpl_eClass1*, retain all the advantages of the family of *eClass* while simplifying the structure adjustment phase significantly that in turn reduces the computational overhead. In [13, 14], *FLEXFIS-class* is presented which is based on FLEXFIS and the rule learning is performed using *evolving vector quantization* [15, 16]. A new data sample is considered to be a candidate for forming a new rule if it is not within the zone of influence of any existing rules. The problem with FLEXFIS-class is that it is unable to handle noise and does not incorporate any strategy to remove outdated rules over time. In [17], a hybrid evolving classifier has been proposed that is a combination of two incremental learning approaches, viz., growing Gaussian mixture model (GGMM) and minimal resource allocating neural network (MRAN). The classifier is semi-supervised as it considers partially labelled data for learning. The idea is that GGMM model is used for clustering unlabeled data so that the generated label (group) can be provided at a later stage to MRAN classifier to update its structure. The accuracy of this classifier is sensitive to two factors: the number of initial Gaussians in GGMM model, and the amount of labelled data used for learning. In [18], some improvements over *eClass* for structure adaptation phase have been suggested. The focus is on the adjustment of the parameters of premise part, mainly prototype centres, of fuzzy rules that constitutes the classifier. This classifier uses Mahalanobis distance to determine the membership degree of a sample. It follows the same principle as *eClass* that is based on calculation of potential value for creation and replacement of rules. In *eClass* , rules are created or replaced only if the new sample bears the highest potential compared to all the existing cluster centres, otherwise the sample is considered to belong to the nearest cluster (or class) and only the radius (not the centre) of that cluster is updated. However, in [18], for each new sample the classifier updates the centre of the cluster and the covariance matrix with highest membership degree using two approaches: *statistical adaptation* and *classification error based adaptation*. The results show performance improvement over *eClass* (with statistical adaptation). However, adjustment of the existing model after every new sample increases the computational overhead and consequent parameter learning convergence is affected. In [19], an evolving Fuzzy Pattern Tree (eFPT) for binary classification has been proposed, where an ensemble of pattern trees is maintained. It consists of a current (active) model and a set of neighbour models. The current pattern tree is used to make predictions, while the neighbouring trees can be considered as anticipated adaptations. When the performance of the current model is significantly low due to concept drift, the current model is replaced by one of the neighbours. The first pattern tree is learned in batch mode using a small set of training samples, which represents the current model. The neighbouring trees are derived from the current model either by expanding or by pruning the tree. Upon arrival of a new data sample, the error rate of the current model and neighbours is determined and if the current model is worse than the neighbours, then the former is replaced by the best performing neighbor. After replacement, the set of neighbouring trees are recomputed. Apparently, the processing requirements increase with the increase in ensemble size and re-computation of the neighbouring models is an overhead. Though, the authors provide some measurements to reduce these computational overheads, the proposed classifier would not be able to perform in real-time.

The learning approaches discussed above do not address all the three key issues (discussed in section I) of data streams together. The proposed classifier incorporates features that

allow it to meet the given fundamental requirements of a stream learning approach.

III. DYNAMCALLY EVOLVING FUZZY CLASSIFIER

The Dynamically Evolving Fuzzy Classifier (DEFC) is a fuzzy rule-based classifier that is built upon a regression model. The classifier consists of first-order MIMO Takagi-Sugeno rules of the following form [10, 20]:

$$Rule^i = IF\ \left(x_1 \sim x_1^{i*}\right)\ AND \cdots AND\ \left(x_n \sim x_n^{i*}\right)$$
$$THEN\ \ \boldsymbol{y}^i = \boldsymbol{x_e}\boldsymbol{\theta} \qquad (1)$$

where $\boldsymbol{y}^i = [y_1^i\, y_2^i\, \cdots\, y_m^i]$ is the m-dimensional output of i^{th} rule, $\boldsymbol{x}_e = [1\, x_1 \ldots\, x_n]$ is the extended input vector, $\boldsymbol{x}^{i*}$is the focal point of the i^{th} rule, $\boldsymbol{\theta}^i$is the matrix of local sub-system parameters given as:

$$\boldsymbol{\theta}^i = \begin{bmatrix} a_{01}^i & a_{02}^i & \cdots & a_{0m}^i \\ a_{11}^i & a_{12}^i & \cdots & a_{1m}^i \\ \vdots & \vdots & \cdots & \vdots \\ a_{n1}^i & a_{n2}^i & \cdots & a_{nm}^i \end{bmatrix} \qquad (2)$$

Thus, each element $y_1^i,\ y_2^i, \cdots\ y_m^i$of m-dimensional output $\boldsymbol{y}^i$ can be represented as:

$$y_1^i = a_{01}^i + x_1 a_{11}^i + \cdots + x_n a_{n1}^i$$
$$y_2^i = a_{02}^i + x_1 a_{12}^i + \cdots + x_n a_{n2}^i$$
$$\vdots$$
$$y_m^i = a_{0m}^i + x_1 a_{1m}^i + \cdots + x_n a_{nm}^i$$

In DEFC, before the rule learning phase, the class label of a given sample is converted to numeric form so that the consequent of a rule can be represented as in (1). For example, for binary classification *class 1* could be represented as $y_{real} = [0 \quad 1]$, and *class 2* as $y_{real} = [1 \quad 0]$. Similarly, for a three class problem, class 1 can be represented as $y_{real} = [1 \quad 0 \quad 0]$, class 2 as $y_{real} = [0 \quad 1 \quad 0]$, and class 3 as $y_{real} = [0 \quad 0 \quad 1]$. Here, the dimension, m of y_{real} is equivalent to the number of classes.

DEFC classifies each incoming sample on-line using the existing rules and when the sample's actual class is available at a later stage it updates the rule-base automatically. Thus, it performs a sequence of classification and model update phases consecutively. Like eClass [11], DEFC can learn rules from scratch. If no rules are available initially, then during classification phase all incoming data samples are assigned a dummy class label. When the real class labels of such samples are available then new rules are incorporated to the model. The classification and model update phases are summarized below:

Classification:

For a given input $\boldsymbol{x}$, the firing strength (τ^i) of each rule is determined using a Gaussian type membership function (μ^i) as given below:

$$\tau^i = \prod_{j=1}^{n} \mu_j^i(x_j)\,, i = [1, N] \qquad (3)$$

$$\mu_j^i(x_j) = e^{-\frac{\left(x_j - x_j^{i*}\right)^2}{2\left(r_j^i\right)^2}} \qquad (4)$$

where $r_j^i, (\, i = [1, N], j = [1, n])$ is the spread of the membership function and represents the zone of influence of the fuzzy rule. The membership function represents the degree of closeness of $\boldsymbol{x}$ to a specific focal point $(\boldsymbol{x}^{i*}, [i = 1, N])$.

To determine the class label of input $\boldsymbol{x}$, the overall output $\hat{\boldsymbol{y}}$ is estimated by weighted sum of outputs of all the rules using (5), and the index corresponding to the highest value element of $\hat{\boldsymbol{y}}$ is designated as the class label of the sample.

$$\hat{\boldsymbol{y}} = \sum_{i=1}^{N} \lambda^i \boldsymbol{y}^i\,;\ \lambda^i = \frac{\tau^i}{\sum_{j=1}^{N} \tau^j} \qquad (5)$$

For example, suppose for a given sample, if the overall estimated output is $\hat{\boldsymbol{y}} = [0.38 \quad 0.46 \quad 0.29]$ in a 3 class problem then the sample is assigned class label 2 because the highest value element is 0.46 and its index is 2.

Model update:

Similar to on-line data driven fuzzy model identification, the rules are formed around focal points [10], where a focal point is the data sample that represents most appropriately all other samples within its zone of influence. The focal points and the respective zones of influence in DEFC are identified using an online clustering method, viz., *Dynamically Evolving Clustering* (DEC) [21, 22]. A focal point and the associated zone of influence are the cluster centre and the cluster radius respectively. In DEFC the clustering is applied to input-output data space.

In DEC each data sample $\boldsymbol{z} = [x\, y_{real}]$ in the data stream is assigned a weight that decreases exponentially with time. If $\boldsymbol{z}$ arrives at time t_k, then its weight $w(\boldsymbol{z}, t)$ at time t is given as:

$$w(\boldsymbol{z}, t) = \gamma^{t - t_k}, \qquad (6)$$

where $\gamma \in (0,1)$ is the decay constant .

The data sample $\boldsymbol{z}$ is considered to be bounded by a hyper-sphere with a predefined radius (initial radius) or bandwidth (ρ). The neighbourhood of a sample is defined in terms of bandwidth. Two data samples are considered direct neighbours if the distance between them is less than or equal to the bandwidth. Also, two data samples are neighbours if there is a chain of data samples in-between and these intermediate samples are direct neighbours of one another. A cluster C is formed by a group of close data samples where each sample is the neighbour of the other. The cluster C is also assigned a weight which is the sum of weights of all the data samples in C at or before a given time and is given as:

$$\delta(C, t) = \sum_{\boldsymbol{z} \epsilon Z(C,t)} w(\boldsymbol{z}, t) \qquad (7)$$

where $\delta(C, t)$ is the cluster weight at time t and $Z(C, t)$ is the set of data samples that constitute the cluster C at time t.

Thus, the weight of a cluster also reduces exponentially with time. The DEC approach uses a notion of *core* and *non-core* cluster. A cluster is considered to be core if its weight is greater than or equal to a pre-specified *weight-threshold* value $(\delta \geq \beta)$, otherwise the cluster is designated as non-core $(\delta < \beta)$. With the progress of time, a core cluster may turn into a non-core, and a non-core may be declared as outliers depending on their weights. A cluster is represented with a compact structure referred to as cluster feature vector (CFV) [23] consisting of six components such as cluster centre, radius, weight, time of arrival of last data sample. All the components of the CFV can be computed incrementally.

At each time step, a data sample $\boldsymbol{z}$ is read and merged or associated to an appropriate cluster and the corresponding feature vector is updated. First, the distance from $\boldsymbol{z}$ to all the existing core clusters is determined. If $\boldsymbol{z}$ is within the neighbourhood (or cluster radius) of nearest core cluster then $\boldsymbol{z}$ is associated to this core cluster. If no such core cluster exists, then the same steps are repeated considering non-core clusters. If no such non-core cluster exists, then $\boldsymbol{z}$ is declared as a new non-core cluster. Note that at this stage, this new non-core cluster's radius is equal to the bandwidth $(r = \rho)$. As more and more samples are merged to this cluster the radius is updated and it grows over time.

When $\boldsymbol{z}$ is merged with an existing core or non-core cluster and if $\boldsymbol{z}$'s associated neighbourhood (hyper-sphere with bandwidth ρ) is not completely within the radius of the cluster then, the CFV is updated (see step2 of Algorithm 1, explained in next paragraph). This involves update of cluster centre, radius, and weight. *If the cluster is core then the update of cluster centre and radius in turn updates the antecedent parameters of the corresponding fuzzy rule.* If the cluster is non-core, and after update the corresponding weight becomes greater than the weight-threshold, then the cluster's status is changed to core. *For every such core cluster a rule is created.*

Algorithm 1 describes the process of computing the centroid (cluster centre) and updating the radius after a new data point is merged or associated to an existing core/non-core cluster. Consider a data point $\boldsymbol{z}$ which is required to be merged with a cluster with centroid $\boldsymbol{c}$. Let r_c be the radius of $\boldsymbol{c}$. Let $\boldsymbol{u}$ be a point such that $\|\boldsymbol{z} - \boldsymbol{u}\| = \rho$ and $\boldsymbol{u}$ is on the line joining $\boldsymbol{c}$ and $\boldsymbol{z}$ (Fig. 1). So, $\boldsymbol{u}$ is a boundary point which is at the boundary of a hyper-sphere bounding $\boldsymbol{z}$, such that $\boldsymbol{z}$ is the centre and ρ is the radius. Any point inside the hyper-sphere is neighbour of $\boldsymbol{z}$ and will also be neighbour of $\boldsymbol{c}$, and thus will be merged with $\boldsymbol{c}$. After processing we need to discard point $\boldsymbol{z}$, but we need to keep its neighbourhood information in order to associate any future point inside the hyper-sphere to the same cluster to which $\boldsymbol{z}$ was associated. To attain this, a bounding sphere is computed in DEC using the algorithm presented in [24] (Algorithm 1). If the point $\boldsymbol{u}$ is outside the current sphere i.e. the hyper-sphere with centre $\boldsymbol{c}$ and radius r_c then the current sphere is updated to a larger sphere that passes through $\boldsymbol{u}$ on one side and the back side of old sphere on the other side. Each new sphere will contain the old sphere and the point $\boldsymbol{u}$ as shown in Fig. 2.

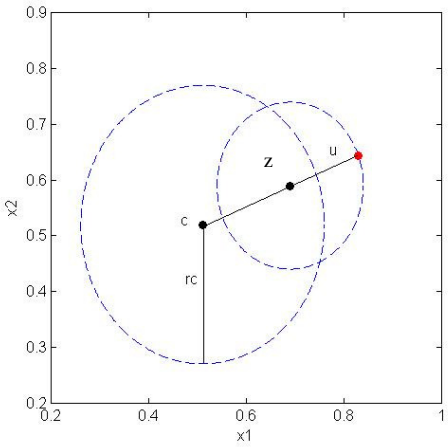

Fig 1. A point x within a cluster with centre c.

Algorithm 1 *update_centre_radius*

1. Compute distance between $\boldsymbol{c}$ and $\boldsymbol{u}$,
   $dist_{sq} = \|\boldsymbol{c} - \boldsymbol{u}\|^2$;
2. **if** $dist_{sq} > r_c^2$ **then**
   a. point $\boldsymbol{u}$ is outside the current sphere;
   b. update radius and centre;
   c. $dist = \sqrt{dist_{sq}}$;
   d. $r_c = (r_c + dist)/2$;
   e. $dist_{new} = dist - r_c$;
   f. $\boldsymbol{c} = (r_c \times \boldsymbol{c} + dist_{new} \times \boldsymbol{u})/dist$;
3. **end if**

After every T_I time interval (where T_I is the *cluster inspection time*), the status of the clusters is changed based on their weight. If any core cluster does not receive new data for a long time, its weight will gradually decrease. When its weight becomes less than the given weight-threshold then its status is changed to non-core. *In such a situation, the corresponding fuzzy rule is also disregarded and is further not involved in classification process.* Similarly, if the weight of

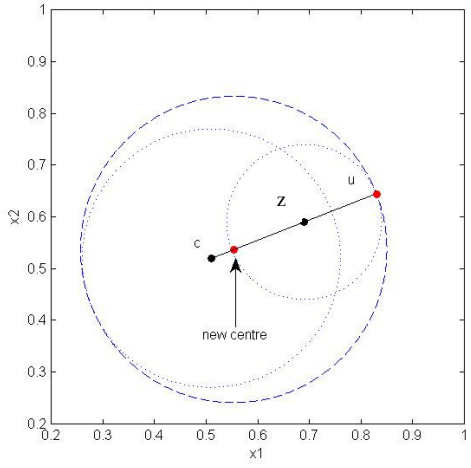

Fig 2. Updated cluster centre and radius.

any non-core cluster becomes less than a given *minimum-weight-threshold* then such clusters are considered as outliers. The cluster inspection time (8) is a function of weight-threshold, and the minimum-weight-threshold (9) is a function of the current time and the time when the cluster received the last data sample.

$$T_I = \log_\gamma\left(\frac{\beta-1}{\beta}\right) \tag{8}$$

$$\eta(t, t_e) = \frac{1-\gamma^{t-t_e+T_I}}{1-\gamma^{T_I}} \tag{9}$$

where T_I is the cluster inspection time, η is minimum-weight-threshold, t is the current time, and t_e is the time when the cluster received the last data point.

It is to be noted that the weight of a non-core cluster can be low if either the clusters are in the initial stage or they are the clusters that have not received sufficient data samples for long time. So, the former non-core clusters should be given enough time before declaring them as outliers and removing them, as they may turn into core clusters later on. Also, one should not wait indefinitely to identify a non-core cluster as outliers. The cluster inspection time and minimum-weight-threshold functions are defined in such a way that non-core clusters are removed at appropriate time and are not removed pre-maturely [21, 22].

The consequent parameters of rules are estimated *locally* using weighted recursive least squares (wRLS) approach using (10) and (11) [10].

$$\boldsymbol{\theta}_k^i = \boldsymbol{\theta}_{k-1}^i + \mathbf{V}_k^i \mathbf{x}_{ek-1}^T \lambda_{k-1}^i (\mathbf{y}_k - \mathbf{x}_{ek-1}\boldsymbol{\theta}_{k-1}^i) \tag{10}$$

$$\mathbf{V}_k^i = \mathbf{V}_{k-1}^i - \frac{\lambda_{k-1}^i \mathbf{V}_{k-1}^i \mathbf{x}_{ek-1}^T \mathbf{x}_{ek-1} \mathbf{V}_{k-1}^i}{1+\lambda_{k-1}^i \mathbf{x}_{ek-1} \mathbf{V}_{k-1}^i \mathbf{x}_{ek-1}^T} \tag{11}$$

where $\mathbf{V}^i$ is the covariance matrix of the i^{th} rule with *(n+1)×(n+1)* dimension, and $\boldsymbol{\theta}_1^i = 0$, $\mathbf{V}_1^i = \Omega\mathbf{I}$, *i=[1,N],* and Ω is set to a large value.

As described in [10], when a new rule is added, its parameters are set to the weighted average of the parameters of already existing *N* rules (12), and the covariance matrix is given by (13). When a new rule replaces an existing rule, then the former inherits the parameters from the latter. In this case also, the covariance matrix is initialized as given by (13).

$$\boldsymbol{\theta}_k^{N+1} = \sum_{i=1}^{N} \lambda^i \boldsymbol{\theta}_{k-1}^i \tag{12}$$

$$\mathbf{V}_k^{N+1} = \Omega\mathbf{I} \tag{13}$$

The flowchart in Fig. 1 summarises the classification and the model update phases of DEFC using DEC approach.

IV. Experimental Results

To evaluate the performance of DEFC, experiments were conducted using two evolving real-datasets. The first dataset is the Electricity dataset (Dataset-1) [25] that is a widely used benchmark for testing adaptive classifiers. We have used the normalized version available from [26]. The dataset covers a

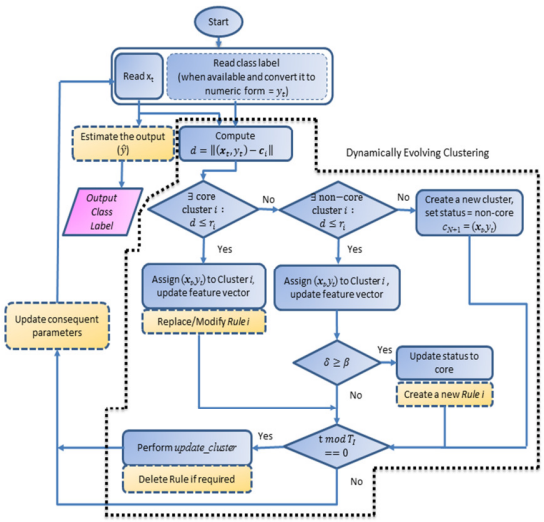

Fig. 3 Classification and Model update in DEFC.

period of two years consisting of 45312 instances and 8 input variables. The task is to predict a rise (UP) or a fall (DOWN) in the electricity price in New South Wales (Australia). The data is subject to concept drift and shift as the electricity prices are affected by changing demands of the consumers depending on seasons, events etc. The second dataset is on heating system choice in California houses (Dataset-2) [27]. The task is to predict the choice of a user among the types of heating systems: gas central, gas room, electric central, electric room, heat pump. The choice depends on installation cost for heating systems (defined for the 5 heating systems), annual operating cost for heating systems (defined for the 5 heating systems), ratio of these two costs, age of the household head, number of rooms in the house, and region. The dataset consists of 900 instances and 14 input variables.

The datasets are considered as pseudo data streams and processing is done on a per-sample basis. The performance was evaluated on a PC with processor speed of 1.20 GHz and 4.0 GB memory. The models were evaluated considering *interleaved test-then-train* method, where first estimation is performed on the current sample $(\boldsymbol{x}_k)$ and then model is updated using the same sample.

DEFC is compared with two state-of-the-art adaptive classifiers, one is a fuzzy rule-based classifier (eClass) [11] and the other is tree-based adaptive classifier (Hoeffding Adaptive trees) [28]. Table I shows a comparison of results obtained from the three classifiers. Both DEFC and eClass1 were developed using MATLAB 7.1 in Windows 7 environment. The input parameters for DEFC, which are actually required for DEC are initialized as, bandwidth $\rho = 0.05$, decay constant $\gamma = 0.99$, and weight threshold $\beta = 1.0001$ [21, 22]. The initial parameters of eClass1 and Hoeffding Adaptive trees are set to default values [10, 11]. The performance of Hoeffding Adaptive trees is evaluated using the MOA framework [29]. It is apparent from the values in the table that for the given evolving datasets, the number of

TABLE I. COMPARISON OF VARIOUS CLASSIFIERS

Classifier	Dataset-1		Dataset-2	
	Accuracy (%)	*#Rule / #Nodes (tree-size)*	*Accuracy (%)*	*#Rules / #Nodes (tree-size)*
DEFC	**82**	**14**	**71.6**	8
eClass1	78.6	29	62.18	10
Hoeffding Adaptive Trees	72.90	146	63.33	**2**

correctly classified instances by DEFC is more compared to other two classifiers. The resulting model is also very compact in case of DEFC which makes it memory efficient. Further, a compact model with few rules can be easily interpreted by the users. With the given initial parameters, the inspection time interval for DEFC is after every 917 samples [21, 22]. In case of Dataset-2, no rules are removed during the processing as the total number of samples is only 900. On the other hand, in case of Dataset-1, after every 917 samples the outdated rules are disregarded and the outliers (non-core clusters) are removed. Therefore, after processing of 45312 instances, the final set of rules consists of only 14 rules.

Fig. 4(a) shows the evolution in number of rules with the progress of the stream considering Dataset-1. The corresponding processing time per sample is shown in Fig. 4(b). It is clear from Fig. 4(a) and 4(b) that the processing time varies from 0.0041 s to 0.0072 s and depends on the number of rules. However, the difference in processing times at various time steps (stream length) is very less and can be neglected. Therefore, we can consider that the processing time per sample in DEFC is constant. The number of rules for Dataset-1 varies from 45 to 14. At the beginning of the stream, the rules were not disregarded, but as the stream progresses some of the rules become outdated and are disregarded. It can be observed from Fig. 4(a) that there is a major change in number of rules from 17 to 36 during the processing of data samples 27000 to 30000. After analyzing the dataset during this period, it is found that the classifier experiences a change in data pattern that led to formation of new rules. For clarity, one of the features of the dataset for the period of 27000 to 30000 is shown in Fig. 5 where the data shift is indicated by a red dotted rectangle. Thus, it can be seen that DEFC adapts with data stream maintaining a compact structure and constant processing time per sample.

V. CONCLUSIONS

The DEFC addresses the key issues of data stream learning in the following way:

The memory requirement is low as DEFC mainly requires the current sample, the covariance matrix (for consequent parameters), and CFVs to be maintained in memory. For each cluster (or rule) a (n+1)×(n+1) covariance matrix, and a CFV are required. Therefore, the memory is dependent on the number of clusters generated by DEC. In

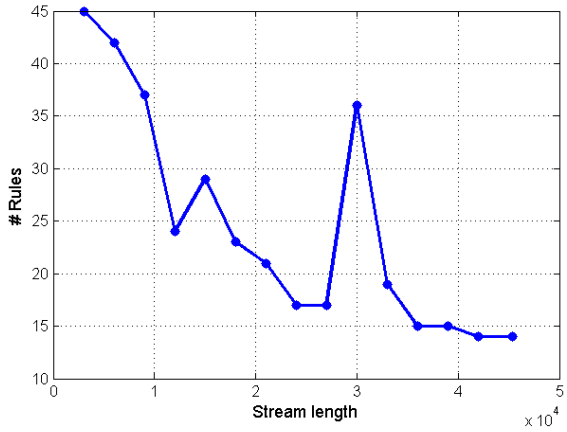

Fig. 4(a) Number of rules vs. stream length

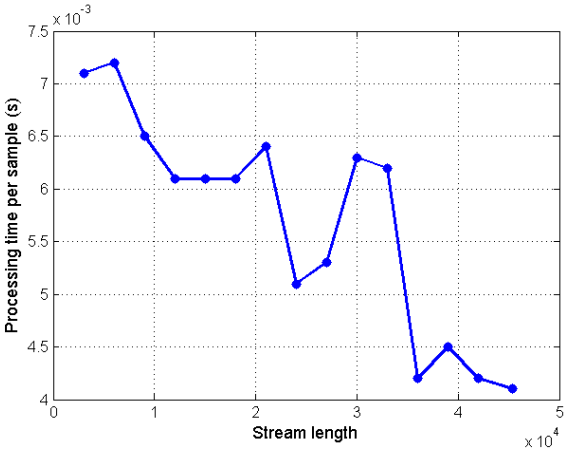

Fig. 4(b) Processing time per sample vs. Stream length

DEC, the creation of too many clusters is avoided by timely removing non-core clusters with very low weight (it can be shown that the number of clusters increase logarithmically with the progress of the stream). The processing time also depends on the number of rules. As DEFC maintains a

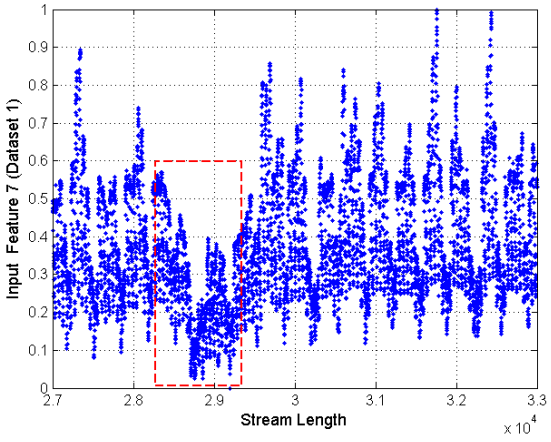

Fig. 5 Input feature 7 of Dataset -1 (stream length 27000 to 300000)

compact model, the processing time required per sample is low and constant. This has been validated by the experimental results.

The incremental nature allows DEFC to learn example by example and update its model whenever required. It creates new rules, updates existing rules, and disregards outdated rules as the data stream progresses. Each of the clusters in DEC is assigned a weight. A rule is formed corresponding to a cluster only when its weight is more than a threshold value. This ascertains that only the clusters that have sufficient data samples associated to it are eligible for forming new rules. Thus, DEFC can effectively distinguish between outliers and a data shift. A data shift is captured only when considerable examples have been seen by the classifier. A rule becomes outdated due to the gradual decrease in the associated weight. This occurs when the core cluster associated with the rule does not receive data sample for a long time. Therefore, the outdated concepts are forgotten over time. These features allow DEFC to dynamically adapt to evolving nature of the data streams and at the same time robust to noise.

The initial experimental results attest that the DEFC performs satisfactorily when data streams are evolving over time. However, in future we intend to perform more experiments to bolster our findings. In DEFC, the user needs to provide three initial parameters, which are actually required by the DEC method to form rules. The most sensitive parameter is the bandwidth parameter that is difficult to set in a streaming environment. At present, we are working on an improved version of DEC that allows for adaptation of the bandwidth parameter.

REFERENCES

[1] G. Widmer and M. Kubat, "Learning in the presence of concept drift and hidden contexts," *Machine Learning,* vol. 23, pp. 69-101, 1996.

[2] J. Gama, *Knowledge Discovery from Data Streams*, First ed. U.S.: Chapman & Hall/CRC., 2010.

[3] C. Fang, W. Yizhou, and C. Zaniolo, "An adaptive learning approach for noisy data streams," in *IEEE International Conference on Data Mining* Brighton, UK, 2004, pp. 351-354.

[4] J. Gama, I. Žliobaité, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A Survey on Concept Drift Adaptation," *ACM Computing Surveys,* vol. 1, p. 35, 2013.

[5] E. Lughofer, *Evolving Fuzzy Systems-Methodologies, Advanced Concepts and Applications* vol. 266. Berlin, Heidelberg: Springer, 2011.

[6] P. Angelov, D. Filev, and N. Kasabov Eds., *Evolving Intelligent Systems: Methodology and Applications*. John Willey and Sons, IEEE Press Series on Computational Intelligence, April 2010.

[7] P. Angelov, *Autonomous Learning Systems from Data Streams to Knowledge inReal Time*. West Sussex, United Kingdom: John Wiley and Sons, Ltd., 2012.

[8] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Transactions on Knowledge and Data Engineering,* vol. 15, pp. 515-528, 2003.

[9] R. Babuška and H. B. Verbruggen, "An overview of fuzzy modeling for control," *Control Engineering Practice,* vol. 4, pp. 1593-1606, 1996.

[10] P. P. Angelov and D. P. Filev, "An approach to online identification of Takagi-Sugeno fuzzy models," *IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics,* vol. 34, pp. 484-498, Feb 2004.

[11] P. Angelov and X. Zhou, "Evolving fuzzy-rule-based classifiers from data streams," *IEEE Transactions on Fuzzy Systems,* vol. 16, pp. 1462-1475, 2008.

[12] R. Dutta Baruah, P. Angelov, and J. Andreu, "Simpl_eClass: Simplified potential-free evolving fuzzy rule-based classifiers," in *IEEE International Conference on Systems, Man, and Cybernetics* Alaska, 2011, pp. 2249-2254.

[13] E. Lughofer, "On-line evolving image classifiers and their appliction to surface inspection," *Image and Vision Computing,* vol. 28, pp. 1065-1079, 2010.

[14] E. Lughofer, P. Angelov, and X. Zhou, "Evolving single-and multi-modal fuzzy classifiers with FLEXFIS-Class," in *IEEE International Fuzzy Systems Conference* London, 2007, pp. 1-6.

[15] E. D. Lughofer, "FLEXFIS: A robust incremental learning approach for evolving Takagi-Sugeno fuzzy models," *IEEE Transactions on Fuzzy Systems,* vol. 16, pp. 1393-1410, 2008.

[16] E. D. Lughofer, "Extensions of vector quantization for incremental clustering," *Pattern Recognition* vol. 41, pp. 995-1011, 2008.

[17] A. Bouchachia, "An evolving classification cascade with self-learning," *Evolving Systems,* vol. 1, pp. 143-160, 2010.

[18] A. Almaksour and E. Anquetìl, "Improving premise structure in evolving Takagi-Sugeno neuro-fuzzy classifiers," *Evolving Systems,* vol. 2, pp. 25–33, 2011.

[19] A. Shaker, R. Senge, and E. HüLlermeier, "Evolving fuzzy pattern trees for binary classification on data streams," *Information Sciences,* vol. 220, pp. 34-45, 2013.

[20] P. Angelov and X. W. Zhou, "Evolving fuzzy systems from data streams in real-time," in *International Symposium on Evolving Fuzzy Systems*, Ambelside, United Kingdom, 2006, pp. 29-35.

[21] R. Dutta Baruah and P. Angelov, "Online learning and prediction of data streams using dynamically evolving fuzzy approach," in *IEEE International Conference on Fuzzy Systems*, Hyderabad, India, 2013, pp. 1-8.

[22] R. Dutta Baruah and P. Angelov, "DEC: Dynamically Evolving Clustering and its application to structure identification of evolving fuzzy models," *IEEE Transactions on Cybernetics,* Issue 9 (IEEE early access article DOI:10.1109/TCYB.2013.2291234), 2013.

[23] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu, "A framework for clustering evolving data streams," in *International conference on Very large data bases - Volume 29*, Berlin, Germany, 2003, pp. 81-92.

[24] J. Ritter, "An efficient bounding sphere," in *Graphics gems*, S. G. Andrew, Ed., ed: Academic Press Professional, Inc., 1990, pp. 301-303.

[25] J. Gama. (2004). *Datasets for concpet drift*. Available: http://www.inescporto.pt/~jgama/ales/ales_5.html

[26] *Stream Datasets*. Available: http://moa.cms.waikato.ac.nz/datasets/

[27] *R Dataset*. Available: http://vincentarelbundock.github.io/Rdatasets/datasets.html

[28] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Advances in Intelligent Data Analysis VIII*. vol. 5772, N. Adams, C. Robardet, A. Siebes, and J.-F. Boulicaut, Eds., ed: Springer Berlin Heidelberg, 2009, pp. 249-260.

[29] *MOA Massive Online Analysis*. Available: http://moa.cms.waikato.ac.nz/