Article info

Article history:
Received 5 October 2020
Received in revised form 17 March 2021
Accepted 4 April 2021
Available online 20 April 2021

Keywords:
Evolving fuzzy systems (EFS)
Data streams
Online bagging of EFS
Autonomous soft pruning and recall of ensemble members
Autonomous evolution of ensemble members
Drift handling

Abstract

Evolving fuzzy systems (EFS) have received increased attention from the community for the purpose of data stream modeling in an incremental, single-pass and transparent manner. To date, a wide variety of EFS approaches have been developed and successfully used in real-world applications which address structural evolution and parameter adaptation in single EFS models. We propose a specific ensemble scheme of EFS to increase their robustness in predictive performance on new stream samples. Our approach relies on an online variant of bagging in which various EFS ensemble members are generated from online bags, that is, the members are updated based on a specific probabilistic online sampling technique, and this with guaranteed convergence to classical sampling in batch bagging. The autonomous pruning of ensemble members is undertaken to omit undesired members with atypically higher errors than other members. We propose two variants, hard pruning where undesired members are deleted forever from the ensemble, and soft pruning where members receive weights to calculate the overall ensemble prediction, according to their single performance; thus, members who are undesired at a certain point of time may be dynamically recalled at a later stage. The autonomous evolution of new ensemble members is carried out whenever a drift in the stream is detected, based on a significantly worsening performance indicator, measured in terms of the Hoeffding inequality. Newer members typically represent the drifted state better and are thus up-weighted compared to older members within an advanced (weighted) calculation of the overall ensemble prediction. The new approach termed online bagged EFS (OB-EFS) was successfully evaluated and compared with single EFS models and related SoA approaches on four data streams from real-world applications (containing various noise levels, drifts and new operating conditions) and showed significantly lower prediction error trend lines.

© 2021 The Author(s). Published by Elsevier Inc. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

1. Introduction

1.1. Motivation and state-of-the-art

Evolving systems and in particular evolving neuro-fuzzy systems (ENFS) have emerged as a prominent methodology over the last decade to properly address the demands of (fast) online modeling and the stream mining processes within a wider range of real-world applications – e.g., predictive maintenance, fault detection and prognosis, time-series forecasting, (bio-)
medical applications, user behavior identification, to name a few (see also [21] for a longer list). This is because they possess the ability i) to adapt quickly to regular system fluctuations and to new operating conditions by expanding their knowledge on the fly, and ii) to properly react to drifts in the system (inducing changing data distributions and/or input–output relations [18] which typically arise in non-stationary environments [33] and/or dynamic and adaptive control operations [26]) by integrating forgetting and out-weighing mechanisms. This is typically achieved in a fully autonomous manner [2,1] by incremental single-pass updates with stream samples, without the necessity of intervention by human operators or experts (as is often needed in the case of classically designed fuzzy systems [37]).

Recent surveys on E(N)FS can be found in [38,21], which compare 50+ of the most widely used approaches and their embedded learning algorithms, together with a list of real-world applications in which they have been successfully implemented and used. All of these approaches (with two exceptions, see below) are operating on a single model basis, that is, one fuzzy system is defined (internally employing a particular structure and rule-based architecture, for which various possibilities have been suggested [38,21]), its parameters recursively updated and its structure evolved over time based on stream samples. A single model may have a certain charm regarding a fast update performance, but it may suffer in predictive performance in the case of higher noise levels in the data. An ensemble of models can typically increase the robustness of performance in such cases, as is known from (classical) batch machine learning scenarios because it can handle the bias-variance tradeoff better than a single model [41.5]. Furthermore, ensembles are potential candidates to deal with drifting data distributions properly in a natural way (especially due to the possibility of the evolution of new ensemble members upon drift alarms [36]) – thus, combining the idea of ensembling with EFS, leading to a kind of online ensembles of EFS, which would be a beneficial methodological direction to increase the robustness and flexibility of current single EFS models at the same time.

**Online ensembles of EFS** have only been loosely addressed so far in the literature: i) in [32], which is a boosting approach along subspace selection (ensemble members operate in different sub-spaces) and which is designed for classification problems (using fuzzy classifiers with particular properties as ensemble members); ii) in [20] the ensemble members are essentially different from each other due to a different set of meta–parameters (constraints on the EOGS objectives); the meta–parameters steer the complexity of the members, in this sense, this approach can be seen as a kind of extended boosting approach with the possibility of including less and more complex ensemble members (whereas in conventional boosting, only weak learners are considered [7,29]). In both approaches, there is no probability-based drawing on streaming samples to increase the significance of ensemble members (and to reduce the effect of noise), and in [20] the number of ensemble members does not change over time.

1.2. Content of this paper – Our approach

Our approach suggests a new adaptive, online ensembling variant by performing an **online bagging of EFS**, where each base member acts as a conventional EFS predictor (with all functionalities regarding evolving, pruning, merging, and splitting of rules according to today’s standards of evolving neuro-fuzzy approaches [38]) and where the members are established based on a sampling technique, which extends the conventional batch bagging procedure to the online, single-pass case. It should be stressed that (conventional batch) bagging is a widely used technique for improving the robustness of the (training of the) models in the case of the availability of a low number of training samples and/or in the presence of significant noise levels. Thus, by adapting the concept of classical batch bagging to the online case for performing incremental, single-pass training of EFS models, we expect a similar robustness improvement of the models as for the batch case. Following the idea of sample selection in classical bagging, we propose a probabilistic online sampling strategy based on a random drawing from the Poisson distribution (Section 3.1), which affects the likelihood that an ensemble member is updated with a single sample one time (classical stream mining/modeling case) or multiple times (up to X repetitions) or not at all. This is different to the approaches in [32,20]: in the former, only the winning model with the highest score in the current sample is updated; in the latter, each new incoming stream sample is used to update all ensemble members likewise. We demonstrate that, with increasing sample size, the online bagged sampling converges to the classical offline bagged sampling in sample distribution and thus inherits its robustness characteristics in the case of noisy data.

Furthermore, our approach integrates two other key issues, namely **autonomous pruning** (Section 3.2) and **autonomous evolving** (Section 3.3) of ensemble members on the fly and on demand. Autonomous pruning is important whenever members fall out of the general error trends of the other members. Such members were either i) unlucky affected by more drawn samples containing higher noise levels or ii) more affected by (local, gradual) drifts than other members. We suggest two pruning options, hard pruning where members are deleted forever from the ensemble (as is also used in previous approaches) and a new form of pruning termed *soft pruning* with integrated recall functionality, that is, a member can be recalled later, which may be important in the case of cyclic drifts (as we show in the results section). Both are able to operate in sample-wise single-pass mode. Soft pruning embeds an assignment of weights to the ensemble members which are then respected in the overall weighted prediction output (Section 3.4). The weights are composed of recent error trends and sample coverage (the latter indicating a kind of extrapolation degree). Thus, members with higher errors and lower coverage (e.g., due to being affected by drifts) are down-weighed in the overall prediction. The autonomous evolution of new ensemble members becomes important whenever drifts are explicitly recognized based on a decreasing performance of the whole ensemble (Section 3.3). Then, new members are evolved solely based on the drifted state, and thus do not contain any older states. Hence, they automatically receive more impact in the advanced (weighted) overall prediction, as members represent-
ing older states are usually automatically down-weighted as they show a lower sample coverage and higher errors in drifted states — autonomous, flexible drift compensation during the prediction stage and not during the learning stage (as basically achieved by related SoA works). Drift detection is performed through an interpretation of the Hoeffding inequality [14] coupled with a hypothesis test to elicit a statistical significance threshold for drift alarms.

The evaluation of our online bagged EFS approach (termed as OB-EFS) was carried out in comparison with (classical) single EFS models and related SoA approaches, and this by applying four variants (to see the effects of the single components in OB-EFS, see Section 4.2): i) native online bagging without pruning and evolution, ii) online bagging with soft pruning, iii) online bagging with soft pruning and evolution of ensemble members (the full OB-EFS algorithm as shown in Algorithm 1). This comparison was achieved based on four noisy data sets from real-world applications (described in Section 4.1), with noise intensity levels ranging from low to very high. One includes a gradual regular drift and one a cyclic drift (with a back-change to the old state after a few hundred samples). The accumulated prediction error trend lines show significantly improved performance of all bagging variants compared to the single EFS model, especially during drift phases and higher noise levels in the data streams (Section 5). Furthermore, the soft pruning option could outperform native bagging (w/o any pruning) and hard pruning on all data sets and especially in the two drift cases. In comparison with related and well-known SoA approaches, OB-EFS could achieve lower error trend lines in large parts of all the streams from the four application scenarios, mainly showed less sensitivity to drifts, and especially resulted in 20%–70% lower final errors at the end of the streams (see Section 5.3).

2. Problem statement

In the case of noisy data samples, regression fits, especially when they are of a nonlinear type, often may provide instable solutions. That is, the regression fits tend to over-fit due to fitting the noise more than the real underlying functional approximation trends [34], increasing the variance error much more than reducing the bias error (please refer to a detailed discussion of this problematic issue in [12]). Bagging, invented by Leo Breiman in the 1990s [6], is a well-known and widely used method for reducing over-fitting effects in the case of noisy data samples [6,16]. It achieves this by bootstrap sampling, drawing \( N \) samples with replacements from an original pool of training samples (typically of size \( N \)). This is repeated \( B \) times to generate \( B \) bags of samples, each of which contains different samples a different number of times due to the probabilistic replacement sampling (see also the subsequent section). For each bag, an own regression fit is produced to generate so-called base members. For new online queries, the average over all predictions (from all members) is used as final predictions. This can be robustly achieved even in the case of a low number of training samples due to the specific sampling (with replacement) strategy and multiple model training [6].

A two-dimensional example of a function approximation problem in form of a parabola is shown in Fig. 1 (dotted dark line). Noisy samples have been drawn from the process describing the parabola function, shown as dark dots. Obviously, the single model (indicated by a dotted grey line) has a much worse regression fit (doubling the error) than the models obtained from the bagged ensemble, no matter whether 10 or 20 members are fitted.

The number of optimal bags plays an important role and can be estimated through the so-called out-of-bag (OOB) error on the whole training set; therefore, the number of bags \( B \) varies from 2 to \( X \) (with \( X \) a significant high number, typically 50 or 100) and for each number, those samples that are not selected form the OOB set, which can be used as a validation set for calculating the OOB error. In particular, the OOB error \( \text{OOBerr}_B \) for a particular number of bags \( B \) is the mean prediction error over all training samples, where for the prediction of one sample \( x \), only those base members are used that did not have \( x \) in their respective bootstrap bags (which they were trained from):

\[
\text{OOBerr}_B = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{|\{B_k, k = 1, \ldots, B | x_i \notin B_k\}|} \sum_{|\{B_k, k = 1, \ldots, B | x_i \notin B_k\}|} f_k(x_i)
\]

(1)

\( N \) denotes the number of samples and \( f_k(x_i) \) denotes the prediction of the \( k \)-th model (trained on bag \( B_k \)) for sample \( x_i \) (which is not included in \( B_k \)). (1) is calculated for various numbers of bags \( B \) and that \( B \) which leads to the lowest OOB error is used as the ideal number of bags.

Now, in the context of data stream modeling, the number of optimal bags may change over time. As the OOB error requires the availability of whole batch sets, it is usually not feasibly applicable in (faster) online learning settings. Therefore, the challenges of establishing online bagged EFS are as follows:

- Probabilistic sampling in a single-pass manner, which converges to the batch sampling in classical bagging.
- Pruning of established ensemble members on the fly and on demand in a single-pass manner.
- Evolution of new ensemble members on the fly and on demand in a single-pass manner.

The first issue is important to ensure the inheritance of the robustness of classical bagging with respect to noise in the data. The last two issues guarantee a dynamically changing ensemble size according to the nature of the current data stream. In our approach, this is achieved by adjusting the size based on the latest performance (e.g., the prediction error trend) characteristics on new streaming samples (preventing time-intensive recalculations of the OOB error on batch sets). Pruning also improves compactness (and thus computational demands) of the ensemble and we propose two variants: hard pruning,
where undesired base members are pruned forever, and a soft pruning option where base members can be smoothly out-
weighed and recalled in the predictions later (accounting for possible cyclic drifts). The evolution of ensemble members
is able to handle drifts in data streams, as new members can more flexibly represent drifted situations than older members
being updated before on older states.

Furthermore, we suggest a new ensembling scheme for predictions, rather than using pure averaging of the predictions of
the single members. Therefore, we exploit the specific structure of fuzzy systems (the base members), and in particular, we
use certainty in their predictions, which can be expressed in a natural way [15] through coverage concepts. This leads to a
compensation of drifts during the prediction stage, as newly evolved members will typically have a higher prediction cer-
tainty in (new) drifted states than older members. In the following subsections, we explain each functionality of our
approach in detail.

3. Online bagging of EFS

3.1. Online sampling according to Poisson distribution

The sampling in a conventional batch bagging approach is done in the following way: $N$ samples are randomly drawn
with replacement from a training set containing $N$ samples. This is repeated $B$ times to create $B$ bags, based on which $B$ base
models (members) are trained. From statistics, it is well-known [13] that each bag then contains each of the training samples
$k$ times with probability:

$$\text{Binomial}(k) = \binom{N}{k} \left( \frac{1}{N} \right)^k \left( 1 - \frac{1}{N} \right)^{N-k}$$ (2)

This binomial distribution converges to the Poisson(1) distribution whenever $N \to \infty$ [13], with

$$\text{Poisson}(1)(k) = \frac{e^{-1}}{k!}$$ (3)

Therefore, it is quite obvious to choose each new incoming stream sample $x$ Poisson(1) times for each bag separately and
thus to update the base member Poisson(1) times with each new sample. We can simulate ‘Poisson(1) times’ by drawing a
random number $\text{rand}_{[0,1]}$ in $[0,1]$ and by checking in which bin (interval) of the Poisson histogram
$$\left( \sum_{i=0}^{\infty} e^{-1} \frac{1}{i!} \right), \left( \sum_{i=0}^{1} e^{-1} \frac{1}{i!} \right), \left( \sum_{i=0}^{2} e^{-1} \frac{1}{i!} \right), \ldots$$

it falls; the bin member $k$ in which the random number falls is used as ‘Poisson(1)-1 times’, i.e. when $k = 1$, no update takes place, when $k = 2$, the member is updated one time with the current
sample, and so on. In formula notation, this means that the number of updates $k_b$ for the $b$-th model (on the current stream
sample $x$) is calculated by:

$$k_b(x) = \min_{l=0,1,\ldots,m} \left( \text{rand}_{[0,1]} \leq \sum_{i=0}^{l} e^{-1} \frac{1}{i!} \right)$$ (4)
Of course, in a practical implementation, we cannot use \( m = \infty \), therefore we restrict it to a maximal value of \( m = M \), i.e. a maximum of \( M \) allows updates for one base member with the same single sample (we set \( M = 5 \) in all our experiments, which already leads to a maximal sum \( \sum_{j=0}^{M} r_j \) of 0.9994, covering nearly the whole area under the probability function).

In order to achieve consistency with the random number drawn from \([0, 1]\) (i.e. it should not exceed \( \sum_{j=0}^{M} r_j \)), \( r_{j, 0} \) is multiplied by \( M \)

An interesting question now is whether the distribution supplied to the base members in the case of (incremental) updates when \( M \) becomes larger, because then it is guaranteed that the online (sample-wise) sampling based on Poisson(1)-draws delivers similar bags and thus base members as in the case of batch sampling, i.e. incremental online bagging converges to (hypothetical) batch bagging over time. This induces a direct inheritance of the robustness of classical bagging with respect to noise in the data to the online case.

Following the considerations in [25], without loss of generality, fixing a specific bag, the distribution of the vector \( k = [k_1, \ldots, k_b] \), where \( k_j \) denotes the number of times the \( j \)-th sample is included in the bag, follows the multinomial distribution \( M(N, (1/N, \ldots, 1/N)) \). This is because each sample is drawn with replacement, thus the probability that it is drawn for each repetition to generate the bag is \( 1/N \). So, according to the statistical model that under \( N \) repetitions with \( N \) different sorts of items (samples in our case), the probability that each sort \( j \) appears \( k_j \) times follows a multinomial distribution (see [28]). On the other hand, in the case of online bagging, each sample is chosen a number of times according to the Poisson(1) distribution, thus the total number of examples drawn follows a Poisson(\( N \)) distribution, because the average expected event rate is \( N \), which is given by:

\[
Poison(N)(t) = \frac{N^t}{t!} e^{-N} \tag{5}
\]

The probability generating function of the Poisson(\( N \)) distribution when \( N \) goes to \( \infty \) is given by [10]:

\[
G(Poisson(N))(z) = e^{N(z-1)} \tag{6}
\]

and as Poisson(\( N \)) multinomial trials \( M(1, 1/N) \) (one trial per single sample) are performed, the generating function for online bagged sampling results in:

\[
G(Poisson(N))(G(M(1, 1/N))) = e^{N(k_1 + \ldots + k_N - 1)} \tag{7}
\]

Furthermore, it is known from [10] that the generating function from the offline (classical) bagged sampling (following the \( M(N, (1/N, \ldots, 1/N)) \) distribution) is:

\[
G(M(N, 1/N, \ldots, 1/N)) = \left( \frac{k_1 + \ldots + k_N}{N} \right)^N \tag{8}
\]

Therefore, when \( N \) goes to \( \infty \), it follows that:

\[
\lim_{N \to \infty} G(M(N, 1/N, \ldots, 1/N)) = \lim_{N \to \infty} \left( 1 + \frac{k_1 + \ldots + k_N - N}{N} \right)^N = e^{N(k_1 + \ldots + k_N - 1)} = e^{N(k_1 + \ldots + k_N - 1)} \tag{9}
\]

The last term in (9) is equal to the term on the right-hand side of (7), which means that the generating function of offline bagged sampling converges to the generating function of online bagged sampling. According to [8], this also implies the convergence of both sampling variants in the same distribution.

In order to improve sample significance and to avoid the issue where online stream samples are completely ignored in all bags and thus are not taken for an update of any base member, we increase \( k_b \) as elicited through the Poisson bins in (4) by 1 for each third bag in cases where it was not drawn for the two base members before:

\[
k_b(x) = \begin{cases} 
(4) + 1 & \text{if } (b \mod 3) = 0 \text{ and } k_{b-1} = k_{b-2} = 0 \\
(4) & \text{otherwise}
\end{cases} \tag{10}
\]

This means that each new sample is used at least one time for updating at least one of three consecutive members. If the condition in the first case in (10) holds for a particular bag in the worst case in all \( N \) samples, this leads to a drawing vector of \( k = [k_1, \ldots, k_N + 1] \) for this bag over all samples, with it \( k \) as in (4), which does not harm the convergence properties between offline and online bagged sampling shown above. This can be easily seen when substituting \( k_1 + \ldots + k_b \) with \( k_1 + \ldots + k_N + N \) in the generating functions (7) and (8).

3.2. Autonomous pruning of ensemble members

The pruning of ensemble members can become important whenever some ensemble members are unluckily trained from more (high) noisy samples than others (and thus their performance becomes atypically lower compared to the performance
of the others) or whenever (local) drifts occur which induces that some ensemble members learned on older samples become outdated (and thus their performance becomes worse).

The performance of a single ensemble member can be measured in terms of its accumulated mean absolute error (MAE) over time. This measure is obtained by predicting new online samples and calculating the errors between predicted and real observed values (once they are available from the process or as being present in streams used for evaluation purposes, see Section 4.2). In a case when whole data chunks are loaded and processed, one can calculate the error on the latest chunk or the average error over the latest chunks.

In the case of single sample-wise update (as we aim in our approach), the MAEs of the single ensemble members are updated incrementally by (here for the $b$-th member):

$$\text{MAE}_b(N) = \frac{(N-1)\text{MAE}_b(N-1) + |y(N) - \hat{y}_b(N)|}{N}$$

with $N$ denoting the $N$-th sample loaded from the stream, $\hat{y}_b(N)$ is the predicted target value of the $b$-th member, $y$ is the real observed value, and $\text{MAE}_b(0) = 0$.

The idea is to check whether some ensemble members have significantly higher errors than others, thus whether they appear atypical compared to the others. Assuming that the accumulated MAEs over the ensemble members are (approximately) normally distributed, we employ the concept of statistical process control [9] to check for significant atypicality. Therefore, the following condition is applied:

$$\exists_{b=1...B} \text{MAE}_b(N) > \mu(\text{MAE}(N)) + n\sigma(\text{MAE}(N))$$

(12)

with

$$\mu(\text{MAE}(N)) = \frac{1}{B} \sum_{b=1}^{B} \text{MAE}_b(N)$$

$$\sigma(\text{MAE}(N)) = \sqrt{\frac{1}{B} \sum_{b=1}^{B} (\text{MAE}_b(N) - \mu(\text{MAE}(N)))^2}$$

and $B$ is the current number of ensemble members and $n$ a factor. Per default $n$ can be set to 2 in order to emphasize a $2\sigma$ confidence interval (covering 96% of the samples). The condition in (12) checks whether any ensemble member has a higher error than the mean error plus $n$ standard deviations over the other ensemble members. This can be done for each new incoming data sample without running into danger that a too fine ‘checking resolution’ may lead to too optimistic prunings, because each MAE$_b$ already contains the whole error over the complete stream (and not only just a small snapshot of it) and is updated smoothly with $N$ becoming larger. Thus, we also suggest pruning starts after a few dozens online stream samples once the MAE-curves have stabilized (see also the results section for examples). Furthermore, a variable factor of $n$ may help to omit too early prunings of members, especially in the case when the variance (and standard deviation) is close to 0. Thus, according to the considerations in [32], $n$ could be adapted by $n = 1.3e^{-\sigma(\text{MAE}(N))} + 0.7$. This means that high $n$’s close to 2 are achieved for low $\sigma$’s, which makes the pruning less active if the variance is low.

In the case when condition (12) is fulfilled for one or more ensemble members, these can be hard-pruned by erasing them from the ensemble and setting $B = B - b_{pruned}$, with $b_{pruned}$ the number of pruned ensemble members (as, e.g., conducted in the related boosting-type approach [32]). In this case, they will not contribute to ensemble predictions on new samples any longer (see Section 3.4), but are gone forever. However, in the case of cyclic drifts, hard pruning could be counter-productive at the point of time when the old state is met again. Previous members which were pruned before could thus become significantly valuable predictors again when the process changes back to an older state, which has already been modeled and thus is reflected in the previous members.

Therefore, we aimed for a soft pruning option with automatic and smooth member recall functionality (rather than a crisp, discontinuous one as, e.g., in [17,32]). To do so, we introduce weights for the ensemble members $w_{b} \in [0, 1]$, which are correlated with their predictive performance in terms of the MAE calculated through (11) and which denote the importance of the members in the prediction scheme (the higher the weight $w_b$, the more important the member, see Section 3.4). Depending on the level of the MAE, the weights are assigned lower or higher values, with smooth transitions inbetween (→ fuzzy weights). In the case when the MAE of an ensemble member $b$ meets condition (11), its weight will receive a value of 0, thus the member becomes completely out-masked, This then mimics the hard pruning case, but with a possible reactivation option, because the member and its MAE stay in the ensemble and will be further updated with new samples. Intuitively, the ensemble member with minimal MAE over all members should receive the highest possible weight of 1, and all other members should receive weights in between 1 and 0. Aiming for a (moderate) linear decreasing trend, this leads to the fuzzy transformation function (from MAEs to in $[0,1]$-weights) as shown in Fig. 2 (for $n = 2$), which is defined as:

$$w_b = \max\left(0, \frac{1}{\min - \mu - n\sigma} \text{MAE}_b + 1 - \frac{\min}{\min - \mu - n\sigma} \right)$$

(14)
where \( \text{min} \), \( \mu \) and \( \sigma \) denote the minimum, mean and standard deviation over the MAEs of all ensemble members (as defined in (13)), respectively. Due to the maximum operator, it is guaranteed that no negative weights are assigned for members with MAEs higher than \( \mu(\text{MAE}(N)) + n\sigma(\text{MAE}(N)) \): all of them receive a weight of zero. These weights will be also respected properly when performing an enhanced prediction scheme over the ensemble members (i.e. members with weights equal to zero are not respected, members with low weights are respected a little etc., see Section 3.4).

### 3.3. Autonomous evolution of ensemble members

During the modeling stage, we aim to evolve new ensemble members whenever a drift seems to arise. A drift represents a new arising operation mode or system states which typically induces changes in data distributions and/or concepts between inputs and targets to be learned and thus makes learned distributions/relations in models outdated [18]. The evolution should ideally compensate drifts, as new members are solely learnt on the drifted state and thus tend to produce lower errors during the new state than older ensemble members already previously learnt on some older states. These lower errors will then result in higher weights \( w_b \), thus putting more emphasis on new members during prediction (see Section 3.4).

The basic idea is to detect a drift based on a particular performance indicator, which is accumulated over time, to reflect the predictive performance of the ensemble on past samples to some extent. Such an accumulation should lead to robustness with respect to possible local (tiny) fluctuations of the ensemble performance which in turn reduces the likelihood of false drift alarms. To detect statistical significance that a performance indicator actually worsens (e.g., the accumulated error trend starts to rise after having been converged), we apply the Hoeffding inequality on recent data chunks. The Hoeffding inequality [14] is defined in the following way:

\[
P(\mu(X) - E[\mu(X)] \geq t) \leq e^{-2w^2t^2} \tag{15}
\]

where \( \mu(X) \) is the empirical mean of a certain random variable \( X \) bounded in \([0, 1]\), \( E[\mu(X)] \) is the expected mean and \( w \) is the number of samples (realizations) of \( X \). Thus, the Hoeffding inequality states that the probability that the difference between empirical (calculated) and expected mean is higher than a threshold \( t \) is lower than an exponential decreasing function in dependency of \( w \): the higher the number of samples, the lower the term on the right hand side. This is intuitive because with a higher number of \( w \) the ‘sample significance’ of the mean rises and thus a larger difference between the empirical mean and the expected mean becomes less probable.

Now, the Hoeffding inequality (bound) can be taken as the backbone for eliciting a threshold for drift alarms. Therefore, the idea is to use a sliding window \( W \) of recent past accumulated performance indicator values \( X_{N-w+1}, X_{N-w+2}, \ldots, X_N \) (with \( N \) the current sample instance) whose rising trend may indicate a drift, and to check whether the mean of the indicator over the first half of the window is statistically equal to the mean over the second half or not. As a performance indicator, the mean absolute error of the whole ensemble can be used which can be incrementally updated with each single sample by (11), applying \( \text{MAE}_{\text{overall}} \) instead of the \( \text{MAE}_b \) on single bags. The mean over the first half of the window can be taken as \( E[\mu(X_{i-N-w+1:N-w/2})] \), because this represents the old, expected (known) state, whereas the mean over the second half is taken as \( \mu(i = X_{N-w/2+1:N}) \). In terms of hypothesis testing, this means that it is checked whether \( H_0 \) defined as

\[
H_0 : \mu(X_{i=N-w/2+1:N}) - E[\mu(X_{i-N-w/2+1:N-w/2})] \leq 0 \tag{16}
\]
can be safely rejected with a significance level of \( \alpha \) (\( \alpha \) typically in \( [0,0.05] \)), and thus a drift should be alarmed, as 
\[ \mu(X_{i-N-w/2+1...N}) \] is significantly higher than 
\[ E[\mu(X_{i-N-w+1...N-w/2})] \] (i.e. a rising error in the second half of the window took place). According to the Hoeffding inequality, this results in checking whether

\[
P(\mu(X_{i-N-w/2+1...N}) - E[\mu(X_{i-N-w+1...N-w/2})] \geq t) \leq \alpha.
\]

(17)

Therefore, the significance threshold \( t \), above which a drift is alarmed, can be elicited as

\[
\alpha = e^{-2w}\ln(2) \rightarrow t = \sqrt{\frac{\ln(\alpha)}{-2w}}
\]

(18)

This is because then 
\[ \mu(X_{i-N-w/2+1...N}) - E[\mu(X_{i-N-w+1...N-w/2})] \geq t \] can theoretically occur (during regular, non-changing phases) with at most a \( (\alpha) \) probability of \( \alpha \); so, when it actually happens, a significant rise in the mean (over the second half of the window) and thus a drift is probable with probability \( 1 - \alpha \). Therefore, \( H_0 \) can be safely rejected. Typical values for \( \alpha \) are 0.05 or 0.01. We used the latter for a pessimistic setting (decreasing the likelihood of false alarms) in all our experiments.

As we perform a sample-wise update of the ensemble and its performance indicators (e.g., MAEs), the window can be sample-wise slid through in a single-pass manner (i.e., the oldest sample leaves the window and the newest sample is added to the window), requiring a maximal memory of \( w \) one-dimensional samples. Then, when the hypothesis test described above is carried out after each window update, this automatically leads to all possible cutting points over the stream (each stream sample is used as cutting point). Therefore, we can always apply the half-half split and do not need to search explicitly for the ideal cutting points (as done in [32]). Another consideration goes to the fact that the application of the Hoeffding inequality requires \([0,1]\)-bounded variables, whereas in our case the performance indicators generally (and also the MAE in particular) can have any range. Therefore, we have to multiply the threshold \( t \) with the actual range of the performance indicator \( \text{range}(\text{perf}) = \max(\text{perf}) - \min(\text{perf}) \) in order to ensure range-independency:

\[
t = \text{range}(\text{perf}) \times \left( \sqrt{\frac{\ln(\alpha)}{-2w}} \right)
\]

(19)

It is important not to use the range over the current window, because small fluctuations in the case of nearly constant trends (e.g., in the case of converged model errors during steady phases etc.) lead to too many drift alarms because \( \text{range}(\text{perf}) \) then becomes very close to 0 (we recognized this during our experiments). Thus, we used the range of the performance indicator \( X \) over a larger past time frame. To omit outliers or to perform a too pessimistic range calculation, e.g., due to non-convergent, fluctuating error trends at the beginning of stream learning, we substitute \( \text{range}(\text{perf}) \) by the interquartile range.

Once a drift has been detected, a new ensemble member is trained based on the samples in the second half of the window, because these indicate and represent the detected drift. Therefore, again bagging is applied in batch form to select \( w/2 \) samples with a replacement for training. Usually \( w/2 \ll N \) (we used \( w = 100 \) in all our experiments), thus the training of the new fuzzy system member on this small windowed set can be expected to be very fast. The newly trained member is included in the ensemble and further updated based on the sampling procedure explained in Section 3.1. Its MAE, \( MAE_{b+1} \), is initialized to the error on the samples not included in the bag from which it is trained. The prediction weight \( w_{b+1} \) of the new member is initialized according to (14).

Finally, we want to emphasize that the autonomous evolution of members addresses the handling/compensation of drifts on a global error level (i.e. the overall ensemble (error) is affected by the drift, thus new members need to be evolved), while in the autonomous pruning technique drifts are handled on a local error level (i.e. only a few ensemble members are affected by the drift and thus produce atypical MAEs and hence are down-weighed and/or pruned). By combining both techniques into one algorithm (see Algorithm 1), both, local drifts and global drifts can be adequately handled.

3.4. Enhanced prediction based on ensemble members

In the conventional batch bagging approach, the predictions of the single ensemble members \( \hat{y}_b \) are averaged to an overall prediction [6], i.e.:

\[
\hat{y} = \frac{1}{P} \sum_{b=1}^{P} \hat{y}_b
\]

(20)

Here, we foresee an alternative scheme in order to account for drifting situations by aiming for a weighted averaging approach, where the weights denote a kind of importance level of the ensemble members for predicting the target values of new samples. Different to the approach in [20], where the weights are learnt through the least squares functional (hence yielding a sort of trained combiners), we propose more flexibility in assigning weights, based on the characteristics of the current data sample (for which a prediction is made) with respect to the ensemble members, which is combined with the performance of the ensemble members. This should ensure a flexible reaction to arising drifts, much more than when upd-
ing learnt weights by recursive mechanisms, which in turn requires forgetting mechanisms whenever drifts are detected to outweigh older learned weights and a forgetting factor steering the degree of forgetting would then induce an additional parameter, whose ideal choice or even adaptation during stream modeling is by far not an easy task.

The idea of our approach stems from the fact that a newly evolved member covers the new drifted data distribution better than older members. For a two-dimensional example in the case of a drifting parabolic function, see Fig. 3. New data samples are shown with bigger dark dots than older ones. Obviously, these new samples lie far away from the old ensemble member, whose rule contours are shown as (three) solid ellipsoids, thus they are not ‘covered’ by the older member. This coverage is better for the new member, as the samples appear close or within the ellipsoidal regions. This also means that extrapolation is much less severe in the case of the new member than in the case of the old member, which typically leads to more safer (more robust) predictions. Hence, we propose the maximal rule membership degree (coverage) of a new sample \( x(N) \) to an ensemble member \( b \) as prediction weight:

\[
\text{cover}_b = \max_{b=1}^{C_b} \phi_{ib}(x(N))
\]  \hspace{1cm} (21)

with \( C_b \) the number of rules in the \( b \)-th ensemble member and \( \phi_{ib} \) the membership degree of the current sample \( x(N) \) to the \( ib \)-th rule in the \( b \)-th ensemble member. This is typically calculated by aggregating membership values to single fuzzy sets along the \( p \) input variables \( x_1(N), \ldots, x_p(N) \) through t-norms [19], i.e. by:

\[
\phi_{ib} = \prod_{j=1}^{p} \phi_{ijb}(x_j(N))
\]

with \( \phi_{ijb} \) the fuzzy set membership degree of input variable \( j \) in the \( ib \)-th rule of the \( b \)-th ensemble member (see [27] for different fuzzy set shapes and concrete t-norm based aggregations often used in fuzzy systems). Please note that it is due to the beneficial model architecture of fuzzy systems that we can easily calculate such coverage degrees, which is not possible with many other machine learning models, neural networks or deep learning architectures. The coverage degree in (21) represents the ‘degree of extrapolation’ (as a prediction certainty measure) of a new sample to member \( b \), and this already in normalized \([0,1]\)-form, with \( 0 = \) maximal extrapolation (no rule fires) and \( 1 = \) perfect coverage (the sample is lying in the rule center).

Additionally, we also apply the accumulated error trends \( MAE_b \) as weights according to the transformation (14), to partially handle drifts properly as well (as older members usually produce higher errors on new drifted states), but additionally to down-weight members with worse performance as evolved from unlucky bags containing more high noisy samples than others. Therefore, the overall prediction for the \( N \)-th stream sample is produced by the following weighted average:

\[
\hat{y}(N) = \frac{\sum_{b=1}^{B} w_b \cdot \text{cover}_b \cdot \hat{y}_b(N)}{\sum_{b=1}^{B} w_b \cdot \text{cover}_b}
\]  \hspace{1cm} (22)

where \( \hat{y}_b(N) \) denotes the prediction produced by the \( b \)-th ensemble member through the fuzzy inference [27] (most often Takagi–Sugeno type inference [39] is employed in the case of EFS [38]).

### 3.5. Overall procedure – Pseudo-code of the algorithm

Combining all aspects as described in the aforementioned subsections, leads to the following sample-wise update algorithm (with single stream sample processing functionality) as shown in the pseudo-code of Algorithm 1, termed as online bagged EFS (OB-EFS).

**Algorithm 1.** Online Bagged Evolving Fuzzy Systems (OB-EFS)

**Input:** new stream sample \( x(N) \), current # of bags \( B \), current ensemble members \( EFS_1, \ldots, EFS_B \),
\( \alpha = 0.01 \), window size \( w \), member error indicators \( MAE_1, \ldots, MAE_B \), ensemble error indicator \( MAE \) over past \( N - w \) samples: \( MAE_{N-w+1:N} \).

**Output:** updated \( EFS_1, \ldots, EFS_B \), updated \( MAE_1, \ldots, MAE_B, MAE \), updated # of bags \( B \).

**Steps 1 (Predictions and Error Updates):**

Obtain predictions \( \hat{y}_1(N), \ldots, \hat{y}_B(N) \) from the \( B \) ensemble members \( EFS_1, \ldots, EFS_B \) on \( x(N) \) using fuzzy inference.

Obtain coverage \( \text{cover}_1, \ldots, \text{cover}_B \) on \( x(N) \) using (21).

Calculate (14) to obtain \( w_1, \ldots, w_B \).

Produce overall prediction \( \hat{y}(N) \) using (22).

Update ensemble member errors \( MAE_1, \ldots, MAE_B \) using (11) with \( \hat{y}_1(N), \ldots, \hat{y}_B(N) \).

Update overall error \( MAE \) using (11) with \( \hat{y}(N) \).

**Step 2 (Sampling and Update):**

**for** \( b = 1 \) to \( B \) **do**

Elicit number of updates \( k_b \) using (4).

if \( (imod3 == 0) \&\&(k_{b-1} == k_{b-2} == 0) \) then

\( k_b = k_b + 1 \).

**end if**

Update \( EFS_b \) with \( x(N) \) \( k_b \) times using \( EFS \) engine.

24
end for
Step 3 (Member Pruning):
for $b = 1$ to $B$
do
   Hard Pruning Case:
   if (12) holds for member $\text{EFS}_b$ with $n = 1.3e^{-0.5(\text{MAE}(N))} + 0.7$ using (13) then
      Prune $\text{EFS}_b$ and its performance indicators.
      $B = B - 1$.
   end if
   Soft Pruning Case:
   Calculate (14) to obtain $w_b$ (if it receives 0, the member will be automatically ignored in the next prediction)
end for
Step 4 (Drift Detection and Member Evolution):
Calculate threshold $t$ using (19).
Calculate mean over MAEs for 1st window half, $\mu_{\text{first}} = \mu(\text{MAE}_{i=N-w+1,\ldots,N-w/2})$.
Calculate mean over MAEs for 2nd window half, $\mu_{\text{second}} = \mu(\text{MAE}_{i=N-w/2+1,\ldots,N})$.
if ($\mu_{\text{first}} - \mu_{\text{second}} \geq t$ calculated by (19)) & (lastdet $< N - w$) (the second condition assures that no new member has been evolved before in the current window) then
   Generate a bag $B + 1$ from $x(N - w/2 + 1), \ldots, x(N)$ past samples (standard batch bagged sampling [6]).
   Create new member $\text{EFS}_{B+1}$ by applying EFS training procedure on the $B + 1$th bag.
   Initialize its weight $w_{B+1}$ to 1.
   Calculate its initial error $\text{MAE}_{B+1}$ on the left out samples from $B + 1$.
   $B = B + 1$.
end if

We want to emphasize that our approach is general in the sense that any EFS learning engine with incremental, single-pass update capabilities can be plugged into the online bagged EFS algorithm for updating the EFS base members $\text{EFS}_1, \ldots, \text{EFS}_B$ – in particular into the last step within Step 2: ‘Update $\text{EFS}_b$ with $x(N)$ $k_b$ times using EFS engine’. This should induce a wider interest and applicability for readers and researchers in the community. We performed a concrete realization to be able to evaluate and test the OB-EFS algorithm (see subsequent section) by employing the generalized smart evolving fuzzy systems (Gen-Smart-EFS) approach [22]. It embeds incremental merging and splitting operations in a sample-wise, single-pass manner (thus, it can handle the single sample processing functionality of Algorithm 1), to provide maximal flexibility of the evolved models according to the characteristics of the stream (e.g., addressing cluster fusion and delamination aspects etc.). It also contains an online, incremental soft dimensionality reduction technique (with more important features receiving higher weights than less important ones), which makes it also applicable to high-dimensional data streams.

![Fig. 3. Two-dimensional example for a drifting functional relation (parabolas in dashed line: in the lower part the original one, in the upper part the drifted one); an old member not covering the new samples well would produce high prediction errors, thus its prediction should be down-weighed compared to a newly evolved member which covers the new samples better.](image)
4. Experimental setup for case study

4.1. Streaming Data Sets

**Real-world application: engine test benches.** One data stream we are dealing with was recorded from the supervision of the behavior of a car engine during simulated driving operations at an engine test bench. Two different test campaigns, covering steady and transient operation were performed. These included the engine speed and torque profiles during a Motor Vehicle Emissions Group cycle, a sportive driving profile on a mountain road and two different synthetic profiles. Several sensors at the engine test bench were installed which measure various important engine characteristics such as the pressure of the oil, various temperatures or emission gases (NOx, CO2 etc.) [23]. Online data comprising 22302 measurements in total (covering the 4 different profiles) were recorded and 42 channels, which could be reduced to the 9 most important ones for the purpose of NOx modelling, according to expert-based and data-driven pre-selection (see [23]). The main task is thus to build an accurate prediction model for the NOx emission content online in an incremental, adaptive manner, as time-intensive (batch) re-training should be prevented when new profiles arise. NOx is the most important gas according to strict standards in the ISO-norm to meet the emission requirements for newly built cars in order to stay in-line the maximal air pollution bounds.

**Real-world application: year prediction MSD data set.** This data set stems from the task to perform major system identification steps in a cold rolling mill system in order to establish (data-driven causal relation) models which can be further used for fault detection and localization purposes, see also [35] for details. The higher the accuracy of the models, the better their fault detection/localization performance. Due to changes in the dependencies between system variables for different production cycles and so-called stitches, it is necessary to update the models to maintain their prediction performance. A partial stream subset was extracted comprising 9486 samples and a causal relation between 6 variables, with the target the expected thickness of the plate and the inputs pre-selected by experts [35]. These 6 variables are expected to be permanently measured, thus an immediate supervised update of the models can be performed, and the models used for supervision purposes (FDI) on the next sample(s) (checking and analyzing predicted versus observed values). It is remarkable that a cyclic drift is embedded towards the end of the stream, which means that a new drifted state starts at around Sample 5600 and ends at around Sample 6400 (reflecting a particular phase during production milling), where the system changed back to the old state (the conventional on-going production phase). This thus serves to examine: i) how well our bagging approach in combination with drift detection together with the evolution of new ensemble members can compensate for the drift (also in comparison with single EFS models and related SoA techniques), and ii) how robust our approach is in the case of cyclic drifts, i.e. how well newly evolved ensemble members during the drift phase are ‘out-masked’ later when the system changes back to the old (regular) state.

**Real-world application: rolling mills.** This data set stems from the task to perform major system identification steps in a cold rolling mill system in order to establish (data-driven causal relation) models which can be further used for fault detection and localization purposes, see also [35] for details. The higher the accuracy of the models, the better their fault detection/localization performance. Due to changes in the dependencies between system variables for different production cycles and so-called stitches, it is necessary to update the models to maintain their prediction performance. A partial stream subset was extracted comprising 9486 samples and a causal relation between 6 variables, with the target the expected thickness of the plate and the inputs pre-selected by experts [35]. These 6 variables are expected to be permanently measured, thus an immediate supervised update of the models can be performed, and the models used for supervision purposes (FDI) on the next sample(s) (checking and analyzing predicted versus observed values). It is remarkable that a cyclic drift is embedded towards the end of the stream, which means that a new drifted state starts at around Sample 5600 and ends at around Sample 6400 (reflecting a particular phase during production milling), where the system changed back to the old state (the conventional on-going production phase). This thus serves to examine: i) how well our bagging approach in combination with drift detection together with the evolution of new ensemble members can compensate for the drift (also in comparison with single EFS models and related SoA techniques), and ii) how robust our approach is in the case of cyclic drifts, i.e. how well newly evolved ensemble members during the drift phase are ‘out-masked’ later when the system changes back to the old (regular) state.

**Real-world application: year prediction MSD data set.** This is taken from the UCI-repository [1] and serves as a benchmark for our stream learning duties. It is a large scale, high-dimensional and complex data stream problem containing 515345 samples and 90 input variables for predicting the release year of audio songs. The input variables are particular audio features extracted from the ‘timbre’ features from The Echo Nest API (for further information please refer to the link to the UCI repository in the footnote). We used the first 20000 samples for a performance comparison between OB-EFS and related techniques because afterwards, the model errors turned out to become fairly constant at a certain level (no further dynamics/changes included). We also used a reduced input dimensionality of 11 most important inputs, which were pre-selected by data-driven forward selection on the whole data set.

**Time-series based forecasting of production quality.** In this case study, we deal with the inspection of micro-fluidic chips used for sample preparation in DNA (deoxyribonucleic acid) sequencing. Originally, bad chips were sorted out once they have already been produced, based on machine learning classifiers (inspecting the surface of the chips). This, however, typically does not prevent unnecessary waste and can even induce greater complications and risks at the production site. Therefore, predicting downtrends in the quality of the chips at an early stage is an important challenge to be addressed. Based on 64 continuously measured process values found to be useful for tracking the production behavior at the injection moulding machine (= first stage of the whole production process), time-series based forecast models should be established from data which accurately predict important quality criteria in the last stage of production (bonding liner).

An important target criterion is the so-called void events, telling the expert how much atypical occurrences on the chips are visible (with a critical value of 100, also see [24] for details). The forecast of its value based on (longer) trends of process values (comprising up to 500 samples) is of utmost importance to recognize problems early in the production. Due to the longer trends, we ended up with an input dimensionality of 64*500 = 32000, which is remarkably high and not realistically processable; thus, partial least squares regression (PLSR) [11] was used in advance to project the 32000-dimensional time-series space to the three most important latent variables (LVs), whose scores are then sent into the data processing environment and used in the evolving learning engines. Three LVs were sufficient to describe the transformed time-series space adequately (see [24]). 900 target measurements could be gathered over a time frame of several months, from which we used the first 100 to establish an initial forecast model, and the remaining 800 for model adaptation. This is necessary, as in our preliminary study in [24] it turned out that static models significantly lose their predictive capabilities over time (not being able...
to stay within the company’s forecast error expectations). It is also remarkable that a (slight, gradual) drift is present in the stream, starting from around Sample # 420 and lasting towards the end of the stream.

The characteristics of the four data sets are summarized in Table 1.

4.2. Evaluation strategy

The evaluation is conducted by performing a direct one-to-one comparison between OB-EFS and single EFS models using the same EFS learning engine (Gen-Smart-EFS [22]), one time in stand-alone mode, one time plugged into Step 2 of Algorithm 1. In both cases, they operate within the same data processing environment (implemented within the MATLAB 2019b environment). To achieve a fair comparison, especially regarding computation times, all tests were made on the same personal computer with a Windows 10 operating system and the same maximal possible resources available (no background programs started etc.).

In this data processing environment, one sample is loaded, processed through the model update engine and immediately discarded, afterwards (thus, simulating a classical single-pass procedure). Furthermore, a prediction is made with the loaded sample before the model update is carried out, which is then compared with the real target value. The absolute errors between predictions and the real target values are accumulated over time as in (11), but also normalized with the range of the target y. This results in a normalized accumulated mean absolute error (NMAE) (which is also comparable among data streams with different target ranges). This is in accordance with the interleaved one-step-ahead test-and-then-train scenario, which is widely used in the data stream mining and machine learning community (see [4]), and also an evaluation variant within the well-known MOA (Massive Online Analysis) framework [3].

In the results section, we show the error trend lines over time as achieved by OB-EFS and by single EFS model in the same plots. This serves for the purpose of a direct comparison whether online bagged EFS achieved some improvement over single EFS models by showing a better decreasing converging behavior or lower trends in major parts of the streams. Thereby, we perform four different OB-EFS variants: i) OB-EFS with only online bagged sampling but no pruning and evolution of members (so, we omit Steps 3 and 4 in Algorithm 1), ii) OB-EFS with onlined bagged sampling and hard pruning (but no evolution, so we omit Step 4 in Algorithm 1), iii) OB-EFS with online bagged sampling and soft pruning (thus, offering the possibility of a member recall option) and iv) OB-EFS with full functionality, i.e. soft pruning + drift detection + evolution of new members (the complete Algorithm 1). We plot the error trend lines of the four variants as four lines (in different styles and colors) in one figure to make a direct comparison. Additionally, we applied three related well-known SoA techniques in EFS to the same data streams (within the same data processing environment to achieve a fair comparison), namely PANFIS [30], pRVFLN [31] and FAOS-PFNN [40], and compared their accumulated error trend lines in the same figure with the best (of the four possible) OB-EFS variants (which was always OB-EFS with full functionality as can be seen from the results below) and with single model GS-EFS.

We fixed the number of starting bags to $B = 20$ in all cases, which typically automatically changes over time, according to the nature of the stream (due to pruning and evolution of ensemble members in Algorithm 1). For the window size $w$ to evaluate the Hoeffding inequality, the same value of 100 samples was used in all test scenarios. For the significance level $\alpha$ on the Hoeffding inequality, to decide whether a new member should be actually evolved or not, we used the pessimistic setting of 0.01 in all test scenarios.

We also show the trends of the number of bags and ensemble members over time, as they developed when applying pruning and evolution options. This is especially interesting in the two drift cases (normal and cyclic drift) included in the two data sets for rolling mills and chip production (see above). We also report the computation times needed to update the whole model on single samples for each data stream, to check with which sample frequency OB-EFS can cope and whether the real-time demands from the applications can be met. In this regard, we also compared the computation times achieved by the single GS-EFS model and by the related SoA techniques.

5. Results

5.1. Results on Stream Regression Sets

Fig. 4 shows the error trend lines (accumulated normalized MAEs) over time on the left hand side and the number of active ensemble members on the right hand side. The various bagging variants are shown in different line styles: native bagging (w/o pruning and evolution) as a dotted line, bagging with hard pruning as a dashed-dotted line, bagging with soft pruning (and possible recall) as a dashed line and bagging with soft pruning and evolution as a solid line. The latter curve is only plotted when drifts are detected and thus new members are evolved, otherwise it becomes identical to the soft pruning option without evolution, as was the case for engine test benches and YearMSD (see below). The single EFS models are plotted in dashed grey lines. Active members are characterized by weights $w_b > 0$, as they contribute (at least a little) to the overall prediction in (22). The top row shows the results for the engine test bench data stream, the middle one the results for the YearMSD data stream and the bottom one the results for the rolling mills.

In the case of engine test benches, the single EFS model shows a significant rise in the accumulated MAE towards the end of the stream, starting at around Sample # 17000. This is due to a new driving condition arising in the data stream which has
different behavior to the other driving conditions seen before [23]. Indeed, more new rules than before are evolved during this phase, but the single model approach takes a bit of time (for more samples) to correctly shape out the positions and especially the consequent hyper-planes of these new rules. This leads to a deterioration of the prediction error and a delayed

Table 1
Characteristics of the applied data sets from the three application scenarios under study.

| Data Set                  | Input Dim. | # of Samples | Noise Level |
|---------------------------|------------|--------------|-------------|
| Stream Regr. Problems     | 42 (8 red.) | 22302        | High        |
| Engine Test Bench         | 5          | 9486         | Low         |
| Rolling Mills             | 90 (11 red.) | 20000      | Medium      |
| Year MSD                  | 32000 (3 proj.) | 800        | Medium      |
| Time-Series Forecasting    |            |              |             |
| Chip Quality Index        |            |              |             |

Fig. 4. Left: error trend lines over time showing a performance comparison between the single EFS model and bagging variants for the three regression stream data sets (top: engine test benches, middle: yearMSD, bottom: rolling mills); right: trend lines of the number of ensemble members over time for the various bagging variants; the evolution of members upon drift detection is indicated by the text in the plots.

28
compensation. This situation is much improved by the bagging variants simply because more samples are drawn with replacement (and this for $B$ models in sum), thus the significance of the models’ overall predictions becomes higher, leading to more stable outputs, which is a well-known fact from batch bagging (see also its motivation in Section 2). This means that online bagging is more robust at the beginning of newly arising conditions in streams per se. It is further remarkable that soft pruning can outperform hard pruning and native bagging, again underlining its stability based on the flexibility of member recalls, as it was carried out between Sample #3000 and #5000 (see the upper-right most plot). This leads to a significantly decreased trend compared to hard pruning (see dashed versus dashed-dotted line). Hard pruning is counterproductive, because it shows an even worse trend than native bagging. This can happen in the case when members, which may become important again at a latter stage, are pruned too early (from 20 down to 17, see the right plot), as is obviously the case here, because they are recalled 1000–2000 samples later when soft pruning is switched on. The update of the whole ensemble with each single sample took about 0.153 s on average in the case of soft pruning with the recall option and negligible less time for hard pruning and native bagging. As the data was recorded with a frequency of 1 Hz, OB-EFS can still cope with real-time demands.

In the case of the YearMSD data, which neither contains any evolutions of new conditions nor drifts of older conditions, a comparison of the methods is thus made when only parameter adaptation and regular rule evolution/splitting steps take place (e.g., due to regular feature range expansions). Clearly, the single EFS model could be again significantly outperformed by all bagging variants, especially from Sample #2000 on, leading to 1.5% lower normalized MAEs, i.e., an error reduction of about 15%. As can be seen from the left middle plot, the error convergence performed fairly smoothly for almost the whole stream (after the start phase) because no drifts arose. It should be emphasized that no drifts were detected, thus no false alarms occurred. Soft pruning with a recall option performed a revival of ensemble members around Sample #3000, which led to a lower error trend than hard pruning (and also than native bagging) for the remaining stream. Obviously, two members were hard-pruned too early, which later lead to the pruning of another three members. The update of the whole ensemble with each single sample took about 0.15 s on average in the case of soft pruning with a recall option and negligible less time for hard pruning and native bagging.

In the case of the rolling mill data, it is interesting to see that the single EFS model has a lower error curve up to the 4000th sample than any bagged variant. The noise in this data set is very low, as can also be seen from the fairly smooth error curves. However, later when drift arises, the single EFS model performs worse than any bagging variant, especially during the back-change towards the end of the stream (while during the drift it performs nearly the same). That is, a single model which automatically adapts to the drift to some extent has lower flexibility than an ensemble model when a back-change occurs. This is somehow intuitive, as in the ensemble model new members may have been evolved during the drift stage (in the case when using an evolution option, see the right lower plot), but models trained on the old state are still present in the case of soft pruning (as they have only been down-weighed for prediction due to a lower coverage during the drift phase). So, when the old state is re-visited, older ensemble members can be directly re-activated for proper predictions without requiring any new samples from the old state to back-adapt its structure and parameters (as needed for a single model because it forgot the older state to some extent due to its adaptation during the drift phase). Furthermore, interestingly the two ensemble members evolved during the drift state do not harm the prediction quality after the drift phase (although one produced 4–5 times higher errors than the others, as we find by conducting a deeper analysis), because they are again out-masked due to the weighted prediction in (22). One of the immediate disappointments was that the evolution of new members during the drift phase (as the drift was detected around Sample #6000) did not help much to decrease the error during this phase compared to the other bagging variants (nearly the same error trend continues). We analyzed this and found that the ‘modelability’ of the problem worsened during the drift, as the drift basically concerned the target concept. We found this by training the models one time solely with samples from the drift phase and one time solely with samples before the drift phase (with the optimal parameter tuning phase included etc.): the errors of the models from the drift phase (on separate validation data) were always 3–4 times higher than the errors of the models before the drift phase. So, the possible degree of compensation of drifts during stream modeling (even though detected correctly) also depends on the change in the ‘modelability’ of the whole learning problem. In fact, when this worsens, an evolution of new members, rules or other structural components cannot achieve the same (low) errors as seen before the drift, always leading to a rising trend of the accumulated error.

5.2. Results on stream-based time-series forecasting

Fig. 5 shows the error trend lines (accumulated normalized MAEs) over time in the upper plot and the number of active ensemble members in the lower one, indicating the various bagging variants in different line styles: native bagging (w/o pruning and evolution) as a dotted line, bagging with hard pruning as a dashed-dotted line, bagging with soft pruning (and possible recall) as a dashed line and bagging with soft pruning and evolution as a solid line. Obviously, the outperformance of the single (classical) EFS model (dashed grey line) by online bagging variants is significant, especially from around Sample #100 on (i.e., after the ‘lead-in’ phase when the errors start to converge). During the end of the streaming process, native bagging and bagging with hard pruning show similar performance as the single EFS model, which can be significantly further improved by soft pruning with and without evolution. It is thereby remarkable, that hard pruning can slightly improve the worsened error trend of native bagging for the last 100 samples by pruning two bad members (see the lower plot of Fig. 5, where the number of members decrease from 17 to 15), which obviously spoiled the overall predictions. It is
even more remarkable that soft pruning is able to achieve a more decreasing error trend from around Sample #200 on than the native bagging and bagging with hard pruning and especially that the evolution of new ensemble members became active during the presence of the drift starting at around Sample #420, indeed, with some delay of about 70 samples (due to a slight gradual intensity), but the evolution of the two additional ensemble members (as indicated in the lower plot) obviously could lead to a significant error reduction (by about 10–15%). During the second half of the stream, the single EFS model was outperformed by the best bagging variant by around 1% normalized MAE, i.e. a decrease of around 4% to around 3% normalized MAE in absolute value, which is a significant error reduction of 25%.

All in all, we can thus conclude that soft pruning with evolution pays off in terms of lower error trends, especially when drifts happen, and this with only a bit more ensemble complexity and thus negligible more computational demands: the update for a single sample took 0.074 s on average in the case of hard pruning, and 0.089 s in the case of soft pruning with evolution, which is fast enough to cope with online demands because in this production system, about each 25th second a new sample is available for the model update.

5.3. Comparison with state-of-the-art methods

Finally, we performed a comparison with related state-of-the-art methods, namely with PANFIS [30] (a novel incremental learning machine with more than 200 references), pRVFLN [31] (parsimonious random vector functional link network including also recurrent links between input feature delays and Chebychev polynomials in the consequent parts) and FAOS-PFNN [40] (a well-known online self-organizing scheme for neuro-fuzzy systems), all of which are well-known and widely used methods in industrial applications and are also emphasized and discussed in recent surveys in evolving neuro-fuzzy systems (see [38] [21]). We therefore used the default parameters as reported in the respective publications in a first test run, but then also tuned the essential parameters therein (3 in the case of PANFIS and pRVFLN, 9 in the case of FAOS-PFNN), to obtain the best possible results. The latter are shown in Fig. 6 for the four data sets as discussed in the preliminary subsections and compared with single model GS-EFS and with the best variant of our OB-EFS approach (using GS-EFS as ensemble members), which was the soft pruning variant in combination with the autonomous evolution of ensemble members. The left upper plot in Fig. 6 shows the error trend lines for the YearMSD data, the right upper plot for the engine test benches, the left lower plot for rolling mills, the right lower plot for time-series based prediction of void events in chip production.

It is interesting to see that our proposed OB-EFS approach performs significantly better than all the other methods in the longer run, ending up with the lowest error in all cases. In particular, a decrease of the final error is achieved by 15–30% in the case of YearMSD (15% in comparison with the second-best approach (GS-EFS) and 30% with the worst approach (PANFIS)), by 17–68% in the case of engine test benches, by around at least 30% in the case of rolling mills and by 19–38% in the case of chip production data. Furthermore, the error trend lines achieved by OB-EFS are the lowest for each of the final three quarters of the streams, except for the rolling mills data set where GS-EFS performs best until the end of the drift, but then has a lower flexibility (showing a higher error trend) than OB-EFS to back-change to the normal state before the drift started, an issue which has been claimed in the methodological description of our approach. In terms of the state-of-the-art methods, FAOS-PFNN seems to work best, especially showing the lowest error trends at the beginning of the engine test benches and chip production streams, however not being able to converge to lower trends than OB-EFS (and also than GS-EFS). In the case of rolling mills, it shows a higher trend line, but interestingly seems to be less affected by the drift as the error increases only marginally, relative to the trend line before the drift (but still ends up being higher than in the case of OB-EFS). Moreover, FAOS-PFNN turns out to be the slowest method as seen in Table 2.

Fig. 5. Upper: error trend lines over time showing a performance comparison between single EFS model and bagging variants for the chip production quality data (multi-stage forecasting of void events); lower: trend lines of the number of ensemble members over time for the various bagging variants; the evolution of members upon drift detection is indicated by the text.
Regarding computation times, Table 2 compares the methods in terms of seconds required to update a single sample on average in all streams. It can be clearly recognized that PANFIS and pRVFLN are significantly faster (by a factor of 10 or more) than the other methods, being able to process single samples within a frequency of more than 100 Hz (less than one millisecond). On the other hand, these two methods turned out to be the worst performers in terms of accumulated accuracy as shown in Fig. 6, significantly worse than OB-EFS (30–70% higher error trends and hardly any convergence in the case of drifts (see the performance on rolling mills)). In this sense, it is a tradeoff between computational resources (depending on the required computation speed due to the sampling frequency) and the maximal allowable model error. However, although OB-EFS performs slower than these related SoA works, it is still able to process the data and update the whole ensemble of models in real-time, as the sampling frequency for the engine test benches, rolling mills and YearMSD data was about 1 Hz (thus, one sample per second is loaded), and in the case of chip production, it was about 0.04 Hz (thus, each 25 s one sample is loaded), whereas the update speed of OB-EFS per sample in these three data sets was significantly below one second. Hence, OB-EFS can be applied without ‘information loss’ and thus would be the first choice because it performed significantly more robustly than the other methods. This is not the case for FAOS-PFNN in the case of YearMSD data (1.22 s update speed), which obviously slows down significantly when the input dimension becomes higher (YearMSD has the highest number of inputs, and engine test benches the second highest, see Table 1); for the other two data sets, it is faster than OB-EFS.

| Method     | Engine | Rolling Mills | YearMSD | Chip Prod. |
|------------|--------|---------------|---------|------------|
| GS-EFS     | 0.007  | 0.008         | 0.014   | 0.002      |
| PANFIS     | 0.0013 | 0.0006        | 0.002   | 0.0003     |
| pRVFLN     | 0.0003 | 0.0002        | 0.0003  | 0.0003     |
| FAOS-PFNN  | 0.172  | 0.0011        | 1.22    | 0.0015     |
| OB-EFS     | 0.153  | 0.0152        | 0.15    | 0.089      |

Table 2
Computation times for model updates on single samples on average (in seconds).

Fig. 6. Error trend lines over time showing a performance comparison between our OB-EFS approach (black solid line markers) and state-of-the-art methods in EFS (dotted and dashed line markers); upper left for YearMSD data, upper right for engine test benches, lower left for rolling mills and lower right for void events in chip production.
6. Conclusion

We demonstrated a new variant of online ensembles of EFS models (as base members), termed OB-EFS, by employing an online variant of bagged sampling in order to improve the significance and robustness of single EFS models, especially in the case of higher noise levels and/or a low number of samples, as discussed in Section 2 for the offline case. The probabilistic bagged sampling from the stream as demonstrated in Section 3.1 converges to the classical batch bagged sampling and thus inherits its robustness aspects. Improved robustness could be verified in the case when new operating conditions arise by achieving lower errors due to a faster integration of new operating conditions with an initial low number of samples available. Furthermore, our approach embeds autonomous (hard and soft) pruning and the evolution of the ensemble members in order to dynamically handle gradual and cyclic drifts, and this on a local member as well as on a global ensemble basis. Empirical results showed decreased error trends when using soft pruning combined with the evolution of ensemble members compared to hard pruning and especially compared to single EFS models. Particularly in the case of drifts and the dynamic evolution of new operating conditions, single EFS models produced higher errors than OB-EFS due to a lower model variety and flexibility. Finally, our method showed significantly improved error trends on all data sets compared to related and well-known state-of-the-art methods throughout three quarters of the stream and especially ended up with significantly lower errors (by 20–70%), while mostly having been slower (except FAOS-PFNN which was slowest of all methods), but still having met real-time capabilities in the respective applications. In the case when the bagged ensembles of EFS become larger due to using many bags at start of the learning process or due to a significant number of bags evolving autonomously during the stream adaptation phase (e.g., because of several drift phases), the computation time of the model updates could be a real limitation of our approach whenever stream samples come in with a high frequency. In such cases, someone may limit the generation of new ensembles once a maximal allowed number is reached or someone may allow hard pruning to become active, which also showed good error trends compared to the related methods.

Future work addresses: i) the integration of advanced prediction schemes, not only including the current coverage degree of ensemble members but also their recent error trends (thus being able to react to drifts also in the target concept), and ii) attempts for an incremental soft dimension reduction by performing subset selection and weighting based on the different bags, which leads to a different subspace for each bag, thus ending up with a kind of mixture of bagging and boosting of EFS.

CRedIT authorship contribution statement

Edwin Lughofer: Methodology, Conceptualization, Software. Mahardhika Pratama: Investigation, Writing – review & editing. Igor Škrjanc: Investigation, Writing - review & editing.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The first and third authors acknowledge the support by the “LCM – K2 Center for Symbiotic Mechatronics” within the framework of the Austrian COMET-K2 program.

References

[1] P. Angelov, Autonomous machine learning (alma): generating rules from data streams, in: Special International Conference on Complex Systems 2011, 2011, pp. 249–256.
[2] P.P. Angelov, Autonomous Learning Systems: From Data Streams to Knowledge in Real-time, John Wiley & Sons, New York, 2012.
[3] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: Massive online analysis, Journal of Machine Learning Research 11 (2010) 1601–1604.
[4] A. Bifer, G. Holmes, R. Kirkby, B. Pfahringer, Data stream mining — a practical approach. Technical report, Department of Computer Sciences, University of Waikato, Japan, 2011.
[5] P. Brazdil, C. Giraud-Carrier, C. Soares, R. Vilalta, Metalearning, Springer, Berlin Heidelberg, 2009.
[6] L. Breiman, Bagging predictors, Machine Learning 24 (2) (1996) 123–140.
[7] M. Collins, R. Schapire, Y. Singer, Logistic regression, adaboost and bregman distances, Machine Learning 48 (1–3) (2002) 253–285.
[8] M. Esquivel, Probability generating functions for discrete real-valued random variables, Theory of Probability & Its Applications 52 (1) (2008) 40–57.
[9] J. Gama, Knowledge Discovery from Data Streams, Chapman & Hall/CRC, Boca Raton, Florida, 2010.
[10] G.R. Grimmett, D.R. Stirzaker, Probability and Random Processes, Oxford Science Publications, New York, 1992.
[11] M. Haenlein, A.M. Kaplan, A beginner’s guide to partial least squares (PLS) analysis, Understanding Statistics 3 (4) (2004) 283–297.
[12] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference and Prediction - Second Edition, Springer, New York Berlin Heidelberg, 2009.
[13] T. Hill, P. Lewicki, Statistics: Methods and Applications, StatSoft, Tulsa, 2007.
[14] W. Hoeffding, Probability inequalities for sums of bounded random variables, Journal of the American Statistical Association 58 (301) (1963) 13–30.
[15] J. Huhne, E. Hüllermeier, FR3: A fuzzy rule learner for inducing reliable classifiers, IEEE Transactions on Fuzzy Systems 17 (1) (2009) 138–149.
[16] M. Islam, X. Yao, S. Nirjon, M. Islam, K. Murase, Bagging and boosting negatively correlated neural networks, IEEE Transactions on Systems, Man and Cybernetics Part B: Cybernetics 38 (3) (2008) 771–784.
[17] Y. Jiang, Q. Zhao, Y. Lu, Ensemble based data stream mining with recalling and forgetting mechanisms, in: Proceedings of the 11th International Conference on Fuzzy Systems and Knowledge Discovery, IEEE press, Xiamen, China, 2014, pp. 430–435.
[18] L. Khamassi, M. Sayed-Mouchaweh, M. Hammami, K. Ghedira, Discussion and review on evolving data streams and concept drift adapting, Evolving Systems 9 (1) (2017) 1–23.
[19] E.P. Klement, R. Mesiar, E. Pap, Triangular Norms, Kluwer Academic Publishers, Dordrecht Norwell New York London, 2000.
[20] D. Leite, I. Škrjanc, Ensemble of evolving optimal granular experts, OWA aggregation, and time series prediction, Information Sciences 504 (2019) 95–112.
[21] E. Lughofer, Evolving fuzzy systems — fundamentals, reliability, interpretability and useability, in: P. Angelov (Ed.), Handbook of Computational Intelligence, World Scientific, New York, 2016, pp. 67–135.
[22] E. Lughofer, C. Cernuda, S. Kindermann, M. Pratama, Generalized smart evolving fuzzy systems, Evolving Systems 6 (4) (2015) 269–292.
[23] E. Lughofer, V. Macian, C. Guardiola, E.P. Klement, Identifying static and dynamic prediction models for NOx emissions with evolving fuzzy systems, Applied Soft Computing 11 (2) (2011) 2487–2500.
[24] E. Lughofer, A.C. Zavoianu, R. Pollak, M. Pratama, P. Meyer-Heye, H. Zörer, C. Eitzinger, T. Radauer, Autonomous supervision and optimization of product quality in a multi-stage manufacturing process based on self-adaptive prediction models, Journal of Process Control 76 (2019) 27–45.
[25] N.C. Oza, S. Russell, Online bagging and boosting, Artificial Intelligence and Statistics (2001) 105–112.
[26] Y. Pan, P. Du, H. Xue, H.-K. Lam, Singularity-free fixed-time fuzzy control for robotic systems with user-defined performance, IEEE Transactions on Fuzzy Systems, 2020, doi: 10.1109/TFUZZ.2020.2990746 (on-line and in press).
[27] W. Pedrycz, F. Gomide, Fuzzy Systems Engineering: Toward Human-Centric Computing, John Wiley & Sons, Hoboken, New Jersey, 2007.
[28] H. Pishro-Nik, Introduction to Probability, Statistics, and Random Processes, Kappa Research, United States, 2014.
[29] R. Polikar, Ensemble based systems in decision making, IEEE Circuits and Systems Magazine 6 (3) (2006) 21–45.
[30] M. Pratama, S.G. Anavatti, P. Angelov, E. Lughofer, PANFIS: A novel incremental learning machine, IEEE Transactions on Neural Networks and Learning Systems 25 (1) (2014) 55–68.
[31] M. Pratama, P. Angelov, E. Lughofer, M.J. Er, Parsimonious random vector functional link network for data streams, Information Sciences 430–431 (2018) 519–537.
[32] M. Pratama, W. Pedrycz, E. Lughofer, Evolving ensemble fuzzy classifier, IEEE Transactions on Fuzzy Systems 26 (5) (2018) 2552–2567.
[33] M. Sayed-Mouchaweh, E. Lughofer, Learning in Non-Stationary Environments: Methods and Applications, Springer, New York, 2012.
[34] C. Schaffer, Overfitting avoidance as bias, Machine Learning 10 (2) (1993) 153–178.
[35] F. Serdio, E. Lughofer, K. Pichler, M. Pichler, T. Buchegger, H. Efendic, Fuzzy fault isolation using gradient information and quality criteria from system identification models, Information Sciences 316 (2015) 18–39.
[36] P. Sidhu, M.P.S. Bathia, An online ensembles approach for handling concept drift in data streams: diversified online ensembles detection, International Journal of Machine Learning and Cybernetics 6 (6) (2015) 883–909.
[37] W. Siler, J.J. Buckley, Fuzzy Expert Systems and Fuzzy Reasoning: Theory and Applications, John Wiley & Sons, Chichester, West Sussex, 2005.
[38] I. Škrjanc, J. Iglesias, A. Sanchis, E. Lughofer, F. Gomide, Evolving fuzzy and neuro-fuzzy approaches in clustering, regression, identification, and classification: A survey, Information Sciences 490 (2019) 344–368.
[39] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, IEEE Transactions on Systems, Man and Cybernetics 15 (1) (1985) 116–132.
[40] N. Wang, M.J. Er, X. Meng, A fast and accurate online self-organizing scheme for parsimonious fuzzy neural networks, Neurocomputing 72 (16–18) (2009) 3818–3829.
[41] Z.H. Zhou, Ensemble Methods: Foundations and Algorithms, Chapman & Hall/CRC Data Mining and Knowledge Discovery Series, Boca Raton, Florida, 2012.