Data Mining and Knowledge Discovery (2021) 35:1941–1971
https://doi.org/10.1007/s10618-021-00769-1

TSK-Streams: learning TSK fuzzy systems for regression on data streams

Ammar Shaker[1] · Eyke Hüllermeier[2,3]

Received: 1 February 2021 / Accepted: 25 May 2021 / Published online: 23 June 2021

Abstract
The problem of adaptive learning from evolving and possibly non-stationary data streams has attracted a lot of interest in machine learning in the recent past, and also stimulated research in related fields, such as computational intelligence and fuzzy systems. In particular, several rule-based methods for the incremental induction of regression models have been proposed. In this paper, we develop a method that combines the strengths of two existing approaches rooted in different learning paradigms. More concretely, our method adopts basic principles of the state-of-the-art learning algorithm AMRules and enriches them by the representational advantages of fuzzy rules. In a comprehensive experimental study, TSK-Streams is shown to be highly competitive in terms of performance.

Keywords Fuzzy rules · TSK systems · Evolving fuzzy systems · Machine learning · Data streams

1 Introduction

In many practical applications of machine learning and predictive modeling, data is produced incrementally in the course of time and observed in the form of a continuous, potentially unbounded stream of observations. Correspondingly, the problem of learning from data streams (Gama 2012) has received increasing attention in recent

Responsible editor: Toon Calders.

✉ Eyke Hüllermeier
eyke@upb.de

Ammar Shaker
ammar.shaker@neclab.eu

1 NEC Laboratories Europe, Heidelberg, Germany

2 Institute of Informatics, University of Munich, Munich, Germany

3 Paderborn University, Warburger Str. 100, 33098 Paderborn, Germany

years. Algorithms for learning on streams must be able to process the data in a single pass, which implies an incremental mode of learning, to detect and handle different types of drift in the data distribution, and, correspondingly, adapt to changes of the underlying data-generating process (Gama et al. 2014; Lu et al. 2019).

A popular approach for learning on data streams, both for classification and regression, is rule induction, in the fuzzy logic and computational intelligence community also known as "evolving fuzzy systems" (Lughofer 2011). Shaker et al. (2017) proposed a method for regression that builds on a very efficient and effective technique for rule induction, which is inspired by the state-of-the-art machine learning algorithm AMRules (Almeida et al. 2013), and combines it with the strengths of fuzzy modeling. Thus, the method induces a set of fuzzy rules, which, compared to conventional rules with Boolean antecedents, has the advantage of producing smooth regression functions.

The method presented in this paper, called TSK-Streams, is a substantially revised and improved variant of (Shaker et al. 2017). The main modifications and novel contributions are as follows:

- We give a concise overview of regression learning on data streams as well as a systematic comparison of existing methods with regard to properties such as discretization of features, splitting criteria for rules, etc. This overview helps to better understand the specificities and characteristics of approaches originating from different research fields, as well as to position our own approach.
- We introduce a new strategy for the induction of TSK fuzzy rules and realize it in the form of two concrete variants: variance reduction and error reduction. While the former is still close to Shaker et al. (2017), the variance reduction approach has not been considered for online learning of fuzzy systems so far. Compared with error reduction and other state-of-the-art methods, it leads to models with superior predictive performance.
- In Shaker et al. (2017), rule antecedents may contain disjunctions and negations, which makes them difficult to understand and interpret. The representation of TSK rules used in this paper is simpler and more concise. This is achieved by means of an improved technique for splitting fuzzy sets (and extending corresponding rules).
- We propose the induction of candidate fuzzy rules using a discretization technique that is based on an extended Binary Search Tree (E-BST) structure. Compared to the three-layered discretization architecture used by Shaker et al. (2017), the use of E-BST for constructing candidate fuzzy sets has a number of advantages in the context of online learning. Most notably, it comes with a reduction of complexity from linear to logarithmic (in the number of candidate extensions).
- Our empirical evaluation is more extensive and comprises a couple of additional large-scale data sets with up to 100k instances. The evaluation is also extended by including an additional method that has been introduced recently (Gomes et al. 2018).

The rest of the paper is organized as follows. Following a brief discussion of related work and systematic exposition of methods for regression on data streams in Sect. 2, we introduce our method TSK-Streams in Sect. 3. Section 4 presents a comprehensive experimental study, in which TSK-Streams is compared to several competitors, prior to concluding the paper in Sect. 5.

2 Learning regression models on data streams

Recall that, in the standard setting of supervised learning, a learner is given access to a set of training data $\mathcal{D} = \big\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\big\} \subset \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is an instance space and $\mathcal{Y}$ the set of outcomes that can be associated with an instance. In the case of regression, $\mathcal{Y} = \mathbb{R}$, i.e., outcomes are real numbers. Most commonly, instances are represented in terms of feature vectors $\boldsymbol{x}_i = (x_{i,1}, \ldots, x_{i,d})^\top \in \mathbb{R}^d$. Given a *hypothesis space* $\mathcal{H}$ (consisting of hypotheses $h : \mathcal{X} \longrightarrow \mathcal{Y}$ mapping instances $\boldsymbol{x}$ to outcomes y) and a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \longrightarrow \mathbb{R}$, the goal of the learner is to induce a hypothesis $h^* \in \mathcal{H}$ with low risk (expected loss)

$$R(h) := \int_{\mathcal{X}\times\mathcal{Y}} \ell(h(\boldsymbol{x}), y)\, d\,\mathbf{P}(\boldsymbol{x}, y), \tag{1}$$

where $\mathbf{P}$ is a joint probability measure on $\mathcal{X} \times \mathcal{Y}$ characterizing the data-generating process. Thus, given the training data $\mathcal{D}$, the learner needs to “guess” a good hypothesis h. Assuming the training examples $(\boldsymbol{x}_i, y_i)$ to be independent and identically distributed (according to $\mathbf{P}$), this choice is commonly guided by the empirical risk $R_{emp}(h) := \frac{1}{N}\sum_{i=1}^{N} \ell(h(\boldsymbol{x}_i), y_i)$, i.e., the performance of a hypothesis on the training data. However, since $R_{emp}(h)$ is only an estimation of the true risk $R(h)$, the hypothesis (empirical risk minimizer) $\hat{h} := \operatorname{argmin}_{h\in\mathcal{H}} R_{emp}(h)$ favored by the learner will normally not coincide with the true risk minimizer $h^* := \operatorname{argmin}_{h\in\mathcal{H}} R(h)$. Moreover, since empirical risk minimization is prone to overfitting the training data, which in turn compromises generalization performance, the learning process is typically regularized.

When learning from data streams, as opposed to learning in “batch mode”, the training data is not given in the form of a static data set $\mathcal{D}$. Instead, the data is produced in an online manner, and training examples are provided one by one. In other words, the data forms a potentially unbounded, continuously evolving sequence $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_i, y_i), \ldots$ of data points. Also, the data generating process is not necessarily stationary, i.e., the $(\boldsymbol{x}_i, y_i)$ are not necessarily generated from the same distribution $\mathbf{P}$. Instead, this distribution may change in the course of time, giving rise to what is called concept drift or concept shift in the literature (Gama et al. 2014; Lu et al. 2019).

2.1 An overview of existing methods

In the machine learning community, research on supervised learning from data streams has mainly focused on classification problems so far. As one of the first methods, Hoeffding trees (Domingos and Hulten 2000) have been proposed for learning classifiers on high-speed data streams. Since then, the tree-based approach has been developed further, and various modifications and variants can be found in the current literature (Bifet and Gavaldà 2009). Closely related to tree-based approaches is the induction of decision rules. For example, the Adaptive Very Fast Decision Rules (AVFDR) method (Kosina and Gama 2012) is an extension of the Very Fast Decision Rules (VFDR) classifier (Gama and Kosina 2011), which learns a compact set of rules in an incrementally manner. Most recently, Bifet et al. (2017) developed an extremely fast version of Hoeffding trees with an implementation that is ready to be used in industrial environments.

Less research has been done on regression for data streams. Notable exceptions include AMRules (Almeida et al. 2013), which is an extension of AVFDR for handling numeric target values, and FIMTDD (Ikonomovska et al. 2011), which induces model trees. In contrast to the machine learning community, the fuzzy systems community has put more emphasis on regression than on classification (Angelov 2002; Angelov et al. 2010; Lughofer 2011). In particular, FLEXFIS (Lughofer 2008) is a method for inducing Takagi–Sugeno–Kang (TSK) rules (Takagi and Sugeno 1985) from data streams.

In the following, we elaborate a bit more on those approaches that are especially relevant for our own method and the experimental study presented later on namely.

In the Adaptive Model Rules (AMRules) approach, the rule premises are represented in the form of conjunctive combinations of literals on the input variables. Moreover, the rule consequents are specified as linear functions of the variables, which are fitted to the data using least squares regression. Each rule maintains various statistics characterising the part of the instance space covered by that rule. Starting with a single literal, each rule is expanded by new literals step by step, using the Hoeffding bound as a selection criterion. A distinction between unordered rule sets and decision lists is made by Almeida et al. (2013). In this paper, the authors propose two prediction and update schemes. In the first approach, the rules are sorted in the order in which they have been learned. For prediction, only the first rule that is activated by an example is used. In the second approach, the rules are treated as a set, and their predictions are aggregated after an inversely proportional weighting to the loss of their corresponding rules. Moreover, all rules activated by an example are updated. Since a better performance was achieved for the second approach, the authors used that one in their study.

Fast Incremental Model Trees with Drift Detection (FIMTDD) is a method for learning model trees for regression. To determine splits of the model tree, candidate attributes are assessed according to how much they they help to reduce the variance of the target variable. Moreover, a linear function on a corresponding subspace is specified for each leaf of the induced tree, and learning these functions is accomplished using stochastic gradient descent. An ensemble of FIMTDD trees was introduced by Ikonomovska et al. (2015) after equipping each tree with the option mechanism, i.e., the standard single split approach is replaced by a multi-split ability to avoid long waiting times before the tie-breaking takes place. Another ensemble version of FIMTDD (adaptive random forest, ARF-Reg) was proposed by Gomes et al. (2018), using an online version of bagging for creating the ensemble members (Oza and Russell 2001).

The Flexible Fuzzy Inference Systems (FLEXFIS) approach by Lughofer (2008) is a method for learning fuzzy rules, or, more specifically, Takagi–Sugeno–Kang (TSK) rules, on data streams. This type of rule will be formally introduced in Sect. 4.1. In contrast to Boolean rules, fuzzy rules are of a gradual nature and can cover an instance to a certain degree, which in turn allows for modulating the influence of a rule on a prediction in a more fine-granular way. In FLEXFIS, the fuzzy support of a rule, i.e., the region it covers in the input space, is determined by (incrementally) clustering the training data and associating each rule with a cluster. Rule consequents are specified in terms of linear functions of the input variables, and the estimation of these functions is successively adapted through recursive weighted least squares (RWLS) (Ljung 1999). While both fuzzy (FLEXFIS) and no-fuzzy (AMRules) methods are capable of performing a weighted aggregation, in the latter case weighting is oblivious to which degree a rule covers a data sample.

The main motivation of our approach is to take advantage of the effectivity and efficiency and algorithmic techniques for rule learning as implemented by methods such as AMRules, and to combine them with the expressiveness of fuzzy rules as used in approaches like FLEXFIS and eTS+ (Angelov 2010) as well as related formalisms such as fuzzy pattern trees (Shaker et al. 2013).

In the following, we provide a more systematic exposition and categorize the learning algorithms discussed above according to several properties. Along the way, we highlight potential advantages of combining different algorithms and their features.

2.2 Trees versus rules

Most tree and rule induction methods are based on refining rules in a general-to-specific manner, i.e., they share the property of moving from general to more specific hypotheses. In FIMTDD, for example, leaf nodes are split into more specific leaf nodes. Likewise, in AMRules and TSK-Streams, rules are specialized by adding terms to the premise part.

Trees can be seen as rule sets with a specific structure. Thus, while a direct transformation from a tree to a set of rules can usually be done in a straight-forward manner, the other direction is not always possible. In AMRules, for example, some of the rules are removed upon detection of a concept change, which makes it impossible to map the current rules to an equivalent tree-model.

FLEXFIS and eTS+ do not follow the aforementioned general-to-specific induction scheme. Instead, they learn and maintain rules in the form of clusters directly in the instance space. In general, these rules cannot be represented in terms of an equivalent tree structure.

2.3 Binary versus gradual membership

The application of fuzzy logic in decision tree and rule learning leads to two important distinctions from conventional learning. First, hard conditions (in rule antecedents) are replaced by soft conditions, so that an example can satisfy a condition to a certain degree. Therefore, in a tree structure, an instance can be propagated to different sibling nodes/leaves simultaneously, perhaps with different weights. Likewise, in a system of rules, it can be covered by multiple rules with different membership degrees.

The second difference is the ability to aggregate the decisions made by different rules in a weighted manner, as done by TSK-Streams, FLEXFIS, and eTS+, instead of merely computing an unweighted average of the outputs of all rules covering an instance. Thus, more weight can be given to the more relevant and less to the less relevant rules.

Likewise, gradual membership allows for more general inference in the case of tree-structures. While decision and model trees restrict tree traversal to a single branch from the root to a leave node, an equivalent fuzzy model tree[1] would follow several such paths simultaneously, branching an instance at an inner node in a weighted manner depending on how much it agrees with the conditions associated with each branch.

2.4 Discretization

Discretization is usually needed to create a finite number of candidate values for splitting points (thresholds) in the case of continuous features; these splitting points are then validated using a splitting criterion to decide how a tree/rule should be extended.

Both AMRules and FIMTDD apply a supervised discretization technique that is tailored to each rule and leaf node; this is achieved by considering the target values of all instances that reached a given leaf node or are covered by a rule.

TSK-Streams, as we will explain later, applies a supervised discretization technique for the creation of fuzzy sets that are evaluated for future extensions.

2.5 Splitting criteria

As already said, refining a model normally means extending a rule with additional conditions, thereby splitting it into two more specific rules or shrinking the region covered by that rule. A splitting criterion is used to find the presumably best among the (typically large) set of candidate splits. To quantify the usefulness of a split, different measures are conceivable.

A splitting criterion employed by many method, including AMRules, is *variance reduction*: For the rule R, the instances N covered by that rule are split into two groups N_1 and N_2 based on an attribute x_j and a threshold v, i.e.,

$$N_1 = \{(\boldsymbol{x}, y) \in N \,|\, x_j \leq v\}\,,$$
$$N_2 = \{(\boldsymbol{x}, y) \in N \,|\, x_j > v\}\,.$$

The sets N_1 and N_2 then specify new rules R_1 and R_2, respectively. Both x_j and v are chosen so as to achieve a maximal reduction of variance

$$Var(N) - \left(\frac{|N_1|}{|N|} Var(N_1) + \frac{|N_2|}{|N|} Var(N_2) \right) \,, \tag{2}$$

where $Var(N)$ is the variance of the target attribute (the y-values) of the instances in N.

[1] The authors are not aware of any fuzzy model tree induction method for regression on data streams.

Variance reduction has its roots in the earliest decision tree induction methods, in which splits are chosen that decrease the impurity of leaf nodes. For categorical target attributes, this is usually put in practice by reducing the information entropy. In the case of classification, the majority class is then used for prediction at a leaf node. In regression, where the target attribute is numerical, averaging is a more reasonable aggregation strategy; it was already adopted by the first regression tree learner CART (Breiman et al. 1984). With the aim of minimizing the sum of squared errors, variance reduction becomes the right splitting criterion, since the sum of weighted variances [the second part of (2)] can be written as the sum of squared errors:

$$\begin{aligned}\frac{|N_1|}{|N|}Var(N_1) + \frac{|N_2|}{|N|}Var(N_2) &= \frac{1}{|N|}\sum_{(\boldsymbol{x}_i, y_i)\in N_1}\big(y_i - \bar{y}(N_1)\big)^2 \\ &\quad + \frac{1}{|N|}\sum_{(\boldsymbol{x}_i, y_i)\in N_2}\big(y_i - \bar{y}(N_2)\big)^2 \ , \end{aligned} \tag{3}$$

where $\bar{y}(N_l) = \frac{1}{|N_l|}\sum_{(\boldsymbol{x}_i, y_i)\in N_l} y_i$ is the (constant) prediction produced by the rule R_l.

M5 (Quinlan 1992), one of the most popular regression approaches, is a tree that is similar to regression trees with the exception of learning a linear function in the leaf nodes, instead of predicting a constant (the average in CART), while employing variance reduction as a splitting criterion. FIMTDD extends M5 for learning model trees from data streams; it also applies variance reduction as a splitting criterion.

Despite the popularity of variance reduction, it has been criticized by Karalič (1992) as “not an appropriate measure for impurity of an example set since example sets with large variance and very low impurity can arise”. Similarly, a set of data points might be perfectly located on a hyperplane, non-orthogonal to the target axis, and still have a high variance.

FLEXFIS and eTS+ do not apply a splitting criterion directly, but utilize an extension mechanism that decides when to add rules to the current rule set. More specifically, FLEXFIS applies an incremental clustering method, namely an incremental version of vector quantization (Gray 1984), such that a new example forms a new cluster if its distance to the nearest cluster is larger than the “vigilance” parameter. This parameter controls the tradeoff between major structural changes (creating a new cluster) and minor adaptations of the current structure. Likewise, eTS+ utilizes a density-based incremental clustering, eClusteting+ (Angelov 2004). In both approaches, the clusters found are eventually transformed into rules.

Finally, we mention that most of the presented approaches consider only a single attribute for splitting, which leads to axis-parallel splits, not only in the standard case (FIMTDD and AMRules) but also in the case of fuzzy methods. FLEXFIS and eTS+ constitute an exception, since they find multivariate Gaussian clusters with non-diagonal covariance matrices.

2.6 Statistical tests versus engineered parameters

Learning on data streams, including the choice of the next split, must be done in an online manner. To answer the question whether or not an additional split is required, i.e., whether or not a significant improvement can be achieved through a split, statistical tests can be applied. A statistical test based on the Hoeffding bound has been extensively used by recent machine learning approaches for classification and regression, including Hoeffding trees, FIMTDD, AMRules, and TSK-Streams.

Instead of applying statistical tests, FLEXFIS and eTS+ make use of more engineered solutions, such as creating a new rule whenever an example is distant from all existing rules, as in FLEXFIS, or when adding an example reduces the density of existing ones, as in eTS+.

3 The learning algorithm TSK-Streams

TSK-Streams is an incremental, adaptive algorithm for learning rule-based regression models in a streaming mode. More specifically, TSK-Streams produces a widely used type of fuzzy rule system called Takagi–Sugeno–Kang (TSK) (Takagi and Sugeno 1985).

3.1 Basic concepts from fuzzy logic

Let us recall that the notion of a *fuzzy set* generalizes the conventional concept of a set in the sense of allowing for partial membership, which means that an element can belong to a set *to a certain degree* (Zadeh 1965). More formally, a fuzzy subset A of a reference set X is characterized in terms of a so-called *membership function* $\mu_A : X \longrightarrow \mathbb{M}$, where $\mathbb{M}$ is a totally or partially ordered set of membership degrees, typically the unit interval [0, 1]. This function can be considered as a generalization of the characteristic function of a set, which is restricted to the membership degrees 0 (no membership) and 1 (full membership). In general, the membership degree $\mu_A(x)$ can be interpreted as the truth degree of the proposition that x is an element of A.

Indeed, like in the classical case, there is a close relationship between set theory and logic. For example, generalized logical operators are used to define generalizations of set-theoretical operations such as intersection and union. A triangular norm (t-norm) is a binary operator $\top : [0, 1]^2 \longrightarrow [0, 1]$, which is commutative, associative, non-decreasing in both arguments, and with neutral element 1 and absorbing element 0 (Klement et al. 2000). Commonly used examples include the minimum, the product, and the Lukasiewicz t-norm $\top(u, v) = \max\{u + v - 1, 0\}$. A t-norm serves as a generalized logical conjunction, and as such, is also used to define the intersection of fuzzy sets: If A and B are fuzzy subsets of X with membership functions μ_A and μ_B, respectively, then the intersection $C = A \cap B$ is characterized by the membership function $\mu_C(x) = \top(\mu_A(x), \mu_B(x))$ for all $x \in X$. Note that, due to its associativity and commutativity, a t-norm can be generalized from a binary operation to a conjunction of any number of elements in a canonical way; we shall write $\top(u_1, \ldots, u_n)$ for the combination of membership degrees $u_1, \ldots, u_n$.

With every t-norm $\top$, one can associate a t-conorm $\perp$ given by $\perp(u, v) = 1 - \top(1-u, 1-v)$. The latter plays the role of a generalized disjunction and can be used, for example, to define the union of fuzzy sets. The t-conorms obtained for the minimum, the product, and the Lukasiewicz t-norm are given by the maximum, the algebraic sum $\perp(u, v) = u + v - uv$ and the t-conorm $\perp(u, v) = \min\{u + v, 1\}$, respectively.

3.2 TSK fuzzy systems

A TSK rule R_i has the following structure:

$$\begin{array}{ll} \text{IF} & (x_1 \text{ IS } A_{i,1}) \text{ AND } \ldots \text{ AND } (x_d \text{ IS } A_{i,d}) \\ \text{THEN} & l_i(\boldsymbol{x}) = w_{i,0} + w_{i,1}x_1 + \ldots + w_{i,d}x_d \ , \end{array} \tag{4}$$

where $(x_1, \ldots, x_d)^\top$ is the feature representation of an instance $\boldsymbol{x} \in \mathbb{R}^d$ and $A_{i,j}$ defines the jth antecedent of R_i in terms of a soft constraint (modeled by a fuzzy set). The consequent part of the rule is specified by the vector $\boldsymbol{\omega} = (w_{i,0}, \ldots, w_{i,d})^\top \in \mathbb{R}^{d+1}$, which defines an affine function of the input features. In what follows, we denote a rule by $R_i = (M_i, \boldsymbol{\omega}_i)$, with M_i the fuzzy sets defining the rule antecedents, and $\boldsymbol{\omega}_i$ the coefficients specifying the linear function.

The soft constraint $A_{i,j}$ is modeled in terms of a fuzzy set with membership function $\mu_j^{(i)} : \mathbb{R} \longrightarrow [0, 1]$. Thus, the predicate $(x_j \text{ IS } A_{i,j})$ has truth degree $\mu_j^{(i)}(x_j)$, which corresponds to the membership degree of x_j in $\mu_j^{(i)}$. The overall degree to which an instance $\boldsymbol{x}$ satisfies the rule premise R_i is

$$\mu_i(\boldsymbol{x}) = \top\left(\mu_1^{(i)}(x_1), \ldots, \mu_d^{(i)}(x_d)\right), \tag{5}$$

where the triangular norm $\top$ models the logical conjunction. We will adopt the Gödel t-norm, which is given by $\top(u, v) = \min(u, v)$. Notice that $A_{i,j}$ might be a void constraint, which corresponds to setting $\mu_j^{(i)} = \mu_{\text{void}} \equiv 1$;

in that case, the feature x_j is effectively removed from the premise of the rule (4).

Now, given an instance $\boldsymbol{x}$ as an input to a TSK system with C rules $RS = \{R_1, \ldots, R_C\}$, each of these rules will be "activated" with the degree (5). Therefore, the system's output is specified by the weighted average of the outputs suggested by the individual rules (see Fig. 1 for an illustration):

$$\hat{y} = \sum_{i=1}^{C} \Psi_i(\boldsymbol{x}) \cdot l_i(\boldsymbol{x}) \ , \tag{6}$$

with

$$\Psi_i(\boldsymbol{x}) = \frac{\mu_i(\boldsymbol{x})}{\sum_{j=1}^{C} \mu_j(\boldsymbol{x})} \ . \tag{7}$$

Fuzzy sets can have membership functions with different shapes and properties (Pedrycz and Gomide 1998). In our approach, we employ the family of the "S-shaped" parametrized functions: a fuzzy set μ has a support and core $[a, d]$ and $[b, c]$, respectively, such that $[a, d] \supset [b, c]$, the degree of membership is 1 inside $[b, c]$ and 0 outside $[a, b]$. The left boundary of the fuzzy set μ is modeled in terms of an "S-shaped" transition between zero and full membership:

$$\mu_S(x) = \begin{cases} 0 & \text{if } x < a \\ 2\left(\frac{x-a}{b-a}\right)^2 & \text{if } a \le x < \frac{a+b}{2} \\ 1 - 2\left(\frac{x-b}{b-a}\right)^2 & \text{if } \frac{a+b}{2} \le x < b \\ 1 & \text{if } b \le x \le c \\ 1 - 2\left(\frac{c-x}{d-c}\right)^2 & \text{if } c < x \le \frac{c+d}{2} \\ 2\left(\frac{d-x}{d-c}\right)^2 & \text{if } \frac{c+d}{2} < x \le d \\ 0 & \text{if } x > d \end{cases} \ . \tag{8}$$

An S-shaped membership function can also be left- or right-unbounded:

$$\mu_{\text{left-ub}}(x) = \begin{cases} 1 & \text{if } x < a \\ 1 - 2\left(\frac{a-x}{b-a}\right)^2 & \text{if } a < x \le \frac{a+b}{2} \\ 2\left(\frac{b-x}{b-a}\right)^2 & \text{if } \frac{a+b}{2} < x \le b \\ 0 & \text{if } x > b \end{cases} \ , \tag{9}$$

$$\mu_{\text{right-ub}}(x) = \begin{cases} 0 & \text{if } x < a \\ 2\left(\frac{x-a}{b-a}\right)^2 & \text{if } a \le x < \frac{a+b}{2} \\ 1 - 2\left(\frac{x-b}{b-a}\right)^2 & \text{if } \frac{a+b}{2} \le x < b \\ 1 & \text{if } b \le x \end{cases} \ . \tag{10}$$

3.3 Online rule induction

The TSK-Streams algorithm (cf. Algorithm 1 for the basic structure) begins with a single default rule and then learns rules in an incremental manner. The default rule has an empty premise for each feature (that is, the membership function μ_{void}) and covers the complete input space. Then, the algorithm continuously checks whether, for any of the rules R_i, one of its extensions could possibly improve the performance of the current fuzzy system.

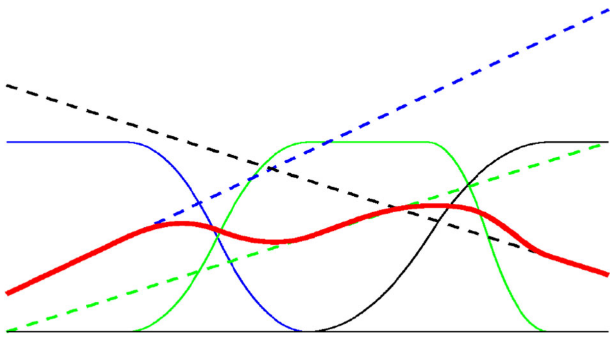

Fig. 1 Illustration of a TSK fuzzy system with three rules (in blue, green, and black, respectively) for the case of a one-dimensional instance space: Each rule is specified in terms of an S-shaped fuzzy set (rule antecedent) and an affine function (rule consequent, dashed line). The function (6) modeled by the TSK system is plotted in red color (Color figure online)

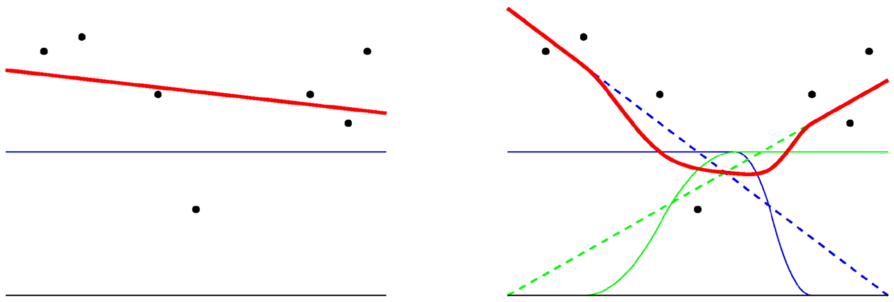

Fig. 2 Illustration of rule expansion: On the left side, the domain is covered by a single fuzzy set (μ_{void}, blue flat line), so that the TSK system produces an affine function. On the right side, the fuzzy set is split into two fuzzy sets (in blue and green, respectively), each of them giving rise to an individual rule with different consequent. The TSK system is now able to fit the data (black points) more accurately (Color figure online)

An expansion of a rule R_i with a predicate (x_j IS $A_{i,j}$) on the jth attribute means that the rule is split into two new rules R_i' and R_i'' with predicates (x_j IS $A'_{i,j}$) and (x_j IS $A''_{i,j}$), respectively, where $A_{i,j} = A'_{i,j} \cup A''_{i,j}$ with membership functions $\mu_j^{\prime(i)}$ and $\mu_j^{\prime\prime(i)}$ satisfying $\mu_j^{(i)}(x) = \bot(\mu_j^{\prime(i)}(x), \mu_j^{\prime\prime(i)}(x))$.

These membership functions are chosen after a fuzzy partitioning of the domain of feature x_j. To this end, we apply a supervised discretization technique that divides a fuzzy set into two new fuzzy sets so as to improve the overall performance (see Fig. 2 for an illustration). Here, we focus on two criteria (cf. the discussion in Sect. 2), to be detailed in the following.

These membership functions are chosen after a fuzzy partitioning of the domain of feature x_j. To this end, we apply a supervised discretization technique that divides a fuzzy set into two new fuzzy sets so as to improve the overall performance. Here, we focus on two criteria (cf. the discussion in Sect. 2), to be detailed in the following.

Algorithm 1: TSK-Streams Algorithm

```
Input: $RS = \{(R_{default}, S_{default} = \Phi)\}, Mode, \delta, \tau$
$Mode \in \{VR, ER\}$
$\delta$: confidence level
$\tau$: tie-breaking constant
1  $n = 0$
2  for $(\boldsymbol{x}_t, y_t) \in Stream$ do
3      for $(R_i, S_i) \in RS$ do
4          if $Mode == VR$ then
5              Update candidates using E-BST according to (11)–(17)
6          else
7              GenUpdateERCandidates($R_i, S_i, (\boldsymbol{x}_t, y_t)$)}
8          UpdateConsequent($RS, R_i, S_i, (\boldsymbol{x}_t, y_t)$)
9      if $Mode == VR$ then
10         ExpandSystemVR($RS, \delta, \tau, n$)
11     else
12         ExpandSystemER($RS, \delta, \tau, n$)
13     $n = n + 1$
```

3.3.1 Variance reduction

Similar to the AMRule principle of reducing the variance, based on the fuzzy set $A_{i,j}$, two fuzzy sets $A'_{i,j}$ and $A''_{i,j}$ are created such that a maximum reduction in the target attribute's variance is achieved. For example, let $A_{i,j}$ be a fuzzy set (for the jth attribute in the rule R_i) characterized by the S-shaped membership function $\mu_j^{(i)}$, which is parametrized by the quadruple (a, b, c, d). Let N_{R_i} be the set of examples $(\boldsymbol{x}, y)$ covered by the rule R_i, i.e., the examples for which $\mu_i(\boldsymbol{x}) > 0$. We then seek to find the value $q \in [a, d]$ such that the reduction in variance

$$Var(S) - \big(w' Var(S') + w'' Var(S'')\big)$$

is maximal, where

$$S = \{y \cdot \Psi_i(\boldsymbol{x}) \mid (\boldsymbol{x}, y) \in N_{R_i}\}, \tag{11}$$

$$S' = \{y \cdot \Psi_i(\boldsymbol{x}) \mid (\boldsymbol{x}, y) \in N'_{R_i}\}, \tag{12}$$

$$S'' = \{y \cdot \Psi_i(\boldsymbol{x}) \mid (\boldsymbol{x}, y) \in N''_{R_i}\}, \tag{13}$$

$$w' = \frac{\sum_{(\boldsymbol{x},y)\in N'_{R_i}} \Psi_i(\boldsymbol{x})}{\sum_{(\boldsymbol{x},y)\in N_{R_i}} \Psi_i(\boldsymbol{x})}, \tag{14}$$

$$w'' = \frac{\sum_{(\boldsymbol{x},y)\in N''_{R_i}} \Psi_i(\boldsymbol{x})}{\sum_{(\boldsymbol{x},y)\in N_{R_i}} \Psi_i(\boldsymbol{x})}, \tag{15}$$

with

$$N'_{R_i} = \{(\boldsymbol{x}, y) \mid (\boldsymbol{x}, y) \in N_{R_i},\ x_j \in [a, q]\}, \tag{16}$$
$$N''_{R_i} = \{(\boldsymbol{x}, y) \mid (\boldsymbol{x}, y) \in N_{R_i},\ x_j \in [q, d]\}, \tag{17}$$

and $Var(S)$ the variance of the set S.

Similar to AMRules and FIMTDD, we achieve the variance reduction by storing candidate values in an extended binary search tree (E-BST). This data structure allows for computing the variance reduction for each candidate value in time that is linear in the size of the tree; moreover, it can be updated in logarithmic time (Ikonomovska 2012). E-BST is an extended binary search tree that stores sufficient statistics to evaluate the variance reduction resulting from the split at that node. Each node in E-BST contains *(i)* the test value at that node, *(ii)* the number of samples $|\{(\boldsymbol{x}_i, y_i)\}|$ reaching that node, the sum of their target attributes ($\sum y_i$), and the sum of the squared target attributes ($\sum y_i^2$), and *(iii)* the same statistics for the samples reaching the right child node of the current node.

3.3.2 Error reduction

Extending the current model with new rules so as to improve the system's overall performance requires, for each existing rule, the creation and evaluation of all possible extensions—evaluating an extension here means determining the empirical performance of the (modified) system as a whole. As before, by a possible rule extension we mean replacing a fuzzy set $A_{i,j}$ in a rule antecedent by new fuzzy sets $A'_{i,j}$ and $A''_{i,j}$, which are produced by bisecting the support of $A_{i,j}$ at some suitable splitting point. Even if these splitting points were organized in a binary search tree structure, the number of updates required after observing a new example would no longer be logarithmic but linear. Indeed, every possible extension means fitting a step-wise linear function, at each splitting value, on the entire training data (or updating the linear function on the new data instance).

To counter the aforementioned problem, we suggest a heuristic that simultaneously chooses a promising splitting value and fits a stepwise linear function for each candidate extension rule. The splitting value is chosen by adaptively shifting (increasing or decreasing) it based on the performance of new candidate rules. More formally, let $A_{i,j}$ be a fuzzy set characterized by the S-shaped membership function $\mu_j^{(i)}$, parametrized by (a, b, c, d), and let N_{R_i} be the set of instances $(\boldsymbol{x}, y)$ covered by the rule R_i. Let $q \in [a, d]$ be the initial splitting value from which $A'_{i,j}$ and $A''_{i,j}$ are constructed via suitable parametrizations $(a, b, q + \rho_1, q + \rho_2)$ and $(q - \rho_2, q - \rho_1, c, d)$ of their membership functions $\mu_j^{\prime(i)}$ and $\mu_j^{\prime\prime(i)}$, respectively. We initialize q by the current mean of the observed values x_j. The values ρ_1 and ρ_2 control the steepness of the S-shaped function and are chosen in proportion to the observed variance. From the membership functions $\mu_j^{\prime(i)}$ and $\mu_j^{\prime\prime(i)}$ and the parent rule R_i, the new candidate rules R'_i and R''_i are created (see lines 1–9 of Algorithm 2).

Upon observing a new example $(\boldsymbol{x}, y)$, both the membership degrees $\mu_j^{\prime(i)}(\boldsymbol{x})$, $\mu_j^{\prime\prime(i)}(\boldsymbol{x})$ and the errors committed by each candidate rule, $err_1 = (\boldsymbol{\omega}^{\prime(i)} \cdot \boldsymbol{x} - y)^2$, $err_2 = (\boldsymbol{\omega}^{\prime\prime(i)} \cdot \boldsymbol{x} - y)^2$, are computed. If the "winner rule", i.e., the candidate rule by which the example is covered the most, commits an error that is larger than the error committed by the other candidate rule (covering the example to a lesser degree), we consider this as an inconsistency. The latter can be mitigated by shifting the splitting value q right or left, in proportion to the error committed by each candidate extension (see lines 11–21 of Algorithm 2).

Algorithm 2: GenUpdateERCandidates – ErrorReduction

```
Input: $R_i, S_i, (\boldsymbol{x}_t, y_t)$:
$R_i = (M_i, \boldsymbol{\omega}_i)$: the rule whose extensions should be created/updated
$M_i$: the set of fuzzy sets conjugated in the premise.
$\boldsymbol{\omega}_i$: the vector of coefficients of the linear function.
$S_i = \{(R_i', R_i'')\}$: Set of candidate extensions of rule $R_i$.
$(\boldsymbol{x}_t, y_t)$: a new training example.
1  if $S_i$ is Empty then
2      for $j \in \{1, \ldots, d\}$ do
3          Update $Mean(x_{tj}), Var(x_{tj})$
4          $M_i' = \{\mu_j^{\prime(i)} = (a, b, q + \rho_1, q + \rho_2)\} \cup M_i \setminus \{\mu_j^{(i)}\}$
5          $M_i'' = \{\mu_j^{\prime\prime(i)} = (q - \rho_2, q - \rho_1, c, d)\} \cup M_i \setminus \{\mu_j^{(i)}\}$
6          $\boldsymbol{\omega}_i' = \boldsymbol{\omega}_i, \quad \boldsymbol{\omega}_i'' = \boldsymbol{\omega}_i$
7          $R_i' = (M_i', \boldsymbol{\omega}_i'), \quad R_i'' = (M_i'', \boldsymbol{\omega}_i'')$
8          $S_i = S_i \cup \{s_j = (R_i', R_i'')\}$
9  else
10     for $j \in \{1, \ldots, d\}$ do
11         Find $s_j = (R_i', R_i'') \in S_i$ s.t.
           $R_i' = (M_i', \boldsymbol{\omega}_i'), R_i'' = (M_i'', \boldsymbol{\omega}_i'') \wedge M_i \setminus \{M_i'\} = M_i \setminus \{M_i''\} = \{\mu_j^{(i)}\}$
12         $m_1 = \mu_j^{\prime(i)}(x_{tj}), \quad error_1 = (\boldsymbol{\omega}_i' \cdot \boldsymbol{x}_t - y_t)^2$
13         $m_2 = \mu_j^{\prime\prime(i)}(x_{tj}), \quad error_2 = (\boldsymbol{\omega}_i'' \cdot \boldsymbol{x}_t - y_t)^2$
14         if $m_1 > m_2 \wedge error_1 > error_2$ then
               /* shift $q$ to the left */
15             $q = q - \eta \Psi_i(\boldsymbol{x}_t)(error_1 - error_2)$
16         else if $m_1 < m_2 \wedge error_1 < error_2$ then
               /* shift $q$ to the right */
17             $q = q - \eta \Psi_i(\boldsymbol{x}_t)(error_1 - error_2)$
18         Update $\mu_j^{\prime(i)} = (a, b, q + \rho_1, q + \rho_2)\}$
19         Update $\mu_j^{\prime\prime(i)} = (q - \rho_2, q - \rho_1, c, d)\}$
       /* Update $\boldsymbol{\omega}_i', \boldsymbol{\omega}_i''$, Algorithm 3 */
20     UpdateConsequent($R_i, S_i$)
```

In the explanations above, we outlined two ways of splitting an S-shaped function into two such functions of similar shape. In the beginning, however, the default rule contains only unbounded fuzzy sets characterized by μ_{void}. A split of an unbounded fuzzy set produces two sets with membership functions $\mu_{\text{left-ub}}(x)$ and $\mu_{\text{right-ub}}(x)$, respectively, which cover the resulting half spaces (with some degree of overlap). Similarly, a split of a right- or left-unbounded membership function leads to a right- or left-unbounded and an S-shaped function.

Recall that AMRules adopts only a single rule from the two candidates emerging from a rule expansion (cf. Sect. 2.2). More specifically, AMRules keeps the rule with minimum weighted variance and discards the other candidate as well as the parent rule from the original rule set. Since the resulting rule set does not form a partition of the instance space, this strategy requires a default rule covering the space that is not covered by any other rule. Motivated by this strategy, we also study the effect of adopting only a single instead of both rule extensions. Thus, we distinguish the following two strategies.

1. Single Extension: Only the best extension is added to the rule set, while the other one is discarded. The parent rule is also discarded unless it is the default rule. The choice of the best rule depends on the criterion used for splitting: either the weighted variance reduction or the weighted SSE.
2. All Extensions: Both extensions are added to the rule set, and the parent rule is removed. This approach makes the whole system of rules equivalent to a tree structure.

The two adaptation strategies will be revisited in the context of change detection in Sect. 3.6. A more detailed exposition of the adaptation strategies is given in Algorithms 5 and 4.

3.4 Rule consequents

FLEXFIS makes use of recursive weighted least squares estimation (RWLS) (Ljung 1999) to fit linear functions as rule consequents. This approach is computationally expensive, as it requires multiple matrix inversions. In our approach, and similar to AMRules, we learn consequents more efficiently using gradient methods.

When a new training instance $(\boldsymbol{x}_t, y_t)$ arrives, TSK-Streams produces a prediction $\hat{y}_t$, the squared error of which can be obtained as follows:

$$E_t = (y_t - \hat{y}_t)^2 \tag{18}$$

$$= \left(y_t - \left(\sum_{R_i \in RS} \frac{\Psi_i(\boldsymbol{x}_t)}{\sum_{R_k \in RS} \Psi_k(\boldsymbol{x}_t)} \sum_{j=0}^{d} \omega_{i,j} x_{t,j} \right) \right)^2 , \tag{19}$$

where RS is the current set of rules. According to the technique of stochastic gradient descent, the coefficients $\omega_{i,j}$ are then moved into the negative direction of the gradient, with the length of the shift being controlled by the learning rate η:

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta \nabla E_t \ . \tag{20}$$

Thus, the following (component-wise) update rule is obtained:

$$\omega_{i,j} \leftarrow \omega_{i,j} - 2\,\eta(y_t - \hat{y}_t)\left(\sum_{R_i \in RS} \frac{\Psi_i(\boldsymbol{x}_t)}{\sum_{R_k \in RS} \Psi_k(\boldsymbol{x}_t)}\, x_{t,j}\right)$$

The process of updating the rule consequents is summarized in Algorithm 3, which also updates the consequents of the rule's extension (when the error reduction strategy is used).

Algorithm 3: UpdateConsequent

```
Input: $RS = \{(R, S)\}, R_i, S_i, (\boldsymbol{x}_t, y_t)$
$RS = \{(R, S)\}$: the set of all rules and their extensions
$R_i = (M_i, \boldsymbol{\omega}_i)$: the rule whose consequent and the consequents of its extensions should be updated
$M_i$: the set of fuzzy sets $\mu'_1, \ldots, \mu'_d$ defining the rule premise
$\boldsymbol{\omega}_i$: the vector of coefficients of the linear function
$S_i = \{(R'_i, R''_i)\}$: Set of candidate extensions of rule $R_i$
$(\boldsymbol{x}_t, y_t)$: a new training example
1  $m_1 = \sum_{R_j \in RS} \mu_j(\boldsymbol{x}_t)$
2  $m_2 = \sum_{R_j \in RS} \mu_j(\boldsymbol{x}_t) l_j(\boldsymbol{x}_t)$
3  $\mu_i(\boldsymbol{x}_t) = \top(\mu_1^{(i)}(x_{t,1}), \ldots, \mu_d^{(i)}(x_{t,d}))$
4  if $\mu_i(\boldsymbol{x}_t) > 0$ then
5      for $(R'_i, R''_i) \in S_i$ do
6          $\mu'_i(\boldsymbol{x}_t) = \top(\mu_1'^{(i)}(x_{t,1}), \ldots, \mu_d'^{(i)}(x_{t,d}))$
7          $\mu''_i(\boldsymbol{x}_t) = \top(\mu_1''^{(i)}(x_{t,1}), \ldots, \mu_d''^{(i)}(x_{t,d}))$
8          $m_1{}' = m_1 - \mu_i(\boldsymbol{x}_t) + \mu'_i(\boldsymbol{x}_t) + \mu''_i(\boldsymbol{x}_t)$
9          $m_2{}' = m_2 - \mu_i(\boldsymbol{x}_t) l_i(\boldsymbol{x}_t) + \mu'_i(\boldsymbol{x}_t) l'_i(\boldsymbol{x}_t) + \mu''_i(\boldsymbol{x}_t) l''_i(\boldsymbol{x}_t)$
10         $\boldsymbol{\omega}'_i = \boldsymbol{\omega}'_i + \eta(y_t - \frac{m_2{}'}{m_1{}'})\left(\frac{\mu'_i(\boldsymbol{x}_t)}{m_1{}'}\boldsymbol{x}_t\right)$
11         $\boldsymbol{\omega}''_i = \boldsymbol{\omega}''_i + \eta(y_t - \frac{m_2{}'}{m_1{}'})\left(\frac{\mu''_i(\boldsymbol{x}_t)}{m_1{}'}\boldsymbol{x}_t\right)$
12     $\boldsymbol{\omega}_i = \boldsymbol{\omega}_i + \eta(y_t - \frac{m_2}{m_1})\left(\frac{\mu_i(\boldsymbol{x}_t)}{m_1}\boldsymbol{x}_t\right)$
```

3.5 Model structure

TSK-Streams adapts the TSK rule system (that is, the fuzzy sets in the rule antecedents and the linear function in the consequents) in a continuous manner. While the adaptations discussed so far essentially concern the parameters of the system, the replacement of a rule by one of its expansions corresponds to a (more substantial) structural change.

For obvious reasons, such changes should be handled with caution, especially when they lead to an increased complexity of the model. Learning methods therefore tend to maintain the current model unless being sufficiently convinced that an expansion will yield an improvement. To decide whether or not a possible expansion should be adopted, the estimated performance difference is typically taken as a criterion: this difference should be *significant* in a statistical sense.

In our algorithm, we make use of Hoeffding's inequality to support these decisions. The latter bounds the difference between the empirical mean $\bar{X}$ of the n i.i.d. random variables $X_1, \ldots, X_n$ (having support $[a, b] \subset \mathbb{R}$) and the expectation $E(\overline{X})$ in terms of

$$P\Big(|\bar{X} - \mathrm{E}(X)| > \epsilon\Big) \le \exp\left(-\frac{2n\epsilon^2}{(b-a)^2}\right) \ . \tag{21}$$

More specifically, when using the error reduction criterion, we replace a rule R_i by two rules R_i' and R_i'', considering the reduction in the sum of squared errors (SSE). That is, the SSE of the current rule set RS is compared with the SSE of all alternative systems $(RS \setminus R_i) \cup \{R_i', R_i''\}$. With SSE_{best} and $SSE_{2ndbest}$ denoting the expansion with the lowest and the second lowest error, respectively, the best expansion is adopted if

$$\frac{SSE_{best}}{SSE_{2ndbest}} < 1 - \epsilon \ , \tag{22}$$

or when ϵ falls below a tie-breaking constant τ. The constant ϵ is obtained from (21) by setting the probability to a desired degree of confidence $1-\delta$, i.e., setting the right-hand side to $1 - \delta$ and solving for ϵ; noting that the ratio (22) is bounded in $]0, 1]$, $b - a$ is set to 1. Algorithm 4 depicts the system expansion procedure when the error reduction criterion is applied. The same technique can be used for the single extension variant, except that the rule R_i is replaced with the extension that achieves the lowest weighted SSE (provided R_i is not the default rule, otherwise R_i is also kept).

As an alternative to the global error reduction criterion, the variance reduction approach checks for the decrease in variance for each rule locally. The Hoeffding inequality is then applied to the ratio of the variance reductions of the best two candidate extensions of the same rule R_i. The procedure that performs the expansion is depicted in Algorithm 5. This strategy can be seen as a model adaptation through local improvements.

Overfitting is a potential problem when a tree- or rule-based system is extended merely based on the measured improvement in the training loss. To circumvent overfitting, pruning is often applied. In our approach, we propose a penalization mechanism to avoid the danger of overfitting due to an excessive increase in the number of rules. This mechanism consists of adding a complexity term C to ϵ. For both extensions (variance reduction and error reduction), C is set to $d^{-2}\sqrt{|RS|}$, with d the number of features and RS the current rule set. Again, we refer to Algorithm 1 for an overview of the TSK-Streams algorithm (for both alternatives, variance and error reduction).

3.6 Change detection

A concept drift may cause a drop in the performance of a rule. To detect such cases, we make use of the adaptive windowing (ADWIN) (Bifet and Gavaldà 2007) drift

Algorithm 4: ExpandSystemER – ErrorReduction

```
Input: $RS = \{(R, S)\}, \delta, \tau, n$
$RS = \{(R, S)\} = \bigcup_{R_i} \{(R_i, S_i)\}$: rules and extensions
    $= \bigcup_{R_i} \{(R_i, \bigcup_{j=1}^{d} \{s_j = (R'_i, R''_i, SSE_{ij})\}))\}$
$SSE_{ij}$: the sum of squared errors committed by the extension $j$ of rule $R_i$
$\delta$: confidence level
$\tau$: tie-breaking constant
$n$: number of examples seen by the current system
$(\boldsymbol{x}_t, y_t)$: a new training example.
1  let $SSE_{current}$ be the SSE of the current system
2  let $s_{pq}$ be the extension with smallest SSE
3  let $s_{pq}$ be the extension with second smallest SSE
4  Update $SSE_{current}$, $s_{pq}$ and $s_{pq}$ on $(\boldsymbol{x}_t, y_t)$
5  $\epsilon = \sqrt{\frac{\ln\left(\frac{1}{\delta}\right)(R)^2}{2n}}$ + complexity
6  $\overline{X} = \frac{1}{n}(SSE_{pq}/SSE_{uv})$
7  $\overline{Y} = \frac{1}{n}(SSE_{pq}/SSE_{current})$
8  if $((\overline{Y} + \epsilon) < 1)$ AND $((\overline{X} + \epsilon) < 1$ OR $\epsilon < \tau)$ then
9      if Single Extension then
10         let $R_{best} \in \{R'_p, R''_p\}$ has the smallest weighted SSE
11         $RS = RS \cup \{(R_{best}, GenUpdateERCandidates(R_{best}, \emptyset, (\boldsymbol{x}_t, y_t))\}$
12         if $R_p$ is not $R_{default}$ then
13             $RS = RS \setminus \{(R_p, S_p)\}$
14     else
15         $RS = RS \cup \{(R'_p, GenUpdateERCandidates(R'_p, \emptyset, (\boldsymbol{x}_t, y_t)),$
16             $(R''_p, GenUpdateERCandidates(R''_p, \emptyset, (\boldsymbol{x}_t, y_t))\}$
17         $RS = RS \setminus \{(R_p, S_p)\}$
```

detector. Compared to the Page–Hinkely test (PH) (Page 1954), which is used by AMRules, ADWIN has the advantage of being non-parametric, which means that it makes no assumptions about the observed random variable. Besides, only a single parameter needs to be chosen, namely the tolerance towards false alarms (δ_{adwin}). In our approach, ADWIN is locally applied in each rule. More specifically, given that an example is covered by a rule, it is applied on the absolute error committed by that rule on this example.

For the single extension strategy, the rule that suffers from a drop of performance can be simply discarded. But in the all extensions strategy and upon detecting a drift in the rule $R_p = (M_p, \boldsymbol{\omega}_p)$, we find its sibling rule $R_q = (M_q, \boldsymbol{\omega}_q)$, from which it differs by only one single literal (i.e., there is a fuzzy set $\mu_j^{(p)} \in M_p$ on the jth attribute that satisfies the following criterion: for all $i \in \{1, \ldots, d\} \setminus \{j\} : \mu_i^{(p)} \in M_q \wedge \mu_i^{(q)} \in M_p$). To remove the rule R_p, it is retracted from the rule set, and its sibling rule R_q is updated by replacing $\mu_j^{(q)}$ with $\mu_j^{(p)} \cup \mu_j^{(q)}$. In case the sibling rule R_q has already been extended before the drift is detected, the same procedure is applied recursively to the children of this rule.

Algorithm 5: ExpandSystemVR – VarianceReduction

```
   Input: $RS = \{(R, S)\}, \delta, \tau, n$
   $RS = \{(R, S)\} = \bigcup_{R_i}\{(R_i, S_i)\}$: rules and extensions
       $= \bigcup_{R_i}\{(R_i, \bigcup_{j=1}^{d}\{s_j = (R'_i, R''_i, VarRed_{ij})\}))\}$
   $VarRed_{ij}$: variance reduction caused by the extension $j$ of rule $R_i$
   $\delta$: confidence level
   $\tau$: tie-breaking constant
   $n$: number of examples seen by the current system
   $(\boldsymbol{x}_t, y_t)$: a new training example.
1  for $(R_i, S_i) \in RS$ do
2      let $s_{best} = (R'_i, R''_i, VarRed_{best}) \in S_i$ has the largest VarRed
3      let $s_{2ndbest} = (R'_i, R''_i, Red_{2ndbest}) \in S$ has the 2nd largest VarRed
4      $\epsilon = \sqrt{\frac{\ln\left(\frac{1}{\delta}\right)(R)^2}{2n}}$ + complexity
5      $\overline{X} = \frac{VarRed_{2ndbest}}{VarRed_{best}}$
6      if $((\overline{X} + \epsilon) < 1 \ OR \ \epsilon < \tau)$ then
7          if Single Extension then
8              let $R_{best} \in \{R'_i, R''_i\}$ has the largest weighted VarRed
9              $RS = RS \cup \{(R_{best}, GenerateExtendedRules(R_{best})\}$
10             if $R_i$ is not $R_{default}$ then
11                 $RS = RS \setminus \{(R_i, S_i)\}$
12         else
13             $RS = RS \cup \{(R'_i, GenerateExtendedRules(R'_i),$
14                 $(R''_i, GenerateExtendedRules(R''_i)\}$
15             $RS = RS \setminus \{(R_i, S_i)\}$
```

4 Empirical evaluation

To compare our method TSK-streams with existing algorithms, we conducted a series of experiments, in which we investigated the algorithms' predictive accuracy, their runtime, and the size of the models they produce.

4.1 Methods, data, and experimental setup

TSK-Streams is implemented in MOA[2] (Massive Online Analysis), which is an open source software framework for mining and analyzing large data sets in a streaming mode (Bifet et al. 2010). In our experiments, TSK-Streams is compared with AMRules, FIMTDD, ARF-Reg, and FLEXFIS. Both AMRules and FIMTDD are implemented in MOA's distribution. We implement ARF-Reg as described in the original paper (Gomes et al. 2018). FLEXFIS is implemented in Matlab. For the (hyper-)parametrization of all methods, we perform grid search as described in Sect. 1.

[2] http://moa.cms.waikato.ac.nz.

Table 1 Data sets

#	Name	Synthetic	Instances	Attributes
1	2dplanes	Yes	40,768	11
2	ailerons	No	13,750	41
3	bank8FM	Yes	8192	9
4	calHousing	No	20,640	8
5	elevators	No	8752	19
6	fried	Yes	40,769	11
7	house16H	No	22,784	16
8	house8L	No	22,784	8
9	kin8nm	–	8192	9
10	mvnumeric	Yes	40,768	10
11	pol	No	15,000	49
12	puma32H	Yes	8192	32
13	puma8NH	Yes	8192	9
14	ratingssweetrs	–	17,903	2
15	BNG-stock	Semi	59,049	10
16	BNG-cholesterol	Semi	100,000	14
17	BNG-echoMonths	Semi	17,496	10
18	BNG-wine-quality	Semi	100,000	14

The test-then-train protocol was used for all experiments. According to this protocol, each instance is used for both testing and training: The model is evaluated on the instance first, and a learning step is carried out afterward. Experiments are performed on benchmark data sets[3] collected from the UCI repository[4] (Dua and Graff 2019) and other repositories[5]; a summary of the type, the number of attributes, and the number of instances of each data set is given in Table 1.

The data sets starting with prefix *BNG-* are obtained from the online machine learning platform OpenML (Bischl et al. 2017); these large data streams are drawn from Bayesian networks as generative models, after constructing each network from a relatively small data set (we refer to van Rijn et al. (2014) for more details).

4.2 Results

In the first part of the evaluation, we compare the four variants of our own method: variance reduction versus error reduction, and the extension using a single candidate versus the extension for both candidates. Let us note that the combination of error reduction with the extension for both candidates essentially corresponds to the previous version by Shaker et al. (2017).

[3] The first 14 data sets are the same as those used in (Almeida et al. 2013).

[4] http://archive.ics.uci.edu/ml/.

[5] https://github.com/renatopp/arff-datasets/tree/master/regression, http://tunedit.org/repo/UCI/numeric.

Table 2 Performance of the different TSK-Streams variants in terms of RMSE (with standard deviation in brackets)

Dataset	TSK-Streams Error Red One Cand.	Rank	TSK-Streams Error Red. Both Cand.	Rank	TSK-Streams Variance Red. One Cand.	Rank	TSK-Streams Variance Red. Both Cand.	2Rank
2dplanes	1.181 (0.031)	$3^{\ominus}$	1.020 (0.001)	2	1.192 (0.042)	$4^{\ominus}$	1.019 (0.003)	1
ailerons	$1.968\times10^{-4}(10^{-6})$	$1^{\oplus}$	2.594×10^{-4} (10^{-5})	2	1.351×10^{-3} (10^{-3})	4	$4.166\times10^{-4}(10^{-4})$	3
bank8FM	3.758×10^{-2} (10^{-4})	$2^{\ominus}$	3.849×10^{-2} (10^{-4})	$4^{\ominus}$	3.805×10^{-2} (10^{-3})	$3^{\ominus}$	3.468×10^{-2} (10^{-4})	1
calhousing	70359 (334)	$2^{\ominus}$	85976 (7054)	$4^{\ominus}$	70639 (262)	$3^{\ominus}$	69025 (190)	1
elevators	0.003 (5.8×10^{-5})	$1^{\oplus}$	0.003 (3.7×10^{-5})	$2^{\oplus}$	0.005 (5.9×10^{-4})	$3^{\oplus}$	0.007 (9.9×10^{-4})	4
fried	2.283 (0.057)	3	2.253 (0.054)	2	2.448 (0.019)	$4^{\ominus}$	2.220 (0.009)	1
house16h	45927 (743)	3	45046 (579)	2	50107 (1968)	$4^{\ominus}$	44721 (771)	1
house8	41110 (1147)	$1^{\oplus}$	45190 (2448)	$2^{\oplus}$	91106 (21705)	4	69907 (8946)	3
kin8nm	0.196 (10^{-3})	$1^{\oplus}$	0.205(10^{-3})	4	0.203 (10^{-4})	$3^{\ominus}$	0.201 (10^{-4})	2
mvnumeric	1.779 (0.090)	$3^{\ominus}$	0.741 (0.052)	$1^{\oplus}$	2.347 (0.199)	$4^{\ominus}$	1.092 (0.073)	2
pol	31.276 (0.632)	$4^{\ominus}$	24.297 (0.903)	$2^{\ominus}$	24.813 (0.829)	$3^{\ominus}$	17.733 (0.136)	1
puma32H	0.0271 (1×10^{-4})	$3^{\ominus}$	0.0274 (2.5×10^{-5})	$4^{\ominus}$	0.025 (5×10^{-4})	$2^{\ominus}$	0.019 (6.9×10^{-4})	1
puma8NH	4.460 (0.036)	$3^{\ominus}$	4.561 (0.003)	$4^{\ominus}$	4.248 (0.029)	$2^{\ominus}$	3.748 (0.040)	1
ratingssw.	1.622 (0.001)	4	1.619 (0.001)	2	1.619 (0.001)	3	1.619 (0.001)	1
BNG-stock	4.057 (0.028)	$2^{\ominus}$	4.262 (0.036)	$4^{\ominus}$	4.076 (0.024)	$3^{\ominus}$	3.845 (0.014)	1
BNG-chol.	49.327 (0.039)	$3^{\ominus}$	49.582 (0.033)	$4^{\ominus}$	49.310 (0.016)	$2^{\ominus}$	49.116 (0.020)	1
BNG-echo	11.895 (0.021)	2	12.009 (0.009)	$4^{\ominus}$	11.926 (0.007)	$3^{\ominus}$	11.849 (0.009)	1
BNG-wine.	0.780 (6.4×10^{-4})	2	0.789 (10^{-4})	$4^{\ominus}$	0.784 (3.9×10^{-4})	$3^{\ominus}$	0.779 (1.7×10^{-4})	1
∅ **Rank**		2.47		2.97		2.94		1.61
Wins/Losses	4/9		3/9		1/14		N/A	

The superscripts $\ominus$ and $\oplus$ indicate statistical significance for the comparison with TSK-Streams (variance reduction with the extension for both candidates)

Table 2 shows the average RMSE (root mean squared error) and the corresponding standard error on ten rounds for each data set. In this table, the last row shows the number of statistically significant wins/losses of the first three against the fourth variant (with variance reduction and consideration of both candidates); these tests apply the Wilcoxon signed-rank test over the paired performances of the 10 iterations with confidence level $\alpha = 0.05$. From the results, the fourth variant appears to be superior to the other variants. Therefore, we adopt this variant (simply referred to as TSK-Streams in the following) and consider it for further comparison with state-of-the-art methods. One way to understand this result is to realize that keeping both candidate extensions for the variance reduction case is better than a single candidate, since the variance reduction does not only reduce the variance in comparison to the original rule, but also leads to decreasing the variance in each of the two candidate rules. The same cannot be said about the error reduction, as it is a measure of the weighted errors in each candidate. Hence, the weighted error of well and poorly performing candidate rules might still lead to reducing the error. In such a case, it would be more reasonable to consider only the best performing candidate, which explains why the single candidate is better when considering the error reduction as criteria. One evidence supporting this claim is that on average, the two candidates strategy produces a smaller number of rules (and shorter runtime) than the single candidate strategy, which means that the latter becomes sometimes slowed down (or gets stuck) with rules that are performing poorly. Tables 3 and 4 show the comparison between the different TSK-Streams variants in terms of model size and runtime, respectively.

Table 5 presents the performance comparison between TSK-Streams and the other approaches, AMRules, FIMTDD, ARF-Reg, and FLEXFIS. Overall, TSK-Streams compares quite favourably and performs best in terms of the average rank statistic. Moreover, at least on 9 of the 18 data sets, its performance is statistically better (also according to the Wilcoxon signed-rank test at significance level $\alpha = 0.05$) than that of any other approach. In spite of the limited number of data sets, the advantage over AMRules and ARF-Reg is even statistically significant; this is not the case, however, for FIMTDD and FLEXFIS (cf. Fig. 3).

Other criteria important for the applicability of an approach in the setting of data streams include model complexity and efficiency. Obviously, these properties are not independent of each other, because more time is needed to maintain and adapt larger models. We measure the two criteria, respectively, in terms of the number of rules/leaf nodes in the model eventually produced by a learning algorithm and the average time (in milliseconds) the algorithms need to process a single instance. We consider the latter more informative than the total runtime on an entire data set (stream), because the processing time per instance is more relevant for the possible application of an algorithm under real-time conditions. Table 6 shows that TSK-Streams tends to produces smaller models than FIMTDD and ARF-Reg, which are still slightly larger than those of FLEXFIS and AMRules. Table 7 shows that TSK-Streams is also slower on average. We would argue, however, that this is not important, as it is still extremely fast in terms of absolute runtime: Being able to predict and learn from each new instance in just a few milliseconds, it certainly meets the requirements for learning on data streams in common practical applications.

Table 3 Size of the learned model of the different TSK-Streams variants

Dataset	TSK-Streams Error red One cand.	Rank	TSK-Streams Error red. Both cand.	Rank	TSK-Streams Variance red. One cand.	Rank	TSK-Streams Variance red. Both cand.	Rank
2dplanes	17.7 (1.4)	2	18.9 (0.6)	3	15.8 (1.2)	1	36.4 (2.5)	4
ailerons	55.8 (5.7)	4	24.2 (1.3)	3	5.9 (0.1)	1	8.6 (0.3)	2
bank8FM	14.0 (2.3)	2	7.7 (0.3)	1	19.4 (4.6)	4	16.3 (0.6)	3
calhousing	20.5 (1.4)	4	14.0 (1.8)	2	14.1 (1.5)	3	9.4 (0.3)	1
elevators	43.5 (3.4)	4	29.2 (0.6)	3	5.1 (0.2)	1	9.5 (0.6)	2
fried	26.0 (0.7)	4	19.3 (0.3)	3	12.1 (0.5)	1	12.1 (0.4)	2
house16h	36.1 (3.2)	3	48.7 (2.5)	4	5.0 (0.3)	1	10.0 (0.3)	2
house8	20.0 (1.7)	3	26.3 (1.8)	4	5.4 (0.5)	1	9.6 (0.3)	2
kin8nm	11.9 (1.7)	4	6.1 (0.1)	3	4.2 (0.3)	1	4.9 (0.2)	2
mvnumeric	27.6 (1.9)	2	28.9 (1.4)	3	15.5 (5.3)	1	33.2 (3.8)	4
pol	45.9 (4.4)	4	20.2 (0.8)	2	7.4 (0.4)	1	26.3 (1.5)	3
puma32H	33.9 (4.9)	4	9.1 (0.1)	1	9.6 (1.3)	2	22.4 (1.4)	3
puma8NH	13.9 (1.2)	4	5.9 (0.2)	1	9.4 (1.9)	2	10.5 (0.6)	3
ratingssw.	2.7 (0.2)	4	2.3 (0.1)	3	2.0 (0.0)	1	2.0 (0.0)	2
BNG-stock	25.1 (0.8)	4	17.2 (0.4)	2	17.8 (5.0)	3	13.3 (0.5)	1
BNG-chol.	164.2 (15.4)	4	78.3 (2.8)	3	40.0 (3.1)	1	42.0 (0.3)	2
BNG-echo	20.0 (1.9)	4	9.8 (0.2)	2	14.1 (5.4)	3	9.5 (1.5)	1
BNG-wine.	51.9 (2.3)	4	43.5 (0.9)	3	25.5 (1.2)	1	28.7 (0.2)	2
∅ **Rank**		3.56		2.56		1.61		2.28

Table 4 Average milliseconds needed to process each instance by the different TSK-Streams variants (training + testing)

Dataset	TSK-Streams Error red One cand.	Rank	TSK-Streams Error red. Both cand.	Rank	TSK-Streams Variance red. One cand.	Rank	TSK-Streams Variance red. Both cand.	Rank
2dplanes	2.36 (0.1)	4	1.64 (0.04)	4	78.82 (6.0)	4	182 (12.1)	4
ailerons	5.07 (0.1)	4	2.74 (0.2)	4	43.3 (1.2)	4	65.34 (2.8)	4
bank8FM	0.7 (0.1)	4	0.26 (0.01)	4	13.98 (2.3)	4	15.36 (0.8)	4
calhousing	1.27 (0.1)	3	0.75 (0.1)	3	38.04 (4.4)	3	24.88 (1.1)	3
elevators	2.78 (0.1)	4	1.87 (0.1)	4	18.83 (0.8)	4	34.17 (1.9)	4
fried	2.91 (0.1)	4	1.93 (0.1)	4	67.83 (2.8)	4	77.46 (5.8)	4
house16h	3.04 (0.2)	4	3.89 (0.3)	4	23.84 (1.0)	4	50.1 (2.6)	4
house8	1.45 (0.1)	4	1.56 (0.1)	4	15.81 (1.6)	4	24.91 (0.6)	4
kin8nm	0.63 (0.1)	4	0.21 (0.0)	4	4.65 (0.4)	4	5.18 (0.3)	4
mvnumeric	3.15 (0.2)	4	2.74 (0.2)	4	86.22 (32.5)	4	141.9 (10)	4
pol	16.94 (2.1)	4	4.28 (0.2)	4	37.1 (3.31)	4	146.98 (10)	4
puma32H	1.69 (0.1)	4	0.66 (0.02)	4	31.26 (3.8)	4	69.95 (5.7)	4
puma8NH	0.63 (0.03)	4	0.2 (0.0)	4	8.75 (1.0)	4	9.4 (0.7)	4
ratingssw.	0.39 (0.01)	3	0.2 (0.0)	3	1.66 (0.0)	3	1.54 (0.03)	3
BNG-stock	3.57 (0.1)	3	2.18 (0.04)	3	122 (22)	3	98.03 (3.3)	3
BNG-chol.	275 (21)	4	201 (7.9)	4	6873 (592)	4	7860 (230)	4
BNG-echo	1.14 (0.1)	3	0.51 (0.01)	3	25.73 (6.8)	3	23.33 (4.5)	3
BNG-wine.	55 (2.5)	4	53 (1.5)	4	2187 (120)	4	2680 (125)	4
∅ **Rank**		1.89		1.11		3.22		3.78

Table 5 Performance comparison of the algorithms with TSK-Streams (variance reduction with the extension for both candidates), in terms of RMSE

Dataset	AMRules	Rank	FIMTDD	Rank	ARF-Reg	Rank	FLEXFIS	Rank	TSK-Streams	Rank
2dplanes	$1.47\ (2 \times 10^{-2})$	$3^{\ominus}$	$1.05\ (8 \times 10^{-4})$	$2^{\ominus}$	$4.1\ (2 \times 10^{-1})$	$5^{\ominus}$	$2.39\ (3 \times 10^{-4})$	$4^{\ominus}$	$1.02\ (2 \times 10^{-3})$	1
ailerons	$4.6 \times 10^{-4}\ (5 \times 10^{-5})$	4	$3.5 \times 10^{-3}\ (10^{-3})$	$5^{\ominus}$	$3.9 \times 10^{-4}\ (2 \times 10^{-5})$	2	$1.9 \times 10^{-4}\ (\times 10^{-6})$	$1^{\oplus}$	$4.1 \times 10^{-4}\ (10^{-6})$	3
bank8FM	$3.63 \times 10^{-2}\ (2 \times 10^{-4})$	$2^{\ominus}$	$3.88 \times 10^{-2}\ (3 \times 10^{-4})$	$4^{\ominus}$	$1.12 \times 10^{-1}\ (2 \times 10^{-2})$	$5^{\ominus}$	$3.64 \times 10^{-2}\ (1 \times 10^{-4})$	$3^{\ominus}$	$3.46 \times 10^{-2}\ (2 \times 10^{-3})$	1
calhousing	$74{,}928\ (6.6 \times 10^{3})$	$4^{\ominus}$	$74{,}498\ (5.7 \times 10^{3})$	$3^{\ominus}$	$84{,}179\ (2 \times 10^{3})$	$5^{\ominus}$	$67{,}177\ (2.5 \times 10^{3})$	$1^{\oplus}$	$69025\ (1.9 \times 10^{3})$	2
elevators	$5.1 \times 10^{-3}\ (10^{-3})$	3	$2.02 \times 10^{-2}\ (5 \times 10^{-3})$	$5^{\ominus}$	$1.2 \times 10^{-2}(4 \times 10^{-3})$	$4^{\ominus}$	$3.64 \times 10^{-3}\ (10^{-5})$	$1^{\oplus}$	$4.3 \times 10^{-3}\ (10^{-3})$	2
fried	$2.54\ (10^{-3})$	$3^{\ominus}$	$2.14\ (10^{-3})$	$2^{\ominus}$	$4.26\ (2 \times 10^{-1})$	$5^{\ominus}$	$2.63\ (3 \times 10^{-3})$	$4^{\ominus}$	$2.07\ (2 \times 10^{-3})$	1
house16h	$55{,}619\ (1.5 \times 10^{3})$	$5^{\ominus}$	$44{,}463\ (10^{3})$	2	$47883\ (4.6 \times 10^{2})$	$3^{\ominus}$	$48{,}401\ (6.7 \times 10)$	$4^{\ominus}$	$44{,}445\ (1.4 \times 10^{2})$	1
house8	$51{,}051\ (2.9 \times 10^{3})$	$4^{\ominus}$	$39{,}041\ (8.8 \times 10^{2})$	$2^{\oplus}$	$56{,}223\ (1.1 \times 10^{3})$	$5^{\ominus}$	$40{,}355\ (8.9 \times 10^{2})$	$3^{\ominus}$	$37031\ (1.2 \times 10^{3})$	1
kin8nm	$2.05 \times 10^{-1}(5 \times 10^{-3})$	$4^{\ominus}$	$1.98 \times 10^{-1}\ (10^{-3})$	2	$2.42 \times 10^{-1}\ (6 \times 10^{-3})$	$5^{\ominus}$	$2.03 \times 10^{-1}\ (10^{-4})$	$3^{\ominus}$	$1.96 \times 10^{-1}\ (10^{-3})$	1
mvnumeric	$2.6\ (6 \times 10^{-2})$	$3^{\ominus}$	$0.95\ (4 \times 10^{-2})$	1	$5.43\ (1)$	$5^{\ominus}$	$3.35\ (2 \times 10^{-1})$	$4^{\ominus}$	$1.09\ (7 \times 10^{-2})$	2
pol	$57.23\ (8.1)$	$4^{\ominus}$	$15.1\ (1.7 \times 10^{-1})$	$1^{\oplus}$	$40.75\ (9 \times 10^{-1})$	$3^{\ominus}$	$59.03\ (8.1 \times 10^{-1})$	$5^{\ominus}$	$17.73\ (1.3 \times 10^{-1})$	2
puma32H	$2.52 \times 10^{-2}\ (4 \times 10^{-4})$	$3^{\ominus}$	$1.71 \times 10^{-1}\ (3 \times 10^{-4})$	$1^{\oplus}$	$3.01 \times 10^{-1}\ (4 \times 10^{-4})$	$5^{\ominus}$	$2.97 \times 10^{-1}\ (4 \times 10^{-5})$	$4^{\ominus}$	$1.92 \times 10^{-1}\ (\times 10^{-3})$	2
puma8NH	$4.26\ (2 \times 10^{-2})$	$3^{\ominus}$	$3.8\ (1.4 \times 10^{-2})$	2	$4.77\ (2.9 \times 10^{-1})$	$5^{\ominus}$	$4.47\ (4 \times 10^{-3})$	$4^{\ominus}$	$3.75\ (4 \times 10^{-2})$	1
ratingssw	$1.61\ (4 \times 10^{-3})$	$3^{\oplus}$	$1.5\ (8 \times 10^{-3})$	$2^{\oplus}$	$1.49\ (5 \times 10^{-3})$	$1^{\oplus}$	$1.6\ (10^{-3})$	$4^{\oplus}$	$1.62\ (10^{-3})$	5
BNG-stock	$4.19\ (2 \times 10^{-2})$	$4^{\ominus}$	$3.49\ (7 \times 10^{-3})$	$1^{\oplus}$	$4.4\ (10^{-1})$	$5^{\ominus}$	$3.87\ (4.4 \times 10^{-2})$	3	$3.85\ (10^{-2})$	2
BNG-chol.	$51.09\ (2 \times 10^{-1})$	$4^{\ominus}$	$50.31\ (1.5 \times 10^{-2})$	$3^{\ominus}$	$51.94\ (1.7 \times 10^{-1})$	$5^{\ominus}$	$49.54\ (10^{-3})$	$2^{\ominus}$	$49.12\ (1.9 \times 10^{-2})$	1
BNG-echo	$12.68\ (6 \times 10^{-2})$	$3^{\ominus}$	$15.31\ (1 \times 10^{-2})$	$4^{\ominus}$	$15.98\ (10^{-1})$	$5^{\ominus}$	$11.81\ (6 \times 10^{-3})$	$1^{\oplus}$	$11.85\ (10^{-2})$	2
BNG-wine.	$8.08 \times 10^{-1}\ (3 \times 10^{-3})$	$4^{\ominus}$	$7.8 \times 10^{-1}\ (3 \times 10^{-4})$	$2^{\ominus}$	$8.26 \times 10^{-1}\ (8 \times 10^{-3})$	$5^{\ominus}$	$7.88 \times 10^{-1}\ (10^{-4})$	$3^{\ominus}$	$7.78 \times 10^{-1}\ (2 \times 10^{-4})$	1
∅ **Rank**		3.5		2.44		4.33		3		1.72
Wins/Losses	1/15		4/9		1/16		5/12		N/A	

The superscripts $\ominus$ and $\oplus$ indicate statistical significance for the comparison with TSK-Streams

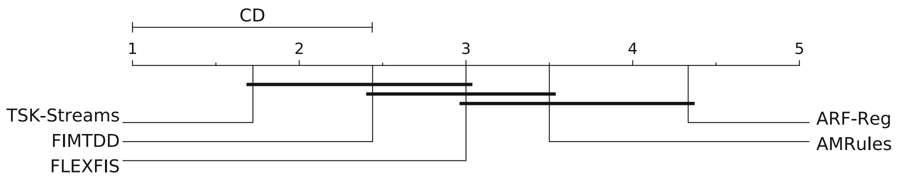

Fig. 3 Comparison of all learners using the Nemenyi test (Demsar 2006). For a confidence level of 0.05, the critical difference (CD) for average ranks is 1.43, which indicates that the advantage of TSK-Rules is statistically significant in the cases of FIMTDD and ARF-Reg but neither for AMRules nor for FLEXFIS

5 Conclusion

In this paper, we introduced a new fuzzy rule learner for adaptive regression on data streams, called TSK-Streams. This method combines the effectivity of concepts for rule induction as implemented in AMRules with the expressivity of TSK fuzzy rules. TSK-Streams as presented in this paper is an improved variant of an earlier version (Shaker et al. 2017); modifications essentially concern all parts of the learning algorithm, including the discretization, the rule extension, and the drift detection.

In an experimental study with real and synthetic data, we compared TSK-Streams with state-of-the-art regression algorithms for learning from data streams: AMRules, FIMTDD, ARF-Reg and FLEXFIS. The results are very promising, especially because our learner achieves the best performance in terms of predictive accuracy. This is remarkable, given that AMRules and FLEXFIS are truly strong (and indeed still competitive) learners—these methods have been developed over many years, and are therefore difficult to beat.

Our current implementation of TSK-Streams can be obtained from our Github repository.[6]

Funding Open Access funding enabled and organized by Projekt DEAL.

Hyperparameter optimization

We perform a hyperparameter optimization for all methods based on Hoeffding's inequality (i.e., TSK-Streams, AMRules, FIMTDD, and ARF-Reg). Three parameters were tuned, *(i)* the confidence level $\delta \in \{10^{-7}, 10^{-5}, 10^{-3}, 10^{-2}, 10^{-1}\}$, *(ii)* the tie breaking threshold $t \in \{0.05, 0.1, 0.2\}$, and *(iii)* the learning rate for stochastic gradient

[6] https://github.com/shaker82/TSK-Streams.

Table 6 Size of the learned model of the algorithms in comparison with TSK-Streams (variance reduction with the extension for both candidates)

	AMRules #rules	Rank	FFIMTDD #leaf nodes	Rank	ARF-Reg #leaf nodes	Rank	FLEXFIS #rules	Rank	TSK-Streams #rules	Rank
2dplanes	9.2 (0.13)	2	154.9 (0.74)	4	11,701.4 (19)	5	1 (0)	1	36.4 (2.5)	3
ailerons	1.9 (0.1)	2	13.4 (0.59)	4	2304.6 (100)	5	1 (0)	1	8.6 (0.25)	3
bank8FM	6.2 (0.24)	2	26.6 (0.45)	4	1569.8 (8)	5	1 (0)	1	16.3 (0.57)	3
calhousing	5.7 (0.2)	2	68.7 (0.95)	4	4079.6 (28)	5	1.7 (0.15)	1	9.4 (0.32)	3
elevators	2.1 (0.1)	2	35.2 (0.58)	4	2801.1 (156)	5	1 (0)	1	6.4 (0.2)	3
fried	9.6 (0.16)	2	150.1 (0.89)	4	8965.5 (7)	5	1.6 (0.29)	1	41.7(0.247)	3
house16h	5.6 (0.16)	2	88.6 (0.7)	4	4465.4 (22)	5	1 (0)	1	23.7 (0.46)	3
house8	3.9 (0.1)	2	67.7 (0.78)	4	4344.3 (7)	5	3.1 (0.75)	1	13.9 (0.26)	3
kin8nm	8.5 (0.16)	2	30.5 (0.32)	4	1807.4 (3)	5	1.6 (0.25)	1	14.6(1.12)	3
mvnumeric	9000 (0)	5	149.6 (0.74)	3	8961.6 (6)	4	3.6 (0.53)	1	33.2 (3.79)	2
pol	3.6 (0.32)	2	48.1 (0.88)	4	2552 (34)	5	1 (0)	1	26.3 (1.5)	3
puma32H	6.6 (0.43)	2	27.4 (0.64)	4	1333 (5)	5	1 (0)	1	22.4 (1.41)	3
puma8NH	4 (0.25)	2	26.2 (0.8)	4	1453 (6)	5	1.1 (0.1)	1	10.5 (0.57)	3
ratingssw	3.8 (0.13)	2	62.8 (1.1)	4	3690.3 (9)	5	7.8 (4.71)	3	2 (0)	1
BNG-stock	8.2 (0.19)	2	225.6 (0.9)	4	13,009.2 (8)	5	6.5 (1.8)	1	13.3 (0.51)	3
BNG-chol	3 (0)	2	3011.5 (5.9)	4	195,593.4 (19221)	5	1 (0)	1	42 (0.35)	3
BNG-echo	6.2 (0.28)	2	48.3 (0.76)	4	3688.4 (20)	5	1.5 (0.47)	1	9.5 (1.49)	3
BNG-wine	2 (0)	2	1197.7 (8)	4	90,261.1 (1498)	5	1.1 (0.1)	1	28.7 (0.25)	3
∅ **Rank**		2.21		3.94		4.94		1.1		2.78

Table 7 Average milliseconds needed to process each instance (training + testing)

	AMRules	Rank	FFIMTDD	Rank	ARF-Reg	Rank	FLEXFIS	Rank	TSK-Streams	Rank
2dplanes	0.38 (6×10^{-2})	2	0.23 (4×10^{-2})	1	4.06 (0.1)	3	54.2 (0.2)	4	182.6 (12)	5
ailerons	0.39 (10^{-2})	2	0.27 (6×10^{-3})	1	3.47 (4×10^{-2})	3	28.6 (6×10^{-2})	4	65.3 (2.8)	5
bank8FM	0.32 (9×10^{-3})	2	0.07 (4×10^{-3})	1	0.97 (4×10^{-2})	3	14.5 (0.2)	4	15.4 (0.8)	5
calhousing	0.31 (6×10^{-3})	2	0.17 (4×10^{-3})	1	2.16 (9×10^{-2})	3	30.8 (0.9)	5	24.9 (1.1)	4
elevators	0.32 (5×10^{-3})	2	0.15 (5×10^{-3})	1	2.39 (8×10^{-2})	3	25 (3×10^{-2})	5	18 (0.7)	4
fried	0.35 (3×10^{-3})	1	0.44 (8×10^{-3})	2	4.96 (0.2)	3	56.6 (2)	4	183.7 (3.8)	5
house16h	0.41 (5×10^{-3})	2	0.32 (5×10^{-3})	1	3.28 (3×10^{-2})	3	33.4 (0.8)	4	91.1 (2.3)	5
house8	0.31 (6×10^{-3})	2	0.18 (6×10^{-3})	1	2.66 (0.1)	3	40.4 (3)	5	30.67 (1.6)	4
kin8nm	0.35 (9×10^{-3})	2	0.08 (4×10^{-3})	1	0.88 (3×10^{-2})	3	15.9 (0.4)	5	9.6 (0.69)	4
mvnumeric	0.34 (4×10^{-3})	1	0.38 (5×10^{-3})	2	4.35 (7×10^{-2})	3	89.4 (7)	4	141 (10)	5
pol	0.49 (6×10^{-3})	2	0.29 (2×10^{-3})	1	4.99 (0.1)	3	37.3 (0.4)	4	146 (10)	5
puma32H	0.5 (8×10^{-3})	2	0.28 (6×10^{-3})	1	2.32 (9×10^{-2})	3	19.8 (8×10^{-2})	4	69 (5.7)	5
puma8NH	0.29 (5×10^{-3})	2	0.07 (5×10^{-3})	1	0.94 (4×10^{-2})	3	21 (4)	5	9.4 (0.7)	4
ratingssw	0.23 (4×10^{-3})	2	0.06 (3×10^{-3})	1	1.02 (3×10^{-2})	3	45 (12)	5	1.54 (3×10^{-2})	4
BNG-stock	0.34 (6×10^{-3})	1	0.64 (8×10^{-3})	2	7.12 (0.3)	3	95 (9.6)	4	98 (3)	5
BNG-chol	0.29 (1×10^{-2})	1	10.87 (7×10^{-2})	2	796 (549)	3	1052 (2)	4	7860 (229)	5
BNG-echo	0.32 (9×10^{-3})	2	0.16 (7×10^{-3})	1	2.2(9×10^{-2})	3	24 (0.8)	5	23 (4)	4
BNG-wine	0.28 (5×10^{-3})	1	9.29 (7×10^{-2})	2	103.5 (2)	3	554 (7)	4	2680 (125)	5
∅ **Rank**		1.72		1.27		3		4.3		4.6

descent update $l \in \{0.02, 0.05, 0.1\}$. On each data set, we perform three rounds of grid search using 5000 instances selected at random. The parameters found are as follows:

- TSK-Streams
  - δ:[10^{-7}, 10^{-5} (house16h), 10^{-3} (elevators), 10^{-2} (pol, fried), 10^{-1} (2dplanes, ailerons, bank8FM, calhousing, house8, kin8nm, mvnumeric, pol, puma32H, puma8NH, ratingssw., BNG-stock, BNG-chol., BNG-echo, BNG-wine.)]
  - t:[0.05 (2dplanes, ailerons, bank8FM, calhousing, elevators, house8, mvnumeric, pol, puma32H, puma8NH, ratingssw., BNG-stock, BNG-chol., BNG-echo, BNG-wine.), 0.1, 0.2 (fried, house16h, kin8nm)]
  - l:[0.02, 0.05 (house8), 0.1 (2dplanes, ailerons, bank8FM, calhousing, elevators, fried, house16h, kin8nm, mvnumeric, pol, puma32H, puma8NH, ratingssw., BNG-stock, BNG-chol., BNG-echo, BNG-wine.)]
- AMRules
  - δ:[10^{-7} (BNG-echo, BNG-wine., ailerons), 10^{-5} (puma32H, puma8NH), 10^{-3} (2dplanes, BNG-chol., BNG-stock, bank8FM, elevators, house16h, house8, kin8nm, mvnumeric, ratingssw.), 10^{-2} (fried, pol), 10^{-1} (calhousing)]
  - t:[0.05 (BNG-chol., BNG-wine., ailerons, calhousing, elevators, puma32H, puma8NH, ratingssw.), 0.1 (bank8FM, house8, pol), 0.2 (2dplanes, BNG-echo, BNG-stock, fried, house16h, kin8nm, mvnumeric)]
  - l:[0.02 (BNG-chol., BNG-wine., ratingssw.), 0.05 (2dplanes, BNG-echo, bank8FM, house16h, house8, kin8nm, pol, puma32H, puma8NH), 0.1 (BNG-stock, ailerons, calhousing, elevators, fried, mvnumeric)]
- FIMTDD
  - δ:[10^{-7} (BNG-echo, BNG-wine., ailerons), 10^{-5} (puma32H, puma8NH), 10^{-3} (2dplanes, BNG-chol., BNG-stock, bank8FM, elevators, fried, house16h, house8, kin8nm, mvnumeric, ratingssw.), 10^{-2} (pol), 10^{-1} (calhousing)]
  - t:[0.05 (BNG-chol., BNG-wine., ailerons, calhousing, elevators, puma32H, puma8NH, ratingssw.), 0.1 (bank8FM, house8,pol), 0.2 (2dplanes, BNG-echo, BNG-stock, fried, house16h, kin8nm, mvnumeric)]
  - l:[0.02 (BNG-chol., ratingssw.), 0.05 (2dplanes, BNG-echo, BNG-wine., bank8FM, house16h, house8, kin8nm, pol, puma32H, puma8NH), 0.1 (BNG-stock, ailerons, calhousing, elevators, fried, mvnumeric)]

Since ARF-Reg is a random forest of FIMTDD, the best parameters for FIMTDD are also used for ARF-Reg. For the other parameters, we adopt the suggested parametrization (Gomes et al. 2018) by setting $\lambda = 6$, the ensemble size $L = 10$, and the number of features $m = \sqrt{d}+1$, with d being the total number of features. FLEXFIS is implemented in Matlab. Its parameters were tuned with the help of a function specifically offered for that purpose. The only exception is the "forgetting parameter", for which the value 0.999 was (manually) found to provide the best performance.

References

Almeida E, Ferreira CA, Gama J (2013) Adaptive model rules from data streams. In: European conference on machine learning and knowledge discovery in databases, ECML/PKDD 2013. Czech Republic, Prague, pp 480–492

Angelov PP (2002) Evolving rule-based models: a tool for design of flexible adaptive systems. Springer, London

Angelov PP (2004) An approach for fuzzy rule-base adaptation using on-line clustering. Int J Approx Reason 35(3):275–289. https://doi.org/10.1016/j.ijar.2003.08.006

Angelov PP (2010) Evolving Takagi–Sugeno fuzzy systems from data streams (eTS+). In: Angelov PP, Filev DP, Kasabov N (eds) Evolving intelligent systems: methodology and applications, Wiley, Hoboken

Angelov PP, Filev DP, Kasabov N (eds) (2010) Evolving intelligent systems: methodology and applications. Wiley, Hoboken

Bifet A, Gavaldà R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the seventh SIAM international conference on data mining, Minneapolis, MN, USA, pp 443–448. https://doi.org/10.1137/1.9781611972771.42

Bifet A, Gavaldà R (2009) Adaptive learning from evolving data streams. In: Proceedings of IDA 2009, 8th international symposium on intelligent data analysis, Lyon, France, pp 249–260. https://doi.org/10.1007/978-3-642-03915-7_22

Bifet A, Holmes G, Kirkby R, Pfahringer B (2010) MOA: massive online analysis. J Mach Learn Res 11:1601–1604

Bifet A, Zhang J, Fan W, He C, Zhang J, Qian J, Holmes G, Pfahringer B (2017) Extremely fast decision tree mining for evolving data streams. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, Halifax, NS, Canada, pp 1733–1742. https://doi.org/10.1145/3097983.3098139

Bischl B, Casalicchio G, Feurer M, Hutter F, Lang M, Mantovani RG, van Rijn JN, Vanschoren J (2017) OpenML benchmarking suites and the OpenML100. arXiv:1708.03731

Breiman L, Friedman J, Stone CJ, Olshen R (1984) Classification and regression trees. Wadsworth and Brooks, Monterey

Demsar J (2006) Statistical comparisons of classifiers over multiple data sets. J Mach Learn Res 7:1–30

Domingos P, Hulten G (2000) Mining high-speed data streams. In: Proceedings of the 6th ACM SIGKDD international conference on knowledge discovery and data mining, Boston, MA, USA, pp 71–80. https://doi.org/10.1145/347090.347107

Dua D, Graff C (2019) UCI machine learning repository [http://archive.ics.uci.edu/ml]. University of California, School of Information and Computer Science, Irvine, CA

Gama J (2012) A survey on learning from data streams: current and future trends. Prog Artif Intell 1(1):45–55. https://doi.org/10.1007/s13748-011-0002-6

Gama J, Kosina P (2011) Learning decision rules from data streams. In: Proceedings of the 22nd international joint conference on artificial intelligence, Barcelona, Catalonia, Spain. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-213

Gama J, Zliobaite I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. ACM Comput Surv 46(4):44:1–44:37

Gomes HM, Barddal JP, Ferreira LEB, Bifet A (2018) Adaptive random forests for data stream regression. In: 26th European symposium on artificial neural networks, ESANN 2018, Bruges, Belgium

Gray R (1984) Vector quantization. IEEE ASSP Mag 1(2):4–29. https://doi.org/10.1109/MASSP.1984.1162229

Ikonomovska E (2012) Algorithms for learning regression trees and ensembles on evolving data streams. PhD thesis, Doctoral Dissertation, Jožef Stefan International Postgraduate School

Ikonomovska E, Gama J, Dzeroski S (2011) Learning model trees from evolving data streams. Data Min Knowl Disc 23(1):128–168. https://doi.org/10.1007/s10618-010-0201-y

Ikonomovska E, Gama J, Džeroski S (2015) Online tree-based ensembles and option trees for regression on evolving data streams. Neurocomputing 150:458–470

Karalič A (1992) Employing linear regression in regression tree leaves. In: Proceedings of the 10th European conference on artificial intelligence. Wiley, New York, ECAI, pp 440–441

Klement EP, Mesiar R, Pap E (2000) Triangular norms. Kluwer Academic Publishers, Dordrecht

Kosina P, Gama J (2012) Handling time changing data with adaptive very fast decision rules. In: European conference on machine learning and knowledge discovery in databases, ECML/PKDD 2012, Bristol, UK. https://doi.org/10.1007/978-3-642-33460-3_58

Ljung L (1999) System identification: theory for the user, 2nd edn. Prentice Hall PTR, Upper Saddle River

Lu J, Liu A, Dong F, Gu F, Gama J, Zhang G (2019) Learning under concept drift: a review. IEEE Trans Knowl Data Eng 31(12):2346–2363

Lughofer E (2008) FLEXFIS: a robust incremental learning approach for evolving Takagi–Sugeno fuzzy models. IEEE Trans Fuzzy Syst 16(6):1393–1410. https://doi.org/10.1109/TFUZZ.2008.925908

Lughofer E (2011) Evolving fuzzy systems: methodologies, advanced concepts and applications. Springer, Berlin

Oza NC, Russell SJ (2001) Online bagging and boosting. In: Proceedings of the eighth international workshop on artificial intelligence and statistics, Key West, FL, USA

Page E (1954) Continuous inspection schemes. Biometrika 41(1–2):100–115. https://doi.org/10.1093/biomet/41.1-2.100

Pedrycz W, Gomide F (1998) An introduction to fuzzy sets: analysis and design. MIT Press, London, UK

Quinlan JR (1992) Learning with continuous classes. In: Proceedings of the Australian joint conference for artificial intelligence. World Scientific, pp 343–348

Shaker A, Senge R, Hüllermeier E (2013) Evolving fuzzy pattern trees for binary classification on data streams. Inf Sci 220:34–45

Shaker A, Heldt W, Hüllermeier E (2017) Learning TSK fuzzy rules from data streams. In: European conference on machine learning and knowledge discovery in databases, ECML/PKDD 2017, Skopje, Macedonia

Takagi T, Sugeno M (1985) Fuzzy identification of systems and its applications to modeling and control. IEEE Trans Syst Man Cybern 15(1):116–132

van Rijn JN, Holmes G, Pfahringer B, Vanschoren J (2014) The Bayesian network generator: A data stream generator. Technical report, Technical Report 03/2014, Computer Science Department, University of Waikato

Zadeh LA (1965) Fuzzy sets. Inf Control 8(3):338–353