Learning TSK Fuzzy Rules from Data Streams

Ammar Shaker(✉), Waleri Heldt, and Eyke Hüllermeier

Department of Computer Science, Paderborn University, Paderborn, Germany
{ammar.shaker,eyke}@upb.de, heldt@mail.upb.de

Abstract. Learning from data streams has received increasing attention in recent years, not only in the machine learning community but also in other research 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. Our method induces a set of fuzzy rules, which, compared to conventional rules with Boolean antecedents, has the advantage of producing smooth regression functions. To do so, it makes use of an induction technique inspired by AMRules, a very efficient and effective learning algorithm that can be seen as the state of the art in machine learning. We conduct a comprehensive experimental study showing that a combination of the expressiveness of fuzzy rules with the algorithmic concepts of AMRules yields a learning system with superb performance.

1 Introduction

Learning from data streams has been a topic of active research in recent years [11]. In this branch of machine learning, systems are sought that learn incrementally, and maybe even in real-time, on a continuous and potentially unbounded stream of data, and which is able to properly adapt themselves to changes of environmental conditions or properties of the data generating process. Systems with these properties have already been developed for different machine learning and data mining tasks, such as clustering and classification.

An extension of machine learning methods to the setting of data streams comes with a number of challenges. In particular, the standard batch mode of learning, in which the entire data as a whole is provided as an input to the learning algorithm, is no longer applicable. Correspondingly, the data must be processed in a single pass, which implies an incremental mode of learning and model adaptation.

Domingos and Hulten [9] list a number of properties that an ideal stream mining system should exhibit, and suggest corresponding design decisions: the system uses only a limited amount of memory; the time to process a single record is short and ideally constant; the data is volatile and a single data record accessed only once; the model produced in an incremental way is equivalent to the model that would have been obtained through common batch learning (on

M. Ceci et al. (Eds.): ECML PKDD 2017, Part II, LNAI 10535, pp. 559–574, 2017.
https://doi.org/10.1007/978-3-319-71246-8_34

all data records so far); the learning algorithm should react to concept change (i.e., any change of the underlying data generating process) in a proper way and maintain a model that always reflects the current concept.

Rule-based learning is a specifically popular approach in the realm of data streams, not only in the machine learning but also in the computational intelligence community, where it has been studied under the notion of "evolving fuzzy systems" [21]. In this paper, we develop a method that combines the strengths of two existing approaches for regression on data streams rooted in different learning paradigms. Our method induces a set of fuzzy rules, which, compared to conventional rules with Boolean antecedents, has the advantage of producing smooth regression functions. To do so, it makes use of an induction technique inspired by AMRules, a very efficient and effective learning algorithm that yields state-of-the-art performance in machine learning.

The rest of the paper is organized as follows. Following a review of related work, we introduce our method in Sect. 3. A comprehensive experimental study, in which the method is compared to several competitors, is presented in Sect. 4, prior to concluding the paper in Sect. 5.

2 Related Work

In the past ten years, learning from data streams has been considered for different learning tasks. Approaches to supervised learning have mostly focused on classification. Here, the Hoeffding tree method [8] has gained a lot of attention, and meanwhile, many modifications and improvements of the original method have been proposed [6]. In addition to the induction of decision trees, the learning of systems of decision rules is supported by several approaches, such as the Adaptive Very Fast Decision Rules (AVFDR) classifier [17]. AVFDR can be seen as an extension of the Very Fast Decision Rules (VFDR) classifier [12] that incrementally induce a compact set of decision rules from a data stream.

Regression on data streams has gained less attention than classification, with a few notable exceptions. AMRules [1] can be seen as an extension of AVFDR to the case of numeric target values. Another approach is based on the induction of model trees [15]. Besides, regression on data streams has been studied quite extensively in the computational intelligence and fuzzy systems community [2, 4,21]. Specifically relevant for us is FLEXFIS [20], which learns a system of so-called Takagi-Sugeno-Kang (TSK) rules [25]. In the following, we describe these methods in some more detail.

FIMTDD (Fast Incremental Model Trees with Drift Detection) is a tree-based approach for inducing model trees for regression on data streams. Similar to Hoeffding trees, it uses Hoeffding's inequality [14] for choosing the best splitting attribute. Since FIMTDD tackles regression problems, attributes are evaluated in terms of the reduction of the target attribute's standard deviation. Each leaf node of the induced tree is associated with a linear function, which is learned (using stochastic gradient descent) in the subspace covering the instances that fall into that leaf node.

AMRules (Adaptive Model Rules) learns rules that are specified by a conjunction of literals on the input attributes in the premise part, and a linear function of the attributes in the consequent. The latter is chosen so as to maximize predictive accuracy in the sense of minimizing the root mean squared error. Adaptive statistical measures are maintained in each rule in order to describe the instance subspace covered by that rule. Each rule is initialized with a single literal and successively expanded with new literals. The best literal to be added, if any, is chosen on the basis of Hoeffding's bound, in a manner that is similar to the expansion of a Hoeffding tree. In their paper [1], the authors distinguish between decision lists and unordered rule sets and, correspondingly, propose two different update and prediction schemes. The first one sorts the set of rules in the order in which they were learned. Only the first rule covering an example is used for prediction and updated afterward. The second strategy updates all the rules that cover an example, and combines these rules' predictions by a weighted sum.[1] The authors also show that the latter strategy outperforms the former one, and hence used it for the rest of their study.

FLEXFIS (Fexible Fuzzy Inference Systems) induces a set of fuzzy rules, making use of fuzzy logic as a generalization of conventional (Boolean) logic [20]. Mores specifically, it uses so-called Takagi-Sugeno-Kang (TSK) rules that are defined by a fuzzy predicate in the premise part and a linear function of the input features in the consequent. As a result, an instance can be covered by a rule to a certain degree, reflecting a degree of relevance of the rule in the corresponding part of the instance space. Correspondingly, the prediction of a TSK system is produced by a weighted average of the outputs of the individual rules. The regions covered by the rules in the input space are defined by means of clustering methods: The instances in the training data are first clustered, and the fuzzy-logical predicates in the rule antecedents are obtained by projecting the clusters to the individual dimensions of this space. For learning on data streams, clustering is done in an incremental way. Moreover, the functions in the rule consequents are adapted using recursive weighted least squares (RWLS) estimation [19].

Our approach essentially seeks to combine the increased expressiveness of fuzzy rules as used by methods such as FLEXFIS, which allows for approximating a regression function is a smoother and much more flexible way, with the efficiency and effectivity of rule induction techniques such as AMRules. In fact, existing methods for learning fuzzy rules, including FLEXFIS and eTS+ [3], are usually slow and computationally inefficient. The complexity is mainly caused by the use of clustering methods, which have the additional disadvantage of producing rules that always contain all input attributes, as well as costly matrix operations (such as inversion) required by RWLS.

[1] In their paper [1], it is not mentioned how the weight of a rule is derived. Based on the delivered implementation, it seems a rule is weighted by the errors it has committed in the past.

3 The TSK-Streams Learning Algorithm

Our method, called TSK-Streams, is an adaptive incremental rule induction algorithm for regression on data streams. The model produced by TSK-Streams is a so-called Takagi-Sugeno-Kang (TSK) fuzzy system [25], a type of rule-based system that is widely used in the fuzzy logic community.

3.1 TSK Fuzzy Systems

A TSK rule R_i is a fuzzy rule of the following form:

$$\begin{aligned} &\text{IF} \quad (x_1 \text{ IS } A_{i,1}) \text{ AND} \quad \ldots \quad \text{AND } (x_d \text{ IS } A_{i,d}) \\ &\text{THEN} \quad l_i(\boldsymbol{x}) = w_{i,0} + w_{i,1}x_1 + w_{i,2}x_2 + \ldots + w_{i,d}x_d, \end{aligned} \tag{1}$$

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. The coefficients $w_{i,0}, \ldots, w_{i,d} \in \mathbb{R}$ in the consequent part of the rule specify an affine function of the features (input attributes).

Modeling the soft constraint $A_{i,j}$ in terms of a fuzzy set with membership function $\mu_j^{(i)} : \mathbb{R} \longrightarrow [0,1]$, the truth degree of the predicate $(x_j \text{ IS } A_{i,j})$ is given by $\mu_j^{(i)}(x_j)$, that is, the degree of membership of x_j in $\mu_j^{(i)}$. Moreover, modeling the logical conjunction in terms of a triangular norm $\top$ [16], i.e., an associative, commutative, non-decreasing binary operator $\top : [0,1]^2 \longrightarrow [0,1]$ with neutral element 1 and absorbing element 0, the overall degree to which an instance $\boldsymbol{x}$ satisfies the premise of the rule R_i is given by

$$\mu_i(\boldsymbol{x}) = \top\left(\mu_1^{(i)}(x_1), \ldots, \mu_d^{(i)}(x_d)\right). \tag{2}$$

In the following, we will adopt the simple product norm, i.e., $\top(u,v) = uv$. Note that $A_{i,j}$ could be an empty constraint, which is modeled by $\mu_j^{(i)} \equiv 1$; this means that the jth attribute x_j does effectively not occur as part of the premise of the rule (1).

Now, consider a TSK system consisting of C rules $RS = \{R_1, \ldots, R_C\}$. Given an instance $\boldsymbol{x}$ as an input, each rule R_i is supposed to "fire" with the (activation) degree (2). Correspondingly, the output produced by the system is defined in terms of a weighted average of the outputs produced by the individual rules:

$$\hat{y} = \sum_{i=1}^{C} \Psi_i(\boldsymbol{x}) \cdot l_i(\boldsymbol{x}), \tag{3}$$

where

$$\Psi_i(\boldsymbol{x}) = \frac{\mu_i(\boldsymbol{x})}{\sum_{j=1}^{C} \mu_j(\boldsymbol{x})}. \tag{4}$$

3.2 Online Rule Induction

TSK-Streams learns rules incrementally, starting with a default rule. This rule has an empty premise and covers the entire input space.

For each rule R_i, TSK-Streams continuously checks whether one of its extensions may improve the performance of the current system. Here, expanding a rule R_i means splitting it into two new rules, which are obtained by adding, respectively, a new predicate $(x_j \text{ IS } A_{i,j})$ and $(x_j \text{ IS } \neg A_{i,j})$ as an additional antecedent. Considering the current rule as the default, the former defines a specialization, while the latter can be seen as what remains of this default. $A_{i,j}$ is modeled in terms of a fuzzy set with membership function $\mu_{j,l}^{(i)}$, which is chosen from a fuzzy partition $\{\mu_{j,1}, \ldots, \mu_{j,k}\}$ of the domain of feature x_j (cf. Sect. 3.3 below), and its negation $\neg A_{i,j}$ is characterized by the membership function $\bar{\mu}_{j,l}^{(i)} = 1 - \mu_{j,l}^{(i)}$. We denote the corresponding expansions by $R_i \oplus \mu_{j,l}^{(i)}$ and $R_i \oplus \bar{\mu}_{j,l}^{(i)}$, respectively.

We distinguish between features x_j that are included by a positive literal $(x_j \in \mu_{j,l}^{(i)})$ and those included by a negative literal $(x_j \in \bar{\mu}_{j,l}^{(i)})$, collecting the indices of the former in the index set I and those of the latter in $\bar{I}$. In a single rule, each attribute is only allowed to occur in a single positive literal. Negative literals are allowed to be added as long as the conjunction of the constraints on x_j does not become too restrictive, thus suggesting a kind of inconsistency (there is not a single value of x_j satisfying the rule premise to a high degree). Details of the rule expansion procedure in pseudocode are given in Algorithm 1.

3.3 Online Discretization and Fuzzification

As a basis for rule expansion, TSK-Streams maintains a fuzzy partition for each feature x_j, i.e., a discretization of the domain of x_j into a finite number of (overlapping) fuzzy sets $\{\mu_{j,1}, \ldots, \mu_{j,k}\}$.

The discretization process is based on the Partition Incremental Discretization (PID) proposed by Gama and Pinto [13]. PID is a technique that builds histograms on data streams in an adaptive manner. In a first layer, continuous input values produced by the data stream are grouped into intervals. A second layer then uses the intervals of the first layer to build histograms, using either equal frequency or equal width binning. In this work, we extend the PID approach as follows:

- Layer 1: This layer discretizes and summarizes the values observed for one input feature into an intial set of intervals.
- Layer 2: This layer merges or splits intervals of the first layer, with the goal to create intervals of equal frequencies.
- Layer 3: This layer transforms the second layer's intervals, which are of the form $X_{j,l} = [b, c]$, into fuzzy sets $\mu_{j,l}$. We employ fuzzy sets with a core $[b, c]$, in which the degree of membership is 1, and support $[a, d] \supset [b, c]$; outside the support, the membership is 0. The boundary of the fuzzy set $\mu_{j,l}$ is modeled in terms of a smooth "S-shaped" transition between full and zero membership:

Algorithm 1. GenerateExtendedRules

Input: $R = (I, M, \bar{I}, \bar{M}, \boldsymbol{\omega})$:
I the index set of the features considered in the premise.
M the set of fuzzy sets conjugated in premise.
$\bar{I}$ the index set of the features negated in the premise.
$\bar{M}$ the set of fuzzy sets whose negated form is conjugated in premise.
$\boldsymbol{\omega}$ the vector of coefficients of the linear function.
$P = \{\mu_{i,j}\}$: the set of all available fuzzy sets. $\mu_{i,j}$ is the jth fuzzy set on the ith feature.
υ the overlapping threshold.
Result: $S = \{(R_1, R_2)\}$: Set of expanded rules

```
for i ∈ {1, ..., d} \ I do
    for μ_{i,j} ∈ P do
        if i ∉ Ī then
            R_1 = (I ∪ {i}, M ∪ {μ_{i,j}}, Ī, M̄, ω)
            R_2 = (I, M, Ī ∪ {i}, M̄ ∪ {μ_{i,j}}, ω})
            S ∪ {(R_1, R_2)}
        else
            find m_{i.} ∈ M
            if μ_{i,j} ∩ m_{i.} < υ then
                R_1 = (I ∪ {i}, M ∪ {μ_{i,j}}, Ī \ {i}, M̄ \ {m_{i.}}, ω)
                m′_{i.} = m_{i.} ∪ μ_{i,j}
                R_2 = (I, M, Ī, M̄ ∪ {m′_{i.} ∪ μ_{i,j}} \ {m_{i.}}, ω})
                S ∪ {(R_1, R_2)}
```

$$\mu_{j,l}(x) = \begin{cases} 0 & \text{if } x < a \\ 2\left(\frac{x-a}{b-a}\right)^2 & \text{if } a \le x < (a+b)/2 \\ 1 - 2\left(\frac{x-b}{b-a}\right)^2 & \text{if } (a+b)/2 \le x < b \\ 1 & \text{if } b \le x \le c \\ 1 - 2\left(\frac{c-x}{d-c}\right)^2 & \text{if } c < x \le (c+d)/2 \\ 2\left(\frac{d-x}{d-c}\right)^2 & \text{if } (c+d)/2 < x \le d \\ 0 & \text{if } x > d \end{cases} . \tag{5}$$

For the fuzzy set $\mu_{j,l}$ associated with $X_{j,l}$, we set $a = b - \alpha \cdot |X_{j,l-1}|$ and $d = c + \alpha \cdot |X_{j,l+1}|$, where $|X_{j,l-1}|$ and $|X_{j,l+1}|$ denote, respectively, the lengths of the left and right neighbor interval of $X_{j,l}$, and $\alpha \in]0, 1[$ is an overlap degree. For the leftmost (rightmost) interval of the partition, we set $a = b$ ($d = c$).

3.4 Learning Rule Consequents

FLEXFIS fits linear functions in the consequent parts of the rules via recursive weighted least squares estimation (RWLS) [19]. Since this approach requires

Algorithm 2. First Layer: Discretization

Input: x: newly observed value
k: the initial number of intervals
λ: exponential weighting factor, $\lambda \in [0, 1]$
Result: $L1Intervals, L1Counts, L1Time$: arrays for the produced intervals, their counts and timestamps of the last update

if *first call* **then**
  initialize $L1Intervals$ with k intervals of equal width
$sum = \sum_i L1Counts[i]$
let t be the current time
let i be the index s.t $x \in L1Intervals[i]$
$sum = sum * \lambda + 1$
$L1Counts[i] = L1Counts[i] * \lambda^{t-L1Time[i]} + 1$
$L1Time[i] = t$
if $(L1Counts[i]/sum) > threshold$ **then**
  Split($L1Intervals[i], L1Counts[i]$)

multiple matrix inversions, it is computationally expensive. Therefore, inspired by AMRules, we instead apply a gradient method to learn consequents more efficiently.

Upon arrival of a new training instance $(\boldsymbol{x}_t, y_t)$, the squared error of the prediction $\hat{y}_t$ produced by TSK-Streams can be computed as follows:

$$E_t = (y_t - \hat{y}_t)^2 = \left(y_t - \left(\sum_{R_i \in RS} \frac{\Psi_i(\boldsymbol{x}_t)}{\sum_{R_k \in RS} \Psi_k(\boldsymbol{x}_t)} \sum_{j=0}^{d} \omega_{i,j} x_{t,j} \right) \right)^2 \tag{6}$$

Invoking the principle of stochastic gradient descent, the coefficients $\omega_{i,j}$ are then shifted into the negative direction of the gradient:

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta \nabla E_t, \tag{7}$$

where η is the learning rate. Component-wise, this yields the following update rule:

$$\omega_{i,j} \leftarrow \omega_{i,j} - 2\,\eta (y_t - \hat{y}_t) \left(\sum_{R_i \in RS} \frac{\Psi_i(\boldsymbol{x}_t)}{\sum_{R_k \in RS} \Psi_k(\boldsymbol{x}_t)} x_{t,j} \right) \tag{8}$$

The process of updating the rule consequents is summarized in Algorithm 4.

3.5 Adaptation of the Model Structure

As outlined above, TSK-Streams continuously adapts the rule system through the adaptation of fuzzy sets used as rule antecedents and linear functions in the rule consequents. While these are adaptations of the system's parameters, the decision to replace a rule by one of its expansions can be seen as a structural change.

Algorithm 3. Second Layer: Histograms

Input:
$L1Intervals, L1Counts$: arrays for the Layer1's intervals and their counts
k: number of intervals
Result: $L2Intervals$: the resulting intervals of the 2nd Layer

$sum = \sum_i L1Intervals[i]$
$maxCap = sum/k$
$currentCap = 0$
$interval.min = L1Intervals[1].min$
while $i <= length(L1Intervals)$ **do**
  while $currentCap + L1Counts[i] < maxCap$ **do**
    $currentCap = currentCap + L1Counts[i]$
    $interval.max = intervalsL1[i].max$
    $i++$
  $newMax = L1Intervals[i].min + \frac{maxCap - currentCap}{L1Counts[i]}.L1Intervals[i].width$
  $interval.max = newMax$
  $L2Intervals.add(interval)$
  $L1Intervals[i].min = newMax$
  $L1Counts[i] = L1Counts[i] - \frac{maxCap - currentCap}{L1Counts[i]}$
  $interval.min = newMax$
  $currentCap = 0$
if $length(L1Intervals) > 1.5K$ **then**
  $L1Intervals = L2Intervals$
  for $i \in \{1, \ldots, k\}$ **do**
    $L1Counts[i] = maxCap$

Needless to say, structural changes should generally be handled with care, especially when increasing the complexity of the model. Therefore, learning methods typically stick to the current model until being sufficiently convinced of a potential improvement through an expansion; to this end, the estimated difference in performance needs to be *significant* in a statistical sense.

Similar to Hoeffding trees [8], AMRules [10], and FIMT-DD [15], we apply Hoeffding's inequality in order to support these decisions. The Hoeffding inequality probabilistically bounds the difference between the expected value $E(X)$ of a random variable X with support $[a, b] \subset \mathbb{R}$ and its empirical mean $\bar{X}$ on an i.i.d. sample of size n in terms of

$$P\Big(|\bar{X} - \mathrm{E}(X)| > \epsilon\Big) \leq \exp\left(-\frac{2n\epsilon^2}{(b-a)^2}\right). \tag{9}$$

More specifically, we decide to split a rule R_i, i.e., to replace the rule with two rules $R_i \oplus \mu_{j,l}^{(i)}$ and $R_i \oplus \bar{\mu}_{j,l}^{(i)}$, by considering the reduction in the sum of squared errors (SSE). To this end, the SSE of the current system (rule set RS) is compared to the SSE of all alternative systems $(RS \backslash R_i) \cup \{R_i \oplus \mu_{j,l}^{(i)}, R_i \oplus \bar{\mu}_{j,l}^{(i)}\}$.

Algorithm 4. UpdateConsequent

Input: $RS = \{(R, S)\}$: the set of all rules and their extensions
$(\boldsymbol{x}_t, y_t)$: the new instance to train on
`/* A rule takes the form` $R = (I, M, \bar{I}, \bar{M}, \boldsymbol{\omega})$ `*/`
`/* A rule extension is` $S = \{(R_1, R_2, SSE)\}$`, where` SSE `is the sum of squared errors committed by this extension */`

$m_1 = \sum_{R_i \in RS} \mu_i(\boldsymbol{x}_t)$
$m_2 = \sum_{R_i \in RS} \mu_i(\boldsymbol{x}_t) l_i(\boldsymbol{x}_t)$
for $R_i \in RS$ **do**
  $\mu_i(\boldsymbol{x}_t) = \top(\bigotimes_{\mu \in M_i} \mu(\boldsymbol{x}_t), \bigotimes_{\mu \in \bar{M}_i} 1 - \mu(\boldsymbol{x}_t))$
  if $\mu_i(\boldsymbol{x}_t) > 0$ **then**
    for $(R_1, R_2, SSE) \in S_i$ **do**
      $\mu_{i1}(\boldsymbol{x}_t) = \top(\bigotimes_{\mu \in M_1} \mu(\boldsymbol{x}_t), \bigotimes_{\mu \in \bar{M}_1} 1 - \mu(\boldsymbol{x}_t))$
      $\mu_{i2}(\boldsymbol{x}_t) = \top(\bigotimes_{\mu \in M_2} \mu(\boldsymbol{x}_t), \bigotimes_{\mu \in \bar{M}_2} 1 - \mu(\boldsymbol{x}_t))$
      $m_1\prime = m_1 - \mu_i(\boldsymbol{x}_t) + \mu_1(\boldsymbol{x}_t) + \mu_2(\boldsymbol{x}_t)$
      $m_2\prime = m_2 - \mu_i(\boldsymbol{x}_t) l_i(\boldsymbol{x}_t) + \mu_1(\boldsymbol{x}_t) l_1(\boldsymbol{x}_t) + \mu_2(\boldsymbol{x}_t) l_2$
      $\boldsymbol{\omega}_1 = \boldsymbol{\omega}_1 + \eta(y_t - \frac{m_2\prime}{m_1\prime}) \left(\frac{\mu_1(\boldsymbol{x}_t)}{m_1\prime} \boldsymbol{x}\right)$
      $\boldsymbol{\omega}_2 = \boldsymbol{\omega}_2 + \eta(y_t - \frac{m_2\prime}{m_1\prime}) \left(\frac{\mu_2(\boldsymbol{x}_t)}{m_1\prime} \boldsymbol{x}\right)$
    $\boldsymbol{\omega}_i = \boldsymbol{\omega}_i + \eta(y_t - \frac{m_2}{m_1}) \left(\frac{\mu_i(\boldsymbol{x}_t)}{m_1} \boldsymbol{x}\right)$

Let SSE_{best} and $SSE_{2ndbest}$ denote, respectively, the expansion with the lowest and the second lowest error. The best expansion is then adopted whenever

$$\frac{SSE_{best}}{SSE_{2ndbest}} < 1 - \epsilon, \tag{10}$$

or when ϵ becomes smaller than a tie-breaking constant τ. The constant ϵ is derived from (9) by setting the probability to a desired degree of confidence $1 - \delta$, i.e., setting the righ-hand side to $1 - \delta$ and solving for ϵ; noting that the ratio (10) is bounded in $]0, 1]$, $b - a$ is set to 1.[2] Refer to Algorithm 5 for details.

Instead of looking for a global improvement of the entire system, an alternative is to monitor the performance of individual rules and to base decisions about rule expansion on this performance. In this case, Hoeffding's bound is applied to the sum of weighted squared errors (SWSE), where the weighted squared error of a rule R_i on a training example $(\boldsymbol{x}_t, y_t)$ is given by

$$WSE_t = \Psi(\boldsymbol{x})(y_t - \hat{y}_t)^2 = \left(\frac{\mu(\boldsymbol{x}_t)}{\sum_{R_j \in RS} \mu_j(\boldsymbol{x}_t)}\right)(y_t - \hat{y}_t)^2.$$

[2] We are aware of theoretical issues caused by the use of the Hoeffding bound, the assumptions of which are normally not all satisfied [23]. Yet, the bound is commonly applied, in spite of these problems, and proved very useful in practice.

This error is then compared with the weighted error of the system in which the rule is replaced by extensions $R_i \oplus \mu_{j,l}^{(i)}$ and $R_i \oplus \bar{\mu}_{j,l}^{(i)}$. The usefulness of such extensions can be checked using the same kind of hypothesis testing as above.

To avoid an excessive increase in the number of rules, also coming with a danger of overfitting, we propose a penalization mechanism that consists of adding a complexity term C to ϵ. For the global variant, we set $C = \frac{1-\log(2)/\log(|RS|)}{\sqrt{d}}$, where RS is the current set of rules and d the number of features. For the local variant, we use $C = \frac{1-\log(2)/\log(|I \cup \bar{I}e|)}{\sqrt{d}}$ when comparing the extensions of a rule $R = (I, M, \bar{I}, \bar{M}, \boldsymbol{\omega})$.

Algorithm 5. ExpandSystem

Input: $RS = \{(R, S)\}$: the set of all rules and their extensions
δ: confidence level
τ: tie-breaking constant
n: number of examples seen so far by the current system
/* A rule takes the form $R = (I, M, \bar{I}, \bar{M}, \boldsymbol{\omega})$ */
/* A rule extension is $S = \{(R_1, R_2, SSE)\}$, where SSE is the sum of squared errors committed by this extension */

$n = n + 1$
let $SSE_{current}$ be the sum of squared errors for the current system
let $(R, S)_{best}$ be the best performing extension with the lowest achieved sum of squared errors SSE_{best}
/* $S_{best} = \{(R_1, R_2, SSE)_{best}\}$ */
let $(R, S)_{2ndbest}$ be the second best performing extension with $SSE_{2ndbest}$
/* $S_{2ndbest} = \{(R_1, R_2, SSE)_{2ndbest}\}$ */
$\epsilon = \sqrt{\frac{\ln\left(\frac{1}{\delta}\right)(R)^2}{2n}}$
$\overline{X} = \frac{1}{n}(SSE_{best}/SSE_{2ndbest})$
$\overline{Y} = \frac{1}{n}(SSE_{best}/SSE_{current})$
if $((\overline{Y} + \epsilon) < 1)$ *AND* $((\overline{X} + \epsilon) < 1$ *OR* $\epsilon < \tau)$ **then**
  $RS = RS \setminus \{(R, S) : R = R_{best}\}$
  $RS = RS \cup \{(R_{best,1}, GenerateExtendedRules(R_{best,1})),$
  $(R_{best,2}, GenerateExtendedRules(R_{best,2}))\}$
  $n = 0$, $SSE_{current} = 0$
  for $R \in RS$ **do**
    reinitialize R

3.6 Change Detection

To detect a drop in a rule's performance, possibly caused by a concept drift, we employ the adaptive windowing (ADWIN) [5] drift detection method. The advantage of this technique, compared to the Page-Hinkely test (PH) [22] used by AMRules, is that ADWIN is non-parametric (it makes no assumptions about the observed random variable). Moreover, it has only one parameter δ_{adwin}, which

represents the tolerance towards false alarms. We apply this change detection method locally in each rule on the absolute error committed by a rule on an example, given that the example is covered by this rule.

Upon detecting a drift in the rule $R_p = (I_p, M_p, \bar{I}_p, \bar{M}_p, \boldsymbol{\omega}_p)$, we find its sibling rule $R_q = (I_q, M_q, \bar{I}_q, \bar{M}_q, \boldsymbol{\omega}_q)$, from which it differs by only one single literal, i.e., there is a fuzzy set $\mu_{i,j}$ that satisfies one of the following criteria: $(\mu_{i,j} \in M_p) \wedge (i \in I_p) \wedge (\bar{\mu}_{i,j} \in \bar{M}_q) \wedge (i \in \bar{I}_q)$ or $(\bar{\mu}_{i,j} \in \bar{M}_p) \wedge (i \in \bar{I}_p) \wedge (\mu_{i,j} \in M_q) \wedge (i \in I_q)$. Removing the rule R_p can simply be achieved by removing it from the rule set and accordingly updating its sibling R_q by either removing $(i, \mu_{i,j})$ from (I_q, M_q) or removing $(i, \bar{\mu}_{i,j})$ from $(\bar{I}_q, \bar{M}_q)$, depending on which of the previous criteria was satisfied. If the sibling rule R_q was already extended before detecting the drift, one simply applies the same procedure to its children.

4 Empirical Evaluation

In this section, we conduct experiments in order to study the performance of TSK-Streams in comparison to other algorithms. More precisely, we analyze predictive accuracy and runtime of the algorithms, the size of the models they produce, as well as their ability to recover in the presence of a concept drift.

4.1 Setup

Our proposed fuzzy learner, TSK-Streams, is implemented under the MOA[3] (Massive Online Analysis) [7] framework, an open source software for mining and analyzing large data sets in a stream-like manner.

In the following evaluations, we compare TSK-Streams with the three methods introduced before: AMRules, FIMTDD, and FLEXFIS. Both AMRules and FIMTDD are implemented in MOA's distribution, and we use them in their default settings with $\delta = 0.01$ and $\tau = 0.05$ for the Hoeffding bound. Regarding the parametrization of TSK-Streams, we use the same values δ, τ, so as to assure maximal comparability with AMRules and FIMTDD. For the discretization, we use the following parameters: the number of intervals $k = 5$, the overlapping threshold $\upsilon = 0.2$, the exponential weighting factor $\lambda = 0.999$, and the overlapping degree $\alpha = 0.15$. FLEXFIS is implemented in Matlab and offers a function for finding optimal parameter values. We used this function to tune all parameters except the so-called "forgetting parameter", for which we manually found the value 0.999 to perform best.

All experiments are conducted using the test-then-train evaluation procedure; this procedure uses each instance for both training and testing. First, the model is evaluated on the instance, and then a single incremental learning step is carried out.

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

4.2 Results

In the first part of the evaluation, we preform experiments on standard synthetic and real benchmark data sets collected from the UCI repository[4] [18] and other repositories[5]; Table 1 provides a summary of the type, the number of attributes and instances of each data set.[6] Table 2 shows the average RMSE and the corresponding standard error on ten rounds for each data set. In this table, the winning approach on each data set is highlighted in bold font, and our approach is marked with an asterisk whenever it outperforms the three competitors. As can be seen, our fuzzy rule learner, both in its global and local variant, is superior to the other methods in terms of generalization performance. In a pairwise comparison, the global variant of TSK-Streams outperforms AMRules and FLEXFIS on 11 of the 14 data sets, and performs better than FIMTDD on 13; the local variant outperforms AMRules, FLEXFIS, and FIMTDD on 8, 11, and 13 data sets, respectively. Using a Wilcoxon signed-rank test, the global variant of our method thus outperforms AMRules, FLEXFIS, and FITDD with p-values 0.067, 0.041 and 0.0008, and the local variant outperforms FITDD with p-value 0.0008.

Table 1. Data sets

#	Name	Synthetic	Instances	Attributes
1	2dplanes	yes	40768	11
2	ailerons	no	13750	41
3	bank8FM	yes	8192	9
4	calHousing	no	20640	8
5	elevators	no	8752	19
6	fried	yes	40769	11
7	house16H	no	22784	16
8	house8L	no	22784	8
9	kin8nm	-	8192	9
10	mvnumeric	yes	40768	10
11	pol	no	15000	49
12	puma32H	yes	8192	32
13	puma8NH	yes	8192	9
14	ratingssweetrs	-	17903	2

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

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

[6] These are the same data sets as those used in the AMRules paper [1].

Table 2. Performance of the algorithms in terms of RMSE.

	AMRules	FIMTDD	FLEXFIS	TSK-streams (global)	TSK-streams (local)
2dplanes	$1.40E+00(0)$	$2.67E+00(0)$	$2.39E+00(0)$	$\mathbf{1.04E+00}^{*}(0)$	$1.05E+00^{*}(0)$
ailerons	$5.83E-04(0)$	$8.00E-04(0)$	$1.91E-04(0)$	$1.81E-04^{*}(0)$	$\mathbf{1.77E-04}^{*}(0)$
bank8FM	$3.81E-02(0)$	$1.25E-01(0)$	$\mathbf{3.64E-02}(0)$	$4.44E-02(0)$	$4.29E-02(0)$
calhousing	$7.00E+04(97)$	$8.44E+04(121)$	$\mathbf{6.72E+04}(71)$	$7.28E+04(90.8)$	$7.26E+04(10.3)$
elevators	$5.80E-03(0)$	$7.94E-03(0)$	$3.64E-03(0)$	$3.79E-03(0)$	$\mathbf{3.42E-03}^{*}(0)$
fried	$2.43E+00(0)$	$3.55E+00(0.02)$	$2.64E+00(0)$	$2.33E+00^{*}(0.01)$	$\mathbf{2.29E+00}^{*}(0)$
house16h	$4.74E+04(427)$	$5.39E+04(435)$	$4.84E+04(15)$	$4.46E+04^{*}(104)$	$\mathbf{4.44E+04}^{*}(52.3)$
house8	$4.07E+04(118)$	$4.67E+04(97.8)$	$4.04E+04(249)$	$\mathbf{4.02E+04}^{*}(142)$	$4.13E+04(106)$
kin8nm	$2.05E-01(0)$	$2.84E-01(0)$	$2.02E-01(0)$	$1.96E-01^{*}(0)$	$\mathbf{1.96E-01}^{*}(0)$
mvnumeric	$2.73E+00(0.02)$	$4.91E+00(0.01)$	$3.35E+00(0.05)$	$1.03E+00^{*}(0.01)$	$\mathbf{9.19E-01}^{*}(0.01)$
pol	$1.96E+01(0.17)$	$3.40E+01(0.24)$	$5.87E+01(0.23)$	$\mathbf{1.90E+01}^{*}(0.11)$	$2.02E+01(0.11)$
puma32H	$2.17E-02(0)$	$4.81E-02(0)$	$2.98E-02(0)$	$1.89E-02^{*}(0)$	$\mathbf{1.86E-02}^{*}(0)$
puma8NH	$4.18E+00(0.01)$	$6.30E+00(0)$	$4.47E+00(0)$	$\mathbf{4.07E+00}^{*}(0.01)$	$4.27E+00(0)$
sweetrs	$1.54E+00(0)$	$\mathbf{1.53E+00}(0)$	$1.61E+00(0)$	$1.60E+00(0)$	$1.61E+00(0)$
Average rank	2.85	4.64	3.28	2.14	2.07

Table 3. Performance of the algorithms in terms of the runtime and model size.

	AMRules	FIMTDD	FLEXFIS	TSK-Streams (global)	TSK-Streams (local)
	Execution time in seconds				
2dplanes	4.80 (0)	0.621 (0)	54.2 (0.13)	5.25 (0.03)	4.49 (0.02)
ailerons	3.23 (0)	0.814 (0)	28.6 (0.04)	18.7 (0.50)	16.3 (0.28)
bank8FM	1.70 (0)	0.321 (0)	14.5 (0.17)	0.829 (0)	1.05 (0)
calhousing	2.93 (0.01)	0.501 (0)	30.8 (0.79)	2.05 (0.01)	1.85 (0.01)
elevators	2.89 (0)	0.544 (0)	25.0 (0.03)	6.64 (0.20)	5.81 (0.03)
fried	4.85 (0.01)	1.13 (0)	56.6 (1.42)	6.54 (0.05)	8.18 (0.12)
house16h	3.67 (0)	0.881 (0)	33.4 (0.74)	9.05 (0.22)	6.05 (0.04)
house8	3.19 (0.01)	0.614 (0)	40.4 (2.35)	2.77 (0.04)	3.23 (0.02)
kin8nm	1.60 (0)	0.318 (0)	15.9 (0.36)	0.854 (0)	1.12 (0)
mvnumeric	5.04 (0)	0.885 (0)	89.4 (5.9)	7.24 (0.12)	13.3 (0.24)
pol	3.51 (0)	0.860 (0)	37.3 (0.39)	75.6 (1.2)	49.7 (0.65)
puma32H	2.47 (0)	0.724 (0)	19.8 (0.07)	8.08 (0.08)	8.89 (0.17)
puma8NH	1.63 (0)	0.308 (0)	21.0 (3.13)	0.745 (0)	1.04 (0)
sweetrs	2.35 (0.01)	0.338 (0)	45.5 (10.3)	0.384 (0)	0.819 (0.01)
Average rank	2.85	1	4.85	3.07	3.21
	Model size				
2dplanes	31.9 (0.19)	140.9 (0.23)	1 (0)	3 (0)	2 (0)
ailerons	5.4 (0.05)	24 (0.25)	1 (0)	3.3 (0.04)	3.1 (0.06)
bank8FM	8.1 (0.10)	26.8 (0.16)	1 (0)	2.9 (0.02)	2 (0)
calhousing	10 (0.06)	64.5 (0.26)	1.7 (0.12)	3 (0)	1.1 (0.02)
elevators	5.1 (0.02)	40.4 (0.39)	1 (0)	3.4 (0.04)	2.1 (0.02)
fried	18.1 (0.12)	119.6 (0.46)	1.6 (0.25)	3.6 (0.05)	3.2 (0.11)
house16h	6 (0.03)	64.9 (0.40)	1 (0)	5.1 (0.13)	2.1 (0.02)
house8	6.4 (0.05)	71.8 (0.29)	3.1 (0.67)	4.8 (0.11)	2.2 (0.03)
kin8nm	4.9 (0.02)	24 (0.22)	1.6 (0.22)	3 (0)	2 (0)
mvnumeric	24.5 (0.17)	130.2 (0.34)	3.6 (0.47)	6.1 (0.10)	6.5 (0.09)
pol	7.8 (0.08)	43 (0.28)	1 (0)	4.8 (0.06)	4.1 (0.07)
puma32H	11.1 (0.12)	28.3 (0.11)	1 (0)	3.3 (0.04)	3.9 (0.10)
puma8NH	6.7 (0.09)	28.1 (0.17)	1.1 (0.08)	3.1 (0.02)	2.2 (0.03)
sweetrs	9.9 (0.06)	63.2 (0.31)	7.8 (4.20)	3 (0)	2 (0)

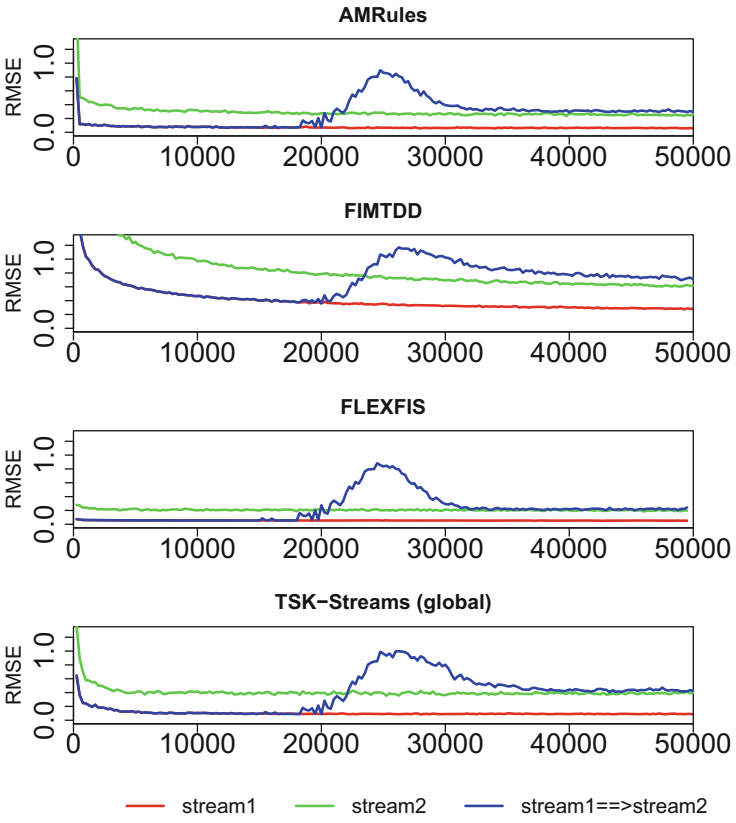

Fig. 1. Performance curves (RMSE, averaged over ten runs) on the distance to hyperplane data, with a drift from squared (red curve) to the cubed distance (green curve) in the middle of the episode. The recovery curve is plotted in blue. Ideally, this curve quickly reaches the performance level of the second stream (green curve). (Color figure online)

Table 3 shows the performance in terms of runtime and model size. TSK-Streams often remains a bit slower than AMRules and FIMTDD. At the same time, however, it is significantly faster than FLEXFIS, reducing runtime by a factor of around 10. Regarding the model size, we report the number of rules (leaves for FITDD) just to give an indicator of model complexity and without implying specific claims.

In the second part of the evaluation, we study the ability of our approach to recover from a performance drop in the presence of a concept drift. To this end, we make use of so-called *recovery analysis* as introduced in [24]. Recovery analysis aims at assessing a learner's ability to maintain its generalization performance in the presence of concept drift; it provides an idea of how quickly a drift is recognized, to what extent it affects the prediction performance, and

how quickly the learner manages to adapt its model to the new condition. The main idea of recovery analysis is to employ three streams in parallel, two "pure streams" and one "mixture", instead of using a single data stream. The mixture stream resembles the first pure stream at the beginning and the second stream at the end, thus it contains a concept drift as a result of modeling the sampling probability as a sigmoidal function. Due to lack of space, we refer the reader to [24] for details of the methodology.

In general, we find that TSK-Streams recovers quite well in comparison to the other methods. As an illustration, we plot the recovery curves (blue lines) for the distance to hyperplane data set in Fig. 1. As can be seen, FLEXFIS exhibits a relatively large drop in performance. FIMTDD does not even manage to recover completely till the end of the stream. Compared to this, TSK-Streams and AMRules recover quite well.

5 Conclusion

In this paper, we proposed TSK-Streams, an evolving fuzzy rule learner for regression that meets the requirements of incremental and adaptive learning on data streams. Our method combines the expressivity and flexibility of TSK fuzzy rules with the efficiency and effectivity of concepts for rule induction as implemented in algorithms such as AMRules.

In an experimental study, we compared TSK-Streams with AMRules, FIMTDD, and FLEXFIS, the state-of-the-art regression algorithms for learning from data streams, on real and synthetic data. The results we obtained show that our learner compares very favorably and achieves superior performance. Moreover, it manages to adapt and recover well after a concept drift.

In future work, we plan to elaborate on extensions and variants of TSK-Rules that may lead to further improvements in performance. These developments will be accompanied by additional experiments and case studies.

Acknowledgments. This work was supported by the Competence Center for Cyber Physical Systems (CPS.HUB NRW).

References

1. Almeida, E., Ferreira, C., Gama, J.: Adaptive model rules from data streams. In: Blockeel, H., Kersting, K., Nijssen, S., Železný, F. (eds.) ECML PKDD 2013. LNCS (LNAI), vol. 8188, pp. 480–492. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40988-2_31
2. Angelov, P.P.: Evolving Rule-based Models: A Tool for Design of Flexible Adaptive Systems. Springer, London (2002). https://doi.org/10.1007/978-3-7908-1794-2
3. Angelov, P.P.: Evolving Takagi-Sugeno fuzzy systems from data streams (eTS+). In: Angelov, P.P., Filev, D.P., Kasabov, N. (eds.) Evolving Intelligent Systems: Methodology and Applications. Wiley, Hoboken (2010)
4. Angelov, P.P., Filev, D.P., Kasabov, N. (eds.): Evolving Intelligent Systems: Methodology and Applications. Wiley, Hoboken (2010)

5. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: Proceedings of 7th SIAM International Conference on Data Mining, Minneapolis, MN, USA, pp. 443–448 (2007)
6. Bifet, A., Gavaldà, R.: Adaptive learning from evolving data streams. In: Proceedings of IDA 2009, 8th International Symposium on Intelligent Data Analysis, Lyon, France, pp. 249–260 (2009)
7. Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B.: MOA: massive online analysis. J. Mach. Learn. Res. **11**, 1601–1604 (2010)
8. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Boston, MA, USA, pp. 71–80 (2000)
9. Domingos, P., Hulten, G.: A general framework for mining massive data streams. J. Comput. Graph. Stat. **12**(4), 945–949 (2003)
10. Duarte, J., Gama, J., Bifet, A.: Adaptive model rules from high-speed data streams. ACM Trans. Knowl. Discov. Data **10**(3), 30:1–30:22 (2016)
11. Gama, J.: A survey on learning from data streams: current and future trends. Prog. Artif. Intell. **1**(1), 45–55 (2012)
12. Gama, J., Kosina, P.: Learning decision rules from data streams. In: Proceedings of 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain (2011)
13. Gama, J., Pinto, C.: Discretization from data streams: applications to histograms and data mining. In: Proceedings of 2006 ACM Symposium on Applied Computing, Dijon, France, pp. 662–667 (2006)
14. Hoeffding, W.: Probability inequalities for sums of bounded random variables. J. Am. Stat. Assoc. **58**(301), 13–30 (1963)
15. Ikonomovska, E., Gama, J., Dzeroski, S.: Learning model trees from evolving data streams. Data Min. Knowl. Discov. **23**(1), 128–168 (2011)
16. Klement, E.P., Mesiar, R., Pap, E.: Triangular Norms. Kluwer Academic Publishers, Dordrecht (2000)
17. Kosina, P., Gama, J.: Handling time changing data with adaptive very fast decision rules. In: Flach, P.A., De Bie, T., Cristianini, N. (eds.) ECML PKDD 2012. LNCS (LNAI), vol. 7523, pp. 827–842. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33460-3_58
18. Lichman, M.: UCI machine learning repository (2013)
19. Ljung, L.: System Identification: Theory for the User, 2nd edn. Prentice Hall PTR, Upper Saddle River (1999)
20. Lughofer, E.: FLEXFIS: a robust incremental learning approach for evolving Takagi-Sugeno fuzzy models. IEEE Trans. Fuzzy Syst. **16**(6), 1393–1410 (2008)
21. Lughofer, E.: Evolving Fuzzy Systems: Methodologies, Advanced Concepts and Applications. Springer, Berlin (2011). https://doi.org/10.1007/978-3-642-18087-3
22. Page, E.S.: Continuous inspection schemes. Biometrika **41**(1–2), 100–115 (1954)
23. Rutkowski, L., Pietruczuk, L., Duda, P., Jaworski, M.: Decision trees for mining data streams based on the McDiarmid's bound. IEEE Trans. Knowl. Data Eng. **25**(6), 1272–1279 (2013)
24. Shaker, A., Hüllermeier, E.: Recovery analysis for adaptive learning from non-stationary data streams: experimental design and case study. Neurocomputing **141**, 97–109 (2014)
25. Takagi, T., Sugeno, M.: Fuzzy identification of systems and its applications to modeling and control. IEEE Trans. Syst. Man Cybern. **15**(1), 116–132 (1985)