Adaptive Model Rules From High-Speed Data Streams

JOÃO DUARTE, LIAAD-INESC TEC
JOÃO GAMA, LIAAD-INESC TEC, Faculty of Economics, University of Porto
ALBERT BIFET, Huawei Noahs Ark Lab

Decision rules are one of the most expressive and interpretable models for machine learning. In this article, we present Adaptive Model Rules (AMRules), the first stream rule learning algorithm for regression problems. In AMRules, the antecedent of a rule is a conjunction of conditions on the attribute values, and the consequent is a linear combination of the attributes. In order to maintain a regression model compatible with the most recent state of the process generating data, each rule uses a Page-Hinkley test to detect changes in this process and react to changes by pruning the rule set. Online learning might be strongly affected by outliers. AMRules is also equipped with outliers detection mechanisms to avoid model adaption using anomalous examples. In the experimental section, we report the results of AMRules on benchmark regression problems, and compare the performance of our system with other streaming regression algorithms.

Q1 CCS Concepts:

Additional Key Words and Phrases: Data streams, regression, rule learning

ACM Reference Format:
João Duarte, João Gama, and Albert Bifet. 2016. Adaptive model rules from high-speed data streams. ACM Trans. Knowl. Discov. Data 10, 3, Article 30 (January 2016), 22 pages.
DOI: http://dx.doi.org/10.1145/2829955

1. INTRODUCTION

Regression analysis is a technique for estimating a functional relationship between a dependent variable and a set of independent variables. It has been widely studied in statistics, pattern recognition, machine learning, and data mining. The most expressive data mining models for regression are model trees [Quinlan 1992] and regression rules [Quinlan 1993a]. In Ould-Ahmed-Vall et al. [2007], a large comparative study between several regression algorithms is presented. Model trees and model rules are among the best performing ones. Trees and rules perform automatic feature selection, being robust to outliers and irrelevant features; exhibit high degree of interpretability; and structural invariance to monotonic transformation of the independent variables.

The authors acknowledge the financial support given by the project ERDF through the COMPETE Programme and by National Funds through Fundação para a Ciência e a Tecnologia (FCT) within the project FCOMP - 01-0124-FEDER-022701. Authors also acknowledge the support of the European Commission through the project MAESTRA (Grant Number ICT-750 2013-612944).
Authors' addresses: J. Duarte, LIADD-INESC TEC, University of Porto, Campus da FEUP, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal; email: joaomaiaduarte@gmail.com; A. Bifet, Huawei Noahs Ark Lab, Units 525-530, Core Building 2, Hong Kong Science Park, Shatin, Hong Kong; email: abifet@waikato.ac.nz; J. Gama, LIADD-INESC TEC, Faculty of Economics, University of Porto, Campus da FEUP, Rua Dr. Roberto Frias, 4200-465 Porto, Portugal; email: jgama@fep.up.pt.

DOI: http://dx.doi.org/10.1145/2829955

One important aspect of rules, and the main advantage over trees, is modularity: each rule can be interpreted individually [Fürnkranz et al. 2012].

Regression problems are one the most frequent learning tasks. The usual batch approaches require that the whole training data are available before learning. Batch algorithms assume that examples are generated at random accordingly to some stationary probability distributions and learn a static model by processing the data multiple times [Gama 2010]. Some regression algorithms, such as the Perceptron algorithm, are incremental by nature. However, turning regression trees and rule-based algorithms incremental require substantial changes. Moreover, these algorithms do not have the capacity to adapt if the target concept evolves over time.

Data streaming learning algorithms face several important challenges. In the data stream computational model, examples are generated sequentially from time-evolving distributions. Data stream learning models need not only to learn with new data, but also forget outdated and no longer relevant data. Therefore, in order to adapt to the most recent state of the nature, data stream algorithms must have mechanisms to increment new examples and decrement old ones. These algorithms should have the capability to learn with high-speed streams since in many applications, such as sensor networks, telecommunication, clickstreams, and financial transactions, examples arrive at extremely high rates. Also, many of these applications require real-time learning and predicting capabilities. Another challenge with streaming data is that a stream is theoretically infinite. However, the memory space and computational capabilities are limited. For this reason, streaming learning algorithms should adapt to the available resources.

In this article, we present the Adaptive Model Rules (AMRules) algorithm, the first one-pass algorithm for learning regression rule sets from time-evolving streams. The work described here is a large extension of the work presented in Almeida et al. [2013a]. The algorithm has been written from scratch and the experimental evaluation has been largely extended. The current version is available in Massive Online Analysis (MOA) [Bifet et al. 2010], which is an open source framework for data stream mining. Another contribution of this article is Random AMRules, an ensemble of adaptive model rules, which is inspired by the Random forests ensemble method [Breiman 2001].

The proposed algorithm can learn ordered or unordered rule sets. The antecedent of a rule is a set of literals (conditions based on the attribute values), and the consequent is a function that minimizes the mean square error of the target attribute computed from the set of examples covered by rule. This function might be either a constant, the mean of the target attribute, or a linear combination of the attributes. Each rule is equipped with online change and anomaly detectors. The change detector monitors the mean square error using the Page-Hinkley (PH) test, providing information about the dynamics of the process generating data. For detecting anomalies, we propose a new method that searches for unlikely input values in particular regions of the instance space. AMRules addresses all the previously referred data streaming challenges. It supports the increment of new examples by continuously growing each rule, and the decrement of non-relevant examples by pruning the rules in which change is detected. Thus, AMRules adapts to time-evolving data. It allows the user to adjust the tradeoff between memory/time costs and accuracy by using an extended binary search tree structure with limited (and parameterized) depth. This structure is used to store summaries of the data needed for learning. Also, since each rule can be learned in parallel, the algorithm can be easily implemented in any distributed real-time stream processing engine.

The article is organized as follows. The next Section presents the related work in learning regression trees and rules from data focusing on streaming algorithms. Section 3 describes, in detail, the AMRules algorithm. Section 4 presents the experimental evaluation using stationary and time-evolving streams. AMRules is compared against other regression systems including batch learners and streaming regression models. The last section presents the lessons learned.

2. RELATED WORK

In the field of machine learning, one of the most popular, and competitive, regression model is system M5, presented by Quinlan [1992]. It builds multivariate trees using linear models at the leaves. In the pruning phase for each leaf, a linear model is built. Later, a *rational reconstruction* of Quinlan's M5 algorithm, M5′, was proposed [Frank et al. 1998]. M5′ first constructs a regression tree by recursively splitting the instance space using tests on single attributes that maximally reduce variance in the target variable. After the tree has been grown, a linear multiple regression model is built for every inner node, using the data associated with that node and all the attributes that participate in tests in the subtree rooted at that node. The linear regression models are then simplified by dropping attributes if this results in a lower expected error on future data (more specifically, if the decrease in the number of parameters outweighs the increase in the observed training error). After this has been done, every subtree is considered for pruning. Pruning occurs if the estimated error for the linear model at the root of a subtree is smaller than or equal to the expected error for the subtree. After pruning terminates, M5′ applies a *smoothing* process that combines the model at a leaf with the models on the path to the root to form the final model that is placed at the leaf.

A widely used strategy consists of building rules from decision (or regression) trees [Quinlan 1993b]. Any tree can be easily transformed into a collection of rules. Each rule corresponds to the path from the root to a leaf, and there are as many rules as leaves. This process generates a set of rules with the same complexity as the decision tree. However, as pointed out by Wang et al. [2003], a drawback of decision trees is that even a slight drift of the target function may trigger several changes in the model and severely compromise learning efficiency. Cubist [Quinlan 1993a] is a rule-based model that is an extension of Quinlan's M5 model tree. A tree is grown where the terminal leaves contain linear regression models. These models are based on the predictors used in previous splits. Also, there are intermediate linear models at each level of the tree. A prediction is made using the linear regression model at the leaf of the tree, but it is *smoothed* by taking into account the prediction from the linear models in the previous nodes in the path, from the root to a leaf, followed by the test example. The tree is reduced to a set of rules, which initially are paths from the top of the tree to the bottom. Rules are eliminated via pruning of redundant conditions or conditions that do not decrease the error.

2.1. Rule Learning from Streaming Data

For classification problems, few rule learning systems from data streams exists in the literature. One of the first classifiers is the system `Facil` [Ferrer-Troyano et al. 2005]. `Facil` uses a multi-strategy approach. The decision model is a set of rules plus a set of training examples. Each decision rules stores a reduced set of positive and negative examples. When classifying a test example, `Facil` find all rules that cover the example. Each rule classifies the example using the nearest-neighbor method using the set of examples stored with that rule. The final classification is obtained using weighted vote. `Facil` uses a forgetting mechanism that can be either explicit or implicit. Explicit forgetting takes places when the examples are older than a user defined threshold. Implicit forgetting is performed by removing examples that are no longer relevant as they do not enforce any concept description boundary.

Rule learning classifiers directly related to the work presented here has been published in Kosina and Gama [2012]. The Hoeffding bound was used to estimate the number of examples required to expand a rule. The main difference is that AMRules, the system described here deals with regression problems.

2.2. Regression Algorithms for Streaming Data

Many methods can be found in the literature for solving classification tasks on streams, but only few exists for regression tasks. To the best of our knowledge, we note only two papers for online learning of regression and model trees. One of the first incremental model trees, was presented by Potts and Sammut [2005]. The authors present an incremental algorithm that scales linearly with the number of examples. They present an incremental node splitting rule, together with incremental methods for stopping the growth of the tree and pruning. The leaves contain linear models, trained using the Recursive Least-Square (RLS)algorithm.

FIMTDD [Ikonomovska et al. 2011] is an incremental algorithm for any-time model trees learning from evolving data streams with drift detection. It is based on the Hoeffding tree algorithm [Domingos and Hulten 2000], but implements a different splitting criterion, using a standard deviation reduction-based measure more appropriate to regression problems. The FIMTDD algorithm is able to incrementally induce model trees by processing each example only once, in the order of their arrival. Splitting decisions are made using only a small sample of the data stream observed at each node, following the idea of Hoeffding trees. FIMTDD is able to detect and adapt to evolving dynamics. Change detection in the FIMTDD is carried out using the PH change detection test [Mouss et al. 2004]. Adaptation in FIMTDD involves growing an alternate subtree from the node in which change was detected. When the performance of the alternate subtree improves over the original subtree, the latter is replaced by the former.

IBLStreams (Instance-Based Learner on Streams) is an extension of MOA that consists of an instance-based learning algorithm for classification and regression problems on data streams by Shaker and Hüllermeier [2012]. IBLStreams optimizes the composition and size of the case base autonomously. When a new example (x_0, y_0) is available, the example is added to the case base. The algorithm checks whether other examples might be removed, either because they have become redundant or they are outliers. To this end, a set C of examples within a neighborhood of x_0 are considered as candidates. This neighborhood is given by the k_{cand} nearest neighbors of x_0, accordingly with a distance function D. The most recent examples are not removed due to the difficulty to distinguish potentially noisy data from the beginning of a concept change.

2.3. Random Rules for Classification Using Data Streams

Random forests [Breiman 2001] consists of a collection or ensemble of simple tree predictors, each capable of producing a response when presented with a set of predictor values. To determine the class of an instance, the method combines the result of various decision trees using a voting mechanism. The classifier is based on the Bagging method [Breiman 1996]. Random forests increase diversity among the classification trees by resampling the data with replacement and by randomly changing the predictive variable sets over the different tree induction processes. Each classification tree is grown using another bootstrap subset X_i of the original dataset X and the nodes are split using the best split predictive variable among a subset of m randomly selected predictive variables [Liaw and Wiener 2002]. This is in contrast with the standard classification tree building, where each node is split using the best split among all predictive variables.

To the best of our knowledge, there have been no publications about random rules for regression until now. However, there are works about random rules for classification. Random Rules [Almeida et al. 2013b] generates an ensemble of rule sets, each one associated with a set of N_{att} attributes, maintaining all properties required when learning from stationary data streams: online and any-time classification, processing each example once.

2.4. Anomaly Detection

The literature in anomaly and outlier detection is huge. Two recent overviews, with excellent references are Hodge and Austin [2004] and Chandola et al. [2009]. Most of the works refer to offline approaches. Two types of anomalies should be considered in anomaly detection [Chandola et al. 2009].

—*Point Anomalies*: if an individual data instance can be considered as anomalous with respect to the rest of the data, then the instance is termed as a point anomaly. This is the simplest type of anomaly and is the focus of the majority of research on anomaly detection.
—*Contextual Anomalies*: if a data instance is anomalous in a specific context. In this case, it is convenient to define:
  —*Contextual attributes:* the contextual attributes are used to determine the context for that instance.
  —*Behavioral attributes:* the attributes with abnormal values in the contexts defined by the contextual attributes.

A relevant aspect is that an observation might be an anomaly in a given context, but an identical data instance (in terms of behavioral attributes) could be considered normal in a different context [Chandola et al. 2009]. This property is a key characteristic in identifying contextual and behavioral attributes for a contextual anomaly detection technique.

3. THE AMRULES ALGORITHM

In this section, we present an incremental algorithm for learning model rules, named Adaptive Model Rules from High-Speed Data Streams (AMRules). AMRules starts with a default rule that is used to progressively grow a rule set. Rules also gradually grow by adding literals to its antecedents. AMRules uses an adaptive window over the most recent examples to make decisions: when to expand a rule. Each rule stores sufficient statistics from a specific landmark window. When a decision is taken, that is, the rule is expanded, the landmark window is reset. The algorithm adapts to concept drifts by monitoring the error of each rule. A rule is removed from the rule set if its online error significantly increases. The stability of the model to concept drifts is guaranteed by the default rule, which is always prepared to make predictions. AMRules is parallelizable since each rule can be learned individually. Therefore, AMRules can be easily implemented in a distributed system. The pseudo-code of the algorithm is given in Algorithm 1.

3.1. Learning a Rule Set

The algorithm begins with an empty rule set (RS), and a default rule $\{\} \rightarrow \mathcal{L}$. Every time when a new training example is available the algorithm verifies if the example is covered by any rule in the rule set (RS), by checking if all the literals are true for the example. Also, change and anomaly detection tests are performed. If a change is detected the rule is removed from the rule set (RS). If an anomaly is detected the example is not considered for learning. We use the PH change detection test to monitor the online error of each rule. Otherwise, the example is used in the rule's learning process. This process consists of updating the sufficient statistics needed for predicting the output value for a new example and expanding the rule. Examples of these statistics are the number of instances covered by the rule, the linear and squared sums of the predicting errors, and information required to decide the best split while expanding a

ALGORITHM 1: AMRules Algorithm

```
  Input: S: Stream of examples
  ordered-set: Boolean flag
  N_min: Minimum number of examples
  λ: Threshold
  α: the magnitude of changes that are allowed
Result: RS  Set of Decision Rules
begin
    Let RS ← {}
    Let defaultRule L ← 0
    foreach example (x⃗, y_k) ∈ S do
        foreach Rule r ∈ RS do
            if r covers the example then
                if not IsAnomaly(example, r) then
                    Call PHTest(error, λ)
                    if Change is detected then
                        Remove the rule
                    end
                    else
                        Update sufficient statistics of r
                        if Number of examples in L mod N_min = 0 then
                            r ← ExpandRule(r)
                        end
                    end
                end
                if ordered-set then
                    BREAK
                end
            end
        end
        if none of the rules in RS triggers then
            Update sufficient statistics of the defaultRule
            if Number of examples in L mod N_min = 0 then
                RS ← RS ∪ ExpandRule(L)
                if defaultRule expanded then
                    Create new L using the statistics not covered by ExpandRule(L)
                end
            end
        end
    end
end
```

rule. The expansion of the rule is considered only after a certain period (N_{min} number of examples). Algorithm 2 describes the expansion of a rule.

The set of rules (RS) is learned in parallel, as described in Algorithm 1. We consider two cases: learning ordered or unordered set of rules. In the former, every example updates statistics of the first rule that covers it. In the latter, every example updates statistics of all the rules that covers it. If an example is not covered by any rule, the default rule is updated.

3.2. Expansion of a Rule

Before discussing how rules are expanded, we will first discuss the evaluation measure used in the attribute selection process. We define the variance ratio (VR) measure of a

ALGORITHM 2: Expandrule: Expanding one Rule

```
Input:
   r: One Rule
   τ: Constant to solve ties
   δ : Confidence
Result: r' : Expanded Rule
begin
   Let X_a be the attribute with greater variance ratio (VR))
   Let X_b be the attribute with second greater VR
   Compute ε = sqrt(R^2 ln(1/δ) / 2n), R = 1 (Hoeffding bound)
   if VR(X_a) − VR(X_b) > ε ∨ ε < τ then
      Extend r with a new condition based on the best attribute
      Release sufficient statistics of L_r
      r ← r ∪ {X_a}
   end
   return r
end
```

split h_A as:

$$VR(h_A) = 1 - \frac{|E_L|}{|E|}\frac{var(E_L)}{var(E)} - \frac{|E_R|}{|E|}\frac{var(E_R)}{var(E)},$$

$$var(E) = \frac{1}{|E|}\sum_{i=1}^{|E|}(y_i - \bar{y})^2 = \frac{1}{|E|}\left[\sum_{i=1}^{|E|} {y_i}^2 - \frac{1}{|E|}\left(\sum_{i=1}^{|E|} y_i\right)^2\right],$$

where E represents the set of examples seen by the rule since its last expansion, E_L and E_R correspond to the subset of E containing the examples whose attribute values are, respectively, less or equal and greater than the value defined in h_A, and $|\cdot|$ is the number of elements in a set. VR can be efficiently computed in an incremental way. To make the actual decision regarding a split, the VR measurements for the best two potential splits are compared, dividing the second-best value by the best one to generate a ratio r in the range 0 to 1. Having a predefined range for the values of the random variables, R, the Hoeffding probability bound (ϵ) [Hoeffding 1963] can be used to obtain high confidence intervals for the true average of the sequence of random variables. The value of ϵ is calculated using the formula:

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}.$$

The process to expand a rule by adding a new condition works as follows. For each attribute X_i, the value of the VR is computed for each attribute value v_j. If the upper bound ($\bar{r}^+ = \bar{r} + \epsilon$) of the sample average is below 1, then the true mean is also below 1. Therefore, with confidence $1 - \delta$, the best attribute over a portion of the data is really the best attribute. In this case, the rule is expanded with condition $X_a \leq v_j$ or $X_a > v_j$. However, often two splits are extremely similar or even identical, in terms of their VR values, and despite the ϵ intervals shrinking considerably as more examples are seen, it is still impossible to choose one split over the other. In these cases, a threshold (τ) on the error is used. If ϵ falls below this threshold and the splitting criterion is still not met, the split is made on the one with a higher VR value and the rule is expanded. The pseudo-code for expanding a rule is presented in Algorithm 2.

The extended binary search tree structure (E-BST) [Ikonomovska et al. 2011] may be used to maintain all possible split points for the numeric attributes. E-BST stores the sufficient statistics for computing VR. We use a modified version of the E-BST structure that limits the maximum number of splitting points to a predefined value (50 by default). This modification reduces memory consumption and speeds up the split selection procedure while having low impact on the error of the learning algorithm.

3.3. Prediction Strategies

The set of rules learned by AMRules can be ordered or unordered. They employ different prediction strategies to achieve "optimal" prediction. In the former, only the first rule that covers an example is used to predict the target example. In the latter, all rules covering the example are used for prediction and the final prediction is decided by aggregating predictions using the mean.

Each rule in AMRules implements three prediction strategies: *(i)* the mean of the target attribute computed from the examples covered by the rule; *(ii)* a linear combination of the independent attributes; and *(iii)* an adaptive strategy, that chooses between the first two strategies, the one with the lower mean absolute error (MAE) in the previous examples. In this case, the MAE is computed following a fading factor strategy. In order to do so, two values are monitored: the total sum of absolute deviations T and the number of the examples used for learning N. When a new example (x, y) arrives for training, T and N are updated as follows: $T \leftarrow \alpha T + |\hat{y} - y|$ and $N \leftarrow \alpha N + 1$, where $\hat{y}$ is the value predicted by the rule and $0 < \alpha < 1$ is a parameter that controls the importance of the oldest/newest examples.

Each rule in AMRules contains a linear model, trained using an incremental gradient descent method, from the examples covered by the rule. Initially, the weights are set to small random numbers in the range −1–1. When a new example arrives, it is standardized considering the mean and standard deviation of the attributes of the examples seen so far. Next, the output is computed using the current weights. Each weight is then updated using the Delta rule: $w_i \leftarrow w_i + \eta(\hat{y} - y)x_i$, where η is the learning rate. The prediction is computed as the "denormalized" value of $\hat{y}$.

3.4. Change Detection

We use the PH [Page 1954] change detection test to monitor the online error of each rule. Whenever a rule covers a labeled example, the rule makes a prediction and computes the loss function (MAE). The PH test is used to monitor the evolution of the loss function. If the PH test signals a significant increase of the loss function, the rule is removed from the rule set (RS).

The PH test is a sequential analysis technique typically used for online change detection. It is designed to detect a change in the average of a Gaussian signal [Mouss et al. 2004]. This test considers a cumulative variable m_T, defined as the accumulated difference between the observed values and their mean until the current moment:

$$m_T = \sum_{t=1}^{T}(x_t - \bar{x}_T - \varphi)$$

where $\bar{x}_T = 1/T \sum_{t=1}^{T} x_t$ and φ corresponds to the magnitude of changes that are allowed.

The minimum value of this variable is also computed: $M_T = \min(m_t, t = 1 \ldots T)$. The test monitors the difference between M_T and m_T: $PH_T = m_T - M_T$. When this difference is greater than a given threshold (λ), we signal a change in the process generating examples. The threshold λ depends on the admissible false alarm rate. Increasing λ will entail fewer false alarms, but might miss or delay change detection.

3.5. Detecting Contextual Anomalies

Detection of outliers and rare events are critical tasks in online learning. Blind learning from these examples might impact the performance of the whole system.

AMRules detects contextual anomalies. Contextual anomalies are characterized by a *context* that refers to the region of the instance space where the anomaly was detected, and behavioral attributes with anomalous values. One example of the type of anomalies we detect is:

Case: 14,571
Rule: $x7 <= 1156$ and $x8 <= 66 \rightarrow y : 7.75$
$x3 = 2 \quad (1.00 \pm 0.03) \quad Prob = 0.002\%$
$x4 = 5 \quad (4.00 \pm 0.03) \quad Prob = 0.002\%$
$x5 = 10 \quad (2052.14 \pm 144.55) \quad Prob = 0.009\%$
$x6 = 100 \quad (2064.88 \pm 374.56) \quad Prob = 0.070\%.$

The 14,571th example is signaled as an anomaly. It is interpreted as follows. The context of the anomaly is given by the conditional part of the rule: $x7 <= 1156$ and $x8 <= 66$. The attributes with suspicious values are $x3 = 2$, $x4 = 5$, $x5 = 10$, and $x6 = 100$, with probabilities 0.002%, 0.002%, 0.009%, and 0.070%, respectively. In the set of examples covered by the rule, the mean value of $x3$ is 1.00 ± 0.03, the mean value of $x4$ is 4.00 ± 0.03, the mean value of $x5$ is 2052.14 ± 144.55, and the mean value of $x6$ is 2064.88 ± 374.56.

Different kinds of rule systems are commonly used in multivariate anomaly detection. The use of AMRules in online detection is one of the advantages the system provides. It can detect possible anomalies during the learning process. The detection process works as follows. When the system reads a new example, the rule set is checked to find the rules that cover the example. The probability $P(X_i = v|\mathcal{L}_r)$ is computed for each value v regarding an attribute X_i given the conditions of a rule r. These probabilities are computed from the consequent of the rule, $\mathcal{L}_r$, that maintains the sufficient statistics required to expand the rule. Low values of these probabilities suggest that the example is an uncommon case in the context of the rule, and it is reported as an anomaly.

A new measure is proposed to perform anomaly detection. It consists of computing the ratio $\frac{P(X_i=v|\mathcal{L}_r)}{1-P(X_i=v|\mathcal{L}_r)}$ for all attributes. When a value v for an attribute X_i is likely $(P(X_i = v|\mathcal{L}_r) > 0.5)$, the ratio gives a positive value. If $P(X_i = v|\mathcal{L}_r) < 0.5$, the ratio gives a negative value. The anomaliness may be assessed by averaging over all ratios, as presented in Equation (1). Logarithms of the ratios are used to avoid numerical instabilities.

$$\begin{aligned} Ascore &= \frac{1}{d}\sum_{j=1}^{d}\log\left(\frac{P(X_i = v|\mathcal{L}_r)}{1 - P(X_i = v|\mathcal{L}_r)}\right) \\ &= \frac{1}{d}\sum_{j=1}^{d}\log(P(X_i = v|\mathcal{L}_r)) - \log(1 - P(X_i = v|\mathcal{L}_r)). \end{aligned} \tag{1}$$

An example is considered to be an anomaly if $Ascore < t$, where t is a user-defined parameter. Usually t is defined to be 0 or a negative value close to 0.

For continuous attributes, the statistics stored in $\mathcal{L}_r$ include the mean and standard deviation of each attribute given the class. Remember that these statistics are computed from the examples covered by the rule. Using these statistics, we can compute $P(X_i = v|\mathcal{L}_r)$ using different strategies, including Normal distribution, Z-scores, etc. From a set of experiments not described here, we selected a variation of the Cantelli's inequality [Bhattacharyya 1987] to estimate $P(X_i = v)|\mathcal{L}_r)$:

$$Pr(|v - \overline{X}_i| \geq k) \leq \begin{cases} \frac{2\sigma_i^2}{\sigma_i^2+k^2}, & \text{if } \sigma_i < k \\ 1, & \text{otherwise} \end{cases}$$

where $\overline{X}_i$ is the mean value of the i^{th} attribute according to $\mathcal{L}_r$.

A relatively new rule, which is a rule that has not been trained with enough examples, would more often tend to report a training example as anomalous. To prevent this situation, only rules that were trained with more than m_{min} examples are used in the anomaly detection.

3.6. Ensembles of Adaptive Model Rules

Ensemble methods have been used as a general method to boost the performance of learning algorithms. In an ensemble, a set of base predictors collaborate in order to solve a task. The machine learning literature about ensembles is huge. Authors converge on at least two points: the ensemble must be diverse and the members of an ensemble must be uncorrelated. A useful analysis to understand why and how an ensemble works is the bias-variance decomposition of the error. The bias-variance profile of an algorithm can be very useful in designing strategies to increase diversity during learning. Regression models with a high-variance profile are affected by perturbing the set of training examples, while low-variance models are affected by perturbing the set of attributes used to train the model.

The profile of AMRules in terms of bias-variance decomposition of the error is low variance. On the basis of this observation, we designed an ensemble of rules model that follows the Random Forests idea: we combine bagging with choosing a random subset of the features for learning the split point for each rule. Note that after the expansion of a rule, a new subset of features is selected at random. We call this ensemble method Random AMRules (RAMRules).

4. EXPERIMENTAL EVALUATION

The main goal of this experimental evaluation is to study the behavior of the proposed algorithm in terms of performance and learning times. We are interested in studying the following scenarios.

—How to grow the rule set? What are the advantages and disadvantages of unordered rule sets over ordered rule sets?
—What is the impact of linear models in rules?
—Which is the impact of change detection ?
—What is the impact of anomaly removal in the performance?
—How does AMRules compare against others Streaming Algorithms?
—How does AMRules compare against others State-of-the-art Regression Algorithms?
—How does AMRules learned models evolve in time?

4.1. Experimental Setup

All our algorithms were implemented in java using the MOA data stream software suite [Bifet et al. 2010]. The performance of the algorithms is measured using the standard metrics for regression problems: MAE and Root Mean Squared Error (RMSE) [Willmott and Matsuura 2005].

Table I. Summary of Datasets

Datasets	# Instances	# Attributes
2dplanes	40768	10
Ailerons	13750	40
Bank8FM	8192	8
CalHousing	20640	8
Elevators	16599	18
Fried	40768	10
House_8L	22784	8
House_16H	22784	16
Kin8nm	8192	8
MV	40768	10
Pol	15000	48
Puma8NH	8192	8
Puma32H	8192	32
FriedD	256000	10
WaveformD	256000	41
Airline	115 Million	10

The experimental datasets include both artificial and real data, as well sets with continuous attributes. We use ten regression datasets from the UCI Machine Learning Repository [Bache and Lichman 2013] and other sources. The datasets used in our experimental work are briefly described here. **2dplanes** this is an artificial dataset described in Breiman et al. [1984]. **Ailerons** this dataset addresses a control problem, namely flying a F16 aircraft. **Bank8FM** a family of datasets synthetically generated from a simulation of how bank-customers choose their banks. CalHousing datasets is composed of eight attributes that describe all the block groups in California from the 1990's Census. The target value is the median house value. **Elevators** this dataset was obtained from the task of controlling a F16 aircraft. **Fried** is an artificial dataset used in Friedman (1991) and also described in Breiman et al. [1984]. **House8L** and **House16H** datasets were collected as part of the 1990 US census and are concerned with predicting the median price of the house based on demographic and state of housing market information. **Kin8nm** dataset is concerned with the forward kinematics of an eight link robot arm. **MV** is an artificial dataset with dependences between the attribute values. **Pol** this is a commercial application described in Weiss and Indurkhya [1995]. The data describe a telecommunication problem. **Puma8NH** and **Puma32H** is a family of datasets synthetically generated from a realistic simulation of the dynamics of a Unimation Puma 560 robot arm. **FriedD** is composed of 256,000 examples generated similarly to the Fried dataset, but contains a drift that starts in the 128,001st instance. **WaveformD** is an artificial dataset containing 256,000 examples generated as described in Breiman et al. [1984], also containing a drift that starts in the 128,001st instance. The dataset consists of three classes of waves labeled, and the examples are characterized by 21 attributes that include some noise plus 19 attributes that are all noise. **Airline** uses the data from the 2009 Data Expo competition. The dataset consists of a huge amount of records, containing flight arrival and departure details for all the commercial flights within the USA, from October 1987 to April 2008. This is a large dataset with nearly 115-million records [Ikonomovska et al. 2011]. Table I summarizes the number of instances and the number of attributes of each dataset.

This method evaluates a model on a stream by testing then training with each example in the stream. AMRules has three main groups of parameters: rule expansion, change detection, and anomaly detection. For the first two groups, we used values usually mentioned in the literature. For all the experiments, we set the parameters regarding the rule expansion to $N_{min} = 200$, $\tau = 0.05$ and $\delta = 0.0000001$, and the PH test parameters to $\lambda = 35$ and $\varphi = 0.005$. For anomaly detection, the reference value for the threshold parameter t is 0 or a negative value close to 0. We were conservative and defined $t = -0.75$. The minimum number of examples that the rule needs to see before performing anomaly detection, m_{min}, was set to 30.

We used two evaluation methods. When no concept drift is assumed, the evaluation method we employ uses the traditional sampling scenario using tenfold cross-validation. All algorithms learn from the same training set and the errors are estimated from the same test set. All the results in the tables are averages from tenfold cross-validation [Kohavi 1995], except for the Airline and Waveform datasets. As pointed out in Gama et al. [2013], in scenarios with concept drift, the appropriate methodology to estimate performance is the prequential error estimate. Also, the fading factor for the MAE computation in the adaptive prediction strategy was defined to $\alpha = 0.99$.

We use the Wilcoxon test to study the significance of the differences in the mean of the evaluation metrics: MAE and RMSE. In all the tables reporting results, the symbol $\triangledown$ (or $\triangle$) indicate when the performance of the algorithm indicated in the column is significantly worst (or better) at a significance level of 95% than the performance of the reference algorithm.

The set of rules learned by AMRules can be ordered or unordered. As they use different learning strategies, they must employ different prediction strategies to achieve optimal prediction. In the former, only the first rule that covers an example is used to predict the example target. In the latter, all rules covering the example are used for prediction and the final target value is decided by a weighted vote.

In regression, the target attribute is numerical, and the loss function is typically measured in terms of the absolute or squared difference between the predicted value and the true output. Corresponding prediction problems can be solved in three ways. In the first method, the target value can be estimated by the weighted mean of the target values of the examples covered by the rule. The second method generates predictions that are the output of the linear models associated with each rule. The third strategy is a combination of these two strategies. When a sample arrives, the absolute or squared difference between predicted and true output is computed using these two strategies, then the one with best results is chosen.

4.2. Experimental Results

In this section, we empirically evaluate the adaptive model rules algorithm. The results come in four parts.

(1) Which is the best strategy to grow rule sets? In the first set of experiments, we compare the AMRules variants.
(2) How do AMRules compare against others streaming algorithms?
(3) How do AMRules compare against others state-of-the-art regression algorithms?
(4) What is the impact of change and anomaly detection in time-evolving data streams?

4.2.1. Comparison between AMRules Variants: Ordered versus Unordered Rule Sets. In this section, we focus on two strategies that we found potentially interesting: use only the first rule that covers an example both for training and predicting; and update the set of rules that covers an example while training and the same set to obtain the prediction using a weighted vote. The former strategy implies using ordered rules (AMRules^o), and the latter using an unordered rule set (AMRules^u). The weights of the votes $w_r \in [0, 1]$ for AMRules^u are inversely proportional to the estimated MAE e_r of each rule r. Let CR be the set of rules that covers a given test example. The weighted prediction of

Table II. Comparison between AMRules Variants: Ordered versus Unordered Rule Sets

	MAE (variance)		RMSE (variance)	
Dataset	AMRules^o	AMRules^u	AMRules^o	AMRules^u
2dplanes	9.41E-01 (4.94E-03)	▽ 1.33E+00 (8.82E-03)	1.22E+00 (1.52E-02)	▽ 1.76E+00 (2.66E-02)
Ailerons	1.61E-04 (1.08E-09)	1.69E-04 (3.20E-09)	4.01E-04 (9.87E-08)	7.79E-04 (2.26E-06)
Bank8FM	2.54E-02 (1.60E-06)	▽ 2.68E-02 (5.29E-06)	3.50E-02 (7.78E-06)	3.67E-02 (4.76E-05)
CalHousing	5.90E+04 (1.60E+08)	5.74E+04 (2.87E+08)	8.06E+04 (2.98E+08)	7.82E+04 (5.21E+08)
Elevators	2.50E-03 (2.78E-07)	2.80E-03 (1.78E-07)	5.00E-03 (2.13E-05)	5.20E-03 (2.11E-05)
Fried	1.87E+00 (1.53E-03)	1.88E+00 (1.76E-03)	2.41E+00 (2.21E-03)	2.43E+00 (3.79E-03)
House8L	2.18E+04 (7.15E+05)	2.18E+04 (5.68E+06)	4.12E+04 (6.42E+07)	4.17E+04 (2.17E+07)
House16H	2.45E+04 (2.22E+06)	2.48E+04 (1.57E+06)	4.37E+04 (3.83E+06)	▽ 4.53E+04 (7.91E+06)
Kin8nm	1.60E-01 (1.27E-05)	△ 1.59E-01 (1.29E-05)	2.01E-01 (2.63E-05)	2.00E-01 (2.71E-05)
MV	1.06E+00 (1.19E-01)	1.06E+00 (7.90E-02)	1.70E+00 (3.24E-01)	1.73E+00 (2.15E-01)
Pol	1.00E+01 (1.15E+00)	▽ 1.13E+01 (8.18E+00)	1.76E+01 (5.32E+00)	▽ 1.94E+01 (9.69E+00)
Puma8NH	3.07E+00 (2.14E-02)	▽ 3.21E+00 (2.64E-02)	3.82E+00 (2.52E-02)	▽ 4.02E+00 (4.30E-02)
Puma32H	1.33E-02 (6.78E-07)	▽ 1.50E-02 (2.22E-06)	1.74E-02 (1.82E-06)	▽ 2.02E-02 (7.73E-06)
FriedD	1.862	1.912	2.410	2.468
WaveformD	0.414	0.462	0.555	0.586
Airline	14.779	14.491	26.551	26.509
Average Rank	1.12	1.88	1.18	1.82
Sig.Diffs (W/L)	-	1/5	-	0/5

AMRules^u is computed as

$$y = \sum_{r \in CR} w_r y_r, \tag{2}$$

$$w_r = \frac{(e_r + \epsilon)^{-1}}{\sum_{i \in CR} (e_i + \epsilon)^{-1}}, \tag{3}$$

where ϵ is a small positive number used to prevent numerical instabilities.

Ordered rule sets specialize one rule at time. As a result they often produce fewer rules than the unordered strategy. Ordered rules need to consider the previous rules and remaining combinations, which might not be easy to interpret in more complex sets. Unordered rule sets are more modular, because they can be interpreted independently.

Table II summarizes the MAE and the RMSE of these variants, and the corresponding variances. The results for the first 13 datasets were obtained using the standard method of tenfold cross-validation, using the same folds for all the experiments included in the study. For the remaining three datasets, which are time-evolving data streams, we present the average prequential error computed over a sliding window of 10,000 instances using a sampling frequency of the same size. The symbols △ and ▽ identify the datasets in which AMRules^u is better or worst than AMRules^o with statistical significance. The last two rows of the table present the average rank of the approaches, and the number of times that AMRules^u was underperformed/outperformed with statistical significance by AMRules^o.

Overall, the experimental results point out that ordered rule sets are more competitive than unordered rule sets in terms of both MAE and RMSE. AMRules^u was significantly better than AMRules^o only in the Kin8nm dataset according to MAE, while AMRules^o outperformed (with statistical significance) AMRules^u in five datasets considering both the MAE and RMSE performance measures.

4.2.2. Comparison between AMRules Variants: Adaptive Model versus Target Mean. Table III compares the results obtained by the AMRules^u using the adaptive and target mean AMRules^{TM} prediction strategies. The adaptive prediction strategy is clearly better than using the rule's target mean. The ordered version achieved the best results in all datasets according to MAE, always with statistical significance in the tenfold

Table III. Comparison between AMRules Variants: Adaptive versus Target Mean Prediction Strategies

	MAE (variance)		RMSE (variance)	
Dataset	AMRules^{o}	AMRules^{TM}	AMRules^{o}	AMRules^{TM}
2dplanes	9.41E-01 (4.94E-03)	▽ 1.48E+00 (1.39E-02)	1.22E+00 (1.52E-02)	▽ 1.92E+00 (2.73E-02)
Ailerons	1.61E-04 (1.08E-09)	▽ 2.67E-04 (2.81E-09)	4.01E-04 (9.87E-08)	3.54E-04 (2.40E-09)
Bank8FM	2.54E-02 (1.60E-06)	▽ 5.83E-02 (6.67E-05)	3.50E-02 (7.78E-06)	▽ 7.95E-02 (1.40E-04)
CalHousing	5.90E+04 (1.60E+08)	▽ 8.41E+04 (4.81E+08)	8.06E+04 (2.98E+08)	▽ 1.04E+05 (6.52E+08)
Elevators	2.50E-03 (2.78E-07)	▽ 4.30E-03 (4.56E-07)	5.00E-03 (2.13E-05)	6.10E-03 (1.21E-06)
Fried	1.87E+00 (1.53E-03)	▽ 2.72E+00 (2.97E-02)	2.41E+00 (2.21E-03)	▽ 3.40E+00 (4.69E-02)
House8L	2.18E+04 (7.15E+05)	▽ 2.64E+04 (7.61E+06)	4.12E+04 (6.42E+07)	4.47E+04 (1.46E+07)
House16H	2.45E+04 (2.22E+06)	▽ 3.16E+04 (1.07E+07)	4.37E+04 (3.83E+06)	▽ 5.07E+04 (9.96E+06)
Kin8nm	1.60E-01 (1.27E-05)	▽ 1.84E-01 (2.13E-05)	2.01E-01 (2.63E-05)	▽ 2.26E-01 (2.32E-05)
MV	1.06E+00 (1.19E-01)	▽ 4.03E+00 (1.33E+00)	1.70E+00 (3.24E-01)	▽ 6.24E+00 (1.95E+00)
Pol	1.00E+01 (1.15E+00)	▽ 1.48E+01 (6.93E+00)	1.76E+01 (5.32E+00)	▽ 2.47E+01 (1.42E+01)
Puma8NH	3.07E+00 (2.14E-02)	▽ 3.49E+00 (2.43E-02)	3.82E+00 (2.52E-02)	▽ 4.37E+00 (2.01E-02)
Puma32H	1.33E-02 (6.78E-07)	▽ 1.62E-02 (1.33E-05)	1.74E-02 (1.82E-06)	▽ 2.15E-02 (3.69E-05)
FriedD	1.862	2.740	2.410	3.440
WaveformD	0.414	0.503	0.555	0.638
Airline	14.779	16.081	26.551	27.520
Average Rank	1.00	2.00	1.07	1.93
Sig.Diffs (W/L)	-	0/13	-	0/10

Table IV. Comparison between AMRules^{o} and Other Streaming Regression Algorithms

	RMSE (variance)			
Dataset	AMRules^{o}	FIMTDD	IBLStreams	Perceptron
2dplanes	1.22E+00 (1.52E-02)	△ 1.04E+00 (9.65E-04)	▽ 1.37E+00 (9.68E-05)	▽ 2.39E+00 (1.06E-03)
Ailerons	4.01E-04 (9.87E-08)	4.14E-02 (1.36E-02)	△ 0.00E+00 (0.00E+00)	1.14E-03 (3.66E-06)
Bank8FM	3.50E-02 (7.78E-06)	4.02E-02 (9.93E-05)	▽ 6.76E-02 (2.87E-05)	▽ 3.92E-02 (1.29E-06)
CalHousing	8.06E+04 (2.98E+08)	▽ 1.45E+05 (5.33E+09)	▽ 1.09E+05 (5.22E+08)	7.51E+04 (3.09E+08)
Elevators	5.00E-03 (2.13E-05)	2.12E+00 (9.51E+00)	5.80E-03 (4.00E-07)	5.70E-03 (3.36E-05)
Fried	2.41E+00 (2.21E-03)	2.18E+00 (2.50E-01)	△ 2.13E+00 (9.62E-03)	▽ 2.64E+00 (2.46E-04)
House8L	4.12E+04 (6.42E+07)	4.34E+04 (4.36E+08)	▽ 5.12E+04 (3.51E+07)	4.28E+04 (5.31E+06)
House16H	4.37E+04 (3.83E+06)	6.83E+04 (5.12E+09)	▽ 7.04E+04 (3.66E+07)	▽ 4.84E+04 (3.75E+07)
Kin8nm	2.01E-01 (2.63E-05)	2.17E-01 (5.87E-03)	△ 1.38E-01 (1.08E-04)	2.03E-01 (1.77E-05)
MV	1.70E+00 (3.24E-01)	1.35E+00 (4.55E+00)	▽ 3.12E+00 (9.33E-03)	▽ 4.50E+00 (6.72E-03)
Pol	1.76E+01 (5.32E+00)	2.21E+01 (3.98E+01)	▽ 2.91E+01 (4.95E-01)	▽ 3.10E+01 (1.69E-01)
Puma8NH	3.82E+00 (2.52E-02)	△ 3.39E+00 (1.39E-02)	▽ 4.35E+00 (3.70E-02)	▽ 4.48E+00 (1.67E-02)
Puma32H	1.74E-02 (1.82E-06)	1.23E+00 (2.30E+00)	▽ 3.85E-02 (1.03E-05)	▽ 2.76E-02 (4.89E-07)
FriedD	2.410	12.628	2.365	2.644
WaveformD	0.555	7.256	1.259	0.647
Airline	26.551	106.949	29.876	26.967
Average Rank	1.57	2.19	1.88	2.75
Sig.Diffs (W/L)	-	2/1	3/9	0/8

cross-validation evaluation. Regarding the RMSE, the results were identical with the following exceptions: AMRules^{TM} was better than AMRules^{o} in the Ailerons dataset; and AMRules^{o} outperformed AMRules^{TM} in all the remaining datasets, but in three of these, the difference was not statistically significant.

4.2.3. Comparison with others Streaming Algorithms. We compare the performance of our algorithm with three others streaming algorithms, FIMTDD, IBLStreams, and Perceptron. FIMTDD is an incremental algorithm for learning model trees, described in Ikonomovska et al. [2011]. IBLStreams is an extension of MOA that consists of an instance-based learning algorithm for classification and regression problems on data streams by Shaker and Hüllermeier [2012]. Perceptron is the linear model used by AMRules. The RMSE evaluation for these algorithms is given in Table IV. The AMRules^{o} produces better overall results since it has the lowest average rank. Considering the 10-fold cross-validation evaluation, AMRules^{o} was significantly better than FIMTDD,

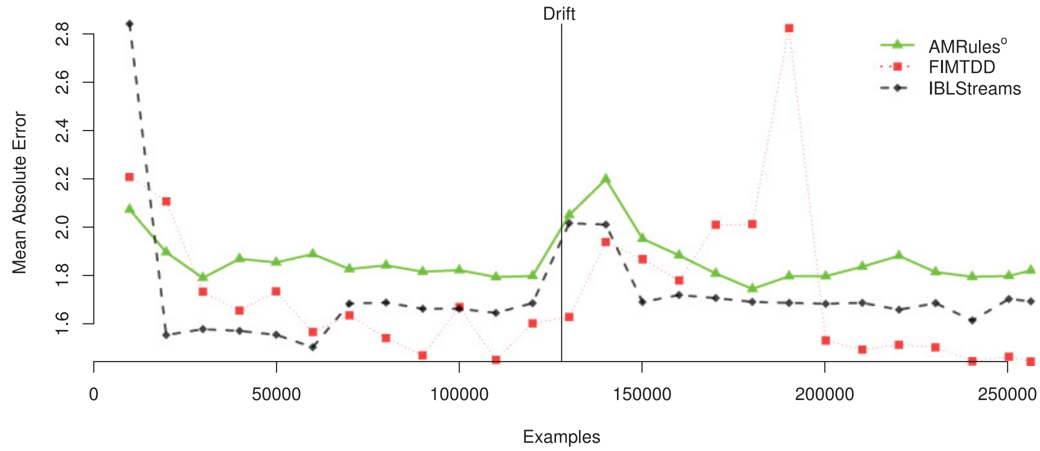

Fig. 1. Evolution of the prequential MAE of streaming algorithms using the dataset FriedD.

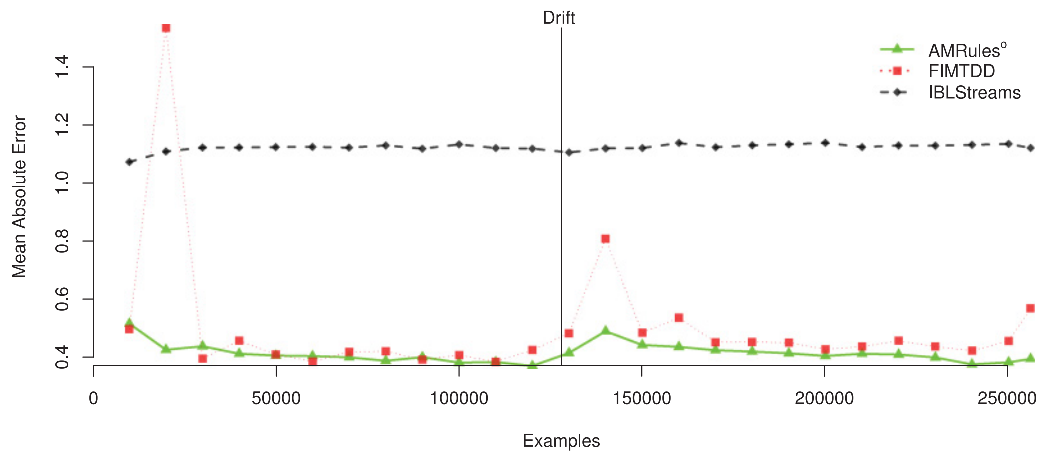

Fig. 2. Evolution of the prequential MAE of streaming algorithms using the dataset WaveformD.

IBLStreams, and Perceptron in one, nine, and eight datasets, respectively, while being significantly worst only in two, three, and zero datasets, respectively.

Figures 1–3 show the evolution of the prequential MAE for the streaming algorithms with time-evolving data streams. Figure 1 depicts the prequential MAE curves using the dataset FriedD, and also illustrates the change point, i.e., the moment the drift begins. It is expected that the MAE of the learning algorithms start high for the first examples, then decrease and stabilize, increased again when the drift occurs, and finally, decrease and stabilize. The AMRules^o and IBLStreams followed this behavior, but not the FIMTDD algorithm which had a huge peak in MAE around the 190,000 examples. In terms of the average MAE, the FIMTDD and IBLStreams performed better than AMRules^o since the average prequential MAE were 1.723, 1.725, and 1.862, respectively. Figure 2 shows the prequential MAE curves for the WaveformD, which also contains a drift starting in the 128,001st example. In this dataset, the performance of AMRules^o and FIMTDD is clearly superior to the performance of IBLStreams. The MAE increased in both AMRules^o and FIMTDD after the drift, but the magnitude was clearly smaller in the case of AMRules^o. FIMTDD also presents an unexpected peak

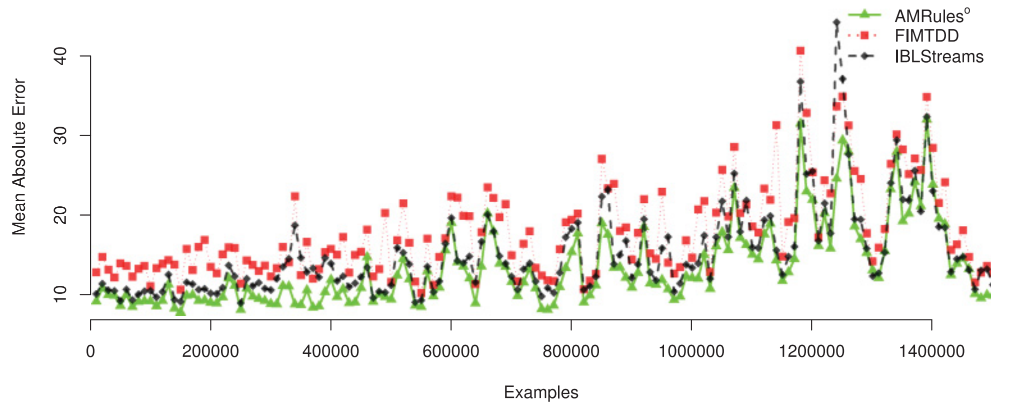

Fig. 3. Evolution of the prequential MAE of streaming algorithms using the dataset Airline.

Table V. Comparison between AMRules^o and Batch Regression Algorithms

	RMSE (variance)			
Dataset	AMRules^o	M5Rules	MLP	OLS
2dplanes	1.22E+00 (1.52E-02)	△ 9.97E-01 (1.15E-04)	1.15E+00 (3.97E-03)	▽ 2.38E+00 (1.04E-03)
Ailerons	4.01E-04 (9.87E-08)	△ 1.80E-04 (1.78E-09)	△ 1.90E-04 (1.00E-09)	2.00E-04 (0.00E+00)
Bank8FM	3.50E-02 (7.78E-06)	△ 3.07E-02 (2.10E-06)	3.36E-02 (1.12E-05)	▽ 3.88E-02 (1.86E-06)
CalHousing	8.06E+04 (2.98E+08)	△ 6.90E+04 (1.32E+08)	9.20E+04 (9.49E+08)	△ 7.03E+04 (1.62E+08)
Elevators	5.00E-03 (2.13E-05)	△ 2.31E-03 (7.88E-08)	△ 2.39E-03 (1.21E-08)	△ 2.90E-03 (1.98E-07)
Fried	2.41E+00 (2.21E-03)	△ 1.61E+00 (4.30E-04)	△ 1.70E+00 (6.69E-02)	▽ 2.63E+00 (2.29E-04)
House8L	4.12E+04 (6.42E+07)	△ 3.23E+04 (1.39E+06)	△ 3.54E+04 (4.79E+06)	4.16E+04 (1.49E+06)
House16H	4.37E+04 (3.83E+06)	△ 3.71E+04 (2.41E+06)	△ 3.90E+04 (1.06E+06)	▽ 4.55E+04 (2.09E+06)
Kin8nm	2.01E-01 (2.63E-05)	△ 1.72E-01 (5.12E-05)	△ 1.63E-01 (1.08E-04)	2.02E-01 (2.10E-05)
MV	1.70E+00 (3.24E-01)	△ 1.97E-02 (4.02E-04)	△ 1.62E-01 (5.79E-04)	▽ 4.49E+00 (6.29E-03)
Pol	1.76E+01 (5.32E+00)	△ 6.64E+00 (6.62E-01)	△ 1.28E+01 (2.84E+00)	▽ 3.05E+01 (1.57E-01)
Puma8NH	3.82E+00 (2.52E-02)	△ 3.20E+00 (3.56E-03)	4.04E+00 (1.69E-01)	▽ 4.46E+00 (1.41E-02)
Puma32H	1.74E-02 (1.82E-06)	△ 8.57E-03 (9.79E-08)	▽ 3.33E-02 (2.25E-06)	▽ 2.68E-02 (3.89E-07)
Average Rank	3.00	1.08	2.31	3.62
Sig.Diffs (W/L)	-	13/0	8/1	2/8

around the 20,000 examples, which may point out some instabilities in the algorithm. Figure 3 presents the MAE curves for the Airline dataset (first 1.5-million examples), which is a real-world problem as described before. The MAE curves have a lot of peaks, which means that the stream is changing over time. As can be seen, in this dataset AMRules^o outperforms the other algorithms since its curve is almost always below the other algorithms' curves and the magnitude of the MAE peaks is also smaller.

4.2.4. Comparison with Others State-of-the-art Regression Algorithms. We compared AMRules with other non-incremental regression algorithms from WEKA [Hall et al. 2009]: M5Rules, Multilayer Perceptron (MLP), and Linear Regression (OLS). The summary of the RMSE results is presented in Table V.

The analysis of these results show that AMRules has, in general, higher RMSE than M5Rules and MLP and higher performance than OLS. Despite not achieving the best average rank, AMRules^o is competitive with batch regression algorithms, being significantly better than OLS in 8 out of 13 datasets. These results were somewhat expected, even in these small datasets, due to the generalization ability of model rules.

4.2.5. Comparison between AMRules Variants: Change Detection versus no Change Detection. Table VI compares the RMSE results achieved by the AMRules^u and a similar version without change detection, in this case, without the PH test (AMRules^{PH}). As

Table VI. Impact of Change Detection

Dataset	Number of Alarms	RMSE (variance) AMRules^{o}	AMRules$^{\neg PH}$
2dplanes	0.1	1.22E+00 (1.52E-02)	1.19E+00 (1.03E-02)
Ailerons	0.6	4.01E-04 (9.87E-08)	3.89E-04 (1.02E-07)
Bank8FM	0	3.50E-02 (7.78E-06)	3.50E-02 (7.78E-06)
CalHousing	0.1	8.06E+04 (2.98E+08)	8.23E+04 (2.69E+08)
Elevators	0	5.00E-03 (2.13E-05)	5.00E-03 (2.13E-05)
Fried	0	2.41E+00 (2.21E-03)	2.41E+00 (2.21E-03)
House8L	0	4.12E+04 (6.42E+07)	4.12E+04 (6.42E+07)
House16H	0	4.37E+04 (3.83E+06)	4.37E+04 (3.83E+06)
Kin8nm	0	2.01E-01 (2.63E-05)	2.01E-01 (2.63E-05)
MV	1.2	1.70E+00 (3.24E-01)	1.58E+00 (1.59E-01)
Pol	0	1.76E+01 (5.32E+00)	1.76E+01 (5.32E+00)
Puma8NH	0	3.82E+00 (2.52E-02)	3.82E+00 (2.52E-02)
Puma32H	0	1.74E-02 (1.82E-06)	1.74E-02 (1.82E-06)
FriedD	3	2.410	2.396
WaveformD	4	0.555	0.557
Airline	2558	26.551	26.545
Average Rank		1.60	1.40
Sig.Diffs (W/L)		-	0/0

expected, the number of alarms for the smaller datasets is very small as these datasets are not time-evolving data streams. As result, the differences between AMRules^{o} and AMRules^{PH} in terms of RMSE have no statistically significance. Regarding the time-evolving datasets, the results for the FriedD and Airline datasets were better without using change detection. This indicates that, in these cases, that have only one drift, the rule set adapted to the change faster than pruning the rule set and start learning new rules from scratch. Note that in AMRules, several alarms may (and should) occur for the same drift, as each rule has its own change detector.

4.3. Anomaly Detection

We evaluate the anomaly detection algorithm embedded in AMRules^{o} on a set of regression problems. The results are presented in Table VII, showing the number of anomalies detected, and the prequential RMSE setting on/off the ability to detect anomalies. In these datasets, no anomalies were detected except for the CalHousing, House8L and Airline datasets. The number of anomalies was very small compared to the size of the dataset and, consequently, the average RMSE values were similar.

Two examples of anomalies detected in the Airline dataset are presented below.

Case: 29256 **Anomaly Score:** -1.93
Rule: $x7 <= 1156$ and $x8 <= 66$ and $x5 <= 1840 \rightarrow y : 5.69$
$x3 = 3 \quad (2.01 \pm 0.09) \quad Prob = 0.018\%$
$x4 = 6 \quad (5.01 \pm 0.03) \quad Prob = 0.018\%$
$x5 = 45 \quad (1680.67 \pm 179.83) \quad Prob = 0.023\%$
$x6 = 12 \quad (1762.60 \pm 186.58) \quad Prob = 0.022\%.$

Case: 541603 **Anomaly Score:** -3.33
Rule: $x4 > 4$ and $x6 <= 1610 \rightarrow y : 5.05$
$x5 = 1755 \quad (1456.6 \pm 33.2) \quad Prob = 0.024\%$
$x6 = 554 \quad (1566.5 \pm 27.5) \quad Prob = 0.001\%$
$x8 = 483 \quad (79.23 \pm 11.8) \quad Prob = 0.002\%$
$x9 = 4243 \quad (390.7 \pm 91.7) \quad Prob = 0.001\%.$

Table VII. The Impact of Anomaly Detection: Results of Tenfold Cross-Validation for AMRules Algorithms

Dataset	Number of Anomalies	RMSE (variance) AMRules^o	AMRules$^{\neg Anom.}$
2dplanes	0	1.22E+00 (1.52E-02)	1.22E+00 (1.52E-02)
Ailerons	0	4.01E-04 (9.87E-08)	4.01E-04 (9.87E-08)
Bank8FM	0	3.50E-02 (7.78E-06)	3.50E-02 (7.78E-06)
CalHousing	35.3	8.06E+04 (2.98E+08)	8.23E+04 (5.73E+08)
Elevators	0	5.00E-03 (2.13E-05)	5.00E-03 (2.13E-05)
Fried	0	2.41E+00 (2.21E-03)	2.41E+00 (2.21E-03)
House8L	0.1	4.12E+04 (6.42E+07)	4.12E+04 (6.42E+07)
House16H	0	4.37E+04 (3.83E+06)	4.37E+04 (3.83E+06)
Kin8nm	0	2.01E-01 (2.63E-05)	2.01E-01 (2.63E-05)
MV	0	1.70E+00 (3.24E-01)	1.70E+00 (3.24E-01)
Pol	0	1.76E+01 (5.32E+00)	1.76E+01 (5.32E+00)
Puma8NH	0	3.82E+00 (2.52E-02)	3.82E+00 (2.52E-02)
Puma32H	0	1.74E-02 (1.82E-06)	1.74E-02 (1.82E-06)
FriedD	0	2.410	2.410
WaveformD	0	0.555	0.555
Airline	294194	26.551	26.535
Average Rank		1.40	1.60
Sig.Diffs (W/L)		-	0/0

Table VIII. Comparison between AMRules^o and RAMRules^o

Dataset	RMSE (variance) AMRules^o	RAMRules^o
2dplanes	1.22E+00 (1.52E-02)	1.23E+00 (7.52E-04)
Ailerons	4.01E-04 (9.87E-08)	4.43E-04 (1.24E-07)
Bank8FM	3.50E-02 (7.78E-06)	▽ 3.88E-02 (8.44E-07)
CalHousing	8.06E+04 (2.98E+08)	7.62E+04 (3.27E+08)
Elevators	5.00E-03 (2.13E-05)	4.50E-03 (1.38E-05)
Fried	2.41E+00 (2.21E-03)	△ 1.95E+00 (1.92E-04)
House8L	4.12E+04 (6.42E+07)	3.81E+04 (3.46E+06)
House16H	4.37E+04 (3.83E+06)	4.42E+04 (1.09E+07)
Kin8nm	2.01E-01 (2.63E-05)	△ 1.97E-01 (2.37E-05)
MV	1.70E+00 (3.24E-01)	▽ 3.45E+00 (1.06E-02)
Pol	1.76E+01 (5.32E+00)	▽ 2.26E+01 (2.11E-01)
Puma8NH	3.82E+00 (2.52E-02)	▽ 4.14E+00 (1.21E-02)
Puma32H	1.74E-02 (1.82E-06)	▽ 2.73E-02 (4.56E-07)
FriedD	2.410	2.171
WaveformD	0.555	0.548
Airline (1M)	20.058	19.688
Average Rank	1.50	1.50
Sig.Diffs (W/L)	-	2/5

4.4. Ensembles of AMRules

We compared the performance of single and ensemble rule sets produced using adaptive model rules. The size of the subset of attributes defined for our experiments was 63.2% of the total number of attributes. The results in Tables VIII and IX report ensembles of 50 AMRules. For the Airline dataset, only the first million examples of the original data set were used to evaluate the performance of Random AMRules. The results for the smaller datasets show that the performance of Random AMRules and AMRules are similar regarding the average rank for the ordered rule sets. Regarding the unordered rule sets, the ensemble methods performed a little better than the base

Table IX. Comparison between AMRules^u and RAMRules^u

	RMSE (variance)	
Dataset	AMRules^u	RAMRules^u
2dplanes	1.76E+00 (2.66E-02)	△ 1.41E+00 (6.66E-04)
Ailerons	7.79E-04 (2.26E-06)	4.36E-04 (9.91E-08)
Bank8FM	3.67E-02 (4.76E-05)	3.90E-02 (8.89E-07)
CalHousing	7.82E+04 (5.21E+08)	7.53E+04 (3.15E+08)
Elevators	5.20E-03 (2.11E-05)	4.60E-03 (1.63E-05)
Fried	2.43E+00 (3.79E-03)	△ 2.16E+00 (2.34E-04)
House8L	4.17E+04 (2.17E+07)	3.82E+04 (3.07E+06)
House16H	4.53E+04 (7.91E+06)	4.45E+04 (1.02E+07)
Kin8nm	2.00E-01 (2.71E-05)	△ 1.97E-01 (2.29E-05)
MV	1.73E+00 (2.15E-01)	▽ 3.51E+00 (5.25E-03)
Pol	1.94E+01 (9.69E+00)	▽ 2.64E+01 (8.24E-01)
Puma8NH	4.02E+00 (4.30E-02)	4.16E+00 (1.46E-02)
Puma32H	2.02E-02 (7.73E-06)	▽ 2.74E-02 (4.89E-07)
FriedD	2.468	2.324
WaveformD	0.586	0.550
Airline (1M)	19.666	19.706
Average Rank	1.63	1.37
Sig.Diffs (W/L)	-	3/3

Table X. Number of Rules for the Variants of AMRules and RAMRules

	Number of rules						
Dataset	AMRules^o	AMRules^u	AMRules$^{\neg PH}$	AMRules$^{\neg Anom.}$	AMRules^{TM}	RAMRules^o	RAMRules^u
2dplanes	20.8	49.5	20.8	20.8	19.4	855.2	954.9
Ailerons	2.9	2.8	3.3	2.9	2.6	101.7	102.3
Bank8FM	5.2	6.3	5.2	5.2	5.2	168.5	172.2
CalHousing	8.4	10.2	8.6	8.1	6.8	871.8	890.8
Elevators	2.8	2.8	2.8	2.8	2.3	169.1	169.1
Fried	8.5	11.9	8.5	8.5	7.9	545.5	619.2
House8L	3.4	4.2	3.4	3.4	3.4	187.4	196.3
House16H	3.0	3.0	3.0	3.0	3.0	227.7	227.3
Kin8nm	3.0	3.0	3.0	3.0	3.0	162.4	161.0
MV	11.7	14.9	12.9	11.7	12.7	391.3	471.9
Pol	4.7	5.3	4.7	4.7	4.7	203.2	178.6
Puma8NH	4.6	6.0	4.6	4.6	4.6	212.1	231.6
Puma32H	8.7	9.3	8.7	8.7	8.7	137.7	137.4
FriedD	25	34	29	25	25	2169	2972
WaveformD	13	14	15	13	11	1883	1985
Airline (1M)	37	58	49	38	41	5901	6252

learners individually. For the time-evolving data streams, Random AMRules outperformed AMRules in all datasets excepting Airlines using unordered rule sets.

4.5. Model Complexity in Terms of Number of Rules

Table X presents the model complexity of the variants of AMRules and RAMRules. By comparing the number of rules of the ordered and unordered rule sets, it can be seen that the number of rules of unordered rule sets tend to be higher than the number of rules of ordered ones, especially in the larger datasets. The AMRules version without change detection usually has more rules than the one equipped with change detection, which is expected since when change is detected the rule is eliminated from the rule set. The complexity of AMRules using the adaptive model and the target mean approaches is similar. Only the Ailerons and Elevators datasets have significant differences (in proportion) in the number of rules. The number of rules of the ensemble methods

Table XI. Relative Learning Times of the Experiments Reported

	Relative Learning Times									
Dataset	AMRules^o	AMRules^u	FIMTDD	IBLStreams	Perceptron	M5Rules	MLP	OLS	RAMRules^o	RAMRules^u
2dplanes	1	1.355	0.602	16.82	0.351	67.25	10.42	0.165	16.1	16.8
Ailerons	1	0.996	0.524	3.85	0.379	5.21	37.53	0.253	17.6	17.4
Bank8FM	1	1.031	0.598	7.13	0.412	29.49	2.29	0.128	11.6	12.4
CalHousing	1	1.071	0.638	2.51	0.388	147.91	4.74	0.138	24.4	24.0
Elevators	1	1.054	0.620	4.43	0.433	12.16	13.36	0.175	21.9	22.4
Fried	1	1.182	0.737	17.73	0.382	1097.27	11.19	0.187	16.5	16.7
House8L	1	1.106	0.721	2.50	0.431	54.01	5.71	0.169	27.8	27.6
House16H	1	1.036	0.698	3.07	0.415	49.33	13.79	0.166	19.1	19.8
Kin8nm	1	1.016	0.697	13.16	0.484	47.53	2.64	0.144	13.1	13.9
MV	1	1.122	0.667	16.57	0.361	57.62	12.90	0.178	15.1	18.6
Pol	1	1.049	0.572	10.74	0.416	11.11	63.79	0.178	21.6	18.3
Puma8NH	1	1.035	0.642	10.76	0.437	31.77	2.32	0.145	12.4	13.4
Puma32H	1	1.033	0.544	6.68	0.351	38.36	15.48	0.171	14.0	14.6
FriedD	1	1.17	2.39	79.31	0.14	-	-	-	65.70	84.20
WaveformD	1	1.24	14.97	106.14	0.20	-	-	-	76.60	106.24
Airline (1M)	1	1.15	0.29	8.72	0.07	-	-	-	98.44	131.92

is clearly higher than the number of rules of AMRules, both using the ordered and unordered sets. This is expected as each ensemble is composed of 50 base learners.

4.6. Learning Times

Table XI reports the relative learning times required for the tenfold cross-validation and prequential evaluation. As AMRules^o generates fewer rules than AMRules^u, it is slightly faster. FIMTDD is usually faster than AMRules^o. However, for the FriedD and WaveformD datasets, AMRules^o performed considerably faster. Being one-pass algorithms, both versions of AMRules are much faster than M5 Rules and MLP. The faster algorithms were the simpler ones, OLS and Perceptron, and the slower ones were the ensembles methods and IBLStreams. Surprisingly, Random AMRules had inferior learning times than IBLStreams in some smaller datasets, despite consisting of ensembles with 50 base learners.

The throughput of AMRules depends on the characteristics of the data stream, mainly on the number of attributes, and the number of rules. In this set of experiences, AMRules processes, on average, around 5k examples per second. Airline is the largest dataset, in terms of the number of examples. AMRules processes more than 8K examples per second in this dataset. Pol is the dataset with largest number of attributes and its throughput is around 3K examples per second. Note that the algorithm was implemented using MOA framework that is designed to run algorithms in a single machine, and the experiments were run in a desktop personal computer (Intel Core i7-4770 CPU, 16-GB RAM). Since AMRules is highly parallelizable (each rule can be learned individually), it could be easily scaled up into multiple machines using a distributed streaming processing engine.

5. CONCLUSIONS

Regression rules are expressive representations of generalizations from examples. Learning regression rules from data streams is an interesting research line that has not been widely explored by the stream mining community. To the best of our knowledge, in the literature there is no method that addresses this issue. In this article, we present a new regression model rules algorithm for streaming and evolving data. The AMRules algorithm is a one-pass algorithm, able to adapt the current rule set to changes in the process generating examples. It is able to induce ordered and unordered rule sets, where the consequent of a rule contains a linear model trained with the perceptron rule.

The experimental results indicate that, in comparison to unordered rule sets, ordered rule sets are more competitive in terms of performance (MAE and RMSE). AMRules is competitive against batch learners even for medium-sized datasets.

A new ensemble method inspired by Random Forests was also introduced and evaluated. Experimental results shown it reduces both MAE and RMSE in time-evolving data streams.

REFERENCES

Ezilda Almeida, Carlos Abreu Ferreira, and João Gama. 2013a. Adaptive model rules from data streams. In *Machine Learning and Knowledge Discovery in Databases (Lecture Notes in Computer Science)*, Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Zelezný (Eds.), Vol. 8188. Springer, 480–492. DOI:http://dx.doi.org/10.1007/978-3-642-40988-2_31

Ezilda Almeida, Petr Kosina, and João Gama. 2013b. Random rules from data streams. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC'13, Coimbra, Portugal, March 18-22, 2013*, Sung Y. Shin and José Carlos Maldonado (Eds.). ACM, 813–814. DOI:http://dx.doi.org/10.1145/2480362.2480518

K. Bache and M. Lichman. 2013. UCI Machine Learning Repository. Retrieved from http://archive.ics.uci.edu/ml.

B. B. Bhattacharyya. 1987. One sided Chebyshev inequality when the first four moments are known. *Commun. Statist.—Theory Methods* 16, 9 (1987), 2789–2791.

Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. MOA: Massive online analysis. *J. Mach. Learn. Res.* 11 (2010), 1601–1604.

Leo Breiman. 1996. Bagging predictors. *Mach. Learn.* 24, 2 (1996), 123–140. DOI:http://dx.doi.org/10.1007/BF00058655

Leo Breiman. 2001. Random forests. *Mach. Learn.* 45, 1 (2001), 5–32.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. 1984. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA. 238 pages.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Comput. Surv.* 41, 3 (July 2009), Article 15, 58 pages. DOI:http://dx.doi.org/10.1145/1541880.1541882

Pedro Domingos and Geoff Hulten. 2000. Mining high-speed data streams. In *Proceedings of the ACM 6th International Conference on Knowledge Discovery and Data Mining*, Ismail Parsa, Raghu Ramakrishnan, and Sal Stolfo (Eds.). ACM Press, Boston, MA, USA, 71–80.

Francisco J. Ferrer-Troyano, Jesús S Aguilar-Ruiz, and José Cristóbal Riquelme Santos. 2005. Incremental rule learning and border examples selection from numerical data streams. *J. Universal Comput. Sci.* 11, 8 (2005), 1426–1439.

Eibe Frank, Yong Wang, Stuart Inglis, Geoffrey Holmes, and Ian H. Witten. 1998. Using model trees for classification. *Mach. Learn.* 32, 1 (1998), 63–76.

Johannes Fürnkranz, Dragan Gamberger, and Nada Lavra. 2012. *Foundations of Rule Learning*. Springer.

João Gama. 2010. *Knowledge Discovery from Data Streams*. CRC Press.

João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. 2013. On evaluating stream learning algorithms. *Mach. Learn.* 90, 3 (2013), 317–346.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.* 11, (2009), 10–18.

V. J. Hodge and J. Austin. 2004. A survey of outlier detection methodologies. *Artificial Intelligence Rev.* 22, 2 (2004), 85–126.

Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *J. Am. Statist. Assoc.* 58, 301 (1963), 13–30.

Elena Ikonomovska, João Gama, and Saso Dzeroski. 2011. Learning model trees from evolving data streams. *Data Min. Knowl. Discov.* 23, 1 (2011), 128–168.

Ron Kohavi. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence—Volume 2 (IJCAI'95)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1137–1143.

P. Kosina and J. Gama. 2012. Very fast decision rules for multi-class problems. In *Proceedings of the 2012 ACM Symposium on Applied Computing*. ACM, New York, NY, USA, 795–800.

A. Liaw and M. Wiener. 2002. Classification and regression by random forest. *R News 2*, 3 (2002), 18–22.

H. Mouss, D. Mouss, N. Mouss, and L. Sefouhi. 2004. Test of page-Hinkley, an approach for fault detection in an agro-alimentary production system. In *Proceedings of the Asian Control Conference*, Vol. 2. 815–818.

ElMoustapha Ould-Ahmed-Vall, James Woodlee, Charles Yount, Kshitij A. Doshi, and Seth Abraham. 2007. Using model trees for computer architecture performance analysis of software applications. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems & Software ISPASS 2007*. IEEE, 116–125.

E. S. Page. 1954. Continuous inspection schemes. *Biometrika* 41, 1/2 (1954), 100–115.

Duncan Potts and Claude Sammut. 2005. Incremental learning of linear model trees. *Mach. Learn.* 61, 1–3 (2005), 5–48.

J. R. Quinlan. 1992. Learning with continuous classes. In *Proceedings of the Australian Joint Conference for Artificial Intelligence*. World Scientific, 343–348.

J. Ross Quinlan. 1993a. Combining instance-based and model-based learning. In *Proceedings of the 10th International Conference on Machine Learning, University of Massachusetts, Amherst, MA, USA, June 27–29, 1993*. Morgan Kaufmann, 236–243.

R. Quinlan. 1993b. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., San Mateo, CA.

Ammar Shaker and Eyke Hüllermeier. 2012. IBLStreams: A system for instance-based classification and regression on data streams. *Evol. Syst.* 3, (2012), 235–249.

Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. 2003. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM Press, Washington, D.C., 226–235. DOI:http://dx.doi.org/10.1145/956750.956778

Sholom M. Weiss and Nitin Indurkhya. 1995. Rule-based machine learning methods for functional prediction. *J. Artificial Intelligence Res.* 3 (1995), 383–403.

C. J. Willmott and K. Matsuura. 2005. Advantages of the mean absolute error (MAE) over the mean square error (RMSE) in assessing average model performance. *Climate Res.* 30 (2005), 79–82.

Received October 2014; revised February 2015; accepted September 2015

QUERIES

Q1: AU: Please supply the CCS Concepts 2012 codes per the ACM style indicated on the ACM website. Please include the CCS Concepts XML coding as well.
Q2: AU: Please provide the issue number in references “Bifet et al. 2010,” “Hall et al. 2009,” “Shaker and Hüllermeier 2012,” “Weiss and Indurkhya 1995,” and “Willmott and Matsuura 2005.”
Q3: AU: Please provide the detail of the publisher in reference “Mouss et al. 2004.”