Adaptive Model Rules from Data Streams

Ezilda Almeida[1], Carlos Ferreira[1], and João Gama[1,2]

[1] LIAAD-INESC TEC, University of Porto
ezildacv@gmail.com, cgf@isep.ipp.pt
[2] Faculty of Economics, University Porto
jgama@fep.up.pt

Abstract. Decision rules are one of the most expressive languages for machine learning. In this paper we present Adaptive Model Rules (AMRules), the first streaming 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 attribute values. Each rule uses a Page-Hinkley test to detect changes in the process generating data and react to changes by pruning the rule set. 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.

Keywords: Data Streams, Regression, Rule Learning

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 [18] and regression rules [19]. In [15], the authors present a large comparative study between several regression algorithms. Model trees and model rules are among the most performant ones. Trees and rules do 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. One important aspect of rules is modularity: each rule can be interpreted per si [6].

In the data stream computational model [7] examples are generated sequentially from time evolving distributions. Learning from data streams require incremental learning, using limited computational resources, and the ability to adapt to changes in the process generating data. In this paper we present the AMRules algorithm, the first one-pass algorithm for learning regression rule sets from time-evolving streams. It near follows FIMT [11], an algorithm to learn regression trees from data streams, and AVFDR [13], a one-pass algorithm for learning classification rules. AMRules can learn ordered or unordered rules. 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 an online change detector. The

change detector monitors the mean square error using the Page-Hinkley test, providing information about the dynamics of the process generating data.

The paper 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 describe 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. Last Section presents the lessons learned.

2 Related Work

In this section we analyze the related work in two dimensions: regression algorithms and incremental learning of regression algorithms.

In the field of machine learning, one of the most popular, and competitive, regression model is system M5, presented by [18]. It builds multivariate trees using linear models at the leaves. In the pruning phase for each leaf a linear model is built. Later, [5] have presented M5′ a *rational reconstruction* of Quinlan's M5 algorithm. 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. Then the linear regression models are 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 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, as it is done in [20]. 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 [22], 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 [19] 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 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. One of the first incremental model trees, was presented by [17]. 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 RLS (Recursive Least Square) algorithm.

The authors of [11] propose an incremental algorithm FIMT for any-time model trees learning from evolving data streams with drift detection. It is based on the Hoeffding tree algorithm [4], but implements a different splitting criterion, using a standard deviation reduction (SDR) based measure more appropriate to regression problems. The FIMT 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. FIMT is able to detect and adapt to evolving dynamics. Change detection in the FIMT is carried out using the Page-Hinckley (PH) change detection test [14]. Adaptation in FIMT 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 [2] that consists of an instance-based learning algorithm for classification and regression problems on data streams by [21]. 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 if 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.

3 The AMRules Algorithm

In this section we present an incremental algorithm for learning model rules to address these issues, named Adaptive Model Rules from High-Speed Data Streams (AMRules). The pseudo code of the algorithm is given in Algorithm 1.

3.1 Learning a Rule Set

The algorithm begin with a empty rule set (RS), and a default rule $\{\} \rightarrow \mathcal{L}$. Every time a new training example is available the algorithm proceeds with checking whether for each rule from rule set (RS) the example is covered by any rule, that is if all the literals are true for the example. The target values of the examples covered by a rule are used to update the sufficient statistic of the rule. Before an example is covered by any rule change detection tests are updated with every example of this rule. We use

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* $\leftarrow \{\}$
  Let *defaultRule* $\mathcal{L} \leftarrow 0$
  foreach *example* $(\boldsymbol{x}, y_k) \in S$ **do**
    foreach *Rule* $r \in RS$ **do**
      if *r covers the example* **then**
        Update change detection tests
        Compute error =$x_t - \bar{x_t} - \alpha$
        Call PHTest(error, λ)
        if *Change is detected* **then**
          Remove the rule
        else
          if *Number of examples in* $\mathcal{L}_r > N_{min}$ **then**
            $r \leftarrow ExpandRule(r)$
            Update sufficient statistics of r
        if *ordered-set* **then**
          BREAK
    if *none of the rules in RS triggers* **then**
      if *Number of examples in* $\mathcal{L}$ *mod* $N_{min} = 0$ **then**
        $RS \leftarrow RS \cup ExpandRule(defaultRule)$
        Update sufficient statistics of the defaultRule

the Page-Hinckley (PH) change detection test to monitor the online error of each rule. If a change is detected the rule is removed from the rule set (RS). Otherwise, the rule is expanded. The expansion of the rule is considered only after certain period (N_{min} number of example). The expansion of a rule is done with Algorithm 2.

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 discuss how rules are expanded, we will first discuss the evaluation measure used in the attribute selection process. [11] describe a standard deviation reduction measure (SDR) for use in determining the merit of a given split. It can be efficiently computed

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 SDR
Let X_b be the attribute with second greater SDR
Compute $\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}}$ (Hoeffding bound)
Compute $ratio = \frac{SDR(X_b)}{SDR(X_a)}$ (Ratio of the SDR values for the best two splits)
Compute $UpperBound = ratio + \epsilon$
if $UpperBound < 1 \vee \epsilon < \tau$ **then**
Extend r with a new condition based on the best attribute
Release sufficient statistics of $\mathcal{L}_r$
$r \leftarrow r \cup \{X_a\}$
return r

in an incremental way. The formula for SDR measure of the split h_A is given below:

$$SDR(h_A) = sd(S) - \frac{N_{Left}}{N} sd(S_{Left}) - \frac{N_{Right}}{N} sd(S_{Right})$$

$$sd(S) = \sqrt{\frac{1}{N}(\sum_{i=1}^{N}(yi - \bar{y})^2)} =$$

$$= \sqrt{\frac{1}{N}(\sum_{i=1}^{N} yi^2 - \frac{1}{N}(\sum_{i=1}^{N} yi)^2)}$$

To make the actual decision regarding a spit, this SDR measure for the best two potential splits are compared, dividing the second-best value by the best one to generate a ratio $ratio$ in the range 0 to 1. Having a predefined range for the values of the random variables, the Hoeffding probability bound (ϵ) [10] 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:

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

where $R^2 = 1$ is the range of the random variable. The process to expand a rule by adding a new condition works as follows. For each attribute X_i, the value of the SDR is computed for each attribute value v_j. If the upper bound $(\bar{ratio} + \epsilon)$ of the sample average is below 1 then the true mean is also below 1. Therefore with confidence $1 - \epsilon$ 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 SDR 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 SDR value and the rule is expanded. The pseudo-code of expanding a rule is presented in Algorithm 2.

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 cover 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 3 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; *iii)* an adaptive strategy, that chooses between the first two strategies, the one with lower MSE in the previous 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 to 1. When a new example arrives, 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 $\hat{y}$ is the output, y the real value and η is the learning rate.

3.4 Change Detection

We use the Page-Hinckley (PH) 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 (MSE or MAD). We use the Page-Hinkley (PH) test [16] 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. The PH test is designed to detect a change in the average of a Gaussian signal [14]. This test considers a cumulative variable m_T, defined as the accumulated difference between the observed values and their mean till the current moment:

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

where $\bar{x_T} = 1/T \sum_{t=1}^{t} x_t$ and δ 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.

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?
  - Update only the first rule that covers training examples. In this case the rule set is ordered, and the corresponding prediction strategy uses only the first rule that covers test examples.
  - Update all the rules that covers training examples. In this case the rule set is unordered, and the corresponding prediction strategy uses a weighted sum of all rules that covers test examples.
- How does AMRules compares against others streaming algorithms?
- How does AMRules compares 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 Massive Online Analysis (MOA) data stream software suite [2]. The performance of the algorithms is measured using the standard metrics for regression problems: Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) [24]. We used two evaluation methods. When no concept drift is assumed, the evaluation method we employ uses the traditional train and test scenario.

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 [1] and other sources. The datasets used in our experimental work are briefly described here. **2dplanes** this is an artificial dataset described in [3]. **Ailerons** this dataset addresses a control problem, namely flying a F16 aircraft. **Puma8NH** and **Puma32H** is a family of datasets synthetically generated from a realistic simulation of the dynamics of a Unimation Puma 560 robot arm. **Pol** this is a commercial application described in [23]. The data describes a telecommunication problem. **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 [3]. **Bank8FM** a family of datasets synthetically generated from a simulation of how bank-customers choose their banks. **Kin8nm** this dataset is concerned with the forward kinematics of an 8 link robot arm. **Airlines** This dataset using 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 very large dataset with nearly 120 million records (11.5 GB memory size) [11]. The Table 1 summarizes the number of instances and the number of attributes of each dataset.

All algorithms learn from the same training set and the performance is estimated from the same test set. In scenarios with concept drift, we use the prequential error estimates [8]. This method evaluates a model on a stream by testing then training with

each example in the stream. For all the experiments, we set the input parameters to: $N_{min} = 200$, $\tau = 0.05$ and $\delta = 0.01$. All of the results in the tables are averaged of ten-fold cross validation [12], except for the Airlines dataset. For all the simulations of evolving data and change detection we set the PH test parameters to $\lambda = 50$ and $\alpha = 0.005$.

Table 1. Summary of datasets

Datasets	# Instances	# Attributes
2dplanes	40768	11
Ailerons	13750	41
Puma8NH	8192	9
Puma32H	8192	32
Pol	15000	49
Elevators	8752	19
Fried	40769	11
bank8FM	8192	9
kin8nm	8192	9
Airline	115Million	11

4.2 Experimental Results

In this section, we empirically evaluate the adaptive model rules algorithm. The results come in four parts. In the first part, we compare the AMRules variants. In the second part we compare AMRules against others streaming algorithms. In the third part, we compare AMRules against others state-of-the-art regression algorithms. In the last part, we assess AMRules models in time-evolving data streams.

Comparison between AMRules Variants. In this section we focus in two strategies that we found potentially interesting. It is a combination of expanding only one rule, the rule that first triggered, with predicting strategy uses only the first rule that covers test examples. Obviously, for this approach it is necessary to use ordered rules ($AMRules^o$). The second setting employs unordered rule set, where all the covering rules expand and the corresponding prediction strategy uses a weighted sum of all rules that covers test examples ($AMRules^u$).

Ordered rule sets specializes one rule at time. As a result it often produces 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 alone.

Table 2 summarize the mean absolute error and the root mean squared error of these variants. Overall, the experimental results points out that unordered rule sets are more competitive than ordered rule sets in terms of MAE and RMSE.

Table 2. Comparative performance between AMRules variants. Results of 10-fold cross-validation except for the AIrlines dataset.

	Mean Absolute Error (variance)		**Root Mean Squared Error (variance)**	
Datasets	AMRules^o	AMRules^u	AMRules^o	AMRules^u
2dplanes	1.23E+00 (0.01)	**1.16E+00** (0.01)	1.67E+00 (0.02)	**1.52E+00** (0.01)
Ailerons	1.10E-04 (0.00)	**1.00E-04** (0.00)	1.90E-04 (0.00)	**1.70E-04** (0.00)
Puma8NH	3.21E+00 (0.04)	**2.66E+00** (0.02)	**4.14E+00** (0.05)	4.28E+00 (0.03)
Puma32H	**1.10E-02** (0.00)	1.20E-02 (0.00)	1.60E-02 (0.00)	**1.00E-04** (0.00)
Pol	**14.0E+00** (25.1)	15.6E+00 (3.70)	**2.30E+01** (44.50)	23.3E00 (4.08)
Elevators	3.50E-03 (0.00)	**1.90E-03** (0.00)	4.80E-03 (0.00)	**2.20E-03** (0.00)
Fried	2.08E+00 (0.01)	**1.13E+00** (0.01)	2.78E+00 (0.08)	**1.67E+00** (0.25)
bank8FM	4.31E-02 (0.00)	**4.30E-02** (0.00)	4.80E-02 (0.00)	**4.30E-02** (0.00)
kin8nm	1.60E-01 (0.00)	**1.50E-01** (0.00)	2.10E-01 (0.00)	**2.00E-01** (0.00)
Airlines	1.42E+01 (0.00)	**1.31E+01** (0.00)	2.44E+01 (0.00)	**2.25E+01** (0.00)
Average Rank	1.7	1.3	1.8	1.2

Comparison with other Streaming Algorithms. We compare the performance of our algorithm with three others streaming algorithms, FIMT and IBLStreams. FIMT is an incremental algorithm for learning model trees, described in [11]. IBLStreams is an extension of MOA that consists in an instance-based learning algorithm for classification and regression problems on data streams by [21].

The performance measures for these algorithms are given in Table 3. The comparison of these streaming algorithms shows that AMRules get better results.

Table 3. Comparative performance between streaming algorithms. Results of 10-fold cross-validation except for the AIrlines dataset.

	Mean Absolute Error (variance)			**Root Mean Squared Error (variance)**		
Datasets	AMRules^u	FIMT	IBLStreams	AMRules^u	FIMT	IBLStreams
2dplanes	1.16E+00 (0.01)	**8.00E-01** (0.00)	1.03E+00 (0.00)	1.52E+00 (0.01)	**1.00E+00** (0.00)	1.30E+00 (0.00)
Ailerons	**1.00E-04** (0.00)	1.90E-04 (0.00)	3.20E-04 (0.00)	1.70E-04 (0.00)	**1.00E-09** (0.00)	3.00E-04 (0.00)
Puma8NH	**2.66E+00** (0.01)	3.26E+00 (0.03)	3.27E+00 (0.01)	**4.28E+00** (0.03)	12.0E+00 (0.63)	3.84E+00 (0.02)
Puma32H	1.20E-02 (0.00)	**7.90E-03** (0.00)	2.20E-02 (0.00)	**1.00E-04** (0.01)	1.20E-02 (0.00)	2.70E-02 (0.00)
Pol	**15.6E+00** (3.70)	38.2E+00 (0.17)	29.7E+00 (0.55)	**23.3E+00** (4.08)	1,75E+03 (1383)	50,7E+00 (0.71)
Elevators	**1.90E-03** (0.00)	3.50E-03 (0.00)	5.00E-03 (0.00)	2.20E-03 (0.00)	**3.00E-05** (0.00)	6.20E-03 (0.00)
Fried	**1.13E+00** (0.01)	1.72E+00 (0.00)	2.10E+00 (0.00)	**1.67E+00** (0.25)	4.79E+00 (0.01)	2.21E+00 (0.00)
bank8FM	4.30E-02 (0.00)	**3.30E-02** (0.00)	7.70E-02 (0.00)	4.30E-02 (0.00)	**2.20E-03** (0.00)	9.60E-02 (0.00)
kin8nm	**1.60E-01** (0.00)	**1.60E-01** (0.00)	9.50E-01 (0.00)	**2.00E-01** (0.00)	2.10E-01 (0.00)	1.20E-01 (0.00)
Airlines	**1.31E+01** (0.00)	1.39E+01 (0.00)	1.45E+01 (0.00)	**2.25E+01** (0.00)	2.30E+01 (0.00)	2.51E+01 (0.00)
Average Rank	1.5	1.9	2.6	1.8	2.0	2.3

Comparison with other State-of-the-Art Regression Algorithms. Another experiment which involves adaptive model rules is showed in Table 4. We compared AMRules with other non-incremental regression algorithms from WEKA [9]. We use the standard method of ten-fold cross-validation, using the same folds for all the algorithms included.

The comparison of these algorithms show that AMRules has lower accuracy (MAE, RMSE) than M5Rules and better accuracy than the others methods. These results were somewhat expected, since these datasets are relatively small for the incremental algorithm.

Table 4. Comparative performance between AMRules^u and other regression algorithms. Results of 10-fold cross validation.

	Root Mean Squared Error (variance)			
Datasets	AMRules^u	M5Rules	MLPerceptron	Linear Regression
2dplanes	1.52E+00 (0.01)	**9.8E-01** (0.01)	1.09E+00 (0.01)	2.37E+00 (0.00)
Ailerons	**1.70E-04** (0.00)	2.00E-04 (0.00)	1.71E-04 (0.00)	2.00E-04 (0.00)
Puma8NH	4.28E+00 (0.03)	**3.19E+00** (0.01)	4.14E+00 (0.20)	4.45E+00 (0.01)
Puma32H	**1.00E-04** (0.00)	8.60E-03 (0.00)	3.10E-02 (0.00)	2.60E-02 (0.00)
Pol	23.3E+00 (4.08)	**6.56E+00** (0.45)	20.1E+00 (15.1)	30.5E+00 (0.16)
Elevators	**2.20E-03** (0.00)	2.23E-03 (0.00)	2.23E-03 (0.00)	2.29E-03 (0.00)
Fried	1.67E+00 (0.25)	**1.60E+00** (0.00)	1.69E+00 (0.04)	2.62E+00 (0.00)
bank8FM	4.30E-02 (0.00)	**3.10E-02** (0.00)	3.40E-02 (0.00)	3.80E-02 (0.00)
kin8nm	2.00E-01 (0.00)	**1.70E-01** (0.00)	1.60E-01 (0.00)	2.00E-01 (0.00)
Average Rank	2.2	1.7	2.6	3.6

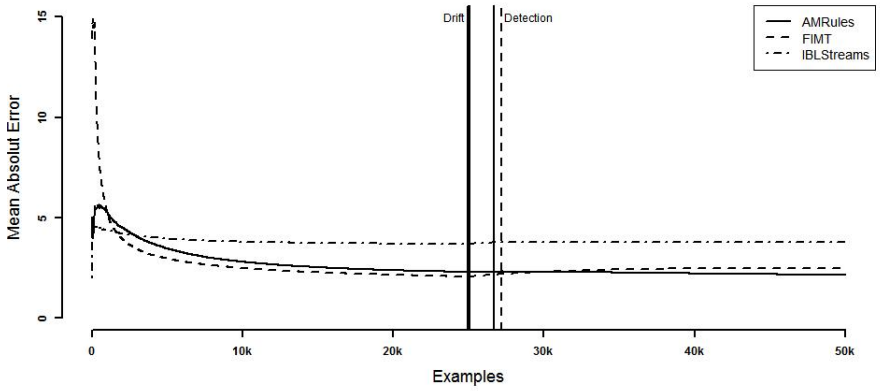

Fig. 1. Evolution of the prequential Mean Absolute Error of streaming algorithms using the dataset Fried.

Evaluation in Time-Evolving Data Streams. In this subsection we first study the evolution of the error measurements (MAE and RMSE) and evaluate the change detec-

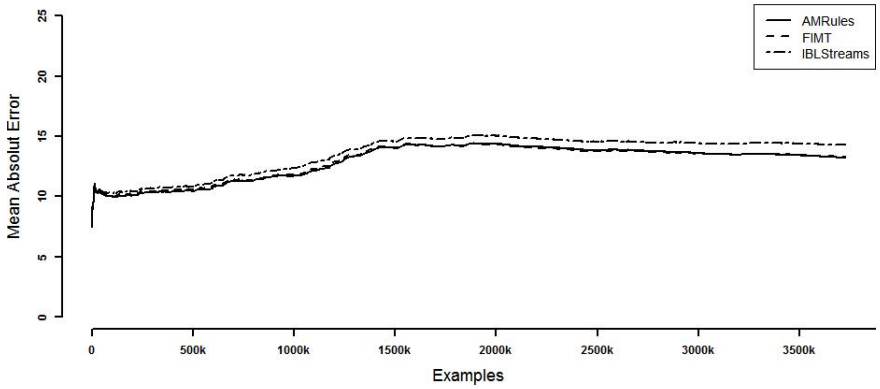

Fig. 2. Evolution of the prequential Mean Absolute Error of streaming algorithms using the dataset Airlines.

tion method. After, we evaluate the streaming algorithms on non-stationary streaming real-world problems, using the Airline dataset from the DataExpo09 competition.

To simulate drift we use Fried dataset. The simulations allow us to control the relevant parameters and to assess the change detection algorithm. Figure 1 depict the prequential MAE curve of the streaming algorithms using the dataset Fried. The figure also illustrates the change point and the points where the change was detected. Only two of the algorithms detected the change (FIMT and AMRules). Table 5 shows the average results over 10 experiments using the Fried dataset. We measure the number of nodes, the number of rules and the Page-Hinckley test delay (number of examples monitored by PH test before the detection). The delay of the Page-Hinckley test is an indication of how fast the algorithm will be able to start the adaptation strategy. These two algorithms obtained similar results. The general conclusions are that FIMT and AMRules algorithms are robust and have better results than IBLStreams.

Table 5. Average results from the evaluation of change detection over 10 experiments.

Algorithms	Delay	Size
AMRules	1484.1	56 (nr. Rules)
FIMT	2095.6	290 (nr. Leaves)
IBLStreams	-	-

Figure 2 illustrates the evaluation of the prequential MAE of the streaming algorithms on a non-stationary streaming real-world problem, the Airlines dataset. FIMT and AMRules exhibit approximately the same behavior, slightly better than IBLStreams.

Learning Times. Table 6 reports the relative learning times required for the 10-cross-validation experiments. Being one-pass algorithms, both versions of AMRules are much faster than M5 Rules and multi-layer perceptron. AMRules^o generates fewer rules being slightly faster than AMRules^u.

Table 6. Relative learning times of experiences reported in the paper. The reference algorithm is AMRules^o.

	Time(seconds)						
Datasets	AMRules^o	AMRules^u	M5Rules	MLPerceptron	LinRegression	IBLStreams	FIMT
2dplanes	1	1.5	317	577	1.0	4627	2.3
Airlerons	1	1.2	535	737	4.6	8845	1.5
Puma8NH	1	1.1	497	113	0.5	1700	2.0
Puma32H	1	1.4	801	280	2.0	1084	2.2
Pol	1	1.4	806	1260	1.4	12133	1.5
Elevators	1	1.5	678	341	0.9	16856	1.8
Fried	1	1.4	5912	359	0.7	154157	1.4
bank8FM	1	1.4	78	73	0.8	17189	0.4
kin8nm	1	1.1	146	75	0.7	29365	0.4
Airlines	1	8.3	-	-	-	203	10.6

5 Conclusions

Regression rules are one of the most expressive languages to represent 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 paper, 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 point out that, in comparison to ordered rule sets, unordered rule sets are more competitive in terms of performance (MAE and RMSE). AMRules is competitive against batch learners even for medium sized datasets.

Acknowledgments

The authors acknowledge the financial support given by the project FCT-KDUS PTDC/ EIA/ 098355/2008 FCOMP -01-0124-FEDER-010053, the ERDF through the COMPETE Programme and by National Funds through FCT within the project FCOMP - 01-0124-FEDER-022701.

References

1. K. Bache and M. Lichman. UCI machine learning repository, 2013.
2. A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, and T. Seidl. MOA: Massive online analysis. *Journal of Machine learning Research (JMLR)*, pages 1601–1604, 2010.
3. L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.
4. Pedro Domingos and Geoff Hulten. Mining High-Speed Data Streams. In Ismail Parsa, Raghu Ramakrishnan, and Sal Stolfo, editors, *Proceedings of the ACM Sixth International Conference on Knowledge Discovery and Data Mining*, pages 71–80, Boston, USA, 2000. ACM Press.
5. Eibe Frank, Yong Wang, Stuart Inglis, Geoffrey Holmes, and Ian H. Witten. Using model trees for classification. *Machine Learning*, 32(1):63–76, 1998.
6. Johannes Fürnkranz, Dragan Gamberger, and Nada Lavra. *Foundations of Rule Learning*. Springer, 2012.
7. João Gama. *Knowledge Discovery from Data Streams*. Chapman & Hall, CRC Press, 2010.
8. João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. On evaluating stream learning algorithms. *Machine Learning*, 90(3):317–346, 2013.
9. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18, 2009.
10. Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
11. Elena Ikonomovska, João Gama, and Saso Dzeroski. Learning model trees from evolving data streams. *Data Min. Knowl. Discov.*, 23(1):128–168, 2011.
12. Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143, 1995.
13. Petr Kosina and João Gama. Handling time changing data with adaptive very fast decision rules. In Peter A. Flach, Tijl De Bie, and Nello Cristianini, editors, *ECML/PKDD (1)*, volume 7523 of *Lecture Notes in Computer Science*, pages 827–842. Springer, 2012.
14. H. Mouss, D. Mouss, N. Mouss, and L. Sefouhi. Test of Page-Hinkley, an approach for fault detection in an agro-alimentary production system. In *Proceedings of the Asian Control Conference*, volume 2, pages 815–818, 2004.
15. ElMoustapha Ould-Ahmed-Vall, James Woodlee, Charles Yount, Kshitij A Doshi, and Seth Abraham. Using model trees for computer architecture performance analysis of software applications. In *Performance Analysis of Systems & Software, 2007. ISPASS 2007. IEEE International Symposium on*, pages 116–125. IEEE, 2007.
16. E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1/2):100–115, 1954.
17. Duncan Potts and Claude Sammut. Incremental learning of linear model trees. *Machine Learning*, 61(1-3):5–48, 2005.
18. J. R. Quinlan. Learning with continuous classes. In *Australian Joint Conference for Artificial Intelligence*, pages 343–348. World Scientific, 1992.
19. J. R. Quinlan. Combining instance-based and model-based learning. pages 236–243. Morgan Kaufmann, 1993.
20. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc. San Mateo, CA, 1993.
21. Ammar Shaker and Eyke Hüllermeier. Iblstreams: a system for instance-based classification and regression on data streams. *Evolving Systems*, 3:235–249, 2012.

22. Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 226–235, Washington, D.C., 2003. ACM Press.
23. Sholom M. Weiss and Nitin Indurkhya. Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research*, 3:383–403, 1995.
24. C. J. Willmott and K. Matsuura. Advantages of the mean absolute error (mae) over the mean square error (rmse) in assessing average model performance. *Climate Research*, 30:79–82, 2005.