Learning Model Rules from High-Speed Data Streams

Ezilda Almeida and **Carlos Ferreira** and **João Gama**
LIAAD - INESC Porto L.A., Portugal

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 in AMRules 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 algorithm with other streaming regression algorithms.

Keywords: Data Streams, Regression, Rule Learning, Change Detection

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, machine learning and data mining. Predicting numeric values usually involves complicated regression formulae. Model trees [14] and regression rules [15] are the most powerful data mining models. 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 Adaptive Model Rules, the first one-pass algorithm for learning regression rule sets from time-evolving streams. AMRules can learn ordered and unordered rules. The antecedent of a rule is a set of literals (conditions based on the attribute values). The consequent of a rule 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. It monitors the mean square error using the Page-Hinkley test, providing information about the dynamics of the process generating data.

The paper is organized has 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. Last Section presents the lessons learned.

2 Related Work

In this section we analyze the related work in two dimensions. One dimension is related to regression algorithms, the other dimension is related to incremental learning of regression algorithms.

In regression domains, [14] presented the system M5. 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.

Cubist [15] 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 step of the tree. A prediction is made using the linear regression model at the terminal node of the tree, but is *smoothed* by taking into account the prediction from the linear model in the previous node of the tree (which also occurs recursively up the tree). 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 and/or combined for simplification.

2.1 Streaming Regression Algorithms

Many methods can be found in the literature for solving classification tasks on streams, but only a few exist for regression tasks. To the best of our knowledge, we note only two papers for online learning of regression and model trees. In the algorithm of [13] for incremental learning of linear model trees the splitting decision is formulated as hypothesis testing. The split least likely to occur under the null hypothesis of non-splitting is considered the best one. The linear models are computed using the RLS (Recursive Least Square) algorithm that has a complexity, which is quadratic in the dimensionality of the problem. This complexity is then multiplied with a user-defined number of possible splits per numerical attribute for which a separate pair of linear models is updated with each training example and evaluated. The Fast Incremental Model Tree (FIMT) proposed in [10], 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, 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. Another data streaming issue addressed in [10] is the problem of concept drift. Data streaming models capable of dealing with concept drift face two main challenges: how to detect when concept drift has occurred and how to adapt to the change. Change detection in the FIMT is carried out using the Page-Hinkley change detection test [11]. Adaptation in FIMT involves growing an alternate subtree from the node in which change was detected.

IBLStreams (Instance Based Learner on Streams) is an extension of MOA that consists in an instance-based learning algorithm for classification and regression problems on data streams by [1]; IBLStreams optimizes the composition and size of the case base autonomously. On arrival of a new example (x_0, y_0), this example is first added to the case base. Moreover, it is checked whether other examples might be removed, either since they have become redundant or since 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_c nearest neighbors of x_0, determined according a distance measure Δ, and the candidate set C consists of the examples within that neighborhood. The most recent examples are excluded from removal due to the difficulty to distinguish potentially noisy data from the beginning of a concept change. Even though unexpected observations should be removed only in the former but not in the latter case.

Algorithm 1: AMRules Algorithm

```
Input:
  S: Stream of examples
  ordered-set: boolean flag
  N_min: Minimum number of examples
  λ: Constant to solve ties
  α: the magnitude of changes that are allowed
  j: rule index
Result: RS  Set of Decision Rules
begin
  Let RS ← {}
  Let defaultRule {} → (L ← NULL)
  foreach example (x_i, y_i) do
    foreach Rule r ∈ RS_j do
      if r covers the example then
        Let ŷ_i be the prediction of the rule r,
        computed using L_r
        Compute error =(ŷ_i − y_i)^2
        Call PHTest(error, α, λ)
        if Change is detected then
          Remove the rule
        else
          Update sufficient statistics of r
          Update Perceptron of r
          if Number of examples in L_r ≥ N_min
          then
            r ← ExpandRule(r)
        if ordered-set then
          BREAK
    if none of the rules in RS triggers then
      Update sufficient statistics of the default rule
      Update Perceptron of the default rule
      if Number of examples in L ≥ N_min then
        RS ← RS ∪ ExpandRule(defaultRule)
```

3 The AMRules Algorithm

The problem of learning model rules from data streams raises several issues. First, the dataset is no longer finite and available prior to learning, it is impossible to store all data in memory and learn from them as a whole. Second, multiple sequencial scans over the training data are not allowed. An algorithm must therefore collect the relevant information at the speed it arrives and incrementally decide about splitting decisions. Third the training dataset may consist of data from different distributions. 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.

The algorithm begins with an empty rule set (RS), and a default rule $\{\} \rightarrow \mathcal{L}$, where $\mathcal{L}$ is initialized to NULL. $\mathcal{L}$ is a data structure used to store the sufficient statistics required to expand a rule and for prediction. Every time a new training example is available the algorithm proceeds with checking

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 $r = \frac{SDR(X_b)}{SDR(X_a)}$ (Ratio of the SDR values for the best two splits)
Compute $UpperBound = r + \epsilon$
if $UpperBound < 1 \vee \epsilon < \tau$ **then**
Extend r with a new condition based on the best attribute $X_a \leq v_j$ or $X_a > v_j$
Release sufficient statistics of $\mathcal{L}_r$
$r \leftarrow r \cup \{X_a \leq v_j or X_a > v_j\}$
return r

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 ($\mathcal{L}$). To detect changes we propose to use the Page-Hinkley (PH) change detection test. If a change is detected the rule is removed from the rule set. Otherwise, the rule might be expanded. The expansion of the rule is considered only after certain minimum number of examples (N_{min}). The expansion of a rule is explained in Algorithm 2.

The set of rules is learned in parallel, as described in Algorithm 1. We consider two cases: learning ordered or unordered set of rules. In the former case, 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.

3.1 Expansion of a Rule

Before discussing how rules are expanded, we will first discuss the evaluation measure used in the attribute selection process. [10] describe a standard deviation reduction measure (SDR) for determining the merit of a given split. It can be efficiently computed in an incremental way. Given a leaf where a sample of the dataset S of size N has been observed, a hypothetical binary split h_A over attribute A would divide the examples in S in two disjoint subsets S_L and S_R, with sizes N_L and N_R respectively. The formula for SDR measure of the split h_A is given below:

$$SDR(h_A) = sd(S) - \frac{NL}{N} sd(S_L) - \frac{NR}{N} sd(S_R)$$

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

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

To make the actual decision regarding a split, the SDR measured for the best two potential splits are compared, by 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, the Hoeffding probability bound (ϵ) [17] 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 SDR 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 $1-\epsilon$ 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 best split with a higher SDR value and the rule is expanded.

3.2 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 using weighted vote.

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.

3.3 Change Detection

The AMRules uses the Page-Hinkley (PH) test [12] to monitor the error and signals a drift when a significant increase of this variable is observed. The PH test is a sequential analysis technique typically used for monitoring change detection in

signal processing. The PH test is designed to detect a change in the average of a Gaussian signal [11]. This test considers a cumulative variable m_T, defined as the accumulated difference between the observed error and the mean of the error till the current moment:

$$m_T = \sum_{t=1}^{T} (e_t - \bar{e}_T - \alpha)$$

where $\bar{e_T} = 1/T \sum_{t=1}^{t} e_t$ and α corresponds to the magnitude of changes that are allowed.

The minimum value of this variable is also computed: $M_T = min(m_t, t = 1 \dots T)$. As a final step, 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 distribution. 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 mean absolut error (MAE) and root mean squared error (RMSE). 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 other streaming algorithms?
- How does AMRules compares against other state-of-the-art regression algorithms?
- How does AMRules learned models evolve in time-changing streams?

4.1 Experimental Setup

All our algorithms were implemented in java using Massive Online Analysis (MOA) data stream software suite [2]. For all the experiments, we set the input parameters of AMRules to: $N_{min} = 200$, $\tau = 0.05$ and $\delta = 0.01$. The parameters for the Page-Hinkley test are $\lambda = 50$ and $\alpha = 0.005$. Table 1 summarizes information about the datasets used and reports the learning rate used in the perceptron learning.

All of the results in the tables 2, 3 and 4 are averaged of ten-fold cross-validation [16]. The accuracy is measured using the following metrics: Mean absolute error (MAE) and root mean squared error (RMSE) [19]. We used two evaluation methods. When no concept drift is assumed, the evaluation method we employ uses the traditional train and test scenario. All algorithms learn from the same training set and the error is estimated from the same test sets. In scenarios with concept drift, we use the *prequential* (predictive sequential) error estimate [8]. This evaluation method evaluates a model sequentially. When an example is available, the current regression model makes a prediction and the loss is computed. After the prediction the regression model is updated with that example.

Datasets

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 [3] and other sources. The datasets used in our experimental work are:

2dplanes this is an artificial data set described in [4]. **Airlerons** this data set 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 [18].The data describes a tele communication problem. **Elevators** this data set is also obtained from the task of controlling a F16 aircraft. **Fried** is an artificial data set used in Friedman (1991) and also described in Breiman (1996,p.139). **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. **Airline** this dataset using the data from the Data Expo competition (2009). The dataset consists of a large 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 120 million records (11.5 GB memory size) [10]. Table 1 summarizes the number of instances and the number of attributes of each dataset.

Table 1. Summary of datasets

Datasets	# Instances	# Attributes	Learning rate
2dplanes	40768	11	0.01
Airlerons	13750	41	0.01
Puma8NH	8192	9	0.01
Puma32H	8192	32	0.01
Pol	15000	49	0.001
Elevators	8752	19	0.001
Fried	40769	11	0.01
Bank8FM	8192	9	0.01
Kin8nm	8192	9	0.01
Airline	115Million	11	0.01

4.2 Experimental Results

In this section, we empirically evaluate the AMRules. The results are described in four parts. In the first part we compare the AMRules variants, the second part we compare AMRules against other streaming algorithms and the third part compare AMRules against other state-of-the-art regression algorithms. The last part presents the analysis of AMRules behavior in the context of time-evolving data streams.

Comparison between AMRules Variants

In this section we focus on 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 cover test examples ($AMRules^u$).

Ordered rule sets specializes one rule at a time, and as a result it often produces less 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 the unordered rule sets are more competitive than ordered rule sets in terms of MAE and RMSE.

Table 2. Results of ten-fold cross-validation for AMRules algorithms

	Mean absolut 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)
Airlerons	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)	3.26E+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)	**1.20E-02** (0.00)
Pol	**14.0E+00** (25.1)	15.6E+00 (3.70)	**23.0E00** (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)

Comparison with other Streaming Algorithms

We compare the performance of our algorithm with three other streaming algorithms, FIMT and IBLStreams. FIMT is an incremental algorithm for learning model trees, addressed in [10]. IBLStreams is an extension of MOA that consists in an instance-based learning algorithm for classification and regression problems on data streams by [1].

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

Comparison with State-of-the-art Regression Algorithms

Another experiment which involves adaptive model rules is shown in Table 4. We compare AMRules with other non-incremental regression algorithms available in WEKA [9]. All these experiments using algorithms are performed using WEKA. 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 is very competitive in terms of (MAE, RMSE) than all the other methods, except M5Rules. AMRules is faster than all the other algorithms considered in this study. These results were somewhat expected, since these datasets are relatively small for the incremental algorithm.

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

Algorithms	Delay	Size
AMRules	1484	56 (nr. Rules)
FIMT	2096	290 (nr. Leaves)
IBLStreams	-	-

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 detection method. After, we evaluate the streaming algorithms on non-stationary streaming real-world problem, we use 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 evaluate the drift detection. Figure 1 and Figure 2 depict the MAE and RMSE curves of the streaming algorithms using the dataset Fried. These figures also illustrate the point of drift and the points where the change was detected. Only two of the algorithms – FIMT and AMRules – were able to detect a change. Table 5 report the average results over ten experiments varying the seed of the Fried dataset. We measure the number of nodes for FIMT, the number of rules AMrules and the the delay (in terms of number of examples) in detection the drift. The delay gives 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. Figures 3 and 4 show the evaluation of the MAE and the RMSE of the streaming algorithms on non-stationary real-world problem. FIMT and AMRules obtain approximately similar behavior in terms of MAD and MSE. Both exhibit somewhat better performance than IBLStreams, but not significantly different.

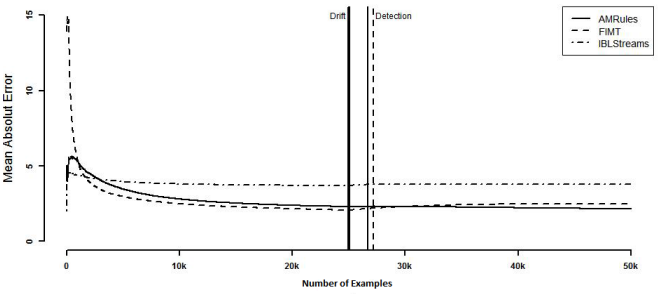

Fig. 1. Mean absolut error of streaming algorithms using the dataset Fried.

5 Conclusions

Learning regression rules from data streams is an interesting approach that has not been explored by the stream mining community. In this paper, we presented a new regression rules approach for streaming data with change detection. The AMRules algorithm is able to learn very fast and the only memory it requires is for storing sufficient statistics of the rules. To

Table 3. Results of ten-fold cross-validation for Streaming Algorithms

	Mean absolut 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)
Airlerons	**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)

Table 4. Results of ten-fold cross-validation for AMRules^u and others Regression Algorithms

	Mean absolute error (variance)				Root mean squared error (variance)			
Datasets	MRules^u	M5Rules	MLPerceptron	LinRegression	MRules^u	M5Rules	MLPerceptron	LinRegression
2dplanes	1.16E+00 (0.01)	**8.00E-01** (0.01)	8.70E-01 (0.01)	1.91E+00 (0.00)	1.52E+00 (0.01)	**9.8E-01** (0.01)	1.09E+00 (0.01)	2.37E+00 (0.00)
Airlerons	**1.00E-04** (0.00)	**1.00E-04** (0.00)	1.40E-04 (0.00)	1.10E-04 (0.00)	**1.70E-04** (0.00)	2.00E-04 (0.00)	1.71E-04 (0.00)	2.00E-04 (0.00)
Puma8NH	3.26E+00 (0.03)	**2.46E+00** (0.00)	3.34E+00 (0.17)	3.64E+00 (0.01)	4.28E+00 (0.03)	**3.19E+00** (0.01)	4.14E+00 (0.20)	4.45E+00 (0.01)
Puma32H	1.20E-02 (0.00)	**6.80E-03** (0.00)	2.30E-02 (0.00)	2.00E-02 (0.00)	1.20E-02 (0.00)	**8.60E-03** (0.00)	3.10E-02 (0.00)	2.60E-02 (0.00)
Pol	15.6E+00 (3.70)	**2.79E+00** (0.05)	14.7E+00 (5.53)	26.5E+00 (0.21)	23.3E+00 (4.08)	**6.56E+00** (0.45)	20.1E+00 (15.1)	30.5E+00 (0.16)
Elevators	1.90E-03 (0.00)	**1.70E-03** (0.00)	2.10E-03 (0.00)	2.00E-03 (0.00)	**2.20E-03** (0.00)	2.23E-03 (0.00)	2.23E-03 (0.00)	2.29E-03 (0.00)
Fried	**1.13E+00** (0.01)	1.25E+00 (0.00)	1.35E+00 (0.03)	2.03E+00 (0.00)	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)	**2.20E-02** (0.00)	2.60E-02 (0.00)	2.90E-02 (0.00)	4.30E-02 (0.00)	**3.10E-02** (0.00)	3.40E-02 (0.00)	3.80E-02 (0.00)
Kin8nm	1.60E-01 (0.00)	**1.30E-01** (0.00)	**1.30E-01** (0.00)	1.60E-01 (0.00)	2.00E-01 (0.00)	**1.70E-01** (0.00)	1.60E-01 (0.00)	2.00E-01 (0.00)

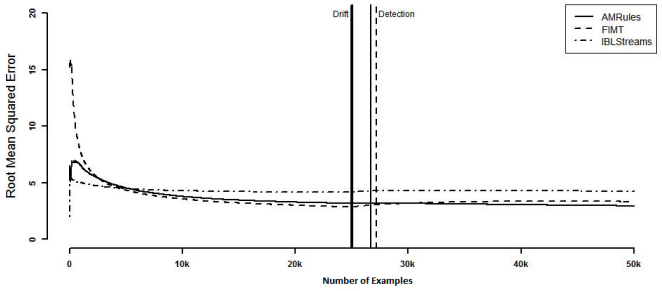

Fig. 2. Root mean squared error of streaming algorithms using the dataset Fried.

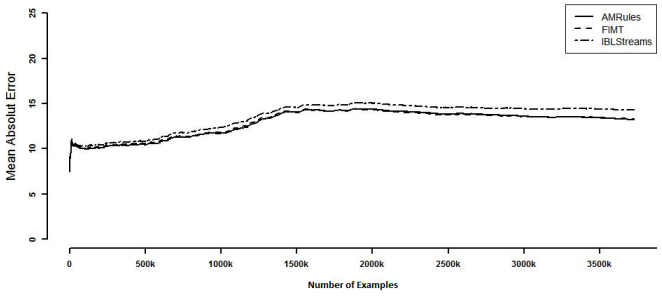

Fig. 3. Mean absolut error of streaming algorithms using the dataset Airlines.

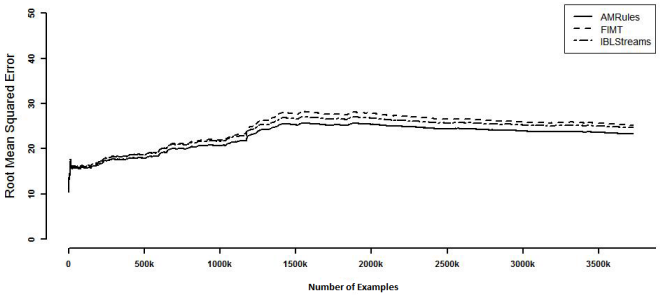

Fig. 4. Root mean squared error of streaming algorithms using the dataset Airlines.

the best of our knowledge, in the literature there is no other method that addresses this issue.

AMRules learns ordered and unordered rule sets. The experimental results point out that unordered rule sets, in comparison to ordered rule sets, are more competitive in terms of error metrics (MAE and RMSE). AMRules achieves better results than the others algorithms even for medium sized datasets. The AMRule algorithm is equipped with explicit change detection mechanisms that signals change points during the learning process. This information is relevant to understand the dynamics of evolving streams.

Acknowledgments:

The authors acknowledge the financial support given by the projects FCT-KDUS (PTDC/EIA/098355/2008); FCOMP - 01-0124-FEDER-010053, the ERDF through the COMPETE

Programme and by Portuguese National Funds through FCT within the project FCOMP - 01-0124-FEDER-022701.

References

1. S. Ammar and H. Eyke. Iblstreams: a system for instance-based classification and regression on data streams. *Evolving Systems*, 3:235–249, 2012.
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. K. E. Blake and C. Merz. Uci repository of machine learning databases. 1999.
4. L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.
5. E. Frank, Y. Wang, S. Inglis, G. Holmes, and I. H. Witten. Using model trees for classification. *Machine Learning*, 32(1):63–76, 1998.
6. J. Frnkranz, D. Gamberger, and N. Lavra. *Foundations of Rule Learning*. Springer, 2012.
7. J. Gama. *Knowledge Discovery from Data Streams*. Chapman & Hall, CRC Press, 2010.
8. J. Gama, R. Sebasti ao, and P. P. Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 329–338, New York, NY, USA, 2009. ACM.
9. M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18, 2009.
10. E. Ikonomovska, J. Gama, and S. Dzeroski. Learning model trees from evolving data streams. *Data Min. Knowl. Discov.*, 23(1):128–168, 2011.
11. H. Mouss, D. Mouss, N. Mouss, and L. Sefouhi. Test f page-hinckley, an approach for fault detection in an agro-alimentary production system. *In Proceedings of the Asian Control Conference*, 2:815–818, 2004.
12. E. S. Page. Continuous inspection schemes. *Biometrika*, 41(1):100–115, 1954.
13. D. Potts and C. Sammut. Incremental learning of linear model trees. *Machine Learning*, 61(1-3):5–48, 2005.
14. J. R. Quinlan. Learning with continuous classes. In *Australian Joint Conference for Artificial Intelligence*, pages 343–348. World Scientific, 1992.
15. J. R. Quinlan. Combining instance-based and model-based learning. pages 236–243. Morgan Kaufmann, 1993.
16. K. Ron. A study of cross-validation and bootstrap for accuracy estimation and model selection. pages 1137–1143, 1995.
17. H. Wassily. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.
18. S. M. Weiss and N. Indurkhya. Rule-based machine learning methods for functional prediction. *Journal of Artificial Intelligence Research*, 3:383–403, 1995.
19. 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.