JMLR: Workshop and Conference Proceedings 36:198–213, 2014 BIGMINE 2014

Ensembles of Adaptive Model Rules from High-Speed Data Streams

João Duarte joao.m.duarte@inesctec.pt
LIAAD-INESC TEC, University of Porto
Campus da FEUP, Rua Dr. Roberto Frias
4200 - 465 Porto, Portugal

João Gama jgama@fep.up.pt
LIAAD-INESC TEC, University of Porto
Faculty of Economics, University of Porto
Campus da FEUP, Rua Dr. Roberto Frias
4200 - 465 Porto, Portugal

Editors: Wei Fan, Albert Bifet, Qiang Yang and Philip Yu

Abstract

The volume and velocity of data is increasing at astonishing rates. In order to extract knowledge from this huge amount of information there is a need for efficient on-line learning algorithms. Rule-based algorithms produce models that are easy to understand and can be used almost offhand. Ensemble methods combine several predicting models to improve the quality of prediction. In this paper, a new on-line ensemble method that combines a set of rule-based models is proposed to solve regression problems from data streams. Experimental results using synthetic and real time-evolving data streams show the proposed method significantly improves the performance of the single rule-based learner, and outperforms two state-of-the-art regression algorithms for data streams.

Keywords: Data Streams, Regression, Ensembles, Rule Learning.

1. Introduction

Data stream mining is an important and active research area which aims to discover patterns from time-evolving data produced continuously and at high-speed (Gama, 2010). The possible applications are enormous since nowadays data is generated everywhere. A few examples are electronic mailing, sensor networks, financial transactions, news feeds, TCP/IP traffic, etc. Traditional off-line algorithms assume data is obtained from a stationary distribution. Thereby, on-line learning algorithms are replacing batch learning algorithms in solving several tasks, especially in rapidly changing environments.

Decision trees and rule-based systems are very useful to the end-user as they require little effort for data preparation: feature selection is performed automatically, and there is no need for attribute scaling. Another advantage is that the resulting model can be easily interpreted. Some decision trees (Domingos and Hulten, 2000; Ikonomovska et al., 2011) and rule-based algorithms (Kosina and Gama, 2012) have been designed to handle data streams.

Ensembles methods improve the performance of learning algorithms (Dietterich, 2000; Mendes-Moreira et al., 2012) both for classification and regression tasks. These methods combine a set of learning models to produce a more accurate prediction. Several strategies for designing ensemble methods have been proposed, many of them using decision trees as base learners (Bibimoune et al., 2013).

In this paper, an on-line ensemble regression method which learns from high-speed data streams is presented. Experimental results show the proposed ensemble method significantly outperforms its single learner version, and two state-of-the-art regression algorithms for data streams. The remaining of this paper is organized as follows. In Section 2 related work is presented and the data stream ensemble problem is defined. Section 3 describes the rule-based algorithm that will be used by the ensemble method proposed in Section 4. The experimental evaluation of the proposed method is presented in Section 5. Section 6 concludes this work.

2. Related Work

Let $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ...\}$ be an unbounded data set where $\mathbf{x}_i$ is a d-dimensional vector $[x_{i1}, x_{i2}, \cdots, x_{id}]^\top$ describing the explanatory variables and y_i is the corresponding response variable. A predictor is a function $\hat{f}(\mathbf{x})$ which maps the input variable $\mathbf{x}$ to an output variable $\hat{y}$ and approximates the true unknown function $f(\mathbf{x}) = y$. The algorithm used to build a predictor is called learner. When the output variable y takes continuous values the learning problem is called regression. When y is categorical the learning problem is referred as classification. In this work we are interested in the former. A good predictor performs the mapping $\hat{f}(\mathbf{x}) = \hat{y}$ accurately. The accuracy is usually measured using a cost function where low cost correspond to high accuracy. Two common measures of performance used to evaluate regression models are the *mean absolute error* (MAE) and the *root-mean-squared error* (RMSE). The MAE and RMSE performance measures are defined as

$$MAE = \frac{1}{n}\sum_{i=1}^{n} \left|\hat{y}_i - y_i\right|, \tag{1}$$

and

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \left(\hat{y}_i - y_i\right)^2}, \tag{2}$$

respectively, where y_i is the true value of the response variable of the i^{th} example, $\hat{y}_i$ the predicted value, and n the number of examples evaluated.

2.1. Regression algorithms for data streaming

Despite many classification algorithms have been proposed for data streams, not many can be found for solving regression tasks. Some methods, such as Kalman and particle filters (Chen, 2003), and artificial neural networks (Briegel and Tresp, 2000), are suitable for on-line regression tasks. However, these techniques do not properly handle concept drifts, which are one important aspect that distinguishes data stream learning from on-line learning. Potts and Sammut (2005) proposed an algorithm that incrementally induces linear model trees. The tree grows only if the confidence of two linear models, one for each side of a possible split, being better estimators than the linear model of the correspondent node is higher than a given threshold. Ikonomovska et al. (2011) proposed an incremental regression algorithm for time-evolving data streams (FIMT-DD) based on the Hoeffding tree classification algorithm (Domingos and Hulten, 2000). Instead of using information gain or the Gini index as split evaluation measure, the algorithm uses the standard deviation reduction measure and, similarly to the Hoeffding tree algorithm, the Hoeffding bound is used to guarantee that the best split is chosen. Shaker and Hüllermeier (2012) proposed an instance-based learning algorithm for data streams (IBLSTREAMS) which can handle both classification and regression problems. It follows the nearest-neighborhood estimation principle which predicts the output value for a new example as an aggregation of the outputs of its nearest examples. The algorithm adaptation and memory management is performed by adding or removing examples to the case base considering three indicators of usefulness: temporal relevance, spacial relevance and consistency.

2.2. Ensemble models for data streaming

An ensemble model $\mathcal{F}$ is composed of a set of predictors $\hat{f}_m$ of the true function f, $\mathcal{F} = \{\hat{f}_1 \cdots, \hat{f}_k\}$. An ensemble predictor $\hat{f}_*$ is obtained by integrating the base learners $\hat{f}_m$. According to Mendes-Moreira et al. (2012), the ensemble process can be divided into three steps: generation, pruning and integration. In the generation step a set of models $\mathcal{F}$ is generated using one or several learning algorithms. In the pruning step redundant models $\hat{f}_m$ may be removed from $\mathcal{F}$. Finally, the strategy to obtain a joint prediction is determined by the integration step.

Some of the most popular ensemble methods are Bagging, Boosting and Random Forests. Batch learning Bagging algorithms build k different versions of the original dataset by resampling with replacement (Breiman, 1996). The ensemble prediction is obtained by majority voting (classification) or averaging the base learners predictions (regression). This approach reduces variance and minimizes the effect of overfitting. Bagging approaches have been adapted to handle data streams by updating each predictor $\hat{f}_m$ a random number of times p_m, such that p_m is sampled from a Poisson distribution (Oza and Russell, 2001). Off-line Boosting methods learn prediction models sequentially by adapting the weights of the training examples according to the error on previous iterations. Examples harder to predict are given more importance while building future predictors (Freund and Schapire, 1997). The weights of the examples may be used to obtain resampled versions of the original data set or used directly by the base learner if it supports examples' weights. Boosting methods are known to reduce the bias of the base learners. Oza and Russell (2001) also proposed an on-line Boosting algorithm. However, when compared to the original versions, the results were not so satisfactory as the on-line Bagging approach. Random Forests (Breiman, 2001), like Bagging, increase diversity among the set ensemble by resampling with replacement. The name of this ensemble method is related with the type of base learners it combines: decision trees. The key difference from the previous ensemble methods is that, at each node of a tree, only a random subset of the features are selected as candidates for splitting. A version of on-line Random Forests has already been proposed and combines the randomized FIMT-DD algorithm with on-line Bagging (Ikonomovska, 2012).

3. Adaptive Model Rules for High-Speed Data Streams

In this section, an incremental algorithm for learning model rules is presented: the Adaptive Model Rules for High-Speed Data Streams (AMRules) (Almeida et al., 2013). The algorithm is able to adapt a current rule to changes of the data stream and to detect anomalous examples. It also allows differentiating the importance of training examples by handling weights. This feature is exploited by the ensemble method proposed in Section 4.

A rule R is an implication in the form $A \Rightarrow C$. A is a conjunction of conditions based on attribute values. These conditions are called literals L and have different forms depending on the type of the attribute. For numerical attributes they may have the form $L = (X_j > v)$, meaning that the j^{th} attribute of an example $\mathbf{x}_i$ must be greater than a real value v, or $L = (X_j \leq v)$ meaning that x_{ij} must be less or equal to v. For categorical data a literal is in the form of $L : (X_j = v)$, where v is a value in the domain of X_j. The C part of the rule is a function that returns the prediction $\hat{y}_i$ if the example $\mathbf{x}_i$ meets the conditions given by A. Let $I(\cdot)$ be a function that returns 1 if the argument is true, and 0 otherwise. Formally, a rule R_l covers an example $\mathbf{x}_i$ if $\left(\prod_{L \in A_l} L(\mathbf{x}_i)\right) = 1$, where $L(\mathbf{x}_i)$ is a function that returns 1 if $\mathbf{x}_i$ satisfies L, and 0 otherwise:

$$L(\mathbf{x}_i) = \begin{cases} I(x_{ij} > v) & \text{if } L \text{ is in the form } (X_j > v) \\ I(x_{ij} \leq v) & \text{if } L \text{ is in the form } (X_j \leq v) \\ I(x_{ij} = v) & \text{if } L \text{ is in the form } (X_j = v) \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Each rule R_l has associated a data structure $\mathcal{L}_l$ containing the information needed to make predictions, find changes, detect anomalies, and the sufficient statistics used for expanding the rule.

3.1. Learning a rule set

Let $\mathcal{R} = \{R_1, \cdots, R_r\}$ be a set of r rules and D a *default rule*. The order of the rules in the set is related with its creation time (R_1 was created before R_2, R_2 before R_3, and so on). A rule set may be ordered or unordered. For the former, only the first rule covering the example should be considered. For the latter, all rules covering the example are taken into account. Let the support $S^u(\mathbf{x}_i)$ of an unordered rule set $\mathcal{R}$ for a given example $\mathbf{x}_i$ be the set of all rules that covers $\mathbf{x}_i$:

$$S^u(\mathbf{x}_i) = \left\{ R_l : \left(\prod_{L \in A_l} L(\mathbf{x}_i) \right) = 1 \right\}. \tag{4}$$

The support $S^o(\mathbf{x}_i)$ of an ordered rule set is the first rule of S^u or an empty set $\emptyset$ if S^u is also an empty set,

$$S^o(\mathbf{x}_i) = \begin{cases} \{R_1\}, R_1 \in S^u(\mathbf{x}_i) & \text{if } |S^u(\mathbf{x}_i)| > 0 \\ \emptyset & \text{otherwise,} \end{cases} \tag{5}$$

where $|\cdot|$ is the cardinality of a set.

The algorithm starts with an empty rule set $\mathcal{R} = \emptyset$ and a default rule D, whose associated data structure $\mathcal{L}_D$ is initialized to $NULL$. When a new training example $\mathbf{x}_i$ is available, the rules belonging to the set $S(\mathbf{x}_i)$ are updated. For each rule $R_l \in S(\mathbf{x}_i)$, the first step is to verify if $\mathbf{x}_i$ is an anomalous example (see subsection 3.5). If so, $\mathbf{x}_i$ is not used to update R_l. The second step is to test R_l for change by monitoring the rule's on-line error (see subsection 3.6). If change is detected R_l is removed from the rule set $\mathcal{R}$. Otherwise, the data structure $\mathcal{L}_l$ associated with R_l is updated and the rule is expanded. In fact, the rule expansion is considered only from time to time (every N_{min} examples) in order to reduce the computational cost. If $\mathbf{x}_i$ is covered by no rule ($S(\mathbf{x}_i) = \emptyset$) the data structure of the default rule $\mathcal{L}_\mathcal{D}$ is updated. D may also be expanded if the number of instances seen by it is a multiple of N_{min}. If D expands it is added to the rule set, $\mathcal{R} = \mathcal{R} \cup D$, and a new default rule D is created.

3.2. Expanding a rule

The process of expanding a rule R_l consists of adding a new literal L to the set of conjunctions A_l. This literal is chosen by determining the attribute and split-point that minimizes some cost-function based on the examples seen so far. Also, a predetermined confidence-level on the split must be guaranteed so that a rule can be expanded.

We use the variance reduction (VR) to determine the merit of a split, which is a variation of the standard deviation reduction measure described in (Ikonomovska et al., 2011). The formula of VR for splitting an numerical attribute X_j using v as split-point is given by

$$VR(X_j, v) = var(E) - \frac{|E_L|}{|E|} var(E_L) - \frac{|E_R|}{|E|} var(E_R), \tag{6}$$

$$var(E) = \frac{1}{|E|} \sum_{i=1}^{|E|} (y_i - \bar{y})^2 = \frac{1}{|E|} \sum_{i=1}^{|E|} {y_i}^2 - \frac{1}{|E|} \left(\sum_{i=1}^{|E|} y_i \right)^2, \tag{7}$$

where E is the set of examples seen by the rule since its last expansion, E_L is the set of examples $\{\mathbf{x}_i \in E : x_{ij} \le v\}$, E_R is the set of examples $\{\mathbf{x}_i \in E : x_{ij} > v\}$, and $\overline{y} = \frac{1}{|E|} \sum_{i=1}^{|E|} y_i$. For a categorical attribute E_L is the set of examples whose j^{th} attribute equals v, and E_R is its complement set.

The number of examples n required to expand a rule is given by the Hoeffding bound (Hoeffding, 1963), presented in Equation 8. It guarantees that the true mean of a random variable r, with range R, will not differ from the sample mean more than ϵ with probability $1 - \delta$.

$$\epsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{8}$$

The best two potential splits are compared, dividing the second-best VR score by the best one to generate a ratio r in the range 0 to 1. To decide if the rule is expanded or not, it is verified if the upper bound of the ratio of the sample average ($r^+ = r + \epsilon$) is below 1. If $r^+ < 1$ the true mean is also below 1, meaning that with confidence $1 - \epsilon$ the best attribute and split-point of the data are the ones being tested. Nevertheless, the measurements of the two best splits are often extremely similar and, despite ϵ decreases considerably as more examples are seen, it is not possible to select which one is better with certainty. In this case, a threshold τ on the error is used, and if $\epsilon < \tau$ the split option with higher VR is chosen to expand the rule.

A modified version of the extended binary search tree (Ikonomovska et al., 2011) was used to maintain the sufficient statistics needed to compute VR. We limited the maximum number of split-points to a predefined value in order to reduce the memory consumption and speed-up the split selection procedure while having low impact in the error of the learning algorithm. Also, weights of the examples are considered while updating the statistics.

3.3. Rule prediction strategies

The rules implement three distinct strategies to make prediction: (1) the weighted target attribute mean of the examples previously covered by the rule; (2) the output of a linear model built with the same examples; and (3) an adaptive strategy which selects at each moment one of the previous strategies based on the on-line estimation of its MAE. Hence, these prediction functions output not only the prediction for a given instance but also the current estimation of the error, which can be used to measure the confidence of the prediction.

The target mean of a rule is computed as $\hat{y}_i = \frac{1}{Z}\sum_{j=1}^{n} w_j y_j$, $Z = \sum_{j=1}^{n} w_j$, where n is the number of examples seen by the rule since its last expansion, and w_i is the weight associated to the i^{th} example. To learn the linear model $\hat{y}_i = \beta_0 + \sum_{j=1}^{d} \beta_j x_{ij}$ the perceptron algorithm is used and is trained using an incremental gradient descent method. When a new training example is available it is standardized considering the means and standard deviations of the attributes of the examples seen so far. Next, the output is computed using the current weights β. Then, the weights are updated using the Delta rule: $\beta_j \leftarrow \beta_j + \eta w_i (\hat{y}_i - y_i) x_{ij}$, where $\hat{y}_i$ is the output of the linear model, y_i the real value and η is the learning rate. The prediction is computed as the "denormalized" value of $\hat{y}_i$.

The on-line estimation of the weighted error, e, follows a fading factor strategy. In order to do so, two values are monitored: the total sum of absolute deviations T and the total sum of the weights of the objects used for learning W. When a new example $(\mathbf{x}_i, y_i)$ arrives for training, T and W are updated as follows: $T \leftarrow \alpha T + w_i|\hat{y}_i - y_i|$ and $W \leftarrow \alpha W + w_i$, where $0 < \alpha < 1$ is a parameter that controls the importance of the oldest/newest examples.

3.4. Rule set prediction strategies

As mentioned before, a rule set may be ordered or unordered. The prediction of an ordered rule set is simply the prediction of the first rule that covers $\mathbf{x}_i$ if $S^o(\mathbf{x}_i) \neq \emptyset$, or the prediction of the default rule D otherwise. The prediction of an unordered set is given by a weighted vote approach, such that the contribution of rules with lower error are higher.

The weights of the votes $\theta_l \in [0, 1]$ for the unordered rule sets are inversely proportional to the estimated mean absolute error e_l of each rule R_l. The weighted prediction of an unordered set is computed as

$$\hat{y}_i = \sum_{R_l \in S(\mathbf{x}_i)} \theta_l \hat{y}_i^l, \; \theta_l = \frac{(e_l + \varepsilon)^{-1}}{\sum\limits_{R_j \in S(\mathbf{x}_i)} (e_j + \varepsilon)^{-1}}, \tag{9}$$

where $\hat{y}_i^l$ is the prediction of R_l, and ε is a small positive number used to prevent numerical instabilities. The prediction uncertainty u_i is estimated as the current weighted error of the rules that cover $\mathbf{x}_i$:

$$u_i = \sum_{R_l \in S(\mathbf{x}_i)} \theta_l e_l. \tag{10}$$

An ordered set has usually less rules than an unordered set because it specializes one rule at time. The ordered rules also need to consider the previous rules which makes the interpretation of complex sets harder. Unordered rule sets are more modular since they can be interpreted alone.

3.5. Detection of anomalies

Detection of outliers or anomalous examples are very important in on-line learning because of the potential impact in the performance of the learner. AMRules detects context anomalies and prevents from using them in the learning process.

Given $\mathbf{x}_i$, for each attribute X_j, the probability $\Pr(X_j = x_{ij}|\mathcal{L}_l)$ of observing the value x_{ij} in a rule $R_l \in S(\mathbf{x}_i)$ is computed using the respective data structure $\mathcal{L}_l$. The univariate anomaliness score for X_j is given by

$$U_j = 1 - \Pr(X_j = x_{ij}|\mathcal{L}_l). \tag{11}$$

The statistics maintained in $\mathcal{L}_l$ include the mean $\overline{X}_j$ and standard deviation σ_j of each attribute. Therefore, Equation 11 may be computed using several strategies, such as, Normal distribution and Z-scores. From a set of experiments not described here, using the Cantelli's inequality (Bhattacharyya, 1987) seems to be an effective strategy. For any real number $b > 0$,

$$\Pr(|x_{ij} - \overline{X}_j| \geq b) \leq \frac{\sigma_j^2}{\sigma_j^2 + b^2}. \tag{12}$$

Therefore, $Uscore_i$ may be computed as $Uscore_j = 1 - \frac{\sigma_j^2}{\sigma_j^2 + |x_{ij} - \overline{X}_j|^2}$. If $Uscore_j$ is higher than a threshold λ_U (typically 90%), x_{ij} is considered an anomaly for the context of R_l.

Assuming the attributes are independent, the joint degree of anomaliness is computed over all attributes whose univariate score is higher than λ_U: $\prod_{j:Uscore_j > \lambda_U} Uscore_j$. In order to avoid numerical instabilities logarithms are applied. Equation 13 presents the degree of anomaliness, normalized into the interval $[0, 1]$, where 1 corresponds to all attributes being anomalous and 0 means none of the attributes are anomalous. If *Ascore* is higher than a threshold λ_M (typically 0.99) the example is discarded from training.

$$Ascore = \frac{\displaystyle\sum_{j:Uscore_j > \lambda_U} \log\left(Uscore_j\right)}{\displaystyle\sum_{j=1}^{d} \log\left(Uscore_j\right)} \tag{13}$$

An anomaly is more likely to be reported when $\mathcal{L}_l$ as seen few examples. Hence, only rules trained with more than a predefined number of examples (after the last expansion) are used for anomaly detection.

3.6. Change detection

The Page-Hinkley (PH) test (Page, 1954) is used to monitor the evolution of the on-line error e_i of a rule. PH test considers a cumulative variable m_n which is defined as the accumulated difference between the observed values e_i and their mean at the current moment:

$$m_n = \sum_{i=1}^{n} e_i - \bar{e}_n - \gamma, \; \bar{e}_n = \frac{1}{n}\sum_{i=1}^{n} e_i \tag{14}$$

where γ correspond to the magnitude of changes that are allowed. The minimum value of m_n at the current moment is also maintained: $M_n = \min_{i=1,\cdots,n} m_i$. When the difference $(m_n - M_n)$ is greater than a giver threshold λ a change is signaled and the correspondent rule is removed from $\mathcal{R}$.

4. Random AMRules: an ensemble of rule sets

In this section, an ensemble of rules model is presented. The motivation is to boost the performance of AMRules since ensemble methods are known to be a general approach to improve the performance of learning algorithms.

Important insights may be obtained by performing the bias-variance decomposition of the error of a learning algorithm (Domingos, 2000). This information is very useful for designing the ensemble strategy. On one hand, regression models with an high-variance profile may be improved by perturbing the set of examples used for training. On the other hand, models with low-variance profile benefit from perturbing the set of attributes used while training the model. We observed that AMRules has a low-variance profile. As such, the ensemble of AMRules model is designed following the Random Forests idea in order to take advantage of both the attributes' and examples' perturbation. The proposed ensemble method is referred as Random AMRules.

4.1. Training an ensemble of rule sets

Random AMRules starts by initializing an ensemble $\mathcal{F}$ with k models $\hat{f}_m$ built using the AMRules regression algorithm. When a rule R_l is created or expanded a subset of the data attributes with size d', $1 \leq d' \leq d$, is randomly chosen. The next split decision for R_l consider only the attributes belonging to this subset. This procedure prevents the models from being correlated.

Every time a training example $(\mathbf{x}, y)$ is available, the on-line error estimation of each predictor $\hat{f}_m$ is updated, and it is sent to each individual AMRules learner $\hat{f}_m$ for training. The on-line error of each $\hat{f}_m$ is estimated using a fading factor strategy, identically to the rule's error estimation described in Subsection 3.3. In order to perturb the training set for each model, we apply an on-line Bagging approach similar to the one described in (Oza and Russell, 2001). For each learning model $\hat{f}_m$ a different weight p_m is sampled from the Poisson distributions, $p_m \sim Poisson(1)$, but instead of updating each model p_m times, we make use of the AMRules capability of handling instance weights. Since the model is only updated once, or never if $p_m = 0$, the computation time for training the algorithm is reduced. The algorithm use for training Random AMRules is presented in Algorithm 1.

```
Algorithm 1: Training Random AMRules
Input: $\mathcal{D}$ - Stream of examples, $k$ - size of the ensemble, $\alpha$ - fading factor
Result: $\mathcal{F}$ - a set of predictors
begin
    Initialize $\mathcal{F} \leftarrow \{\hat{f}_1, \cdots, \hat{f}_k\}$ using AMRules as base learner
    Initialize $T_m \leftarrow 0$, $W_m \leftarrow 0$, $m \in \{1, \cdots, k\}$
    foreach $(\mathbf{x}, y) \in \mathcal{D}$ do
        foreach $m \in \{1, \cdots, k\}$ do
            $p_m \sim Poisson(1)$
            if $p_m > 0$ then
                Update on-line error $T_m \leftarrow \alpha T_m + p_m|\hat{f}_m(\mathbf{x}) - y|$, $W_m \leftarrow \alpha W_m + p_m$
                Update predictor $\hat{f}_m \leftarrow$ AMRules$(\hat{f}_m, \mathbf{x}, y, p_m)$
```

4.2. Predicting with an ensemble of rule sets

The prediction $\hat{y}_*$ of Random AMRules is computed as a linear combination of the estimations produced by the models $\hat{f}_m \in \mathcal{F}$: $\hat{y}_* = \hat{f}_*(\mathbf{x}) = \sum_{m=1}^{k} \theta_m \hat{f}_m(\mathbf{x})$. The weights θ_m can be computed using different weighting functions $\mathcal{V}$. The most straightforward approach is using an uniform weighting function, such that all the predictors have the same importance, $\theta_m = \frac{1}{k}$. However, the current information about the on-line error e_m of each individual predictor $\hat{f}_m$ can be used define the weights. It is expected that predictors with lower error should output more accurate predictions. Therefore, its contribution in the weighting function should be higher. Instead of using e_m, which measures the overall current error of the rule set associated to $\hat{f}_m$, the prediction uncertainty u_m defined in Equation 10 may also be used. For the former case, the weighting function is *constant* in the sense that no matter the example $\mathbf{x}$, the weights are already defined. For the later case, the weighting function is *dynamic* since the weights depend on $\mathbf{x}$ because the prediction uncertainty is computed regarding the rules covering $\mathbf{x}$. Any weighting function $\mathcal{V}$ that considers an error/uncertainty measure can be used to obtain the weights for the ensemble's prediction. For instance, similarly to Equation 9, the weights for an inverse-error weighting function may be defined as:

$$\theta_m = \frac{(b_m + \varepsilon)^{-1}}{\sum\limits_{m=1}^{k} (b_m + \varepsilon)^{-1}}, \tag{15}$$

where b_m is e_m or u_m, depending if the voting type $\mathcal{T}$ is static or dynamic. The pseudo-code of the prediction mechanism is presented in Algorithm 2.

5. Experimental Evaluation

In this section, the experimental evaluation of Random AMRules is presented. The influence of the number of attributes and size of the ensemble is analyzed, and variants of the proposed ensemble method are assessed. Also, the performance of Random AMRules is compared with the performance of its base learner.

```
Algorithm 2: Predicting with Random AMRules
Input: x - example, $\mathcal{V}$ - weighting function, $\mathcal{T}$- voting type.
Result: $\hat{y}_*$ - the prediction
begin
    foreach $m \in \{1, \cdots, k\}$ do
        $(\hat{y}_m, u_m) \leftarrow \hat{f}_m(\mathbf{x})$
    if $\mathcal{T} =$ static then
        $e_m \leftarrow \frac{T_m}{W_m}, \forall m \in \{1, \cdots, k\}$
        $\{\theta_1, \cdots, \theta_k\} \leftarrow \mathcal{V}(\{e_1, \cdots, e_k\})$
    else
        $\{\theta_1, \cdots, \theta_k\} \leftarrow \mathcal{V}(\{u_1, \cdots, u_k\})$ ; // $\mathcal{T} =$ dynamic
    $\hat{y}_* = \sum_{m=1}^{k} \theta_m \hat{f}_m$
```

5.1. Data sets

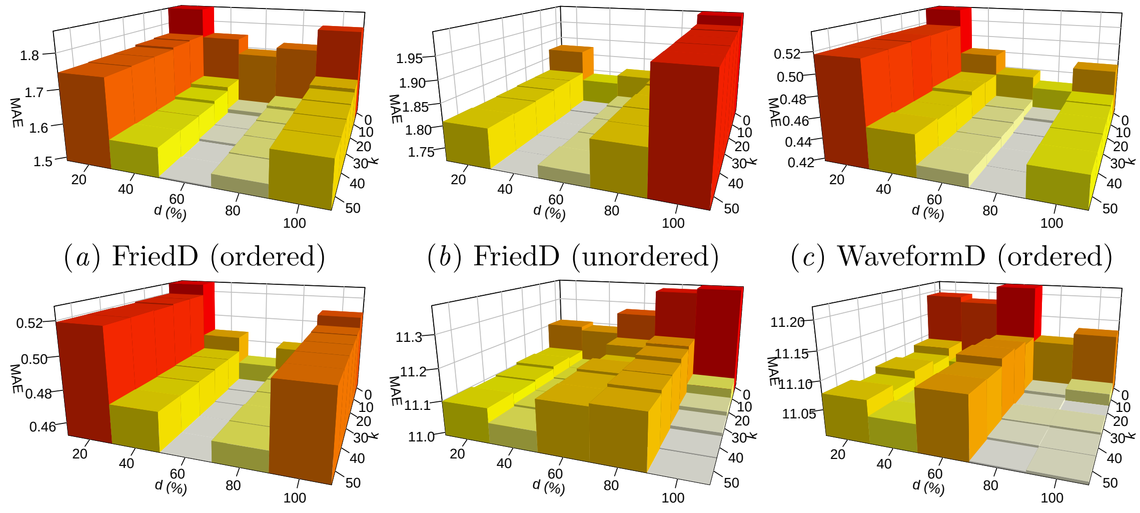
Three data sets were used to evaluate Random AMRules and variants. **FriedD** is an artificial dataset composed of 256000 examples generated similarly to the dataset described in (Breiman et al., 1984), but contains a drift that starts at the 128001^{st} instance. **WaveformD** is an artificial data set also containing 256000 examples generated as described in (Breiman et al., 1984), and a drift which starts at the 128001^{st} instance. The data set has 3 classes of waves labeled, and the examples are characterized by 21 attributes that include some noise plus 19 attributes that are all noise. **Airlines1M** 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 120 million records (11.5 GB memory size) (Ikonomovska et al., 2011). In our experiments we used only the first million examples of the original data set in order to perform our experiments quicker.

5.2. Experimental results

The experimental results presented in this subsection were obtained using prequential evaluation with a sliding window of 10000 examples. The sliding window is only used to store the MAE and RMSE measurements, so, it does not influence the behavior of the learning algorithms. AMRules and Random AMRules algorithms were implemented using *MOA: Massive Online Analysis* framework (Bifet et al., 2010). The values of the main parameters of AMRules were set to $\delta = 10^{-7}$, $\tau = 0.05$, $N_{min} = 200$, $\lambda = 35$ and $\gamma = 0.05$. The α parameter for the estimation of the on-line errors was set to $\alpha = 0.99$. For each rule and attribute, the maximum number of split-points was defined as 50.

5.2.1. Number of attributes and size of the ensemble

A study on the influence of the ensemble size and the number of attributes was performed for the purpose of evaluating the behavior of Random AMRules with variations of these parameters. Figure 1 presents the average MAE of ten runs of Random AMRules using



(*a*) FriedD (ordered) (*b*) FriedD (unordered) (*c*) WaveformD (ordered)

(*d*) WaveformD (unordered) (*e*) Airlines1M (ordered) (*f*) Airlines1M (unordered)

Figure 1: Average MAE values for 10 runs of Random AMRules using different numbers of attributes and sizes of the ensemble, both for ordered (RAMRules^o) and unordered (RAMRules^u) rule sets.

uniform-weighted voting, with ordered and unordered rule sets, for each combination of $k \in \{1, 10, 20, 30, 40, 50\}$ and $d \in \{\lfloor 0.2n \rfloor, \lfloor 0.4n \rfloor, \lfloor 0.6n \rfloor, \lfloor 0.8n \rfloor, n\}$, where n is the number of examples of the data set.

As the size of the ensemble k increases, the error is expected to converge to its minimum. However, the more base learners are trained the higher the computational requirements will be. Thus, k should be determined by increasing the value of k until the error stabilizes. It can be seen that when the ensemble is a singleton ($k = 1$) the average MAE is clearly higher than when several predictors are combined. For $k \geq 30$ the MAE seems to stabilize for all numbers of attributes and data sets. The best results for FriedD data set were obtained using 60% and 40% of the attributes by combining ordered and unordered rules sets, respectively. For the WaveformD data set the best performances were achieved using 80% of the attributes for the ordered rule sets and 60% for the unordered rule sets. In both these data sets, using a subset of the features for expanding the rules is clearly advantageous. However, the lowest average MAE for the Airlines1M data set was obtained using all attributes.

5.2.2. Comparison between Random AMRules, AMRules, FIMT-DD and IBLSTREAMS

The prequential RMSE of AMRules Random AMRules algorithms for FriedD, WaveformD and Airlines1M data sets are shown in Figure 2, considering both ordered (AMRules^o, RAMRules^o) and unordered (AMRules^u, RAMRules^u) rule sets. Also, the RMSE of FIMT-DD and IBLSTREAMS algorithms are presented for comparison. The results for FIMT-DD were obtained using the available implementation in MOA framework. The experiments with IBLSTREAMS were performed using the code available at www.uni-marburg.de/

fb12/kebi/research/software/iblstreams. For Random AMRules, the size of the ensembles was set to $k = 30$ since it provides a good trade-off between error and computational cost, and the number of features varies depending on the data set and the type of rule sets. The number of features was selected according to the best results presented before.

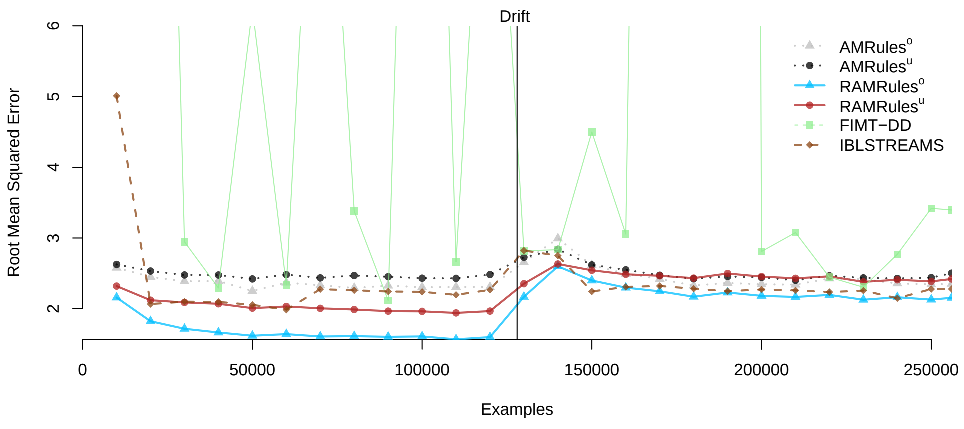
As expected, the ensemble methods outperform the single-learner versions in all data sets. The behavior of FIMT-DD is unstable, achieving the poorest performance in all data sets. The reason for this instability should be related with the linear model in the leaves of the tree (a Perceptron trained using an incremental stochastic gradient descent method, identical to the one use by AMRules). This linear model requires the observation of some examples before being reliable. The adaptive approach of AMRules prevents these instabilities from occurring. The Random AMRules and AMRules models using ordered rule sets obtained lower RMSE than the models obtained using unordered rule sets for the FriedD and WaveformD data sets. It can be seen that the RMSE increases after the drifts occur and then decreases as more examples are evaluated. For the FriedD data set, the best results were achieved by RAMRules^o, followed by RAMRules^u and IBLSTREAMS. When comparing RAMRules^u with IBLSTREAMS, the former obtained lower RMSE before the drift, and the later performed better after the drift occur. For the WaveformD data set, the best results were again obtained by RAMRules^o, followed by RAMRules^u, AMRules^o and AMRules^u. In this data set, these algorithms clearly perform better than IBLSTREAMS and FIMT-DD. For the Airlines1M data set, the Random AMRules outperformed the other algorithms. However, in this case, the behavior of the ensemble method using ordered and unordered rule sets is very similar. In fact, the unordered rule set version achieved a little less RMSE (19.556) than the ordered version (19.595).

5.2.3. Comparison between weighting vote functions

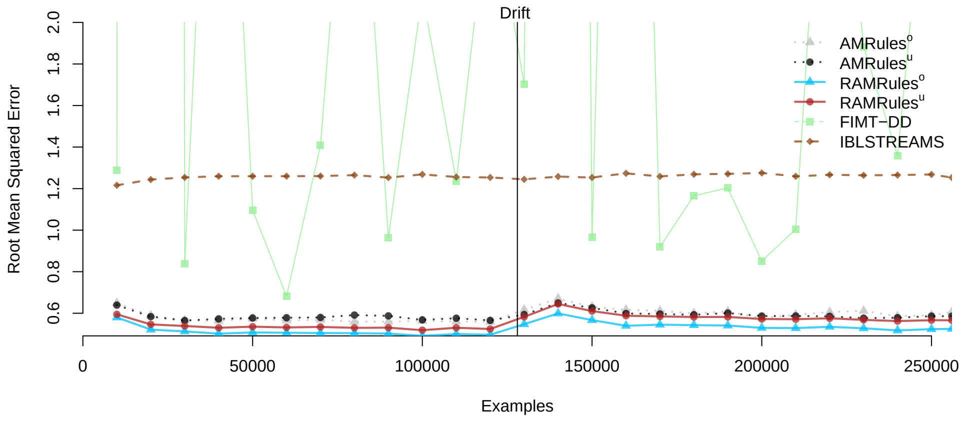
Figure 3 compares the performance of different weighting vote functions for Random AMRules: uniform (U), and inverse-error weighting function both for static (IS) and dynamic (ID) voting types. For ordered rule sets, the U and IS voting strategies have very similar behaviors (the lines almost perfectly overlap). Nonetheless, the IS strategy achieved slightly less RMSE than U in all data sets. For unordered rule sets, the best strategy for FriedD data set was IS. In this case, it can be seen that the RMSE using IS is clearly smaller than U, specially after the drift occurs. For the WaveformD data set, the best strategy was ID, followed by IS, which clearly outperformed the uniform voting approach. In this case, using a dynamic voting type was beneficial. For the Airlines1M dataset, the results were very similar for all cases but, again, IS strategy obtained slightly less RMSE than the others.

6. Conclusions

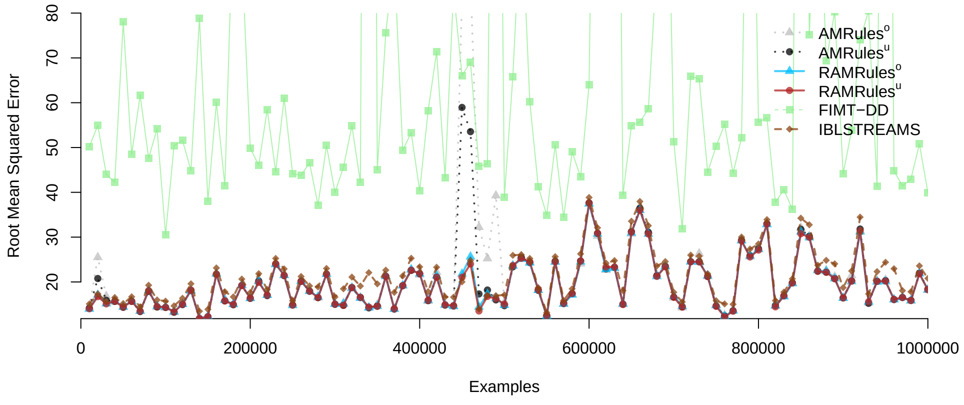
A new rule-based ensemble method for regression capable of detect change and anomalous examples in a data stream was proposed. The proposed method perturbs both the set of examples and the set of attributes in order to reduce the error on regression tasks. The method was evaluated using different weighting vote functions for obtaining the ensembles' predictions, and compared to its base learner and two state-of-the-art regression algorithms for data streams.



(*a*) FriedD

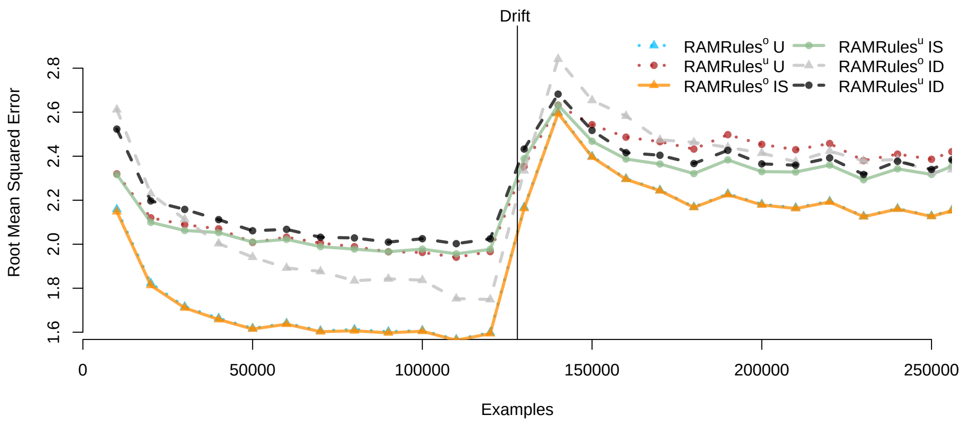


(*b*) WaveformD

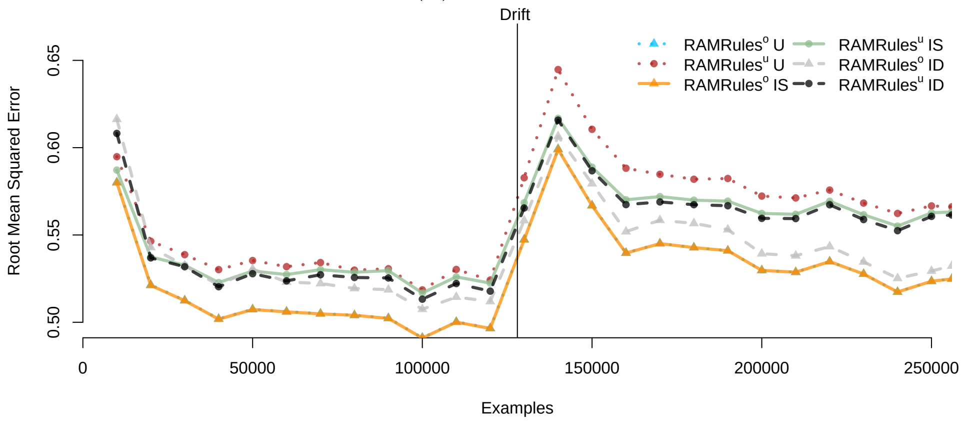


(*c*) Airlines1M

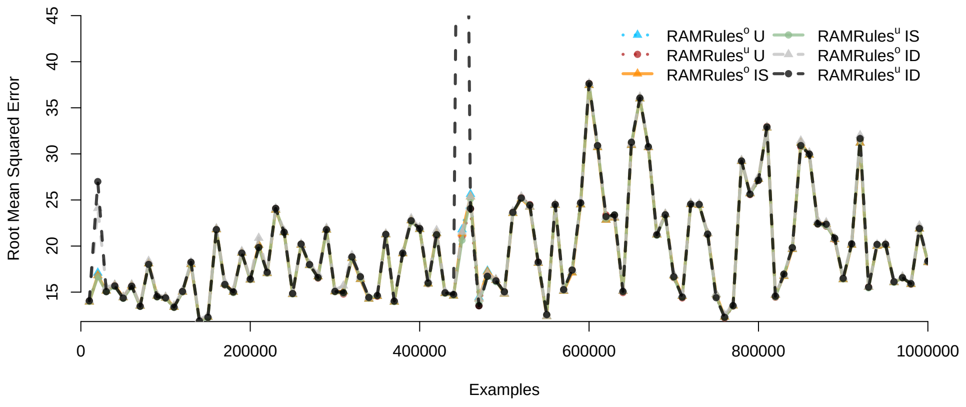
Figure 2: Prequential root-mean-squared error of AMRules and Random AMRules for FriedD, WaveformD and Airlines1M data sets.



(a) FriedD



(b) WaveformD



(c) Airlines1M

Figure 3: Prequential root-mean-squared error of Random AMRules using different weighting vote functions for FriedD, WaveformD and Airlines1M data sets.

Experimental results in two synthetic data sets and one real non-stationary data stream shown that combining the predictions of several rule sets lead to better predictions. In two of the data sets, ensembles built using ordered rule sets obtained clearly better performances than using unordered rule sets. For the other data set, the behavior of both types of rule sets was similar, but with a little advantage for the unordered version. Also, the proposed ensemble method outperformed the other two state-of-the-art algorithms. Regarding the weighting vote functions used for aggregating the estimates of the set of predictors, assigning more relevance to the predictors with less on-line error was a good strategy, especially when using unordered rule sets.

Acknowledgments

The authors acknowledge the financial support given by the project ERDF through the COMPETE Programme and by National Funds through 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). The projects "NORTE-07-0124-FEDER-000056/59" are financed by the North Portugal Regional Operational Programme (ON.2 O Novo Norte), under the National Strategic Reference Framework (NSRF), through the Development Fund (ERDF), and by national funds, through the Portuguese funding agency, Fundação para a Ciência e a Tecnologia (FCT).

References

Ezilda Almeida, Carlos Ferreira, and João Gama. Adaptive model rules from data streams. In Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip elezn, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 8188 of *Lecture Notes in Computer Science*, pages 480–492. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-40987-5.

BB Bhattacharyya. One sided chebyshev inequality when the first four moments are known. *Communications in Statistics-Theory and Methods*, 16(9):2789–2791, 1987.

Mohamed Bibimoune, Haytham Elghazel, and Alexandre Aussem. An Empirical Comparison of Supervised Ensemble Learning Approaches. In *International Workshop on Complex Machine Learning Problems with Ensemble Methods COPEM@ECML/PKDD'13*, pages 123–138, September 2013.

Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. Moa: Massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, August 2010. ISSN 1532-4435.

Leo Breiman. Bagging predictors. *Machine Learning*, pages 123–140, 1996.

Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA, 1984.

Thomas Briegel and Volker Tresp. Robust neural network regression for offline and online learning. In *Advances in Neural Information Processing Systems*, pages 407–413, 2000.

Zhe Chen. Bayesian filtering: From kalman filters to particle filters, and beyond. *Statistics*, 182(1):1–69, 2003.

Thomas G Dietterich. Ensemble methods in machine learning. In *Multiple classifier systems*, pages 1–15. Springer, 2000.

Pedro Domingos. A unified bias-variance decomposition. In *Proceedings of 17th International Conference on Machine Learning.*, pages 231–238, 2000.

Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM, 2000.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. of computer and system sciences*, 55(1):119–139, 1997.

João Gama. *Knowledge Discovery from Data Streams*. Chapman and Hall / CRC Data Mining and Knowledge Discovery Series. CRC Press, 2010. ISBN 978-1-4398-2611-9.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Elena Ikonomovska. *Algorithms for Learning Regression Trees and Ensembles on Evolving Data Streams*. PhD thesis, Jožf Stefan International Postgraduate School, 2012.

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

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

João Mendes-Moreira, Carlos Soares, Alípio Mário Jorge, and Jorge Freire De Sousa. Ensemble approaches for regression: A survey. *ACM Comput. Surv.*, 45(1):10:1–10:40, December 2012. ISSN 0360-0300.

Nikunj C. Oza and Stuart Russell. Online bagging and boosting. In *In Artificial Intelligence and Statistics 2001*, pages 105–112. Morgan Kaufmann, 2001.

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

Duncan Potts and Claude Sammut. Incremental learning of linear model trees. *Machine Learning*, 61(1-3):5–48, 2005.

Ammar Shaker and Eyke Hüllermeier. Iblstreams: a system for instance-based classification and regression on data streams. *Evolving Systems*, 3:235–249, 2012. ISSN 1868-6478.