Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 27, Num. 1 (2020) 36-47

RESEARCH ARTICLE

An Online Tree-Based Approach for Mining Non-Stationary High-Speed Data Streams

Uma Abordagem Baseada em Árvore Online para Mineração de Fluxos de Dados Não-Estacionários em Alta Velocidade

Isvani Inocencio Frías Blanco [1], Agustín Alejandro Ortiz Díaz [2]*,Fabiano Baldo [2],Laura María Palomino Mariño [3].

Abstract: This paper presents a new learning algorithm for inducing decision trees from data streams. In these domains, large amounts of data are constantly arriving over time, possibly at high speed. The proposed algorithm uses a top-down induction method for building trees, splitting leaf nodes recursively, until none of them can be expanded. The new algorithm combines two split methods in the tree induction. The first method is able to guarantee, with statistical significance, that each split chosen would be the same as that chosen using infinite examples. By doing so, it aims at ensuring that the tree induced online is close to the optimal model. However, this split method often needs too many examples to make a decision about the best split, which delays the accuracy improvement of the online predictive learning model. Therefore, the second method is used to split nodes more quickly, speeding up the tree growth. The second split method is based on the observation that larger trees are able to store more information about the training examples and to represent more complex concepts. The first split method is also used to correct splits previously suggested by the second one when it has sufficient evidence. Finally, an additional procedure rebuilds the tree model according to the suggestions made with an adequate level of statistical significance. The proposed algorithm is empirically compared with several well-known induction algorithms for learning decision trees from data streams. In the tests, it is possible to observe that the proposed algorithm is more competitive in terms of accuracy and model size using various synthetic and real-world datasets.
Keywords: Data stream — decision tree — incremental learning — machine learning — online learning

Resumo: Este artigo apresenta um novo algoritmo de aprendizagem que induz árvores de decisão a partir de fluxos de dados. Nesses domínios, grandes quantidades de dados chegam constantemente ao longo do tempo, possivelmente em alta velocidade. O algoritmo proposto usa um método de indução descendente para construir árvores dividindo os nós folha recursivamente até que nenhum deles possa ser expandido. O novo algoritmo combina dois métodos de divisão para a indução da árvore. O primeiro método é capaz de garantir, com significância estatística, que cada divisão escolhida seria a mesma escolhida usando exemplos infinitos. Ao fazer isso, o objetivo é garantir que a árvore induzida online esteja próxima do modelo ótimo. No entanto, esse método de divisão geralmente precisa de muitos exemplos para tomar uma decisão sobre a melhor divisão, o que afeta a melhoria da precisão do modelo de aprendizagem preditiva on-line. Portanto, o segundo método é usado para dividir os nós mais rapidamente, acelerando o crescimento da árvore. O segundo método de divisão se baseia na observação de que árvores maiores podem armazenar mais informações sobre os exemplos de treinamento e representar conceitos mais complexos. O primeiro método de divisão também é usado para corrigir divisões sugeridas anteriormente pelo segundo método, quando existirem evidências suficientes. Por fim, um procedimento adicional reconstrói o modelo de árvore de acordo com as sugestões feitas com um nível adequado de significância estatística. O algoritmo proposto é comparado empiricamente com vários algoritmos de indução bem conhecidos para aprender árvores de decisão a partir de fluxos de dados. Nos testes é possível observar que o algoritmo proposto é competitivo em termos de precisão e tamanho do modelo usando vários conjuntos de dados sintéticos e reais.
Palavras-Chave: Fluxo de dados — árvore de decisão — aprendizado incremental — aprendizado de máquina — aprendizado online

[1] *LexisNexis Risk Solutions Sao Paulo, Brazil*
[2] *CCT-UDESC, Santa Catarina State University, Bazil*
[3] *CIN-UFP, Pernambuco Federal University, Pernambuco, Brazil*
***Corresponding author:** agaldior@gmail.com
DOI: http://dx.doi.org/10.22456/2175-2745.90822 • **Received:** 10/03/2019 • **Accepted:** 17/10/2019

1. Introduction

An emerging challenge for learning algorithms is to be able to process unbounded streams of data constantly arriving at high speed. Common sources of these data streams include the Internet, computer networks, phones, and sensors [1]. Because of the large size, learning algorithms are allowed to read the input data only a small number of times, using limited computing and storage resources. Classical batch learning algorithms are unsuitable for these domains, as they assume that all input data are available at the beginning of the processing, and they are not able to provide online answers. Many incremental and online approaches have therefore been developed, and they are currently a powerful tool for learning from data streams [2].

Decision trees have successfully been used in classification tasks both for batch [3] and online learning [4, 5, 6]. Essentially, a decision tree learner splits leaf nodes in a recursive way, until none of them can be expanded. In online learning, the training data are possibly infinite, and the main problem is then to decide how many examples are necessary to guarantee, with a statistically significant result, that a split chosen at a given moment is the same as that chosen using infinite examples [5, 6]. The most prominent approaches try to induce a decision tree that is identical, theoretically, to that induced by a batch learning algorithm from the entire data stream [4, 5, 6, 7]. Thereby, they aim at ensuring that the tree induced online is close to the optimal model.

To solve the problem, various research work used interval estimation [5, 8, 6, 9]. These approaches rank the possible splits by using a split evaluation function, and the best split is selected when an adequate confidence level can be guaranteed. A higher confidence level used for splitting can provide, with higher probability, a decision tree model that is more similar to that one induced from the entire data stream. Several approaches have computed confidence intervals making no assumption regarding the form of the probability distribution of the input data [5, 8, 6, 9].

Non-parametric interval estimations are commonly favored because real data rarely follow well-known probability density functions. However, the most common split evaluation functions need large deviation bounds to guarantee appropriate confidence levels. These large deviation bounds require more examples being processed for splitting leaf nodes. Therefore, in practice, these well-founded statistical approaches are outperformed in predictive accuracy by algorithms able to split more quickly without theoretical guarantees [7, 10].

For example, a well-known family of algorithms uses the Hoeffding bound for interval estimates [11, 12, 10]. However, it has recently been shown that this family does not guarantee the user-predefined confidence level for splitting [6] and thus, they often split nodes more quickly than algorithms ensuring appropriate confidence levels. This fast splitting can lead to bad split being installed in internal nodes and to large trees being induced. However, larger trees are also able to store more information about the training instances and to represent more complex target concepts. Naturally, speeding up the tree growth can lead to higher levels of predictive accuracy [13]. In order to split leaf nodes with less training data and to speed up the tree growth, various approaches have focused on computing tighter probabilistic bounds for entropy [7], gini index [7], misclassification error [14] or even a combination of these heuristic measures [14].

Some work have also assumed normality in order to obtain tighter bounds [15, 14]. This problem has to lead us to study methods of finding good splits, instead of best. For such, in this paper, we propose a new learning algorithm, named Online Hybrid Tree (OHyT), which combines two different split methods for the tree induction. The first one is based on the split method used by the IADEM family of algorithms [4, 16, 8]. It guarantees statistical significance on each split chosen for internal nodes. As mentioned, this guarantee often requires too many training examples, delaying the tree growth significantly and thus, harming the online predictive accuracy of the decision tree model. The second method tries to overcome this drawback by splitting leaf nodes more quickly, without being exhaustive in finding the best split. During the tree induction, the first split method is also used to correct splits previously suggested by the second one. Then, an additional algorithm rebuilds the tree model according to the suggestions made by the IADEM split method. The empirical study shows that new online hybrid algorithm often reaches higher levels of predictive accuracy than the algorithm which only expands the tree when statistical significance can be guaranteed on each split chosen (IADEM-2 [8]).

The rest of this paper is structured as follows. In Section 2 we review some outstanding research works dealing with the online induction of decision trees from data streams. Then, the proposed algorithm, OHyT, is discussed in Section 3. Section 4 presents an empirical study that shows the performance of the new decision tree algorithm over both synthetic and real data; in this section, we compare the new algorithm with respect to two state-of-the-art learning algorithms based on decision trees: IADEM-2 [8] and the Very Fast Decision Tree algorithm [5]. Finally, Section 5 presents the conclusions, summarizing the most notable results and proposing future work.

2. Related work

Domingos and Hulten [5] proposed the Very Fast Decision Tree algorithm (VFDT) in order to induce decision trees from extremely large datasets. VFDT was designed for situations in which the examples are constantly arriving over time and thus, they cannot possibly be stored on disk because of the large dataset size. To estimate the number of examples needed to split a leaf node, VFDT uses Hoeffding's bound, aiming at guaranteeing that the split test chosen is the same as that chosen using infinite examples.

However, Rutkowski et al. [6] recently showed that Hoeffding's bound is not an adequate tool to solve the problem of estimating this number of examples, and proposed to use

McDiarmid's bound instead. Unfortunately, the probabilistic bounds derived are too large, and they perform poorly in practice [7]. De Rosa and Cesa-Bianchi [7] also proposed another method that uses McDiarmid's bound, providing additional empirical evidence that their trees are more accurate than trees with the same number of leaves generated by other algorithms.

Although the VFDT algorithm applies Hoeffding's bound erroneously, in practice, it often reaches very satisfactory predictive accuracy. This performance is caused because VFDT is able to split leaf nodes very quickly. This fast splitting leads to large trees being induced. Larger trees are able to store more information about the training examples, and thus they can be more accurate even having bad split tests. Therefore, it may be possible to induce decision trees that are more accurate than trees induced by VFDT with the same number of nodes [7]. However, VFDT can reach higher levels of predictive accuracy with the same number of examples.

Alternative approaches used Gaussian approximations instead of these probabilistic bounds in order to compute tighter confidence intervals [15, 9]. OnlineTree2 [17] gives a trivial solution, fixing this number of examples to a threshold. Recently, Rutkowski et al. [14] proposed to combine a splitting criterion based on the misclassification error with the Gini index, showing that such a combination is a promising research area.

The IADEM family of algorithms [4, 16, 8] also estimates confidence intervals by using Chernoff's and Hoeffding's bounds. Different from VFDT, this family assumes that a relative frequency stored in a given leaf node is a sum of independent random variables, not the heuristic measure. Then, Chernoff's and Hoeffding's bounds are applied for this sum. The tree construction by this family uses these estimates into account when splitting leaf nodes.

The algorithm proposed in the next section takes advantage of two different split methods. The first one aims at splitting leaf nodes fast, with a simple method to control the tree size. The second split method, which is able to estimate split tests with high statistical confidence, is used to correct previous split tests suggested by the first method.

3. Online Hybrid Tree

In this section, we present OHyT, an online algorithm based on decision trees for mining high-speed data streams (see Algorithm 1). OHyT starts the tree induction with a single root node and processes all the training examples in their temporal order. It stores no example, but each node only maintains the statistics needed for the tree induction. As an example arrives, it traverses the tree into the corresponding leaf, depending on the split tests currently installed in the internal nodes. For this example, the statistics stored in both, the internal and leaf nodes belonging to this path (from the tree root to the leaf) are updated accordingly.

Procedure OnlineHybridTree :

S is a sequence of examples

$\dot{g}$ is a split evaluation function

δ is one minus the desired probability of choosing the correct attribute at any given node

τ is a tie threshold for splitting

ν controls the maximum number of internal nodes allowed to split by the tie threshold

Result: A decision tree HT updated according to the current training instances

```
1  begin
2    Let HT be a tree with a single leaf (the root)
3    forall training examples in S do
4      Sort example into leaf l using HT
5      Update sufficient statistics in the internal
         nodes belonging to the path for the tree root
         to leaf l
6      Update sufficient statistics in leaf l
7      if examples seen at l are not all of the same
         class then
8        Compute ġ for each possible split
9        Let ȧ be the split with the highest value of
           ġ
10       Let ḃ be the split with the lowest value of
           ġ
11       if IADEM_Test( ȧ, ḃ) then
12         Replace l with an internal node i that
             splits on X_a
13         Mark the internal node i as split by
             IADEM
14         forall branches of the split ȧ do
15           Add a new leaf with initialized
               sufficient statistics
16       else
17         Let ṅ(S) be the number of training
             examples seen by leaf l
18         if ṅ(S) > τ then
19           Let ν′ be fraction between the
               number of internal nodes split by
               the tie-breaking threshold and
               the total number of nodes of the
               current tree
20           if ν′ ≥ ν then
21             ReviseSplit (l)
22           else
23             Let ȧ be the split suggested by
                 the parent of l with greatest
                 height and marked as split by
                 the tie-breaking threshold
24             if ȧ == null then
25               Let ȧ be the split with
                   highest value of ġ in l
26             Replace l with an internal
                 node i that splits on ȧ
27             Mark the internal node i as
                 split by tie-breaking
                 threshold
28             Store l in the split node i and
                 maintain it updated
                 according to new training
                 examples
29             forall branches of the split do
30               Add a new leaf with
                   initialized sufficient
                   statistics
```

Algorithm 1: Online Hybrid Tree induction algorithm.

OHyT uses the IADEM split method in the tree induction. OHyT also splits leaf nodes by an alternative method based on a tie-breaking threshold (τ), to be used when the IADEM split method cannot make a decision on the best split even with a large number of training examples.

During the tree induction, OHyT tries to guarantee a statistically significant result for each split test installed in the internal nodes. Thus, if a given leaf node is split by the tie-breaking threshold, the internal node resulting from this expansion maintains the sufficient information needed to periodically check the previous split decision over time. This periodical checking is again performed by IADEM. If OHyT decides that a different split should be installed in a previous internal node, it changes to a stage in which the splits suggested by the tie-breaking threshold, in the descendant leaf nodes, are chosen in order to rebuild the tree accordingly.

Although the tie-breaking threshold can speed up the tree growth and the convergence of the learning model, it can also harm the quality of the tree (e.g., number of nodes), because tree expansions occur more often without enough information to guarantee good splits. In Algorithm 1, the parameter ν restricts the number of internal nodes created by this alternative split mechanism. The parameter ν is simply defined as a percent of the total number of internal nodes. Then, if the number of internal nodes created by the tie-breaking split method exceeds this percent, OHyT disables the alternative split method.

Next subsections provide more details about the three main components of OHyT: the IADEM split method (Section 3.1), the alternative one based on the tie-breaking threshold (Section 3.2), and the mechanism to revise and change split tests previously installed in internal nodes (Section 3.3).

3.1 Splitting with Confidence

To build a decision tree, it is necessary to find, at each leaf node, the split that is the best in discriminating the classes in the input data. In online learning, this split often involves a single attribute. The possible splits are usually ranked by using some split evaluation function, which measures dependencies between splits and classes. There is a great variety of split evaluation functions, and many investigations have measured the effectiveness of these functions for obtaining good decision trees. However, various studies showed that, in general, there are no significant differences in choosing different split evaluation functions [3].

Each internal node of OHyT has associated one attribute, which is involved in the corresponding split test. Similar to previous approaches [10, 18], decision trees induced by OHyT have a border of virtual nodes. Figure 1 shows a possible decision tree generated by the algorithm, in this case considering four possible splits ($\dot{a}$, $\dot{b}$, $\dot{c}$, and $\dot{d}$). The leaves that form the tree border are called real leaves, and the nodes for all possible splits of real leaves are called virtual leaves. The leaves (real or virtual) have counters that allow the algorithm to estimate a set of variables needed for the tree induction.

Procedure ReviseSplit :

l is a leaf node

Result: A decision tree with split tests revised by IADEM

```
1  begin
2      forall internal node i in the path from tree root to
         the leaf node l do
3          if i is marked as split by misclassification
             error then
4              Compute G_l'(X_i) for each attribute in the
                 leaf l' stored in the internal node i
5              Let X_a be the attribute with highest G_l
6              Let X_b be the attribute with lowest G_l
7              if IADEM_Test(X_a,X_b) then
8                  if i was split by attribute X_a then
9                      Mark i as split by IADEM
10                 else
11                     Let X_c the attribute involved in the
                         split test currently installed in
                         the internal node i
12                     ReplaceSplit (i,X_c)
```

Algorithm 2: Algorithm that checks whether a split test previously installed in an internal node by using the accuracy gain, is the same as that suggested by IADEM.

The process of splitting a leaf consists of removing this real leaf and replacing it with an internal node, in accord with the corresponding virtual node. The virtual leaves related to this split become new real leaves and children of the new internal node.

Let $\mathbf{S}$ be the set of training examples seen at a given leaf node and $\dot{n}(\mathbf{S})$ be the cardinality of $\mathbf{S}$. The OHyT algorithm uses the split method of IADEM to solve the problem of deciding, with statistical significance, how many examples are necessary at each node to select the best split test. The IADEM family uses Hoeffding's (ε_H) and Chernoff's (ε_C) bounds to compute a final confidence interval ($\varepsilon_{\text{IADEM}}$), for a sum of independent random variables ($\overline{X} = \sum_{i=1}^{\dot{n}(\mathbf{S})} X_i$):

$$\varepsilon_H = \sqrt{\frac{1}{2\dot{n}(\mathbf{S})} \ln(1/\delta)} \tag{1}$$

$$\varepsilon_C = \frac{1}{2\dot{n}(\mathbf{S})} \left(3\ln\frac{2}{\delta} + \sqrt{9\ln^2\frac{2}{\delta} + 12\dot{n}(\mathbf{S})\overline{X}\ln\frac{2}{\delta}} \right)$$

$$\varepsilon_{\text{IADEM}} = \begin{cases} 1 & \dot{n}(\mathbf{S}) = 0 \\ \min\{\varepsilon_H, \varepsilon_C, 1\} & \dot{n}(\mathbf{S}) > 0 \end{cases}$$

The IADEM family assumes Boolean values drawn according to independent random variables (e.g., 1 if an example

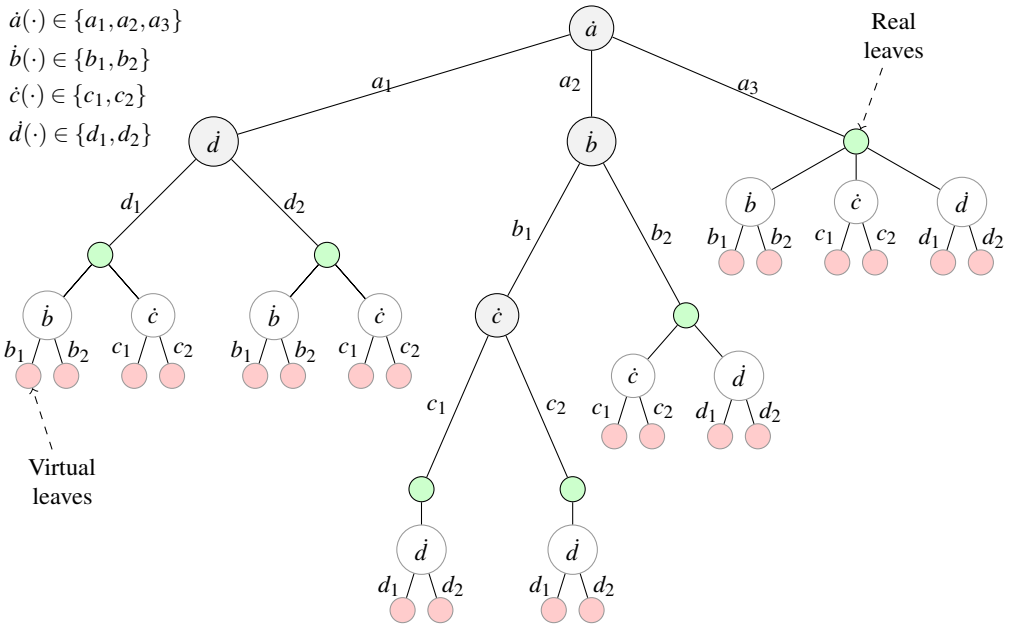

Figure 1. Example of a decision tree induced by the Online Hybrid Tree algorithm.

has a given class label and 0 otherwise). The leaf nodes store relative frequencies in accord with these random variables. Each relative frequency has then associated a confidence interval. Therefore, even with a single tree structure, we can say that the IADEM family of algorithms maintains a variety of different trees because each relative frequency is not unique (it has associated a confidence interval). For each possible combination of values in the corresponding confidence interval having a possible tree, and every tree having vectors in leaf nodes whose components are relative frequencies. Each vector must meet certain restrictions; for example, the sum of the probabilities of reaching a leaf node must be one.

Ramos and Morales [4] also proposed an algorithm to calculate confidence intervals for any heuristic measure in leaf nodes. Different from VFDT, in a given leaf node, the best split test is compared with the worst one based on these interval estimates, assuming that the strict determination of the best split test is not decisive to achieve good trees. If these two confidence intervals are very different (e.g., a split test is statistically better than the worse one) or very similar (implying that all split tests have the same importance), then the IADEM family selects the best split test to expand the corresponding leaf node.

3.2 Breaking Ties in Split Evaluations

It may be that the contending splits are approximately equally good in discriminating the classes in the input data and thus, the split evaluation function would approximately take the same value for these splits. In this situation, the decision tree algorithm would require too many training examples to decide between them, only based on the confidence intervals. If the different split options have similar gain, waiting too long to make a decision on the best split can harm the accuracy of the tree because the tree growth can get paused for a long time.

Domingos and Hulten [5] proposed a tie-breaking parameter τ to be used when various contending splits cannot be separated by only comparing confidence intervals. This way, the current best split is chosen if the probabilistic bound (ε_H) is less than this threshold (τ). This tie-breaking parameter can also be viewed as fixing a number of training example to make a decision on the best split, as the threshold (τ), the probabilistic bound (ε_H) and the number of examples ($\dot{n}(\mathbf{S})$) are related by Equation (1). This method is therefore similar to that one used by OnlineTree2 [19, 17], which fixed the number of training examples seen to evaluate the contending splits.

The configuration of this threshold can also be viewed as adjusting the importance that the best split has for the decision tree algorithm. Larger values give less importance to selecting the best split but can also speed up the tree growth. OHyT controls the tree growth rate by means of the tie-breaking threshold τ and the parameter ν (see Algorithm 1). The threshold τ forces to split on the best-observed split at a given point, without having enough information to make a decision on the best split with a statistically significant result.

OHyT also maintains the balance between splits made by this alternative split method and splits made by the IADEM test. This balance is controlled by means of the parameter ν. OHyT revises the coherence of previous splits if, in the path traversed by the training example, the tree has reached the maximum number of splits made by the accuracy gain test, or by reaching the tie-breaking threshold.

3.3 Revising Splits

As mentioned, the tree growth rate is regulated by means of the parameter ν (see Algorithm 1), which fixes the maximum

number of internal nodes that are allowed to split by the accuracy gain, or by the tie-breaking parameter τ. When this number reaches the parameter ν, OHyT checks the corresponding splits by using the IADEM split method (see Algorithm 2), in order to again enable both split mechanisms.

To change a split previously installed in an internal node, Algorithm 3 performs a recursive call for all the children. Thus, there are two possible options to be considered in this recursive algorithm: (1) installing the required split in a leaf (lines 2–6); or (2) installing the split in an internal node (lines 7–16). In a leaf node, the algorithm simply forces the leaf to split. In an internal node, the algorithm firstly ensures that all its children have split accordingly, to then move up the required split from the children, move down the current split to be changed, and update the parent-child relationships in accord with the new structure.

Figure 2 illustrates an example in which Algorithm 3 installs a new split ($\dot{b}$), different from the previous one ($\dot{a}$), in the root node of a given decision tree. In this basic situation, the decision tree has three different split types ($\dot{a}$, $\dot{b}$, $\dot{c}$), whose outputs are independent each other ($\dot{a} \in \{a_1, a_2\}$; $\dot{b} \in \{b_1, b_2\}$; $\dot{c} \in \{c_1, c_2\}$). We can see that after the required restructuring, all paths from the root to leaf nodes remain constant (e.g., the leaf l_4 is always reached by passing through the path $\dot{a} = a_2, \dot{b} = b_2, \dot{c} = c_1$).

Procedure ReplaceSplit :

node is a tree node

$\dot{a}$ is the split test to be installed in *node*

Result: The node i splits by $\dot{a}$

```
1  begin
2      if node is a leaf then
3          Replace node with an internal node i that
             splits on ȧ
4          Mark the internal node i as split by IADEM
5          forall branches of the split do
6              Add a new leaf with initialized sufficient
                 statistics
7      else
           /* node is an internal node  */
8          if node splits by attribute ȧ then
9              Mark node as split by IADEM
10             return
11         else
12             forall child of node do
13                 ReplaceSplit (child,ȧ)
14             Let ḃ be the attribue involed in the split
                 test installed in node
15             Split node by attribute ȧ, split the children
                 of node by attribute ḃ and update
                 parent-child relationships according to
                 the new structure
16             Mark the internal node node as split by
                 IADEM
```

Algorithm 3: Algorithm that changes a split test previously installed in an internal node by splitting the necessary leaves and moving up split tests.

4. Empiriral Study

In this section, we evaluate OHyT using both synthetic and real datasets. We compare the new algorithm with respect to two baseline learners based on decision trees: VFDT and IADEM-2. We principally evaluate the stability of the induced model and the evolution of the learning over time. All the experiments were performed over MOA [10], a framework for online analysis. It provides a collection of evaluation tools, a great variety of algorithms and several methods to generate artificial data streams.

We assessed the performance of OHyT considering three important performance measures in online learning: accuracy, model's complexity (in terms of tree nodes) and processing time. We calculate these metrics online, in order to measure how the learning process evolves over time [20, 21, 10]. Specifically, we used a *test-then-train* approach in all the experiments [22, 23]. It basically consists in calculating measures as each example arrives (test step); and then, the example is made available to the method which continues with the learning (train step) [12]. We calculated metrics by means of a sliding window considering only the last performance measurements (test-then-train with a forgetting factor) [22]. It has been shown [23] that the test-then-train approach with forgetting mechanisms converges on the measurement calculated with the holdout approach.

Therefore, at each new example, the classifiers were first tested and then trained. During the learning process, accuracy was computed with respect to a sliding window of size 1000 [10, 24]. We computed the instantaneous accuracy of classifiers every 1000 examples processed by means of the fraction between the number of well-classified examples in this sliding window and the window's size. Measurements of the model's complexity were also computed every 1000 examples, whilst we measured the processing time that each algorithm took to process all examples.

We set VFDT with the default configuration they have in MOA [25, 10]. VFDT, OHyT, and IADEM-2 handle numeric attributes via the Gaussian approximation [25, 10] with a maximum number of bins fixed to 10. All algorithms predict in leaf nodes with a na ive Bayes approach. Unless otherwise stated, in OHyT, each experiment used $\tau = 2000$ and $\nu = 10\%$.

4.1 Synthetic Data

The performance of the algorithms was evaluated using common benchmark generators of data streams [10]. Table 1 summarizes the main characteristics of these generators. Thus, the data streams used in the experiments presented noise and irrel-

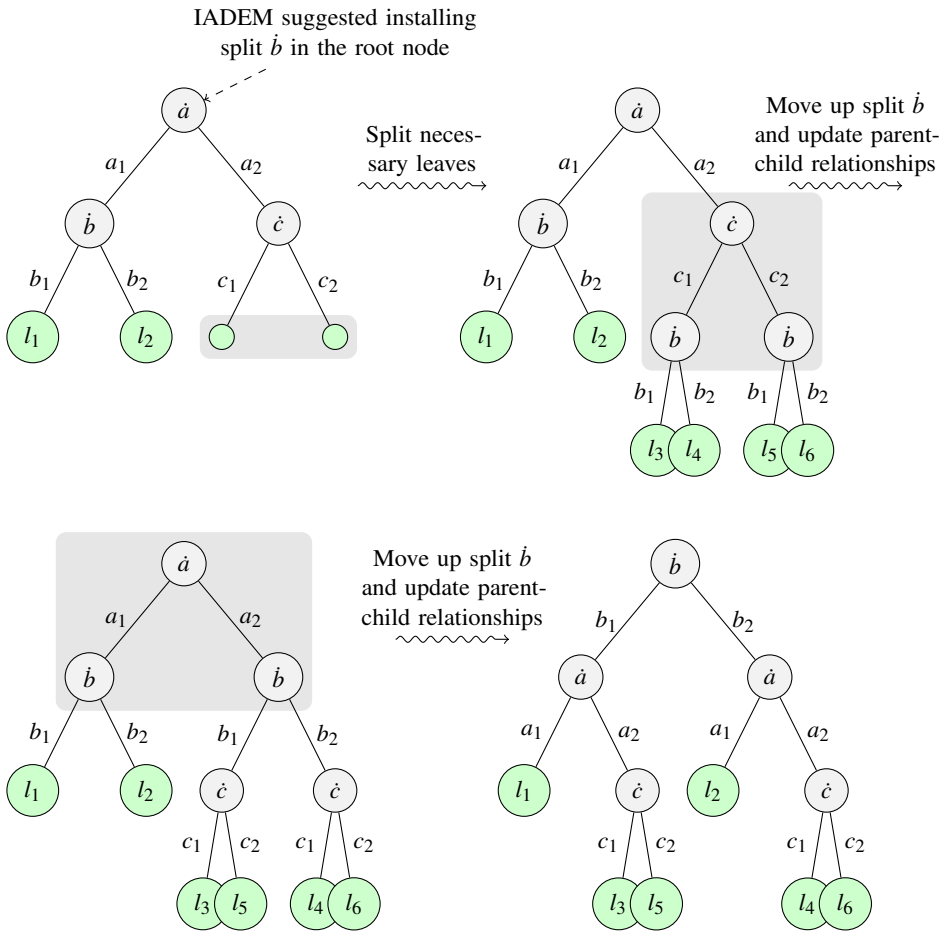

Figure 2. Algorithm 3 transforming a decision tree in order to install a different split (B) in the root node.

evant attributes, nominal and numeric attributes, and different types of target functions.

The Random Tree Generator [5] produces data streams by means of a decision tree. To construct the tree (the target function), it chooses attributes at random to split and assigns a random class label to each leaf. The training and test examples are generated by assigning uniformly distributed random values to attributes, and this attribute values determine the class label via the tree. In this experiment, two random trees were generated as target functions [25]. The first data stream was generated by a random tree having ten nominal attributes with five values each, ten numeric attributes, two classes, a tree depth of five, with leaves starting at level three and a $0,15$ chance of leaves thereafter. The second stream was generated by a random tree having 50 nominal attributes with five values each, 50 numeric attributes, two classes, a tree depth of ten, with leaves starting at level five and a $0,15$ chance of leaves thereafter. We added 10% noise to both data streams.

The Radial Base Function Generator [25] fixes a number of random centroids. Each centroid has a random position, standard deviation, class label, and weight. New examples are generated by selecting a centroid at random so that centroids

Table 1. Main characteristics of the data streams generators used in the experimental study.

Dataset	Nominal	Numeric	Classes
Simple Random Tree Generator	10	10	2
Complex Random Tree Generator	50	50	2
Simple Radial Base Functions		10	2
Complex Radial Base Functions		50	2
Simple Waveform Generator		21	3
Complex Waveform Generator		40	3
Hyperplane 1		10	2
Hyperplane 2		15	2
Hyperplane 3		20	2
Seven segments LED display (5%, 10% and 15% of noise)	24		10
STAGGER Generator (three target functions)	3		2
AGRAWAL Generator (ten target functions)	6	3	2

with higher weight are more likely to be chosen. The attribute values are randomly drawn from a Gaussian distribution with standard deviation determined by the chosen centroid. The chosen centroid also determines the class label of the example. Two data streams were generated by this type of target function: (1) 100 centroids and 10 attributes, (2) 1000 centroids and 50 attributes.

The Waveform Generator [10] differentiates between three different classes of waveform, and the data stream is generated from a combination of two or three base waves. We included two versions of the problem: the first one with 21 numeric attributes; and the second one which introduces an additional 19 irrelevant attributes and adds noise to all attributes.

A hyperplane in d-dimensional space is the set of vectors $\overrightarrow{\mathscr{A}} \in \mathbb{R}^n$ that satisfy the equation

$$\sum_{i=1}^{d} w_i a_i = y$$

where a_i is the ith component of $\overrightarrow{\mathscr{A}}$. In our setting, examples for which $\sum_{i=1}^{d} w_i a_i \geq y$ are labeled positive, and examples for which $\sum_{i=1}^{d} w_i a_i < y$ are labeled negative [10]. We generated three different data streams varying the dimension of the hyperplane: $d = 10$, $d = 15$ and $d = 20$. The generator included 5% of random noise to each class. The weights w_i are initialized randomly in the interval $(0;1]$.

In the LED Generator [10], the goal is to predict the digit displayed on a seven-segment LED display. In our experiments, we used 24 binary attributes, 17 of which are irrelevant. We generated three different data streams by varying the noise level for each attribute, introducing, respectively, 5%, 10% and 15% chances of being inverted.

STAGGER generates concept functions introduced by Schlimmer and Granger [26]. The concepts are boolean functions of three attributes encoding objects: *size* (*small*, *medium*, and *large*), *shape* (*circle*, *triangle*, and *rectangle*), and *color* (*red*, *blue*, and *green*). We generated data streams in accord with three different classification functions: $(size = small) \wedge (color = red)$, $(color = green) \vee (shape = circle)$, or $(size = medium) \vee (size = large)$ [10].

Finally, AGRAWAL generates streams according to ten different predefined functions which rules binary class labels from the attributes [27]. The generator produces streams containing nine attributes: six numeric and three categorical ones. These attributes describe hypothetical loan applications [25, 10].

Therefore, we ran algorithms over data streams generators according to Table 1 (25 artificial data streams in total). There were 1000000 training examples per data stream. In order to assess the performance of the algorithms, the accuracy and the number of nodes were calculated every 1000 examples by a test-then-train approach, as described in Section 4. The experiment was repeated 30 times under this setting. For each run, we changed the random seed of the instance generators. Tables 2 and 3 summarize the performance of the algorithms regarding predictive accuracy and number of nodes for these 25 data streams, in terms of number of wins, losses and ties.

Table 2. Algorithm's results regarding predicting accuracy. There were 1.000.000 training instances, 25 artificial dataset generators, and 30 runs for each configuration. The accuracy was computed by using the test-then-train methodology with forgetting factors.

Algorithms	Against VFDT			Against IADEM-2		
	Wins	Ties	Losses	Wins	Ties	Losses
OHyT	14	4	7	21	4	0
VFDT	-			13	3	9

Table 3. Algorithm's results regarding number of nodes. There were 1.000.000 training instances, 25 artificial dataset generators, and 30 runs for each configuration.

Algorithms	Against VFDT			Against IADEM-2		
	Wins	Ties	Losses	Wins	Ties	Losses
OHyT	13	0	12	0	0	21
VFDT	-			0	0	21

Table 2 shows that the new algorithm often reached higher levels of predictive accuracy in the artificial datasets considered. It is noteworthy that OHyT never was outperformed by IADEM-2. The Wilcoxon Signed-Rank test with the two-tailed hypothesis and with a significance level of 0.01 yields significant statistical difference. The value of z is -4.0145 and the p-value is less than 0.01. However, when performing the same Wilcoxon test, statistically significant results are not achieved between the OHyT and VFDT algorithms despite the fact that the OHyT algorithm obtains higher accuracy values over 14 of the analyzed data sets. In 7 of the cases analyzed, the VFDT algorithm obtains higher accuracy values.

On the other hand, Table 3 reports that VFDT and OHyT are approximately equivalent regarding the size of the tree induced. The experiment also showed that they induced decision trees with similar size on average. For example, VFDT induced, on average, trees with 297 nodes, whilst OHyT induced trees with 275 nodes on average.

As mentioned, IADEM-2 guarantees a statistically significant result for each split installed in decision nodes. As a result, IADEM-2 requires more training examples to make a decision on the best split and thus, it induces very small trees. Table 3 gives evidence of this fact, as it shows that IADEM-2 always induced smaller trees than OHyT and VFDT (on average, IADEM-2 induces trees with 37 nodes). However, IADEM-2 has also associated slower learning, which is negatively reflected in online predictive accuracy.

Figures 3 and 4 show the values of the predictive accuracy and number of nodes respectively averaged over the 25 datasets involved in the experiment. In these figures, we include the performance of OHyT varying the tie-breaking threshold τ. We can note that OHyT has a good performance for various of these configurations, inducing, on average, accurate decision trees with approximately the same number of

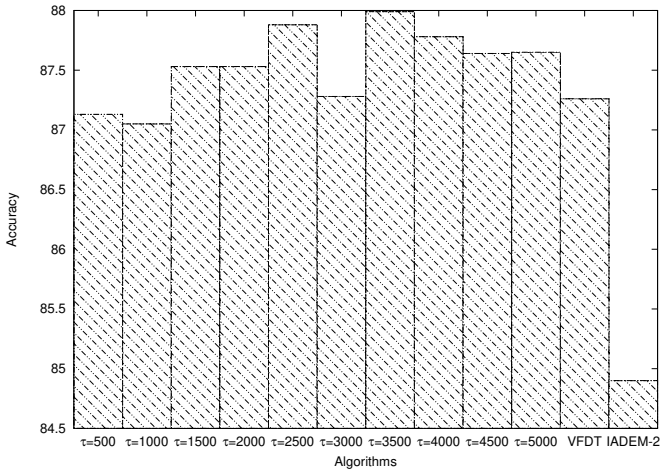

Figure 3. Algorithms' accuracy averaged over 25 synthetic datasets (in a single run), including various values of the tie-breaking threshold τ in OHyT (fixing the parameter $\nu = 0.10$).

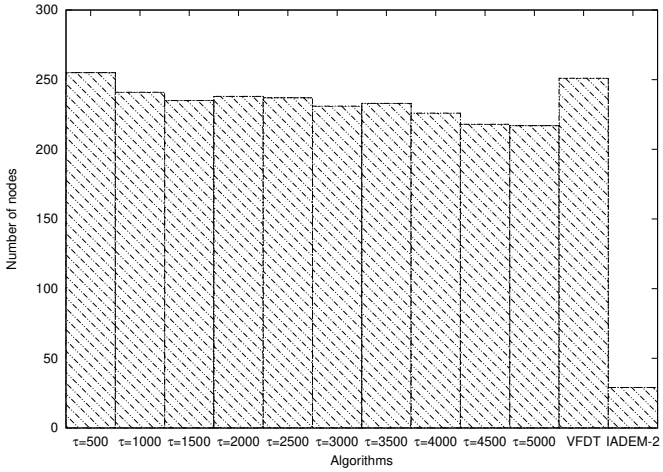

Figure 4. Tree size of the induced models averaged over 25 synthetic datasets (in a single run), including various values of the tie-breaking threshold τ in OHyT (fixing the parameter $\nu = 0.10$).

Table 4. Main characteristics of the real datasets used in the experimental study.

Dataset	Instances	Nominal	Numeric	Missing values	Classes
Elec2	45.312	1	7	yes	2
Adult	48.842	7	6	yes	2
CoverType	58.1012	44	10	no	7
KDDCup99	494.021	7	34	no	2
Nursery	12.960	8	0	no	5
Poker	1.025.010	10	0	no	2
Spam corpus 2	9.323	500	0	no	2
Usenet 1	1.500	100	0	no	2
Usenet 2	1.500	100	0	no	2

nodes.

As described in the previous session, τ is a tie-breaking threshold for splitting. This threshold controls the minimum necessary value of training examples in a node to apply the second method of division of the OHyT algorithm. Figures 3 and 4 use a range of acceptable values for this parameter τ. In this range, as shown in the experiments, the accuracy values remain stable. It is important to note that low values of the parameter τ can harm the quality of the tree because tree expansions occur more often without enough information to guarantee good splits. On the other hand, high values of this parameter (τ) would greatly reduce the use of the second method of division, which is one of the contributions of this work.

4.2 Real Data

The selected real-world datasets have been used in various studies about online learning. The final objective of designing learning algorithms is to apply then to real-world problems, so the benefit of evaluating methods with these datasets explains their presence. For these datasets, we evaluate the methods processing the training examples online in their temporal order. At each new example, the classifier is first tested and then trained. Again, we used a test-then-train approach to calculate the accuracy and number of nodes. In this experiment, we also include the processing time that each algorithm took to process all examples. We experiment with the Electricity Market dataset (Elec2)[1] and other eight datasets obtained from the UCI Machine Learning Repository:[2] Adult, Forest Cover Type (CovType), 10% of the KDD Cup 1999 data, Nursery (KDDCup99), Poker Hand (Spam), the Spam Assassin collection (Spam), and two datasets based on the 20 newsgroup collection (Usenet1 and Usenet2). Table 4 summarizes the main characteristics of these datasets.

[1] http://moa.cms.waikato.ac.nz/datasets/

[2] http://www.ics.uci.edu/mlearn/MLRepository.html

Table 5. Algorithms' result in real world datasets. The accuracy and number of nodes were computed by using the test-then-train methodology with forgetting factors.

Algorithms	Measures	Elec2	Adult	CovType	KDDCup99	Nursery	Poker	Spam	Usenet1	Usenet2
OHyT	Precision	**79,50 ± 7,00**	78,33 ± 5,39	**82,51 ± 6,39**	**99,69 ± 1,16**	**89,19 ± 6,42**	64,78 ± 3,19	**90,92 ± 6,19**	**61,77 ± 5,38**	70,70 ± 1,64
	Nodes	37,98 ± 26,04	15,73 ± 9,10	399,72 ± 251,49	488,69 ± 179,32	7,42 ± 3,24	168,78 ± 91,02	6,47 ± 2,41	1,00 0,00	1,00 0,00
	Time (sec.)	5	5	454	612	1	394	5	0	0
VFDT	Precision	76,86 ± 9,14	75,37 ± 6,05	79,69 ± 7,87	99,66 ± 1,97	88,95 ± 5,37	**71,67 ± 9,35**	90,31 ± 8,28	60,40 ± 6,46	**70,80 ± 1,69**
	Nodes	31,23 ± 12,48	30,79 ± 19,66	183,16 ± 104,98	83,30 ± 23,10	12,81 ± 5,30	113,60 ± 80,61	4,58 ± 1,79	1,00 0,00	1,00 0,00
	Time (sec.)	1	1	51	37	0	31	2	0	0
IADEM-2	Precision	76,21 ± 8,80	**82,07 ± 1,22**	73,29 ± 9,39	98,22 ± 7,29	86,25 ± 9,93	60,52 ± 5,50	90,81 ± 6,09	**61,77 ± 5,38**	70,70 ± 1,64
	Nodes	17,29 ± 9,89	2,91 ± 0,42	48,74 ± 18,38	4,73 ± 0,79	4,08 ± 1,14	27,26 ± 14,29	5,74 ± 1,74	1,00 0,00	1,00 0,00
	Time (sec.)	3	2	61	28	1	89	4	0	0

Table 5 shows the average and standard deviation of the fraction among the number of the well-classified examples and the total of the examples every 100 examples processed. The accuracy is computed with respect to a sliding window with a size 100 [23, 10].

First, taking into account the accuracy values shown in Table 5, we can verify that the results of the OHyT algorithm were visibly promising. The OHyT algorithm obtained results superior to those obtained by the VFDT and IADEM-2 algorithms in six of the nine experiments performed on real bases. However, when performing the Wilcoxon Signed-Rank test with the two-tailed hypothesis and with a significance level of 0.05 no statistically significant difference was obtained. As a second aspect, we can highlight that the tree models generated by the IADEM-2 algorithm are remarkably smaller in all experiments with real data. As mentioned above, IADEM-2 requires more examples of training in each node of the tree to make a decision about the best division. Therefore, it induces very small trees since in general, it makes fewer divisions. The size of the trees generated by the OHyT and IADEM-2 algorithms are approximately equivalent.

However, table 5 also reflects that OHyT is more time-consuming than VFDT and IADEM-2. OHyT checks the splits of internal nodes previously suggested by the tie-breaking threshold. In the presence of numeric attributes, the computational cost of revising previous splits increases significantly, because different from nominal attribute values, the number of candidate splits increases as the tree grows. Therefore, a future line of investigation is to give a more intelligent method for splitting numerical attributes without decreasing predictive accuracy significantly.

4.3 Final Considerations

In this session, two comparison scenarios were structured, the first used real data sets and the second synthetic data sets. In both scenarios, we have analyzed the behavior of the OHyT algorithm compared to two other algorithms, VFDT and IADEM-2. The three analyzed algorithms induce decision trees. For the comparative analysis, three parameters were used, accuracy, runtime, and tree size. All three algorithms behaved consistently in both scenarios. The IADEM-2 algorithm shows in the experiments that it is capable of generating very small decision trees. In addition, this algorithm offers statistical guarantees in the division decisions in each internal node of the tree. However, in order to offer these statistical guarantees, IADEM-2 may need many instances to make its decisions to divide the nodes of the tree or not, therefore its learning process is usually a bit slow. On average, their accuracy values were the least promising of the three algorithms. However, in terms of runtime, IADEM-2 was in second place just after the VFDT algorithm. IADEM-2 is a recommended algorithm for data stream analysis, especially when it has stabilized the tree model, that is, once it has analyzed enough training instances.

The VFDT algorithm obtained as a result of the experiments the best execution time of the three analyzed algorithms in both real and synthetic data sets. On the other hand, although its mean accuracy showed no significant statistical difference in relation to the OHyT algorithm, its accuracy results were numerically inferior, loosing in 14 cases against 7 over the synthetic sets and 7 against two over the real sets. The VFDT algorithm is highly recommended for scenarios where you work in real-time where the runtime is a crucial factor. It is a very fast algorithm and with suitable accuracy.

The OHyT algorithm obtained promising results in terms of accuracy values compared to the other two algorithms. In addition, it has the characteristic of faster learning, which needs a few instances to make decisions of division in the nodes of the tree. In addition, these decisions are supported by a statistical guarantee. All these characteristics can be considered positive for online learning. OHyT is a highly rec-

ommended algorithm for scenarios where the best accuracy values are needed. However, in real-time learning scenarios, when time can be a vital factor, your choice should be analyzed more deeply, due to its worst results in runtime the experiments.

5. Conclusions

In this paper, we have presented a new learning algorithm for the online induction of decision trees. The proposed algorithm, Online Hybrid Tree (OHyT), uses two different split methods for building decision trees. The objective of incorporating a second split method is to increase the convergence of the learning algorithm, namely, the usage of fewer training examples to reach similar levels of accuracy. Therefore, the first split method guarantees statistical significance for each split chosen. This method uses probabilistic bounds to estimate the best splitting attribute, and it may wait too long to make a decision on this split. Waiting too long to decide can slow down the learning of the tree, this situation being particularly harmful at the beginning of the tree induction because the decision tree model is often more inaccurate. The second method is then able to split leaf nodes more quickly, without being exhaustive in finding the best. During the tree building process, OHyT also uses the first split method to revise splits previously suggested by the second one, and an additional algorithm rebuilds the tree model according to this revision, trying to ensure an adequate statistically significant result for each split.

The computational complexity of the proposed algorithm does not depend on the number of examples processed so far. Additionally, OHyT is able to learn with a single pass over the training data. These characteristics make the new algorithm suitable for learning from possible unbounded data streams.

We have empirically compared OHyT with various previous algorithms based on decision trees, including the well-known Very Fast Decision Tree system. According to the empirical study, OHyT is more competitive in accuracy and model size, considering several synthetic and real-world datasets. We expect to continue this research by focusing on the discretization methods applied to OHyT. In the presence of numeric attributes, the computational cost of revising previous splits increases significantly, because different from nominal attribute values, the number of candidate splits increases as the tree grows. A future line of investigation is to give a more intelligent method for splitting numerical attributes without decreasing predictive accuracy significantly.

6. Acknowledgment

This work was partially funded by the Coordination for the Improvement of Higher Education Personnel – CAPES – Brazil (Finance Code 001), and the Foundation of Supporting to Research and Innovation of Santa Catarina State – FAPESC – Brazil.

Author contributions

All authors had the same level of contribution.

References

1 GAMA, J. *Knowledge Discovery from Data Streams*. [S.l.]: Chapman and Hall/CRC, 2010.

2 GAMA, J. et al. A survey on concept drift adaptation. *ACM Computing Surveys*, v. 46, n. 4, 2014.

3 MURTHY, S. Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery*, Hingham, MA, USA, v. 2, n. 4, p. 345–389, dez. 1998.

4 RAMOS, G.; MORALES, R. A new method for induction decision trees by sampling. In: *Neurocolt Workshop on Applications of Learning Theory*. Barcelona, Spain: [s.n.], 2000.

5 DOMINGOS, P.; HULTEN, G. Mining high-speed data streams. In: *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2000. (KDD '00), p. 71–80.

6 RUTKOWSKI, L. et al. Decision trees for mining data streams based on the McDiarmid's bound. *IEEE Transactions on Knowledge and Data Engineering*, v. 25, n. 6, p. 1272–1279, 2013.

7 ROSA, G.; CESA-BIANCHI, N. Splitting with confidence in decision trees with application to stream mining. *International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2015.

8 del Campo, J. et al. Improving the performance of an incremental algorithm driven by error margins. *Intelligent Data Analysis*, v. 12, n. 3, p. 305–318, 2008.

9 RUTKOWSKI, L. et al. Decision trees for mining data streams based on the gaussian approximation. *IEEE Transactions on Knowledge and Data Engineering*, v. 26, n. 1, p. 108–119, 2014.

10 BIFET, A. et al. MOA: Massive Online Analysis. *Journal of Machine Learning Research*, v. 11, p. 1601–1604, 2010.

11 DOMINGOS, P.; PAZZANI, M. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, v. 29, n. 2/3, p. 103–130, 1997.

12 HULTEN, G.; SPENCER, L.; DOMINGOS, P. Mining time-changing data streams. In: *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2001. p. 97–106.

13 FERNáNDEZ, M. et al. Do we need hundreds of classifiers to solve real world classification problems? *J. Mach. Learn. Res.*, v. 15, n. 1, p. 3133–3181, 2014.

14 RUTKOWSKI, L. et al. A new method for data stream mining based on the misclassification error. *IEEE Transactions on Neural Networks and Learning Systems*, v. 26, n. 5, p. 1048–1059, 2015.

15 JIN, R.; AGRAWAL, G. Efficient decision tree construction on streaming data. In: *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2003. (KDD '03), p. 571–576.

16 RAMOS, G.; del Campo, J.; MORALES, R. Incremental algorithm driven by error margins. *Lecture Notes in Artificial Intelligence*, v. 4265, p. 358–362, 2006.

17 NÚÑEZ, M.; FIDALGO, R.; MORALES, R. Learning in environments with unknown dynamics: Towards more robust concept learners. *Journal of Machine Learning Research*, v. 8, p. 2595–2628, 2007.

18 del Campo, J. et al. Improving prediction accuracy of an incremental algorithm driven by error margins. In: *Proceedings of the 4th International Workshop on Knowledge Discovery from Data Streams*. [S.l.: s.n.], 2006. (IWKDDS'06), p. 57–66.

19 Núñez, M.; FIDALGO, R.; MORALES, R. On-Line learning of decision trees in problems with unknown dynamics. In: *Proc. 4th Mexican Int. Conf. on Advances in Artificial Intelligence*. [S.l.]: Springer-Verlag, 2005. p. 443–453.

20 GAMA, J.; CASTILLO, G. Learning with Local Drift Detection. In: *Proceedings of the 2nd International Conference on Advanced Data Mining and Applications*. [S.l.: s.n.], 2006. p. 42–55.

21 BIFET, A. *Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*. [S.l.]: IOS Press, 2010. v. 207. 1-212 p.

22 GAMA, J.; RODRIGUES, P.; CASTILLO, G. Evaluating algorithms that learn from data streams. In: *Proceedings of the 2009 ACM Symposium on Applied Computing*. New York, NY, USA: ACM, 2009. (SAC '09), p. 1496–1500.

23 GAMA, J.; SEBASTIÃO, R.; RODRIGUES, P. Issues in evaluation of stream learning algorithms. In: *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. New York, NY, USA: ACM, 2009. (KDD '09), p. 329–338.

24 GAMA, J.; SEBASTIÃO, R.; RODRIGUES, P. On evaluating stream learning algorithms. *Machine Learning*, v. 90, n. 3, p. 317–346, 2013.

25 KIRKBY, R. *Improving Hoeffding Trees*. Tese (Doutorado) — University of Waikato, 2007.

26 SCHLIMMER, J.; GRANGER, R. Incremental learning from noisy data. *Machine Learning*, v. 1, n. 3, p. 317–354, 1986.

27 AGRAWAL, R.; IMIELINSKI, T.; SWAMI, A. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, v. 5, n. 6, p. 914–925, 1993.