Just Change on Change: Adaptive Splitting Time for Decision Trees in Data Stream Classification

Daniel Nowak Assis
Programa de Pós-Graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná
Curitiba, Paraná, Brazil
daniel.nassis@ppgia.pucpr.br

Jean Paul Barddal
Programa de Pós-Graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná
Curitiba, Paraná, Brazil
jean.barddal@ppgia.pucpr.br

Fabrício Enembreck
Programa de Pós-Graduação em Informática (PPGIa)
Pontifícia Universidade Católica do Paraná
Curitiba, Paraná, Brazil
fabricio@ppgia.pucpr.br

ABSTRACT

Hoeffding Trees are well-established decision trees for classifying streaming data. The Hoeffding bound was widely used in a static periodic manner, applying the bound for impurity measures to determine whether leaf nodes should split. However, this approach does not account for the tree state and its leaf nodes over time. We hypothesize that splitting when data distribution and accuracy changes occur in leaf nodes enhances decision tree performance. This paper introduces the use of change detection algorithms that dictate the moment a split will happen. First, in the local approach, each leaf node has a change detector that monitors either the error rate or purity of a leaf node and a global one, where a detector monitors statistics from the leaf nodes where the instances arrive. Results show that our methods had competitive results while being more efficient regarding processing time than state-of-the-art Hoeffding-based Trees since the periodic and constant evaluation of splits is costly.

CCS CONCEPTS

• **Computing methodologies → Online learning settings**; • **Classification and regression trees**; • **Machine Learning algorithms**;

KEYWORDS

Decision Trees, Stream mining, Split conditions

ACM Reference Format:
Daniel Nowak Assis, Jean Paul Barddal, and Fabrício Enembreck. 2024. Just Change on Change: Adaptive Splitting Time for Decision Trees in Data Stream Classification. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Decision Trees are successful methods for mining tabular data due to their simplicity, interpretability, and prediction quality. Since the work of [11], they are feasible and efficient algorithms for mining potentially infinite arriving streaming data. Data stream mining algorithms must face challenges that broaden memory and time processing issues since storing all arriving instances can collapse the system due to the lack of memory. Instances can arrive hastily, and the algorithms must process an instance as fast as the arrival of another instance; otherwise, the machine learning model will not be able to predict new arriving data and will be outdated [16].

Conference'17, July 2017, Washington, DC, USA
2024. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Another intrinsic problem of streaming data is concept drift [20]. Concept drift is known as changes throughout time in the probability properties of data. Concept drifts can decrease the accuracy of machine learning models, which must swiftly detect and adapt to those situations to not jeopardize the entire learning process.

The Hoeffding Tree [11] algorithm adapts standard decision trees [7, 22] to be incremental and learn from new incoming instances. A user-given parameter (known as Grace Period) determines the frequency at which a leaf node will attempt to split. If the number of observations at a leaf node is divisible by the Grace Period, a split attempt based on the Hoeffding bound will occur. This distributes the cost of verifying the best splits at leaf nodes along the stream.

Although Hoeffding Trees are the leading choice as base learners of ensembles [14] and are widely used in streaming problems, the periodic evaluation of splits performed by Hoeffding trees are static, in the sense that they do not consider the state of the tree throughout time. This paper proposes two new decision tree algorithms: Local and Global Adaptive Streaming Trees (LAST and GAST, respectively). These algorithms assume that increases in error rate and impurity of leaf nodes are ideal moments for performing a split. These statistics are constantly evaluated by a change detector algorithm, implying that our approach is adaptive and considers the evolution of the state of the leaf nodes and the stream.

LAST and GAST integrate change detection algorithms into the incremental decision tree. Results show that our approach can be more accurate and efficient than Hoeffding Tree variants in time processing and memory usage.

The contributions of this work are summarized below:

- The proposition of two new decision tree algorithms for stream mining.
- A comparison of decision tree-based models for data stream mining.
- An empirical demonstration that our proposition can be more accurate than Hoeffding Tree variants while demanding less processing time.

This paper is divided as follows. Section 2 formalizes data stream classification and concept drift problems. Section 3 discusses the Hoeffding Tree algorithm and variants proposed in the literature. Section 4 introduces our methods. Section 5 reports the experimental results obtained. Finally, Section 6 concludes this paper and states future works.

2 DATA STREAM MINING

The ginormous amount of data generated in different contexts, like social media, sensors, and smart cities, gathered researchers' attention to develop data stream mining algorithms. Classification is the task of predicting a discrete class given a vector of nominal, numerical, or mixed data. Data stream classification can be formalized as follows: given continuously arriving instances i of the form $(\vec{x}_n, y_n)$ where y_n is a discrete class label and $\vec{x}_n$ is a d-dimensional vector of attributes, such that potentially $n \rightarrow \infty$, the goal is to create a predictive model that better outputs y_n given $\vec{x}_n$ as an input $(f : \vec{x} \rightarrow y)$.

Another fact that gathered the attention of researchers is that an assumption of data stationary may not hold in streaming scenarios, i.e., concept drifts might happen over time. A concept drift is said to occur if in two timestamps t and $t + \Delta$ in a stream S, while $\Delta > 1$ and $P_t(X,Y) \neq P_{t+\Delta}(X,Y)$. Since $P_n(X,Y) = P_n(X)P_n(Y|X)$, two types of drifts are noticeable. Changes on $P_n(Y|X)$ mean that the decision boundaries changed (known as real drifts) and are known to affect the accuracy of the models, and changes in $P_n(X)$ are changes in the distribution of attribute values (known as virtual drifts). We reference the reader to the following work for more details about concept drift [20].

3 HOEFFDING TREES AND VARIANTS

This section presents existing Hoeffding Tree-based algorithms.

3.1 Hoeffding Tree

Domingos and Hulten proposed the Hoeffding Tree (HT) algorithm and VFDT (Very Fast Decision Tree) system in [11].

Algorithm 1 presents the VFDT algorithm. The two main components of the VFDT system are the Hoeffding bound constraint to prevent overgrowth and the periodic evaluation of splits in leaf nodes. The Grace Period (*GP*) is a user-given parameter that sets the frequency a leaf node will attempt to split. Given n_l the number of samples seen at leaf l, if $n_l \bmod GP = 0$, a split attempt will ensue (line 8, Algorithm 1).

The Hoeffding bound [17] is a theorem that presents a probability inequality for the difference between the mean $(\overline{X})$ and expected value $(\mathbb{E}[X])$ of a set of random variables. With a level of confidence δ, one can derive that $\overline{X} - \mathbb{E}[X] \geq \epsilon$, where

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

and R is the range of the random variables, and n is the number of random variables. The Hoeffding Tree applies this bound for impurity measures to determine if X_a, the attribute with the highest impurity measure ΔG (such as gini index [7] or information gain[22]), is the ideal attribute to split on a split attempt. Given X_b, the attribute with the second-highest impurity measure, if $\Delta G(X_a) - \Delta G(X_b) \geq \epsilon$, a split will occur on X_a (line 13, Algorithm 1).

The $\lim_{n \to +\infty} \epsilon = 0$, and if in a node $\Delta G(X_a)$ and $\Delta G(X_b)$ have similar values, a split will take many observations to split. To relax the Hoeffding constraint in these situations, the Hoeffding Tree has a tie threshold. Given τ (user-given threshold), a split will ensue when $\tau > \epsilon$.

If d is the number of attributes, v is the maximum number of values per attribute, c is the number of classes, and l is the number of leaf nodes, the Hoeffding tree algorithm requires $O(ldvc)$ memory to store the necessary counts.

Algorithm 1 *VFDT*, adapted from [11]

```
Input: $S$: a data stream,
    $X$: a set of attributes,
    $\Delta G(\cdot)$ : a split evaluation function,
    $\delta$ : one minus the desired probability of choosing the correct attribute at any given node,
    $GP$: Grace Period (the frequency a leaf node will attempt to split)
    $\tau$ : Tie threshold
Output: $DT$: a decision tree
 1: Let $DT$ be a tree with a single leaf $l_1$ (the root)
 2: Let $n_{l_1} \leftarrow 0$                ▷ number of observations in the leaf
 3: for each sample $(\vec{x}, y) \in S$ do
 4:     Sort $\vec{x}$ into a leaf $l$ using $DT$
 5:     Classify $\vec{x}$ with the majority class in $l$
 6:     Update $l$ statistics using $(\vec{x}, y)$
 7:     $n_l \leftarrow n_l + 1$
 8:     if $n_l \bmod GP = 0 \wedge \neg(l$ contains samples from only one class) then
 9:         Compute $\Delta G(X_i)$ for each $X_i \in X_l$ stored in $l$
10:         Let $X_a$ be the attribute with highest $\Delta G$
11:         Let $X_b$ be the attribute with second highest $\Delta G$
12:         Let $\epsilon \leftarrow \sqrt{\frac{R^2 \log(\frac{1}{\delta})}{2n}}$                ▷ Hoeffding bound
13:         if $(\Delta G(X_a) - \Delta G(X_b) > \epsilon \vee \tau > \epsilon) \wedge X_a \neq \emptyset$ then
14:             Replace $l$ by leaf nodes that split on $X_a$
15:             for each leaf node $l_i$ from splitting on $X_a$ do
16:                 Let $n_{l_i} \leftarrow 0$
17:             end for
18:         end if
19:     end if
20: end for
21: return $DT$
```

3.2 Additions to the Hoeffding Tree algorithm

An aspect that established Hoeffding Trees as accurate decision trees was the addition of Naive Bayes at leaf nodes [13]. It was observed that Naive Bayes could outperform decision trees in initial streaming scenarios where the decision tree has trained with little data. In [19], the authors propose the selection of Naive Bayes or majority class strategy prediction according to the method with the highest accuracy at the leaf.

Another critical aspect of Hoeffing Trees is memory management since storing all instances from a data stream is unfeasible. Leaf nodes maintain statistics from data, such as histograms for nominal data, and rely on forgetting mechanisms. We refer the interested reader to [19] for more details.

3.3 CVFDT

The Concept-adapting Very Fast Decision Tree (CVFDT) [18] adapts the Hoeffding Tree to concept drifts implicitly (the model's internal mechanisms adapt to concept drift). The CVFDT gives higher weights for the most recent instances. The main idea of the algorithm is to maintain a sliding window of data and ensure that the tree statistics (like splits that happened with the Hoeffding bound) are still consistent and valid.

When more instances arrive, there is the possibility that the Hoeffding bound is not valid for specific nodes. Periodically, the algorithm checks for new attributes to split on nodes that the Hoeffding inequality is not valid anymore and creates subtrees. If the subtree's accuracy surpasses the existing subtree's, the constructed background subtree replaces the existing one.

3.4 Hoeffding Adaptive Tree

The main problem of CFVDT is to estimate the correct sliding window size for each problem. The authors in [5] propose the Hoeffding Adaptive Tree (HAT) to overcome this issue. For each node of the tree, an ADaptive WINdowing (ADWIN) [4] algorithm monitors the accuracy of the branches of the node. If the algorithm triggers a change, a new subtree is created and replaces the node and its branch if the subtree is more accurate.

ADWIN is one of the most famous and used algorithms for change detection and is widely used as an explicit drift detector (monitors concept drift given the output of a model, which resets in case of triggering a concept drift). ADWIN creates a window W and checks if all the subwindows W_1 and W_2 in W have similar mean according to the Hoeffding bound. If so, W_1 is discarded and $W = W_2$. Due to the cost of verifying each subwindow of W can have a high computational cost, the authors propose an efficient version of ADWIN that uses exponential histograms [9].

Since ADWIN memory complexity is $O(\log W)$, HAT complexity is $O(ndvc \times \log W)$ as n is the number of nodes.

3.5 EFDT

In [21] the authors propose the Extremely Fast Decision Tree (EFDT) algorithm. In the split attempt process, instead of comparing $\Delta G(X_a) - \Delta G(X_b) \geq \epsilon$, the authors propose comparing $\Delta G(X_a)$ with the occasion of any split occurrence, or $\Delta G(X_a) - \Delta G(X_\emptyset) \geq \epsilon \equiv \Delta G(X_a) \geq \epsilon$. In situations where $\Delta G(X_a)$ is high and could enhance the tree's predictive performance, EFDT is not affected when $\Delta G(X_b)$ has a similar value to $\Delta G(X_a)$.

Like CVFDT and HAT, EFDT also has a re-evaluation mechanism of previous splits. However, the EFDT re-evaluation process is similar to the Hoeffding Tree split attempt process. The user defines the frequency intermediate nodes attempt a re-evaluation of splits. Given $\Delta G(X_n)$, the split quality at the time node n split, n and its branch is replaced by a new split on a newly evaluated X_a if $\Delta G(X_a) - \Delta G(X_n) \geq \epsilon$. In other words, it is a comparison between a new split and the previous one done at the terminal node.

Since EFDT must store data at intermediate nodes to perform re-evaluation of splits, it requires $O(ndvc)$ memory, as n is the number of nodes.

4 LOCAL AND GLOBAL ADAPTIVE STREAMING TREES

Hoeffding-based Trees, per se, have a static and periodic evaluation of splits. Even when an evident change occurs in the purity of a leaf node, a leaf node will possibly split if the number of observations at the leaf node reaches a number divisible by the Grace Period. And even when little change ensued, the greedy evaluation of attributes and their values that compose the best split will still be performed.

For that reason, we propose new trees that constantly monitor the statistics of leaf nodes to determine ideal moments for splitting. Figure 1 shows how adaptability can be well suited in incremental decision trees.

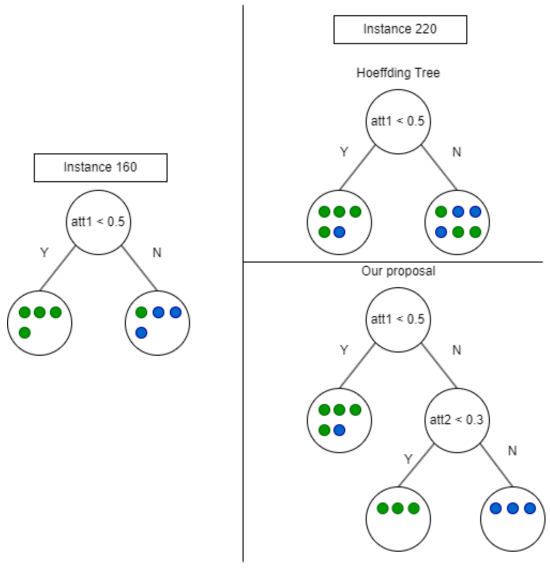

Figure 1: Illustration of the adaptability of our method in comparison to Hoeffding Trees with $GP = 160$**.**

Detecting increases in real-valued variables (such as error rates and impurity measures) are well known in streaming scenarios, and many algorithms were proposed in the literature for this task [20]. Therefore, we integrate change detection algorithms into incremental decision trees. We propose two forms of integrating change detection algorithms into decision trees, and they are explained as follows.

Local Adaptive Streaming Tree (LAST): Algorithm 2 presents the LAST algorithm. LAST creates and maintains change detectors only at leaf nodes (lines 3 and 21, Algorithm 2). At the leaf, change detectors can monitor either the impurity (line 10, Algorithm 2) or error rate (line 12, Algorithm 2). If a change detector at the leaf triggers a change, then a split will ensue if $\Delta G(X_a) > 0.0$ (line 17, Algorithm 2).

Monitoring the impurity of a leaf node favors the majority class strategy, and monitoring error rates consider Naive Bayes's performance, i.e., if its performance is superior predictions based on the majority class.

Change detectors are constantly updated with arriving instances, meaning that these algorithms track how the stream evolves.

Global Adaptive Streaming Tree (GAST): Algorithm 3 presents the GAST algorithm. In GAST, only a single drift detector monitors the impurity of all the leaf nodes. The drift detector receives information from more than one leaf node, having a general overlook of the tree. But at the same time, when the drift detector triggers that a change occurred, the probability that the leaf node will be the appropriate one for splitting equals $\frac{1}{|L|}$ as $|L|$ is the number of leaf nodes in the tree.

GAST does not monitor error rate because this is exactly what explicit drift detectors do. If GAST monitored the error rate and had an explicit drift detector, the tree would always have just a root node, since the time for splitting and resetting would be the same.

Algorithm 2 LAST

```
Input: $S$: a data stream,
    $X$: a set of attributes,
    $\Delta G(\cdot)$ : a split evaluation function,
    $H(\cdot)$ : impurity measure analogous to $\Delta G(\cdot)$
    $\psi$ : A change detection algorithm
    $useH$: boolean variable. If true, $\psi$ has input $H(\cdot)$, else $\psi$ input
    is 1 for misclassifications, otherwise 0
Output: $DT$: a decision tree
1: Let $DT$ be a tree with a single leaf $l_1$ (the root)
2: Let $n_{l_1} \leftarrow 0$    ▷ number of observations in the leaf
3: $l_{1\psi} \leftarrow \psi$    ▷ create a change detector at leaf
4: for each sample $(\vec{x}, y) \in S$ do
5:     Sort $\vec{x}$ into a leaf $l$ using $DT$
6:     Classify $\vec{x}$ with the majority class in $l$
7:     Update $l$ statistics using $(\vec{x}, y)$
8:     $n_l \leftarrow n_l + 1$
9:     if $useH$ then
10:        Update $l_\psi$ with $H(l)$
11:    else
12:        Update $l_\psi$ with $\mathbb{1}\{DT(\vec{x}) \neq y\}$
13:    end if
14:    if $l_\psi$ detected change $\wedge \neg$ ($l$ contains samples from only
       one class) then
15:        Compute $\Delta G(X_i)$ for each $X_i \in X_l$ stored in $l$
16:        Let $X_a$ be the attribute with highest $\Delta G$
17:        if $(\Delta G(X_a) > 0.0) \wedge X_a \neq \emptyset$ then
18:            Replace $l$ by leaf nodes that split on $X_a$
19:            for each leaf node $l_i$ from splitting on $X_a$ do
20:                Let $n_{l_i} \leftarrow 0$
21:                $l_{i\psi} \leftarrow \psi$ ▷ create a change detector at each leaf
22:            end for
23:        end if
24:    end if
25: end for
26: return $DT$
```

Both LAST and GAST apply the softest split constraint ($\Delta G(X_a) > 0.0$), meaning that the change detectors control how the tree grows.

LAST requires $O(\psi ldvc)$ memory, as ψ is the memory complexity of the change detection algorithm applied, and GAST requires $O(\psi + ldvc)$.

Algorithm 3 GAST

```
Input: $S$: a data stream,
    $X$: a set of attributes,
    $\Delta G(\cdot)$ : a split evaluation function,
    $H(\cdot)$ : impurity measure analogous to $\Delta G(\cdot)$
    $\psi$ : A change detection algorithm
Output: $DT$: a decision tree
1: Let $DT$ be a tree with a single leaf $l_1$ (the root)
2: Let $n_{l_1} \leftarrow 0$    ▷ number of observations in the leaf
3: for each sample $(\vec{x}, y) \in S$ do
4:     Sort $\vec{x}$ into a leaf $l$ using $DT$
5:     Classify $\vec{x}$ with the majority class in $l$
6:     Update $l$ statistics using $(\vec{x}, y)$
7:     $n_l \leftarrow n_l + 1$
8:     Update $\psi$ with $H(l)$
9:     if $\psi$ detected change $\wedge \neg$ ($l$ contains samples from only
       one class) then
10:        Compute $\Delta G(X_i)$ for each $X_i \in X_l$ stored in $l$
11:        Let $X_a$ be the attribute with highest $\Delta G$
12:        if $(\Delta G(X_a) > 0.0) \wedge X_a \neq \emptyset$ then
13:            Replace $l$ by leaf nodes that split on $X_a$
14:            for each leaf node $l_i$ from splitting on $X_a$ do
15:                Let $n_{l_i} \leftarrow 0$
16:            end for
17:        end if
18:    end if
19: end for
20: return $DT$
```

Table 1: Decision Trees accuracy (%).

Dataset	HT	EFDT	HAT	LAST	GAST	LAST$_H$
LED$_a$	69.03	69.87	73.73	**73.93**	70.81	68.57
LED$_g$	68.65	69.72	**72.60**	71.49	70.11	67.84
SEA$_a$	86.42	86.41	**88.81**	86.61	84.44	86.59
SEA$_g$	86.42	86.37	**88.51**	86.38	84.45	86.51
AGR$_a$	77.83	82.87	**88.20**	83.94	79.52	79.99
AGR$_g$	75.63	80.09	**86.70**	80.58	77.11	75.76
RBF$_m$	45.49	51.27	61.75	**64.11**	48.93	62.75
RBF$_f$	32.29	31.87	**39.16**	36.70	32.05	32.95
HYPER	78.77	81.59	**86.69**	79.41	70.91	70.91
Outdoor	57.33	59.58	57.27	60.40	**60.68**	59.92
Elec	79.20	80.64	83.39	**81.78**	79.90	78.84
Rialto	31.35	**57.74**	30.62	56.33	21.57	56.07
Airlines	65.08	65.27	63.81	**65.52**	65.29	65.15
CovType	80.31	84.67	81.89	**87.52**	83.98	85.73
Nomao	92.23	93.93	93.78	**94.68**	94.32	91.99
Poker	76.07	76.60	66.87	76.40	77.27	**80.13**
NOAA	73.43	73.23	73.53	**73.92**	72.97	72.23
Overall avg. Rank	4.76	3.53	2.76	**1.94**	4.0	3.94
Real avg. Rank	4.75	3.0	4.37	**1.75**	3.25	3.87
Synth. avg. Rank	4.78	4.0	**1.33**	2.11	4.67	4.0

Bold values indicate the best results per data set

5 EXPERIMENTS AND RESULTS

In this section, we introduce the experimental protocol adopted, followed by the results obtained and discussion.

5.1 Experimental Protocol

We assess the predictive performance with the accuracy obtained in a test-then-train validation strategy, where every instance is used first for testing and then for training, known as prequential evaluation [12]. We also evaluate the methods for time processing and memory usage based on CPU time and RAM (GB) per hour used.

Experiments were conducted using the Massive Online Analysis (MOA) framework[6], and the source code for our proposal is also made publicly available[1]. All experiments were done in a Intel(R) Xeon(R) CPU E5649 @ 2.53GHz with 32 GB of RAM. All the trees evaluated had default parameters from MOA (Grace Period = 200, level of confidence = 10^{-7}, and impurity measure is information gain). For LAST and GAST, we decided to run experiments with the ADWIN [4] change detector with default parameters. An advantage of ADWIN is that it accepts real values as input (as in [15]), and most of the change detectors in the literature accept only binary inputs.

We refer to the LAST version that monitors the impurity as LAST_H and the one that monitors the error rate simply as LAST.

Experiments were held with 17 datasets, eight real-world and nine synthetic datasets. The synthetic datasets and parameters used are discussed as follows.

LED [7]. This generator produces 24 boolean features, while 17 of them are irrelevant. Each feature has a 10% of being inverted, simulating noise. We simulated three abrupt (LED_a) and gradual (LED_g) drifts.

SEA [23]. This generator produces 3 numerical features (f_1, f_2, f_3). If $f_1 + f_2 \leq \theta$, the class has value 1, otherwise 0. In this dataset, we simulated three abrupt (SEA_a) and gradual (SEA_g) drifts by changing the values of θ.

AGRAWAL [2]. This generator has six nominal features and three numerical features. Ten distinct functions map two classes. In this dataset, we simulate three abrupt (AGR_a) and gradual(AGR_g) datasets.

RBF. This generator produces ten features and 5 class values. Data is generated based on the radial basis function (RBF). Centroids are generated in random positions and mapped with a standard deviation value, a weight, and a class label. In this dataset, incremental drifts are simulated by changing the centroids' position at a continuous rate. The parameters used were 50 centroids at a speed change of 10^{-4} (moderate, RBF_m) and 10^{-3} (fast, RBF_f).

HYPER [18]. A hyperplane is a flat, $(n-1)$ dimensional subset of that space that divides it into two disconnected parts. Drifts can be simulated incrementally by changing the decision boundary implied. HYPER was parametrized with 10 features and a magnitude of change of 10^{-3}.

[1]https://sites.google.com/view/just-change-on-change

Table 2: Decision trees size (# of nodes).

Dataset	HT	EFDT	HAT	LAST	GAST	LAST_H
LED_a	229	278	37	**15**	77	897
LED_g	221	299	78	**29**	95	873
SEA_a	753	357	541	7	**5**	95
SEA_g	743	350	1049	11	**5**	91
AGR_a	1351	966	325	**53**	181	385
AGR_g	1422	623	1234	187	**170**	924
RBF_m	219	1112	**46**	3277	357	4267
RBF_f	139	170	**82**	1455	199	507
HYPER	1087	795	618	225	**1**	**1**
Outdoor	**1**	17	**1**	11	33	41
Elec	**57**	103	97	103	99	61
Rialto	9	164	9	185	**1**	197
Airlines	8582	15146	91376	8873	**6008**	12461
CovType	339	893	**172**	941	1213	1367
Nomao	38	31	**10**	36	106	39
Poker	301	543	**3**	683	1475	921
NOAA	13	15	**0**	7	69	7
Overall avg. Rank	3.35	3.82	**2.59**	2.88	3.0	4.24
Real avg. Rank	2.12	3.5	**2.0**	3.38	4.0	4.25
Synth. avg. Rank	4.44	4.11	3.11	2.44	**2.11**	4.22

Bold values indicate the best results per data set. The smaller the tree size, the higher the ranking

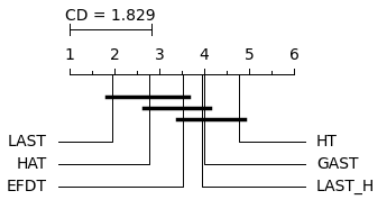

Figure 2: Nemenyi test on accuracy

The real-world datasets used were Outdoor, Elec, Rialto, Airlines, CovType, Nomao, Poker, and NOAA. More details on the used datasets can be found in the auxiliary repository given above.

Tables 1 and 2 show the accuracy and size (number of nodes) of Decision Trees, respectively. In the case of the tree size, the lower the tree size, the better the ranking. The CPU-Time results obtained for real and synthetic data are displayed in figures 3 and 4. Figures 5 and 6 show the RAM Hours in real-world and synthetic data, respectively. The algorithms are sorted by median values in ascending order.

Figure 2 shows a Nemenyi test with 95% confidence in the decision trees accuracy. The Nemenyi test compares the average ranking of algorithms [10]. If two algorithms have an absolute difference of mean ranking greater than the critical distance (CD), they can be considered statistically different.

5.2 Discussion

The results obtained show competitive results with state-of-the-art decision trees. Some interesting points can be highlighted by comparing the decision tree accuracy and cost, which depict how our methods can be well suited for mining streaming data.

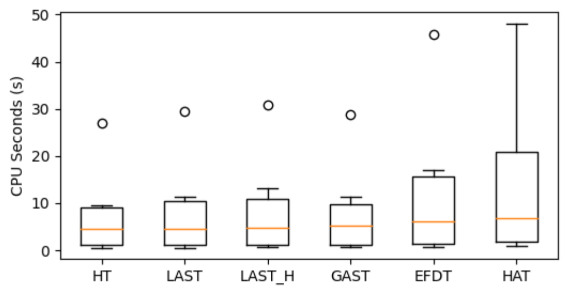

Figure 3: CPU Time (s) for trees in real-world datasets

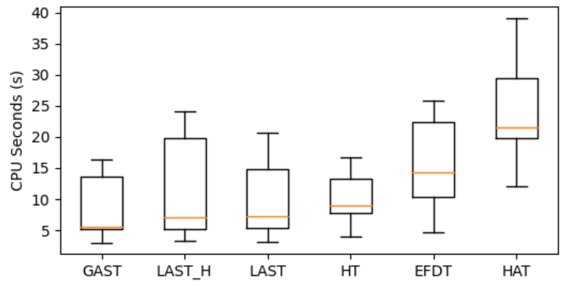

Figure 4: CPU Time (s) for trees in synthetic datasets

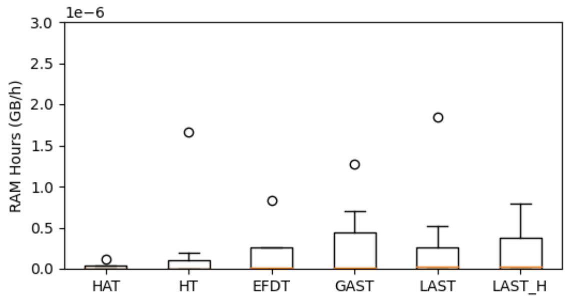

Figure 5: RAM-Hours (GB/Hour) for trees in real-world datasets

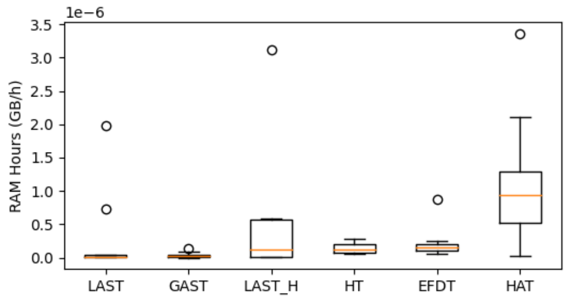

Figure 6: RAM-Hours (GB/Hour) for trees in synthetic datasets

LAST monitoring error rates had the best-reported results regarding average accuracy ranking. LAST outperformed HAT in real-world datasets and had the second-best results regarding synthetic datasets while presenting median and upper quartile CPU-Time lower than EFDT in real and synthetic data and similar tree size ranking. LAST$_H$ and GAST could outperform Hoeffding Tree but could not outperform EFDT. One can notice in the Nemenyi test (Fig. 2) that LAST was statistically different from GAST, while HAT was not.

In synthetic datasets, it is evident that if a dataset causes a high accuracy decrease in the leaf nodes, LAST will probably overgrow due to its soft split condition, such as in the RBF dataset. But LAST could also show that in datasets with a low number of changes, LAST could outperform state-of-the-art trees while significantly having a smaller size, such as in the AGR$_g$ dataset that the Hoeffding Tree had 1298 more nodes than LAST.

In real datasets, this is not clear since drift types in real-world datasets are not known. However, it is evident that LAST could outperform EFDT in 6 out of 8 real-world datasets while having similar size regarding average ranking.

Even with similar size in real-world datasets, EFDT presented median (6.14 seconds) and upper quartile (22.43 seconds) CPU-Time greater than LAST (4.56 and 10.29 seconds, respectively). LAST median and upper quartile processing time is similar to HT (4.52 and 9.31 seconds, respectively), suggesting that constantly evaluating statistics at nodes and search for the best splits only when the detector triggers a change has a similar cost as periodically and constantly searching for best splits with a lower size tree. In synthetic datasets, LAST and GAST presented lower median CPU Time.

One advantage of our algorithms is that the user does not need to specify the Grace Period and τ threshold values, since the optimal value of these parameters depends on each scenario. Change detectors can also have parameters, but some are well-established regarding change detection, and some change detectors even have no parameters at all.

HAT presented the lowest number of nodes in real-world datasets due to the reevaluation of the splits strategy applied but had CPU-Time upper quartile (20.79 seconds) greater than EFDT CPU-Time upper quartile (15.57 seconds). In synthetic datasets, HAT median CPU-Time (21.39 seconds) is similar to EFDT upper quartile (22.43 seconds). These results depict how LAST is suitable for mining streaming data by presenting higher ranking compared to HAT and being more efficient compared to EFDT in CPU-Time.

LAST presented similar upper quartile (2.53×10^{-7}) RAM Hours usage as EFDT (2.59×10^{-7}) in real-world datasets but presented higher median processing time (2.23×10^{-8} in comparison to EFDT 1.02×10^{-8} Ram Hours usage). Due to low tree size, LAST presented lower RAM-Hours than HT in synthetic datasets.

LAST$_H$ and GAST presented similar results regarding CPU-Time and RAM-Hours to LAST (except in synthetic datasets where LAST$_H$ presented superior CPU-Time upper quartile) but could not outperform EFDT in accuracy. By receiving as input the impurity of the leaf nodes, LAST$_H$ and GAST do not take into account the performance of the Naive Bayes in the leaf nodes, which is a key component of streaming decision trees efficiency [13].

6 CONCLUSIONS

In this work, we proposed LAST and GAST, where change detectors control how the tree grows. LAST and GAST dictate splitting time when a change in error rate or impurity of leaf nodes ensues. At the same time, Hoeffding Trees do periodic evaluations, not considering how the tree and the stream evolve. LAST and GAST presented competitive results against state-of-the-art decision trees while requiring less CPU Time.

In future works, we plan to apply stronger split constraints to LAST and GAST to avoid unnecessary splits, known as regularization techniques [3]. We also plan to apply the change detection strategy in EFDT and HAT leaf nodes. We also plan to explore how our method performs with state-of-the-art ensemble methods.

The HAT has been applied in cybersecurity applications [1, 8], and our results motivate us to explore if LAST can outperform HAT in such applications.

7 ACNKOWLEDGMENTS

This work was financed by the Pontifícia Universidade Católica do Paraná (PUCPR) through the PIBIC Master – Combined Degree program.

REFERENCES

[1] Uttam Adhikari, Thomas H. Morris, and Shengyi Pan. 2018. Applying Hoeffding Adaptive Trees for Real-Time Cyber-Power Event and Intrusion Classification. *IEEE Transactions on Smart Grid* 9, 5 (2018), 4049–4060. https://doi.org/10.1109/TSG.2017.2647778

[2] R. Agrawal, T. Imielinski, and A. Swami. 1993. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering* 5, 6 (1993), 914–925. https://doi.org/10.1109/69.250074

[3] Jean Paul Barddal and Fabricio Enembreck. 2019. Learning Regularized Hoeffding Trees from Data Streams. In *Proceedings of the 34rd Annual ACM Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 08-12, 2019.*

[4] Albert Bifet and Ricard Gavaldà. 2007. Learning from Time-Changing Data with Adaptive Windowing. *Proceedings of the 7th SIAM International Conference on Data Mining* 7. https://doi.org/10.1137/1.9781611972771.42

[5] Albert Bifet and Ricard Gavaldà. 2009. Adaptive Learning from Evolving Data Streams. In *Advances in Intelligent Data Analysis VIII*, Niall M. Adams, Céline Robardet, Arno Siebes, and Jean-François Boulicaut (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 249–260.

[6] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Philipp Kranen, Hardy Kremer, Timm Jansen, and Thomas Seidl. 2010. MOA: Massive Online Analysis, a Framework for Stream Classification and Clustering. In *Proceedings of the First Workshop on Applications of Pattern Analysis (Proceedings of Machine Learning Research, Vol. 11)*. PMLR, Cumberland Lodge, Windsor, UK, 44–50. https://proceedings.mlr.press/v11/bifet10a.html

[7] Leo Breiman. 1984. Classification and Regression Trees. Wadsworth Statistics, Wadsworth, Belmont, CA,.

[8] Diego Guarnieri Corrêa, Fabrício Enembreck, and Carlos N. Silla. 2017. An investigation of the hoeffding adaptive tree for the problem of network intrusion detection. In *2017 International Joint Conference on Neural Networks (IJCNN)*. 4065–4072. https://doi.org/10.1109/IJCNN.2017.7966369

[9] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 2002. Maintaining Stream Statistics over Sliding Windows. *SIAM J. Comput.* 31, 6 (2002), 1794–1813. https://doi.org/10.1137/S0097539701398363

[10] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research* 7 (2006), 1–30.

[11] Pedro Domingos and Geoff Hulten. 2000. Mining High-Speed Data Streams. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Boston, Massachusetts, USA) *(KDD '00)*. Association for Computing Machinery, New York, NY, USA, 71–80. https://doi.org/10.1145/347090.347107

[12] J. Gama, R. Sebastião, and P.P. Rodrigues. 2013. On evaluating stream learning algorithms. *Machine Learning* 90 (2013), 317—-346.

[13] João Gama, Ricardo Rocha, and Pedro Medas. 2003. Accurate Decision Trees for Mining High-Speed Data Streams *(KDD '03)*. Association for Computing Machinery, New York, NY, USA, 523–528. https://doi.org/10.1145/956750.956813

[14] Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. 2017. A Survey on Ensemble Learning for Data Stream Classification. *ACM Comput. Surv.* 50, 2, Article 23 (mar 2017), 36 pages. https://doi.org/10.1145/3054925

[15] Heitor Murilo Gomes, Jean Paul Barddal, Luis Eduardo Boiko Ferreira, and Albert Bifet. 2018. Adaptive random forests for data stream regression. In *26th European Symposium on Artificial Neural Networks, ESANN 2018, Bruges, Belgium, April 25-27, 2018.*

[16] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. 2017. Adaptive random forests for evolving data stream classification. *Machine Learning* 106 (2017), 1469–1495.

[17] Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *The collected works of Wassily Hoeffding* (1963), 409–426.

[18] Geoff Hulten, Laurie Spencer, and Pedro Domingos. 2001. Mining Time-Changing Data Streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California) *(KDD '01)*. Association for Computing Machinery, New York, NY, USA, 97–106. https://doi.org/10.1145/502512.502529

[19] R.B. Kirkby. 2007. *Improving hoeffding trees*. Ph. D. Dissertation. The University of Waikato.

[20] Jie Lu, Anjin Liu, Fan Dong, Feng Gu, João Gama, and Guangquan Zhang. 2019. Learning under Concept Drift: A Review. *IEEE Transactions on Knowledge and Data Engineering* 31, 12 (2019), 2346–2363. https://doi.org/10.1109/TKDE.2018.2876857

[21] Chaitanya Manapragada, Geoffrey I. Webb, and Mahsa Salehi. 2018. Extremely Fast Decision Tree. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 1953–1962. https://doi.org/10.1145/3219819.3220005

[22] J.R Quinlan. 1992. C4.5: Programs for Machine. Morgan Kaufmann Publishers, 340 Pine Street, 6th Floor San Francisco, CA 94104 USA.

[23] W. Nick Street and YongSeog Kim. 2001. A Streaming Ensemble Algorithm (SEA) for Large-Scale Classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Francisco, California) *(KDD '01)*. Association for Computing Machinery, New York, NY, USA, 377–382. https://doi.org/10.1145/502512.502568