Behavioral insights of adaptive splitting decision trees in evolving data stream classification

Daniel Nowak Assis[1], Jean Paul Barddal[1†], Fabrício Enembreck[1†]

[1*]Programa de Pós Graduação em Informática, Pontifícia Universidade Católica do Paraná, Rua Imaculada Conceição 1155, Curitiba, 80215-901, Paraná, Brazil.

Contributing authors: daniel.nowak.assis@gmail.com; jean.barddal@ppgia.pucpr.br; fabricio@ppgia.pucpr.br;
[†]These authors contributed equally to this work.

Abstract

Decision Trees are leading in state-of-the-art architectures for classifying high-speed data streams. Hoeffding-based Trees have dominated the field and are more efficient than batch counterparts because they are incremental and distribute the cost of greedy best-split evaluations throughout the stream through periodic evaluation. However, this splitting mechanism is invariant to the state of the stream and the tree, as periodic evaluations cannot capture the state of data or performance of the tree in between assessments. In this work, we significantly outline the main behaviors of a novel decision tree that can deal with high-speed data streams and adapt to performance decays considering the tree state, namely the Local Adaptive Streaming Tree (LAST) and Hoeffding-based Trees . We also provide a comprehensive benchmark of online decision trees, analyze how split moments affect performance, how the trees react to increases in the number of features and classes, and how the trees behave in concept drift scenarios. Results show that (i) LAST presents the best results, regardless of the change detector selected, (ii) LAST's strategy of branching the tree based on performance decays is the reason it outperforms other decision trees, (iii) decision trees present similar CPU-Time, but trees with split reevaluation are more costly in memory and (iv) LAST presents superior results in datasets with abrupt changes, while datasets with gradual changes depend on choosing detectors that are more suitable for gradual changes.

Keywords: Data Streams, Decision Tree, Concept Drift

1 Introduction

Data Stream Mining comprises the development of algorithms capable of predicting and learning at any time from sequential, continuously arriving, and fast-paced instances. Data stream applications are becoming ubiquitous, as the amount of data generated in real-time is increasing with the development of new technologies, such as bank transactions, sensor applications, social media, cybersecurity, and many other applications of online machine learning [1].

In real-world applications, the assumption that data are stationary cannot hold since a fundamental problem in data stream mining is concept drift, where probabilistic changes in the data at feature, class level, or both can affect the predictive performance of the algorithm over time. Consequently, algorithms must be able to react to these changes that can ensue at different speed rates.

Decision Trees are one of the principal algorithms for data stream classification. Research on decision trees has remained active for 60 years since its inception due to their interpretability, robustness, and utility in various applications [2]. Hoeffding Tree [3] is a popular approach for learning decision trees from streaming data, as well as the core for many state-of-the-art (SOTA) decision trees and ensembles for data stream mining [4–6]. The principal component of the Hoeffding Tree algorithm is the splitting mechanism, where the tree trains incrementally, gathering data at leaf nodes and attempting to split periodically. A Hoeffding test is performed at each split attempt to determine if the leaf will split in the feature that maximizes the purity of the child nodes. This mechanism distributes the cost of the greedy best-split evaluation throughout the stream, and it presents an efficient algorithm for dealing with data streams compared to batch decision trees, such as CART [7] or C4.5 [8].

To the best of our knowledge, the works on incremental decision trees for mining streaming data are extensions of Hoeffding Trees, preserving the idea of periodic evaluation of splits. Hoeffding Trees were extended with reevaluation of prior splits [4, 5, 9] or by changing the Hoeffding bound applied in the split attempt [5, 10–12].

However, the periodic split mechanism cannot capture the evolution of the stream and tree performance between split evaluations, being invariant to them. Even when an evident change occurs in the purity of a leaf node, a leaf node may split in a posterior split attempt. Even when little change ensues, the greedy evaluation of attributes and their values that compose the best split will still be performed, consuming unnecessary processing time and not significantly changing the tree structure.

To overcome this limitation of Hoeffding Trees, the novel **Local Adaptive Streaming Tree (LAST)** [13] incorporates adaptive splitting mechanism, where change detection algorithms [14] track either the performance or data distribution purity locally (at leaf nodes). This way, LAST reacts to performance decays through splitting and evaluates the best split only when the detector triggers a change. Our experiments demonstrate that LAST outperforms Hoeffding-based Trees due to its superior splitting mechanism. This paper provides robust experiments that depict insights into Hoeffding-based Trees and LAST as an ablation.

We provide *i)* an evaluation of the predictive quality and computational cost decision trees in 37 datasets, where we identify the best decision trees and change detectors for LAST given their predictive quality and computational cost, *ii)* an analysis of the

effect of splitting in the predictive quality over time, that for the best of our knowledge, was never done before, *iii)* an analysis of the impact of varying the number of classes and features, that is rarely performed and *iv)* an analysis of the effect of concept drift in the predictive quality, and best and worst case scenarios given the type of drift.

The contributions and highlights of this paper are summarized as follows:

- A large and most comprehensive benchmark of decision trees in the online learning literature considering different scenarios and concept drift types and sources.
- LAST with error-rate monitoring presents the best results in accuracy ranking among all evaluated datasets, regardless of the concept drift detector.
- LAST with HDDM$_A$ presented the best results, being able to present good results in both real and synthetic data, being the recommended configuration for LAST. LAST with HDDM$_A$ lower computational cost compared to detectors such as DDM and EDDM.
- LAST presents similar CPU-Time among all decision trees. Still, it presents lower memory usage while presenting superior predictive quality compared to state-of-the-art decision trees that have a reevaluation of splits mechanisms, such as EFDT and HAT.
- A novel experimental setup and visualization (through illustrations of performance and splitting points simultaneously) that Hoeffding-based Trees are mostly invariant to the performance of the tree and leaf node, and splitting in performance decay makes LAST surpass Hoeffding-based Tree's performance.
- A standard set of experiments for evaluation of decision trees. Including a benchmark, novel evaluation on how splitting affects the decision tree accuracy throughout time, how algorithms react to varying the number of classes and features (that is rarely done in the literature), and how algorithms react to concept drift in real-world and synthetic data.
- LAST performs better than Hoeffding-based methods in abrupt changes. In contrast, good performance in datasets with gradual concept drift depends on choosing detectors designed to deal better with gradual changes, such as EDDM.
- A depiction of a challenging drifting real-world scenario in which HAT could not react to drift as well as common synthetic data with simulated drift and a discussion of of Hoeffding Adaptive Tree's disparity in performance between real-world and synthetic data.

This paper is organized as follows. Section 2 presents the main concepts and definitions related to data streams and concept drift. Section 3 presents works in incremental decision trees for data stream mining, including our previously proposed Local Adaptive Streaming Tree. Section 4 reports the experimental protocol and results obtained. Finally, Section 5 concludes this paper and states future works.

2 Background and Definitions

Data streams are sequences generated over time without an expectation to end and are commonly regarded as potentially infinite. In the context of data stream mining, the data generated is a set of feature vectors, $DS = \{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_t, \vec{x}_\infty\}$, where $\vec{x}_t$ is a d-dimensional observed at an instance t.

The problem of data stream classification, which is the scope of this work, follows the traditional classification problem, where a classifier $f : \vec{x}_t \rightarrow y_t$ learns how to map a vector of features $\vec{x}_t$ into a set of distinct labels $y \in Y$ (classes) over time, hereafter depicted as t. Data stream classification differs from traditional batch classification in terms of memory and time processing requirements for mining streams efficiently, as it is unfeasible to store potentially infinite data. Ideally, a classifier must be able to process an instance as fast as the arrival of another example. Otherwise, data must be discarded (leaving the system outdated for more recent instances) or stored (which may lead to storage of more data and system collapse due to lack of memory) [15].

A relevant challenge of data streams is the concept drift phenomenon. Concept drift regards changes in the probabilistic properties of data throughout time. Given two time stamps t and $t+\Delta$, with $\Delta > 0$, a concept drift occurred if $P_t(\vec{x}, y) \neq P_{t+\Delta}(\vec{x}, y)$. As $P(\vec{x}, y) = P(\vec{x})P(y|\vec{x})$, three sources of drift are observable. Changes only in $P(\vec{x})$, at a feature level, where class mapping remains unchanged but changes in features happen, are commonly regarded as virtual drift, changes in $P(y|\vec{x})$, where the class mapping changes but features remain unchanged, known as real drift, and when both co-occur [14].

Concept drifts may also be categorized into four types according to how they occur in time. Abrupt drifts occur if a concept C_1 suddenly changes to a concept C_2, whereas gradual drifts occur if the shift between the prior and posterior concepts takes longer. Additionally, incremental drifts are said to happen if changes between 2 concepts occur in the face of intermediary concepts. Finally, reoccurring concepts are said to occur if a previous concept reappears over time.

For the interested reader in the concept drift problem, we recommend the following papers [14, 16].

3 Incremental Decision Trees

This section reviews state-of-the-art decision trees designed for mining high-speed data streams.

3.1 Hoeffding Trees

Hoeffding Trees (HT) and the Very Fast Decision Tree (VFDT) system [3] are some of the most popular choices of classifiers for streaming data at a monolithic and ensemble level.

Hoeffding trees are centered on the theory of the Hoeffding bound for performing splits and incrementally building the tree throughout the stream. The user defines the frequency in which leaf nodes perform split attempts based on the Hoeffding bound, known as the *Grace Period (GP)* parameter. At every GP samples observed at the leaf node, i.e., $n_l \bmod GP = 0$), a greedy best split search is performed, and a split happens if the bound presented in Equation 1 holds.

$$\Delta G(X_a) - \Delta G(X_b) > \epsilon = \sqrt{\frac{R^2 \log(\frac{1}{\delta})}{2n_l}} \tag{1}$$

The components of Equation 1 include X_a being the feature with the best purity measure $\Delta G(\cdot)$ (such as information gain or gini index), X_b the feature with second-best purity measure, R the range of the $\Delta G(\cdot)$ function, n_l the number of samples in the leaf node prior to splitting and δ result in a $(1-\delta)$ confidence level that X_a is an appropriate feature for splitting.

In some cases, where $\Delta G(X_a) - \Delta G(X_b)$ is low, and the bound tends to zero as $n_l \to \infty$, potentially many samples will be required to choose one with high confidence. However, the selected feature makes no difference since their $\Delta G(\cdot)$ values are similar. Therefore, the authors extend the Hoeffding bound constrain with a tie threshold τ (user-given), such that the constraint is $\Delta G(X_a) - \Delta G(X_b) > \epsilon \vee \tau < \epsilon$.

Figure 1 illustrates the training process of Hoeffding Trees.

Assuming that 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 $\mathcal{O}(ldvc)$ memory to store the necessary counts.

In [17], the authors proposed the support of Naive Bayes at leaf nodes for classifying data, and in [18], the authors further enhanced this mechanism by selecting between majority voting and Naive Bayes at leaf nodes according to the method with the highest accuracy in the leaf. This adaptive strategy is commonly used in the literature, and we also followed it when assessing all trees in this study.

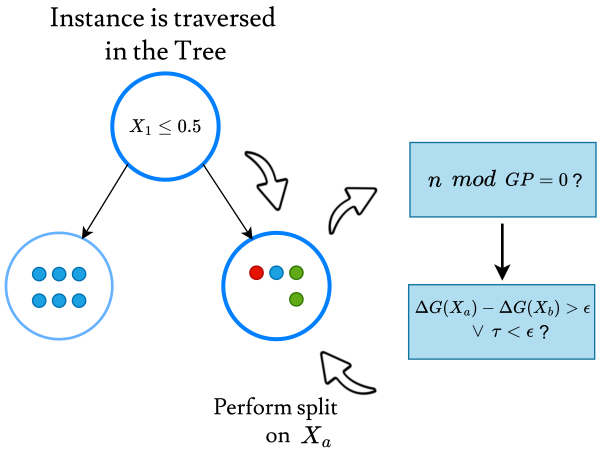

Fig. 1: Training Process of Hoeffding Trees.

3.2 Extremely Fast Decision Tree

In [5], the Extremely Fast Decision Tree (EFDT) algorithm is proposed, in which authors claim faster convergence to batch trees, thus giving rise to the "extremely" term. Instead of comparing the two best features in the Hoeffding bound, the tree

compares the best feature against the case where no split is performed, or simply $\Delta G(X_a) > \epsilon$.

EFDT also has a post-pruning technique similar to the splitting phase. As described earlier, this technique adds more cost to the tree since it must perform more greedy best-split searches and must store data in non-terminal nodes. As more instances arrive, it is possible that better splits can replace the ones performed priorly. As in splitting, the user defines the frequency at which split reevaluations of splits are performed in non-terminal nodes. If a new split on X_a is better than the split performed at a non-terminal node X_{prior}, i.e., $\Delta G(X_a) - \Delta G(X_{\text{prior}}) > \epsilon$. If true, a split on X_a replaces X_{prior} and all its child nodes.

Since EFDT must store data at intermediate nodes to reevaluate splits, it requires $\mathcal{O}(ldvc)$ memory, as n is the number of nodes.

3.3 Hoeffding Option Tree

Hoeffding Option Trees [9] adapt Option trees [19, 20] for a streaming setting by coupling option nodes to Hoeffding Trees. Option nodes allow instances to traverse multiple decision paths (the instance traverses all the child nodes of an option node). The primary motivation of Option trees is to reduce the instability of the tree by aggregating multiple decision paths through majority voting or weighted voting. In [9], the authors propose adding new nodes to option nodes, similarly to the splitting mechanism. Periodically, an addition of an option node is tested through the Hoeffding bound. Given X_a, a feature not contained in the option node that maximizes the purity in case of a split, and $X_{\max}$, the feature with max purity in the option node, a node is added to the option if $\Delta G(X_a) - \Delta G(X_{\max}) > \epsilon$. The authors also delimit a maximum number of nodes in an option node (set as five).

3.4 Hoeffding Adaptive Tree

Hoeffding Adaptive Tree (HAT) [4] is a version of Hoeffding Trees with adaptive non-terminal nodes. Each non-leaf node has an ADaptive WINdowing (ADWIN) [21] change detector that monitors the error rates of instances that traverse the non-terminal nodes. If the change detector triggers an increase in error in instances that traverse the non-terminal node, a new subtree with a single split is built and replaces the non-terminal node and its child nodes if it is more accurate. Since ADWIN memory complexity is $\mathcal{O}(\log(W))$, HAT complexity is $\mathcal{O}(\log(W)ndlc)$ as n is the number of nodes.

3.5 PLASTIC

PLASTIC [22] is an extension of EFDT. Instead of reevaluating splits, it performs a restructure of the branch analyzed. Plasticity refers to the idea that the order of features that the instance traverses in the tree does not affect the final prediction, such that $(X_1 \rightarrow X_2 \rightarrow \text{leaf}) \equiv (X_2 \rightarrow X_1 \rightarrow \text{leaf})$. Instead of pruning the tree branch and splitting in a new feature, as in EFDT, PLASTIC restructures the branch aiming to position the best feature at the top of the branch analyzed, and the plasticity idea permits this to happen.

3.6 LAST: Local Adaptive Streaming Tree

All of the trees previously described perform periodic split attempts. Since the creation of the leaf node, split attempts are performed at the moment that the leaf node gathers $\{GP, 2GP, 3GP, \dots\}$ instances until it passes the Hoeffding test and ensues a split. However, this mechanism is invariant to the stream's state and tree performance. Furthermore, the tree cannot react to changes in performance between split attempts, and the bound does not consider that a performance decay occurred. If no change occurs, the tree still attempts to split, search the best split greedily constantly, and the split will not result in significant changes in the tree and waste unnecessary processing time.

A change detection algorithm can track performance or class distribution purity changes. For this sake, we generalize error rate-based drift detection [14] for determining splitting points. Error rate-based drift detection assumes that decay in the algorithm's performance is a concept drift. However, in the context of leaf nodes, decay in performance can also signify a model drift, where the arrival of more data shows that the model did not learn enough, or the Naive Bayes, even though it is the method with better accuracy, the majority class would be the best method in a short term.

Therefore, the Local Adaptive Streaming Tree (LAST) is an algorithm that applies change detection algorithms for determining splitting points. Figure 2 shows the training process of LAST, and Algorithm 1 shows LAST pseudocode. At the leaf, change detectors can monitor either the error rate ($\mathbb{1}\{DT(\vec{x}) \neq y\}$, Alg. 1, lines 11 and 12) or impurity (entropy, if information gain is used as split criteria, or Gini if Gini Index is used, Alg. 1, lines 9 and 10). If a change detector at the leaf triggers a change, then a split will ensue if $\Delta G(X_a) > 0$ (Alg. 1, line 17).

Interpretability is also enhanced in the tree throughout time, as it is unintuitive to understand why Hoeffding-based Trees grow due to a statistical test and blind recurrent splitting attempts, while in LAST, it is possible to see how the tree grows due to decays in predictive quality.

LAST requires $\mathcal{O}(\psi ldvc)$ memory, as ψ is the memory complexity of the change detection algorithm applied, as every leaf node has a change detector.

LAST is already present in modern data stream mining frameworks such as River [23] in the following link[1] and MOA (Massive Online Analysis) [24] in the following link[2].

[1] https://github.com/online-ml/river/blob/main/river/tree/last_classifier.py
[2] https://github.com/Waikato/moa/blob/master/moa/src/main/java/moa/classifiers/trees/LAST.java

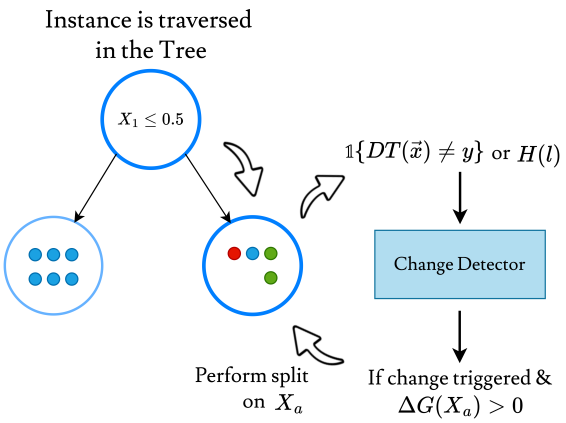

Fig. 2: Training Process of LAST.

4 Experiments and Results

The experiments outlined in this section are designed to provide a comprehensive evaluation of the algorithms' performance, focusing on predictive quality and computational cost across a wide range of scenarios. For this sake, we first compare the algorithms' overall performance. In addition, we also identify the best change detectors appropriate for LAST. We provide an analysis of the split momentum of the decision trees throughout time to understand how splits affected the tree's predictive quality. With synthetic datasets, we evaluate the effect of changing the number of features and classes in the classifiers' performance, especially the computational cost in this scenario. Moreover, we present the impact of concept drift on the performance of the classifiers. We formulate the objective of these experiments with the following research questions (RQ):

RQ1: Which classifiers have the best predictive quality and computational cost performance?
RQ2: What is the effect of splitting on the predictive quality of decision trees?
RQ3: How does the number of classes and features affect online decision trees' predictive quality and computational cost?
RQ4: What is the effect of different types and sources of concept drift in the performance of decision trees?

Finally, we also highlight and make available a repository with additional results of this paper under the following link: https://sites.google.com/view/bi-paper.

Algorithm 1 LAST

Input: S: a data stream,
X: a set of attributes,
$\Delta G(\cdot)$: a split evaluation function,
$H(\cdot)$: impurity measure analogous to $\Delta G(\cdot)$
ψ : A change detection algorithm
$useD$: boolean variable. If true, ψ has input $H(\cdot)$, else ψ input is 1 for misclassifications, otherwise 0
Output: DT: a decision tree

Let DT be a tree with a single leaf l_1 (the root)
Let $n_{l_1} \leftarrow 0$ ▷ number of observations in the leaf
$l_{1\psi} \leftarrow \psi$ ▷ create a change detector at leaf
for each sample $(\vec{x}, y) \in S$ **do**
    Sort $\vec{x}$ into a leaf l using DT
    Classify $\vec{x}$ with the majority class or Naive Bayes in l
    Update l statistics using $(\vec{x}, y)$
    $n_l \leftarrow n_l + 1$
    if $useD$ **then**
        Update l_ψ with $H(l)$
    else
        Update l_ψ with $\mathbb{1}\{DT(\vec{x}) \neq y\}$
    end if
    if l_ψ detected change $\wedge \neg$ (l contains samples from only one class) **then**
        Compute $\Delta G(X_i)$ for each $X_i \in X_l$ stored in l
        Let X_a be the attribute with highest ΔG
        if $(\Delta G(X_a) > 0) \wedge X_a \neq \emptyset$ **then**
            Replace l by leaf nodes that split on X_a
            for each leaf node l_i from splitting on X_a **do**
                Let $n_{l_i} \leftarrow 0$
                $l_{i\psi} \leftarrow \psi$ ▷ create a change detector at each leaf
            end for
        end if
    end if
end for
return DT

4.1 Experimental Protocol

The evaluation metrics for all algorithms are Accuracy, Kappa Majority, and Kappa Temporal in a test-then-train validation protocol, where every instance is classified and then trained with the original label, also known as prequential validation [25]. Predictive quality metrics were averaged in 20 evenly spaced points throughout the stream, as more points did not show significantly different results. This strategy considers the tree's performance throughout the stream and avoids methods that can outperform

others only at the end of the stream. Also, tree size (number of nodes), CPU-Time (in seconds), and memory usage (in RAM-Hours (GB/h)) are evaluated.

All experiments were performed in the Massive Online Analysis (MOA) framework [24], in an Intel(R) Core(TM) i7-12700H @ 2.30 GHz with 16 GB of RAM. The source code for our proposal is available in modern data stream mining frameworks, as noted earlier. All tree parameters were set to default from MOA.

For LAST, we performed experiments with the ADWIN [21], DDM [26], EDDM [27], HDDM$_A$ [28], HDDM$_W$ [28], RDDM [29], MDDM$_A$ [30], MDDM$_E$ [30] and MDDM$_G$[30] drift detectors. All detectors are available in the MOA framework and the default parameters have been used.

All real-world datasets were taken from [31], and synthetic datasets were generated in the MOA framework. Table 1 shows the datasets configurations, with the number of instances, features classes, percentage of instances that belong to the majority class, and concept drift type. In experiments that vary the number of classes and features, the RBF and RTG datasets were used with 50000 instances and while the number of classes changed in $\{2, 10, 50, 100, 200\}$ and the number of features in $\{10, 50, 100, 200, 300, 400, 500\}$, such that we performed experiments with all combinations of these sets of classes and features (RBF with 2 classes and 10 features, 2 classes and 50 features, and so forth).

Pairwise, one-sided Wilcoxon signed-rank tests were performed as post-hoc tests, as described in [32] and [33]. We also generate a plot similar to [34], where a line contains the algorithms' average rankings, and a line connects algorithms that are not statistically different.

4.2 Comparison between state-of-the-art decision trees

In this section, we focus on answering the first research question (**RQ1**) that regards an analysis of which decision tree classifiers achieve the best predictive quality and computational resources usage. In particular, we provide a comparison between state-of-the-art decision trees considering predictive quality and computational cost. LAST with error-rate monitoring is referred to as LAST, while LAST with class distribution purity monitoring is referred to as LAST$_D$.

Figure 3 shows a Wilcoxon test for mean accuracy throughout the stream. LAST, regardless of the detector used, presented the best results. Apart from MDDM$_E$ and HDDM$_W$, all the detectors had results statistically different from EFDT, while HAT does not. The change detectors HDDM$_A$ and MDDM$_A$ presented the best rankings. LAST$_D$ did not present a superior ranking compared to EFDT, except for ADWIN. However, LAST$_D$ with ADWIN did not present results statistically different from EFDT. Splitting the tree in moments when the error rate increases has shown superiority regarding the construction of a tree with good predictive quality (further explored in Section 4.3). Yet, monitoring the class distribution purity did not present trees with good predictive quality. We argue that either the detectors are not effective in detecting changes in real variables, or there is no correlation between changes in the class distribution and tree splitting moments that would result in higher predictive quality.

PLASTIC has shown the worst ranking compared to Hoeffding Trees, as authors do experiments in what they define as "lightweight versions of state-of-the-art decision

Table 1: Datasets configurations.

Dataset	# Samples	# Features	# Classes	Majority Class (%)	Concept Drift
AGR-f(7,8,9,10,9,8,7)$_{ra}$	1.000.000	9	2	52,83	Reocurring-Abrupt
AGR-f(7,8,9,10,9,8,7)$_{rg}$	1.000.000	9	2	52,83	Reocurring-Gradual
AGR-f(1,2,3,4,5,6,7,8,9,10)$_{a}$	1.000.000	9	2	52,83	Abrupt
AGR-f(10,9,8,7,6,5,4,3,2,1)$_{a}$	1.000.000	9	2	52,83	Abrupt
AGR-f(1,2,3,4,5,6,7,8,9,10)$_{g}$	1.000.000	9	2	52,83	Gradual
AGR-f(10,9,8,7,6,5,4,3,2,1)$_{g}$	1.000.000	9	2	52,83	Gradual
SEA-f(1,2,3,4,3,2,1)$_{ra}$	1.000.000	3	2	59,91	Reocurring-Abrupt
SEA-f(1,2,3,4,3,2,1)$_{rg}$	1.000.000	3	2	59,91	Reocurring-Gradual
SEA-f(1,2,3,4)$_{a}$	1.000.000	3	2	59,91	Abrupt
SEA-f(1,2,3,4)$_{g}$	1.000.000	3	2	59,91	Gradual
SEA-f(4,3,2,1)$_{a}$	1.000.000	3	2	59,91	Abrupt
SEA-f(4,3,2,1)$_{g}$	1.000.000	3	2	59,91	Gradual
LED-f(7,5,3,1,3,5,7)$_{ra}$	1.000.000	24	10	10,28	Reocurring-Abrupt
LED-f(7,5,3,1,3,5,7)$_{rg}$	1.000.000	24	10	10,28	Reocurring-Gradual
LED-f(1,3,5,7)$_{a}$	1.000.000	24	10	10,28	Abrupt
LED-f(1,3,5,7)$_{g}$	1.000.000	24	10	10,28	Gradual
LED-f(7,5,3,1)$_{a}$	1.000.000	24	10	10,28	Abrupt
LED-f(7,5,3,1)$_{g}$	1.000.000	24	10	10,28	Gradual
RBF$_{s}$	1.000.000	10	5	30,01	Incremental
RBF$_{m}$	1.000.000	10	5	30,01	Incremental
RBF$_{f}$	1.000.000	10	5	30,01	Incremental
HYPER$_{s}$	1.000.000	10	2	50	Incremental
HYPER$_{m}$	1.000.000	10	2	50	Incremental
HYPER$_{f}$	1.000.000	10	2	50	Incremental
Outdoor	4.000	21	40	4,11	Unknown
Elec	45.312	8	2	57,41	Unknown
Rialto	82.250	27	10	10,00	Unknown
Airlines	539.383	7	2	55,47	Unknown
CoverType	581.012	54	7	48,75	Unknown
Nomao	34.465	119	2	71,44	Unknown
Poker	829.201	10	10	47,78	Unknown
NOAA	18.158	8	2	69,74	Unknown
INSECTS$_{a}$	52.848	33	6	16,07	Abrupt
INSECTS$_{i}$	57.018	33	6	11,56	Incremental
INSECTS$_{g}$	24.150	33	6	15,76	Gradual
Asfault	8.066	62	5	55,59	Unknown
LADPU	22.950	96	10	10	Unknown

trees" by defining a max depth of 20; however, Hoeffding Trees "amnesia" effect upon discarding instances when splitting mitigates overfitting and constructs a model that constantly learns with new samples [35]. Limiting the trees depth to 20 suppresses the amnesia effect, which results in the good results of Hoeffding Trees, and we judge that a depth of 20 is high for preventing overfitting. Therefore, by experimenting with the original algorithm configurations of each trees as in MOA, Hoeffding Trees present superior results compared to PLASTIC.

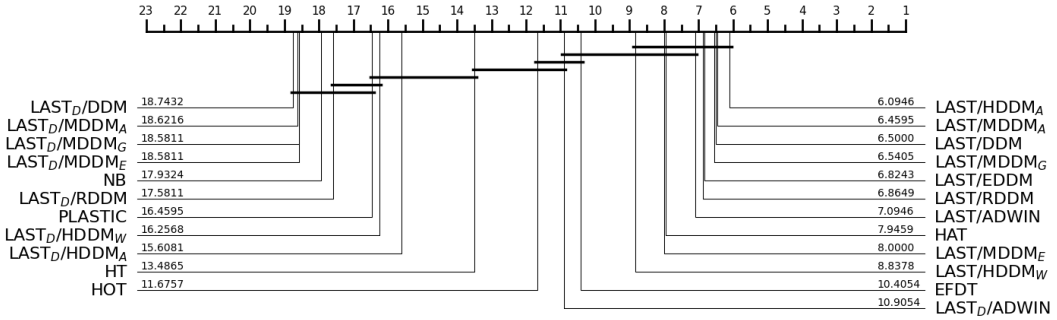

Fig. 3: Mean accuracy ranking throughout the stream for the *baseline*, LAST e LAST$_D$ with drift detectors

Figure 4 presents a Wilcoxon test in tree size (number of nodes). LAST with detectors MDDM, HDDM, and DDM did not present results statistically different from HT, EFDT, and HAT, while LAST with detector ADWIN was the only one that presented results statistically different from EFDT. PLASTIC presented the largest tree size ranking, as the restructuring phase of the tree can result in even larger trees.

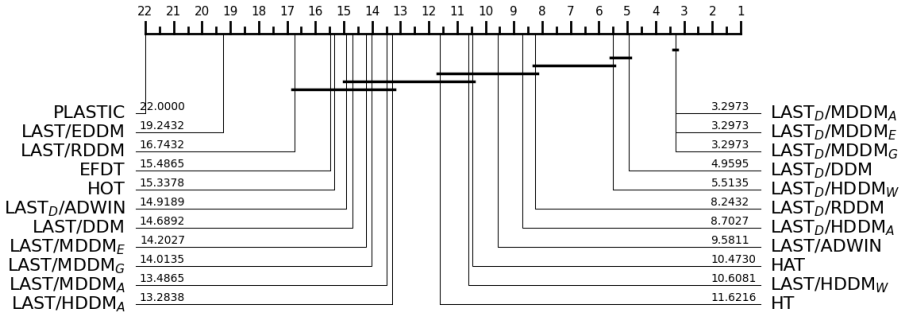

Fig. 4: Tree size (number of nodes) ranking of the baseline, LAST and LAST$_D$ with drift detectors

Table 2 show the mean rankings of the metrics accuracy, Kappa$_M$, Kappa$_T$, number of nodes, CPU-Time and RAM-Hours. LAST with the detectors RDDM, EDDM, and DDM presented the best results in accuracy, Kappa$_M$, Kappa$_T$ in real-world datasets, but not the best results in synthetic datasets, showing a disparity, and also presented the worst results in tree size, CPU-Time and RAM-Hours in both real-world and synthetic datasets. The same behavior of these three detectors can be explained by how similar they detect drift, as they seek to identify a deviation from the standard behavior considering descriptive statistics.

Table 2: Mean ranking per metric and total time to process all datasets in real-world and synthetic scenarios.

Dataset Type		NB	HT	HOT	EFDT	HAT	PLASTIC
Real-World	Acc	18.31(20)	15.69(16)	13.08(12)	9.85(11)	14.85(14)	17.31(17)
	Kappa_M	18.23(20)	15.77(16)	14.77(13)	9.92(11)	15.08(14)	17.54(18)
	Kappa_T	18.31(20)	15.85(16)	13.08(12)	10.00(11)	15.23(14)	17.54(18)
	Size	-	7.38(7)	11.04(10)	13.85(13)	6.92(6)	20.35(21)
	CPU-Time	**1.08(1)**	9.35(8)	15.92(16)	18.54(20)	18.85(21)	21.08(23)
	Total-Time	**54.45(1)**	123.83(8)	149.81(12)	207.50(21)	182.33(17)	1494.29(23)
	Ram-Hours(10^{-6})	9.08(8)	7.85(6)	12.00(10)	16.77(19)	15.08(16)	23.00(23)
Synthetic	Acc	17.73(19)	12.29(14)	10.92(12)	10.71(11)	**1.00(1)**	16.00(15)
	Kappa_M	17.73(19)	12.29(14)	10.96(12)	10.67(11)	**4.21(1)**	16.04(15)
	Kappa_T	17.73(19)	12.29(14)	10.96(12)	10.71(11)	**4.21(1)**	16.00(15)
	Size	-	13.96(15)	18.17(20)	16.54(19)	12.75(11)	19.25(22)
	CPU-Time	**1.00(1)**	14.08(14)	20.75(22)	19.38(20)	20.50(21)	22.96(23)
	Total-Time	**95.30(1)**	254.80(11)	649.55(22)	422.56(19)	580.05(21)	6663.32(23)
	Ram-Hours(10^{-6})	6.21(6)	14.42(15)	19.67(22)	17.62(19)	19.08(21)	23.00(23)

Table 2 : Continued.

Dataset Type		LAST ADWIN	DDM	EDDM	HDDM_A	HDDM_W	RDDM
Real-World	Acc	7.31(9)	6.46(7)	6.00(3)	5.62(2)	6.85(8)	**1.00(1)**
	Kappa_M	7.15(9)	6.38(7)	6.23(6)	5.31(2)	6.62(8)	**5.23(1)**
	Kappa_T	7.23(9)	6.46(7)	6.31(6)	5.38(2)	6.69(8)	**5.31(1)**
	Size	10.88(9)	16.85(19)	20.62(22)	16.04(18)	11.62(12)	18.35(20)
	CPU-Time	10.38(9)	16.23(18)	20.62(22)	15.92(16)	10.58(10)	17.38(19)
	Total-Time	125.67(9)	197.38(19)	266.02(22)	182.50(18)	109.50(7)	204.66(20)
	Ram-Hours(10^{-6})	10.77(9)	16.85(20)	21.00(22)	15.54(17)	12.00(10)	18.62(21)
Synthetic	Acc	6.98(6)	6.52(3)	7.27(7)	6.35(2)	9.92(10)	7.71(8)
	Kappa_M	7.02(6)	6.65(4)	7.27(7)	6.40(2)	9.75(10)	7.83(8)
	Kappa_T	7.02(6)	6.52(3)	7.31(7)	6.35(2)	9.83(10)	7.79(8)
	Size	8.96(7)	13.85(13)	19.25(22)	11.96(10)	10.06(9)	16.12(18)
	CPU-Time	11.69(10)	12.62(12)	17.04(19)	12.17(11)	9.54(9)	15.75(18)
	Total-Time	241.89(10)	308.30(18)	427.34(20)	286.64(13)	221.50(9)	304.94(17)
	Ram-Hours(10^{-6})	9.83(9)	13.62(13)	18.50(20)	12.08(11)	10.21(10)	16.25(18)

Table 2 : Continued.

Dataset Type		LAST MDDM_A	MDDM_E	MDDM_G	LAST_D ADWIN	DDM
Real-World	Acc	6.23(6)	6.23(6)	6.15(4)	9.77(10)	17.46(18)
	Kappa_M	6.08(4)	6.15(5)	6.08(4)	9.69(10)	17.15(17)
	Kappa_T	6.23(4)	6.23(4)	6.15(3)	9.77(10)	17.15(17)
	Size	14.08(14)	14.69(16)	14.62(15)	15.35(17)	5.50(5)
	CPU-Time	13.81(14)	13.62(12)	13.69(13)	14.23(15)	4.96(4)
	Total-Time	171.12(14)	176.70(15)	177.98(16)	154.70(13)	100.66(6)
	Ram-Hours(10^{-6})	13.92(14)	14.23(15)	13.92(14)	15.69(18)	4.69(4)
Synthetic	Acc	6.58(4)	8.96(9)	6.75(5)	11.52(13)	19.44(23)
	Kappa_M	6.58(3)	8.92(9)	6.71(5)	11.48(13)	19.44(23)
	Kappa_T	6.62(4)	8.96(9)	6.75(5)	11.44(13)	19.44(23)
	Size	13.38(12)	14.15(16)	13.90(14)	15.08(17)	4.65(5)
	CPU-Time	14.10(15)	14.31(16)	13.00(13)	15.54(17)	4.15(3)
	Total-Time	287.59(15)	286.92(14)	283.80(12)	292.45(16)	159.45(2)
	Ram-Hours(10^{-6})	13.67(14)	14.58(16)	13.58(12)	15.75(17)	3.92(2)

HDDM_A presented inferior computational cost compared to RDDM, EDDM, and DDM and is the method with the second-best ranking in accuracy in both real and synthetic data, while other methods usually show a disparity. All MDDM variants obtained lower computational cost ranking while presenting good predictive quality in both real-world and synthetic datasets. MDDM_A presented the best results among MDDM variations. HDDM_A e MDDM variants had superior results compared to EFDT in predictive quality while presenting lower CPU-Time and RAM-Hours in real-world and synthetic datasets. ADWIN and HDDM_W depict a good trade-off between predictive quality and computational cost, as they present superior results in predictive

Table 2 : Continued.

Dataset Type		LAST$_D$ HDDM$_A$	HDDM$_W$	RDDM	MDDM$_A$	MDDM$_E$	MDDM$_G$
Real-World	Acc	14.00(13)	15.38(15)	17.96(19)	18.81(23)	18.69(22)	18.69(22)
	Kappa$_M$	13.77(12)	15.46(15)	17.88(19)	18.58(23)	18.46(22)	18.46(22)
	Kappa$_T$	13.85(13)	15.54(15)	17.96(19)	18.65(23)	18.54(22)	18.54(22)
	Size	11.31(11)	8.12(8)	5.42(4)	3.35(2)	3.35(2)	3.35(2)
	CPU-Time	11.08(11)	7.92(7)	6.08(6)	5.58(5)	4.62(3)	4.50(2)
	Total-Time	136.17(11)	72.95(4)	136.09(10)	73.22(5)	72.92(3)	72.91(2)
	Ram-Hours(10^{-6})	12.69(12)	8.08(7)	4.73(5)	3.92(3)	2.85(2)	**2.73(1)**
Synthetic	Acc	16.48(16)	16.73(17)	17.38(18)	18.52(21)	18.52(21)	18.52(21)
	Kappa$_M$	16.56(16)	16.60(17)	17.33(18)	18.52(21)	18.52(21)	18.52(21)
	Kappa$_T$	16.48(16)	16.69(17)	17.33(18)	18.52(21)	18.52(21)	18.52(21)
	Size	7.35(6)	4.12(4)	9.75(8)	3.25(2)	3.25(2)	3.25(2)
	CPU-Time	8.19(7)	4.12(2)	9.02(8)	4.98(4)	6.00(6)	5.10(5)
	Total-Time	207.94(8)	164.95(6)	196.84(7)	160.92(3)	162.06(5)	161.12(4)
	Ram-Hours(10^{-6})	7.42(7)	**3.83(1)**	9.81(8)	3.92(2)	4.92(5)	4.10(4)

quality compared to EFDT while they present similar computational costs to Hoeffding Trees. ADWIN presented the closest Total time (the time it took the algorithm to run all the datasets) to HT. ADWIN, HDDM, and MDDM have more strict change detection constraints with statistical bounds (Hoeffding and McDiarmid) in comparison to RDDM, EDDM, and DDM, and methods based in statistical bound and descriptive statistics are noticeable, as methods based in statistical bounds present smaller tree size than methods based in descriptive statistics, therefore fewer change detections. Given the results, we recommend LAST with HDDM$_A$ as a standard configuration for LAST.

ADWIN was the only change detector in which the algorithm LAST$_D$ obtained comparable results to the best methods but presented higher computational cost and lower predictive quality compared to LAST with ADWIN. More unnecessary detections were probably triggered by the algorithm when monitoring class distribution. Another possibility is that there is a low correlation between changes in class distribution and appropriate moments for splitting the tree.

For a more in-depth analysis of the trees, we choose three change detectors for LAST and three baseline algorithms for a better understanding of the behavior of these algorithms. We selected LAST with HDDM$_A$ and MDDM$_A$ because they obtained the best ranking in accuracy and inferior computational cost compared to RDDM, EDDM and DDM, and ADWIN detector, by the good trade-off between accuracy and statistically different results compared to EFDT, in opposition to HDDM$_W$. For baselines, we choose HT, EFDT, and HAT because they are widely used in the literature and present good results.

Figure 6 shows a scatter plot of the mean accuracy throughout the stream for HAT and LAST with the selected change detectors. In real-world datasets, LAST presented a number of better results compared to HAT, greater or equal to 11 out of 13 datasets. All the trees have shown a statistical difference in a Wilcoxon test with p-value inferior to 0.05. In synthetic datasets, HAT presented more wins than LAST. None of the trees presented a statistical difference compared to HAT in a Wilcoxon test with p-value 0.05. Section 4.4 discusses the disparity in results in real-world and synthetic data by the characteristic of the simulated concept drift.

Figure 7 shows a scatter plot of the mean accuracy throughout the stream for EFDT and LAST with the selected change detectors. In real-world data, LAST with MDDM$_A$ was the best tree in terms of win/loss ratio compared to EFDT, with 12 wins and one loss. LAST with detectors HDDM$_A$ e MDDM$_A$ presented a statistical

difference compared to EFDT with a p-value inferior to 0.05 in a Wilcoxon test. In synthetic data, LAST presented a number of better results than EFDT, which was greater or equal to 17 out of 24 datasets. All LAST variations presented a statistical difference compared to EFDT with a p-value inferior to 0.05.

Figure 5 provides a sensibility analysis to LAST most important parameter, e.g, of its detector. HDDM$_A$ contains only one significant parameter, that is the level of confidence for detecting drift. We tested values of confidence $\delta = \{10^{-1}, 10^{-2}, \ldots, 10^{-8}\}$. Any statistical difference between parameters values is observable and the default parameter value ($\delta = 10^{-3}$) obtained similar ranking compared to the best parameters $(10^{-2}, 10^{-4}, 10^{-6})$.

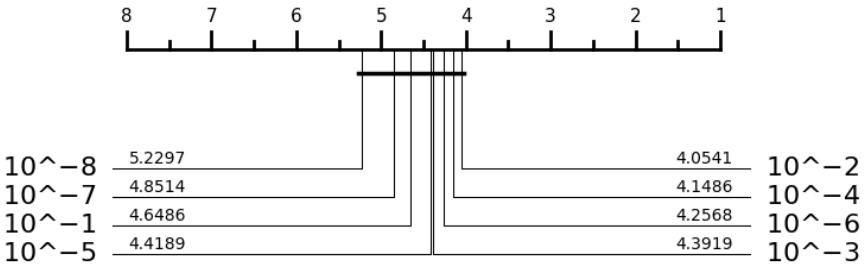

Fig. 5: Tree size (number of nodes) ranking of the baseline, LAST and LAST$_D$ with drift detectors

In conclusion, LAST, with error-rate monitoring, has demonstrated itself as an efficient classifier, independently of the change detector used. LAST$_D$ with ADWIN was the only among LAST$_D$ variations that presented the best results, yet presented higher computational cost and lower predictive quality compared to LAST with ADWIN. We observed that change detectors based on statistical bound (HDDM, MDDM, and ADWIN) are less computationally intensive than descriptive statistics (RDDM, EDDM, and DDM) and can produce predictive quality as good as them. ADWIN presented a good trade-off in predictive quality and computational cost compared to state-of-the-art decision trees, as it presented higher mean accuracy and ranking compared to EFDT and presented a similar computational cost compared to HT. A parameter sensitivity analysis also shows no statistical difference between varying the values of parameters of the best performing detector HDDM$_A$.Therefore, addressing question **RQ1**.

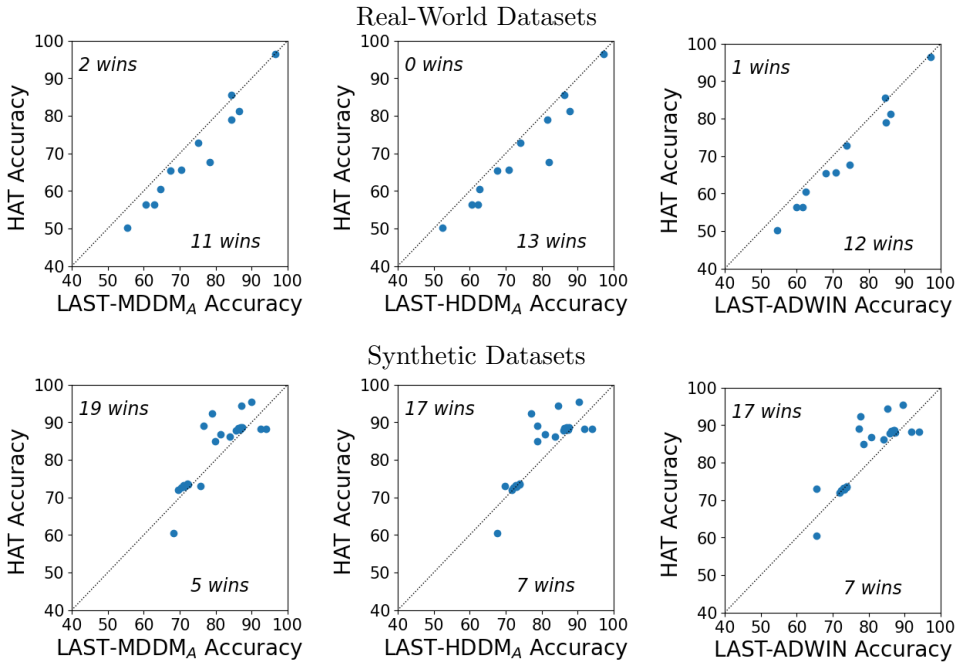

Fig. 6: Comparison of mean accuracy (%) computed throughout the stream between HAT and LAST with change detector in real and synthetic datasets. Each point represents the result in a single dataset.

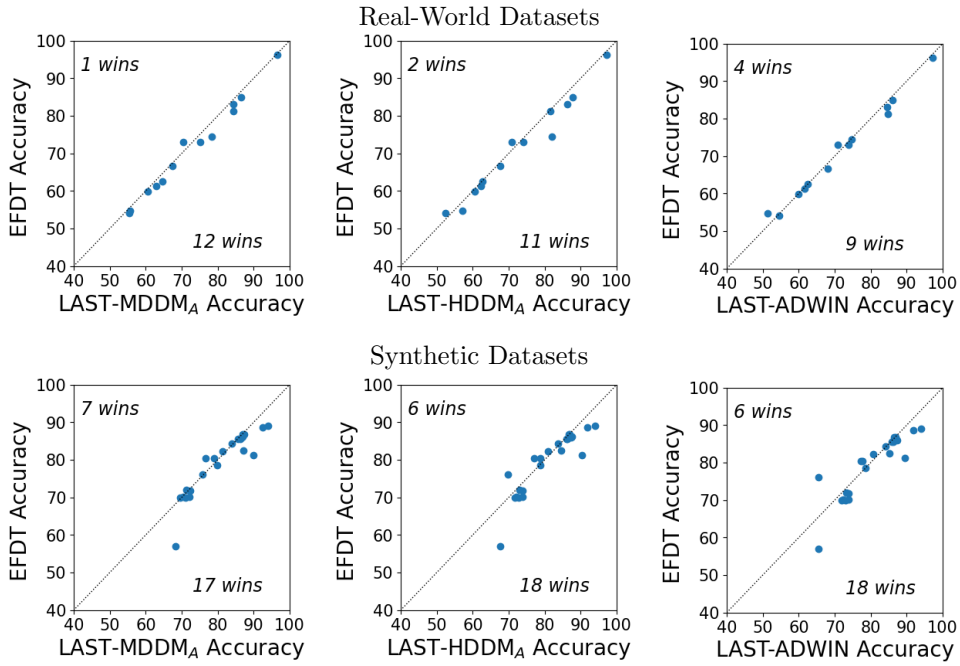

Fig. 7: Comparison of mean accuracy (%) computed throughout the stream between EFDT and LAST with change detectors in real and synthetic datasets. Each point represents the result in a single dataset.

4.3 Split momentum of decision trees over time

To address **RQ2**, this section presents the effect of splitting the tree in the accuracy of online decision trees. Therefore, we present the accuracy over time for selected datasets, in which vertical lines in the color of the algorithm denote a node split.

The datasets selected were the ones that permit the visualization of the accuracy over time without the interference of the number of splits on the figure. For some datasets, we limited the number of instances to 15,000 for visualization purposes. For LAST, we present results only for ADWIN, as it produces smaller trees than HDDM$_A$ and MDDM$_A$.

Figure 8 shows the accuracy over time with splitting points for the Asfault, Outdoor, and NOAA datasets. These scenarios show that Hoeffding-based trees are invariants to the performance of the tree, and splitting moments when an accuracy decay is observed makes LAST a better tree than Hoeffding-based trees when they are stable or increase the accuracy.

In the Asfault dataset, HT presents the same performance as LAST with ADWIN, as both have the same initial state (an Adaptive Naive Bayes), until a performance decrease is observed by Hoeffding Trees, and LAST could react to the performance decrease through splitting and maintaining a stable performance. EFDT presented lots of unnecessary splits, especially in the initial moments of the stream, presenting

results worse than those of Hoeffding Trees. At the first performance decay of the dataset, the detector triggered a change. However, as no split presented gain greater than 0, the tree did not split.

In Outdoor and NOAA datasets, LAST with ADWIN presented a similar behavior, in which splitting the tree on performance decays stabilizes or increases the tree performance. In the Outdoor dataset, the fourth split of EFDT resulted in superior performance compared to LAST, which split after the performance decay was observed. However, further in the stream, LAST split the tree in a moment of performance decay that presented superior performance to EFDT. In the NOAA dataset, EFDT also presents a split moment near point 15,000 that reduced the difference in accuracy between LAST and EFDT.

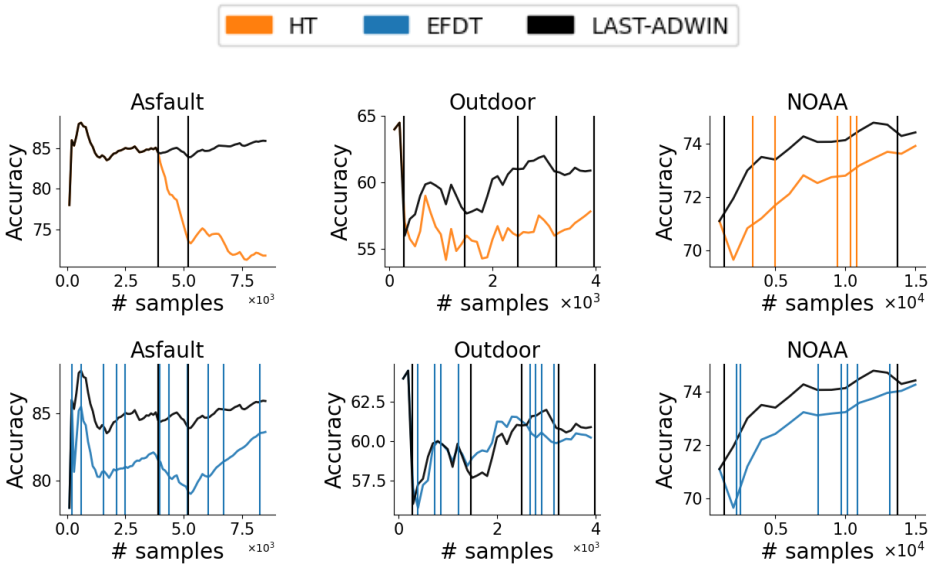

Fig. 8: Accuracy (%) over time for Hoeffding Trees, EFDT and LAST with ADWIN in Asfault, Outdoor and NOAA datasets.

Figure 9 shows the accuracy over time with splitting points for the Elec, Poker, and Nomao datasets. The Elec and Poker datasets show moments where splitting the tree does not affect the performance of the trees since the difference in the accuracy of the trees remains stable over time, as between instances 7,500 and 12,500 in the Poker dataset. However, LAST presents higher stability in moments of performance decay, as in the fifth split in the Poker dataset. In the Nomao dataset, LAST presents performance superior to HT and EFDT by detecting a change in the initial moment of the stream between instances 12,500 and 15,000.

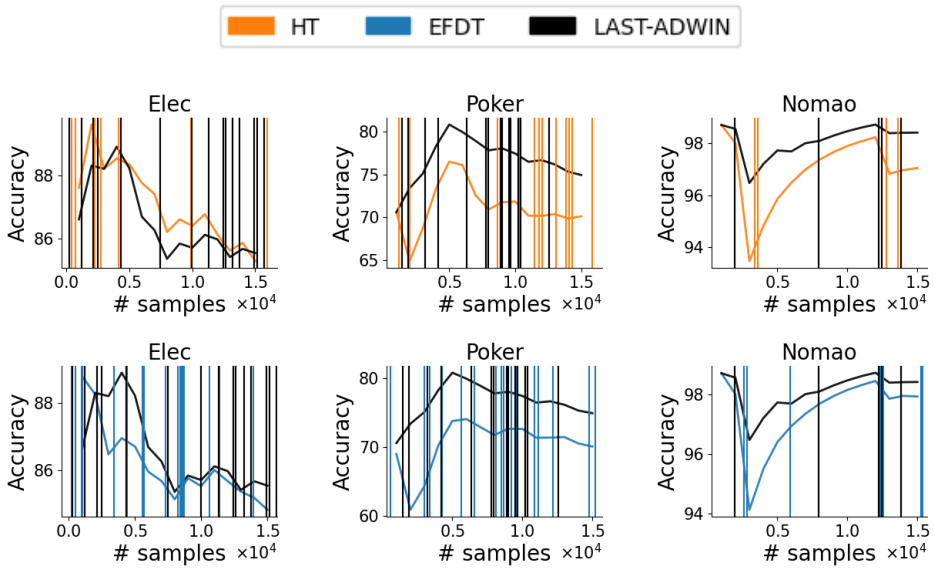

Fig. 9: Accuracy (%) over time for Hoeffding Trees, EFDT, and LAST with ADWIN in the Elec, Poker, and Nomao datasets.

In conclusion, LAST presents higher performance compared to HT and EFDT, reacting quickly to performance decay by splitting the tree. We also show examples where EFDT splitted the tree before performance decay, which could present higher performance to LAST or at least approach LAST performance. However, more cases of LAST reacting to performance decay successfully by splitting the tree are observable. In general, splitting the tree into cases of performance decay presented superior results, while periodic splitting is mostly invariant to the performance (as we observed in a few cases in which EFDT could react to performance decay after it happened), addressing **RQ2**.

4.4 Variation of the number of classes and features

To address **RQ3**, in this section we evaluate in this section the behavior of algorithms in RTG and RBF scenarios when set up with different numbers of features and classes. Figure 10 presents the accuracy of the Hoeffding Tree in the RBF and RTG datasets and follows the same pattern for all trees, i.e., accuracy decay when there is an increase in the number of classes and features. This behavior is expected, given that decision trees suffer from the curse of dimensionality [36].

Figure 11 compares HT and LAST accuracy, CPU-Time, RAM-Hours, and number of nodes for the RBF and RTG datasets. Concerning accuracy, LAST presents the highest values in most cases. HT presents a higher tree size compared to LAST, especially with ten features that obtained the highest tree size (between 30 and 40 for the RBF dataset and between 25 and 30 for the RTG dataset). It is noticeable that the Hoeffding bound is more conservative when the number of features increases,

and good splits can't be done since they will need to rely on tie-breaking, while a split could be beneficial way earlier. For CPU-Time and RAM-Hours, HT and LAST present similar results.

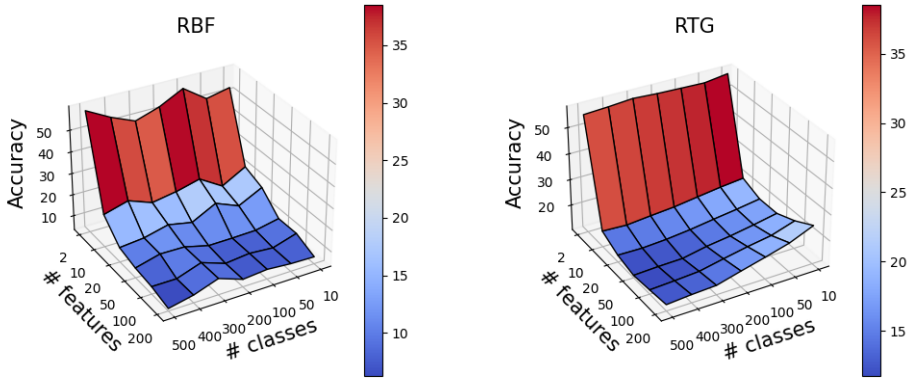

Fig. 10: Hoeffding Tree's accuracy (%) when varying the number of classes and features in the RBF and RTG.

Figure 12 shows a comparison between EFDT and LAST accuracy, CPU-Time, RAM-Hours, and number of nodes for datasets RBF and RTG. Regarding accuracy, LAST also presents values superior to EFDT in most of the scenarios. Considering the number of nodes, it is possible to notice that EFDT implies a softer Hoeffding bound compared to the one used in [3], and in RTG, EFDT had a greater number of nodes between 200 and 400 features. For CPU-Time, LAST and EFDT obtained similar results. For RAM-hours, the greater the number of classes and features, the greater is EFDT memory usage, related to the increase in number of nodes and storage of statistics about instances in each node of the tree.

Figure 13 shows a comparison between HAT and LAST accuracy, CPU-Time, RAM-Hours, and number of nodes for RBF and RTG datasets. In accuracy, LAST once again presents higher accuracy in most of the cases. Regarding the number of nodes, HAT presents a number of nodes inferior to LAST, as the bound is more conservative when the number of features increases, as described earlier. In CPU-Time, LAST and HAT presented similar results. For RAM-Hours, HAT presented higher memory usage, as in the RBF dataset, that the greater the number of classes in a scenario with ten features, the higher the memory usage by HAT. Since the tree grows deeper, more change detectors are created at non-leaf nodes and increase HAT memory usage.

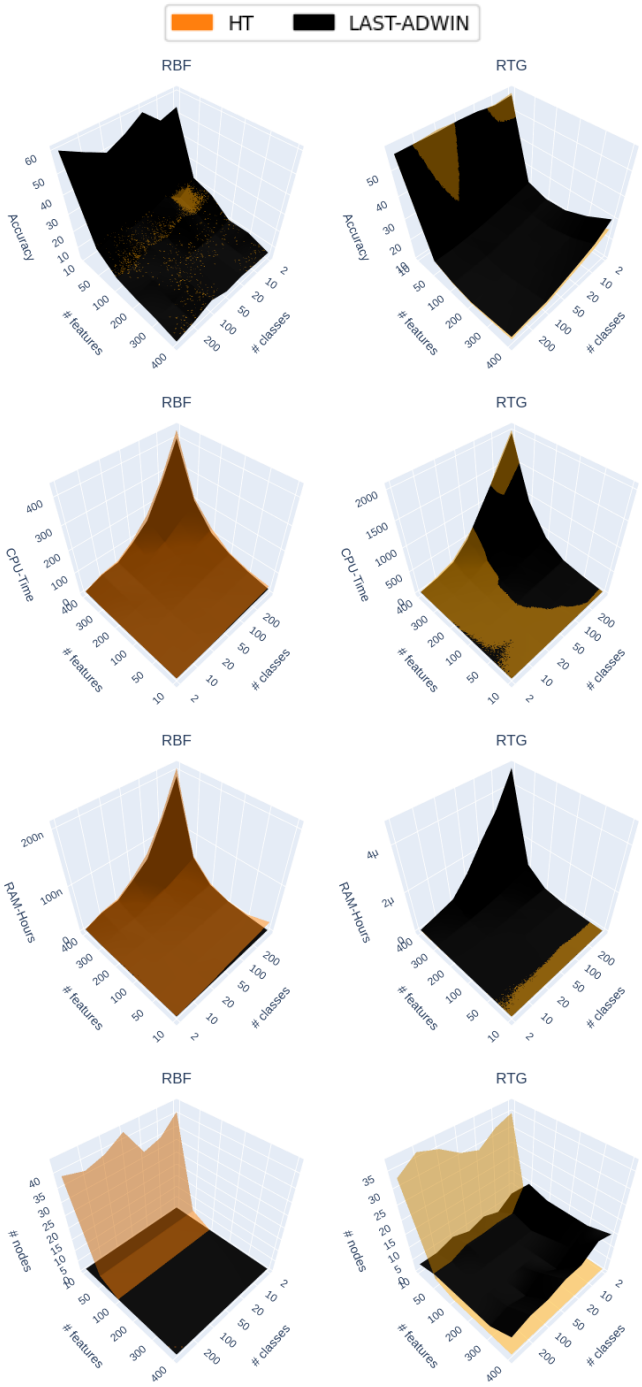

Fig. 11: Hoeffding Trees and LAST with ADWIN accuracy (in %), CPU-Time (in seconds), RAM-Hours (in GB/h) and number of nodes in RBF and RTG datasets.

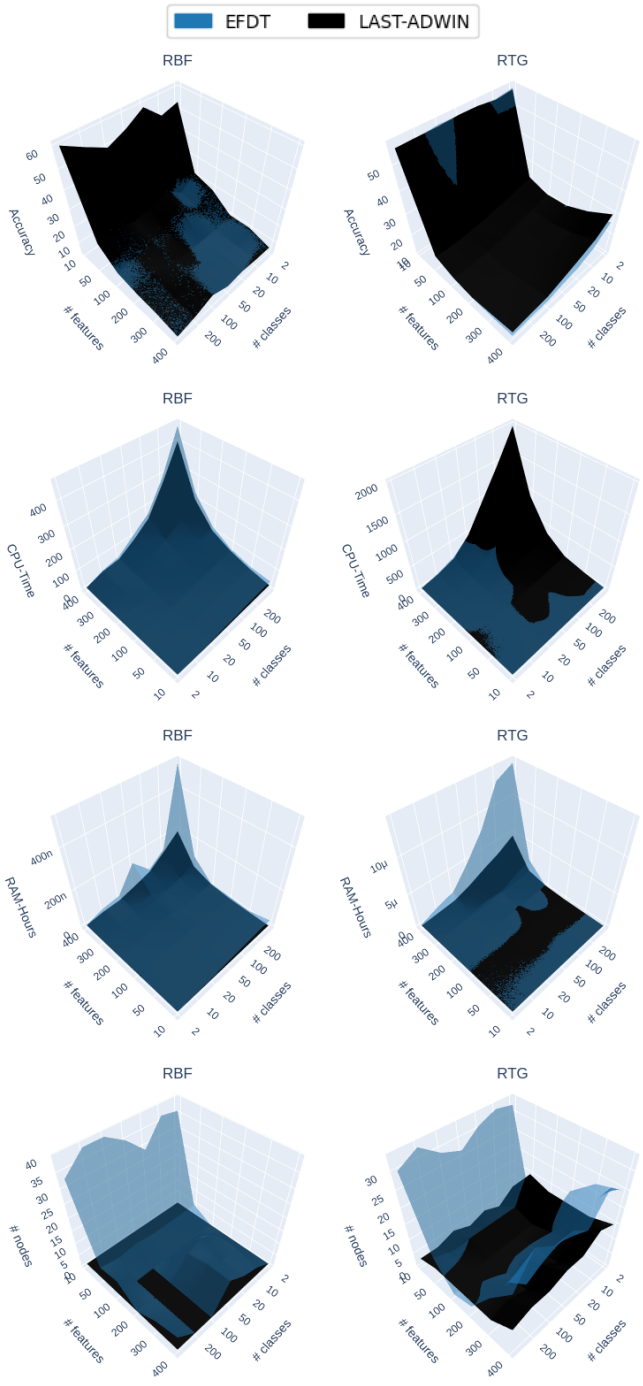

Fig. 12: EFDT and LAST with ADWIN accuracy (in %), CPU-Time (in seconds), RAM-Hours (in GB/h) and number of nodes in RBF and RTG datasets.

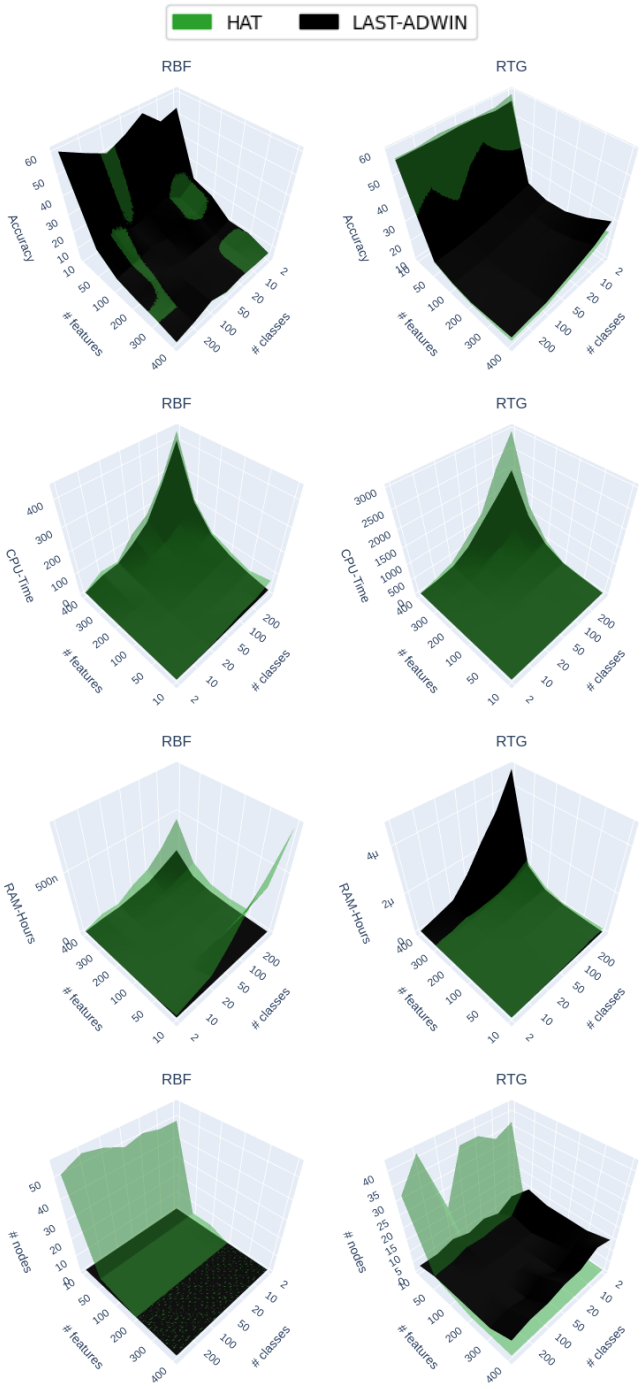

Fig. 13: HAT and LAST with ADWIN accuracy (in %), CPU-Time (in seconds), RAM-Hours (in GB/h) and number of nodes in RBF and RTG datasets.

Figure 14 shows the distribution of CPU-Time by varying the number of features in the RBF and RTG datasets. Algorithms are ordered by median, and the subset of features selected contains results from all the number of classes. In general, all methods presented a similar distribution. However, HAT and EFDT presented the highest in the first quartile, median, and third quartile.

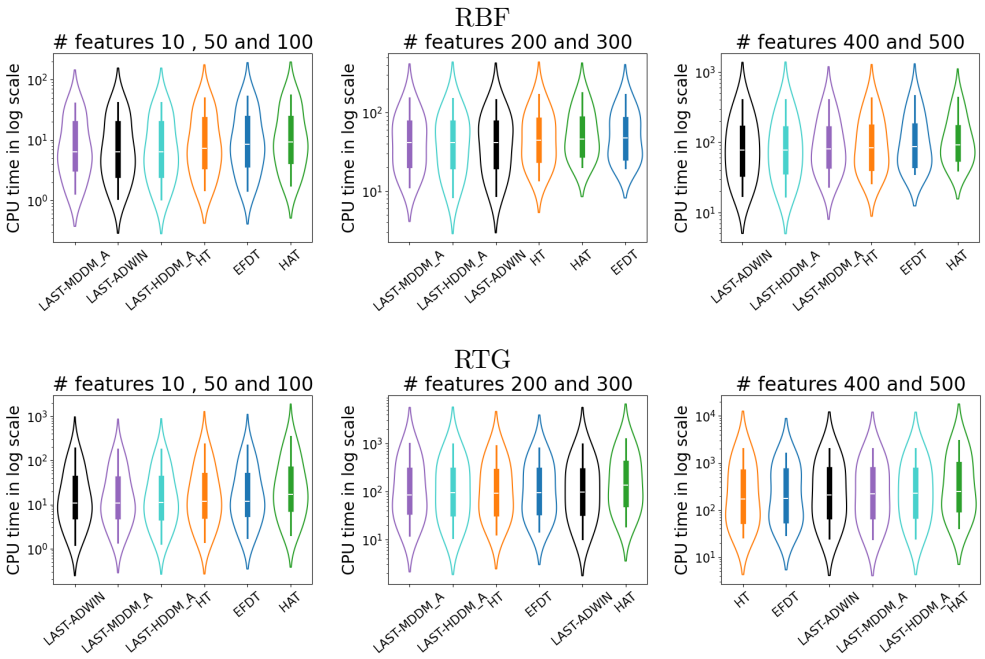

Fig. 14: Distribution of CPU-Time (in seconds and log scale) in RBF and RTG datasets.

Figure 15 shows the distribution of RAM-Hours by varying the number of features in the RBF and RTG datasets. Differently from CPU-Time, HAT presented a higher first quartile than most of the algorithm's median, being the method with the highest memory cost. EFDT presented a median and third quartile higher than that of the Hoeffding Tree. LAST, independently of the chosen detector, presented memory usage similar to that of Hoeffding Trees.

Answering question **RQ3**, the Hoeffding bound was more conservative as the number of features increased, while LAST did not present much variation in the number of nodes. The methods obtained similar CPU-Time, but methods with reevaluation of splits, such as EFDT and HAT, presented higher memory cost compared to Hoeffding Tree and LAST, as they need to store information about instances in non-terminal nodes. HAT presents the highest cost in memory since besides storing statistics about instances in non-terminal nodes, it needs to maintain change detection algorithms in non-terminal nodes.

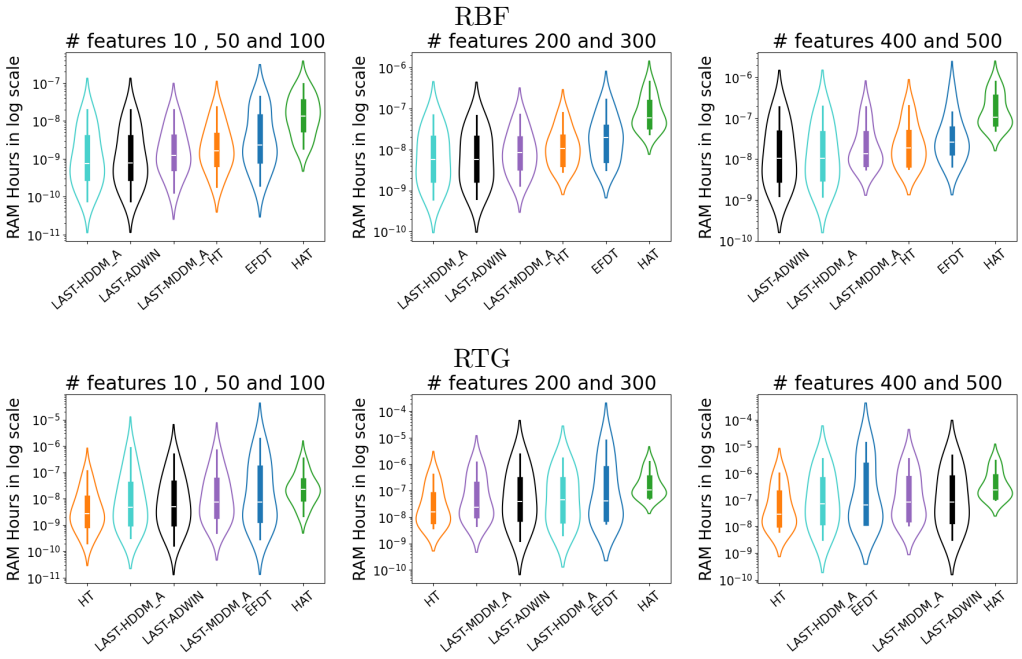

Fig. 15: Distribution of RAM-Hours (in GB/h and log scale) in RBF and RTG datasets.

4.5 Concept Drift

To address question **RQ4**, this Section presents the accuracy over time for datasets with concept drift. In all features, vertical lines in red represent the moment of concept drift in the dataset. We provide results by the native classifier, without the usage of explicit drift detectors to reset or replace the classifier by background learners, as recommended by [16].

Quoting from the authors: ***Retraining is not (always) the best option.*** *While this approach works well with ensembles due to their varied classifiers, our experiments have shown that when handling drifts that only affect specific boundaries or when the classifier is capable of self-adaptation, avoiding complete replacement of the established boundaries results in better performance. Furthermore, retraining the classifier for each identified drift may lead to redundant work due to the high occurrence of false alarms, even with state-of-the-art drift detectors.*

In datasets with abrupt change, with reoccurring change (Figure 16), or not (Figure 18), LAST performed better than HT and EFDT in most of the moments of the datasets. While in datasets with gradual change, with reoccurring change (Figure 17), or not (Figure 19), the difference in performance between HT, EFDT, and LAST is lower, especially in the AGR and SEA datasets. LAST the selected detectors are more efficient in abrupt changes, such that the method adapts better to the change when it stabilizes fast and splitting is enough to adapt to the change. Detectors that were

designed to deal better with gradual changes, such as EDDM, obtained higher mean accuracy throughout the stream than HT and EFDT. The paper repository makes available these results.

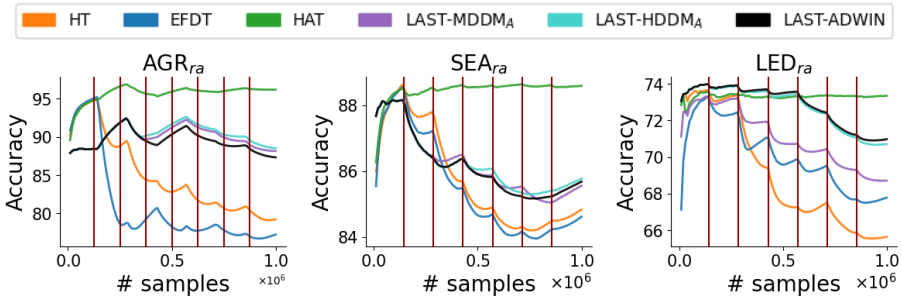

Fig. 16: Accuracy (%) of decision trees in datasets that simulate reoccurring abrupt changes.

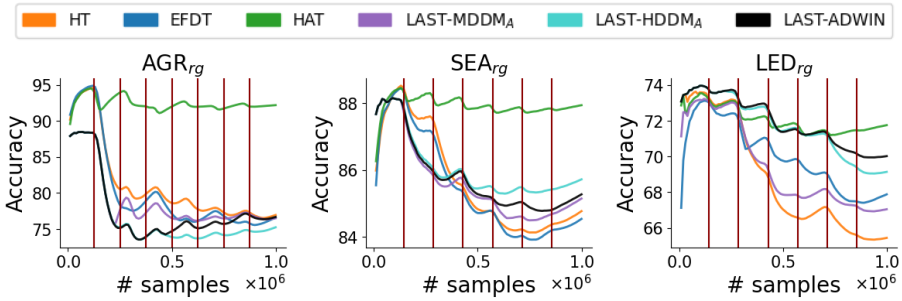

Fig. 17: Accuracy (%) of decision trees in datasets that simulate reoccurring gradual changes.

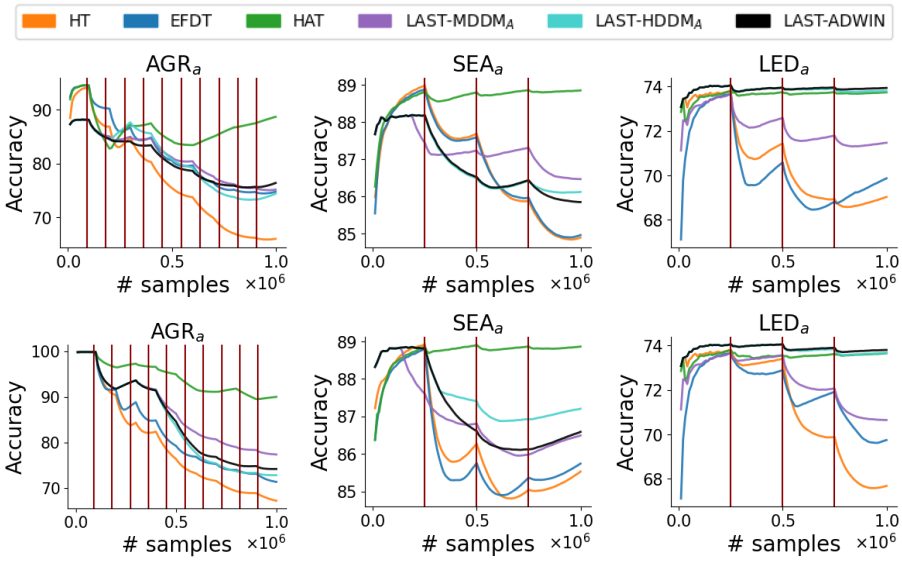

Fig. 18: Accuracy (%) of decision trees in datasets that simulate abrupt changes.

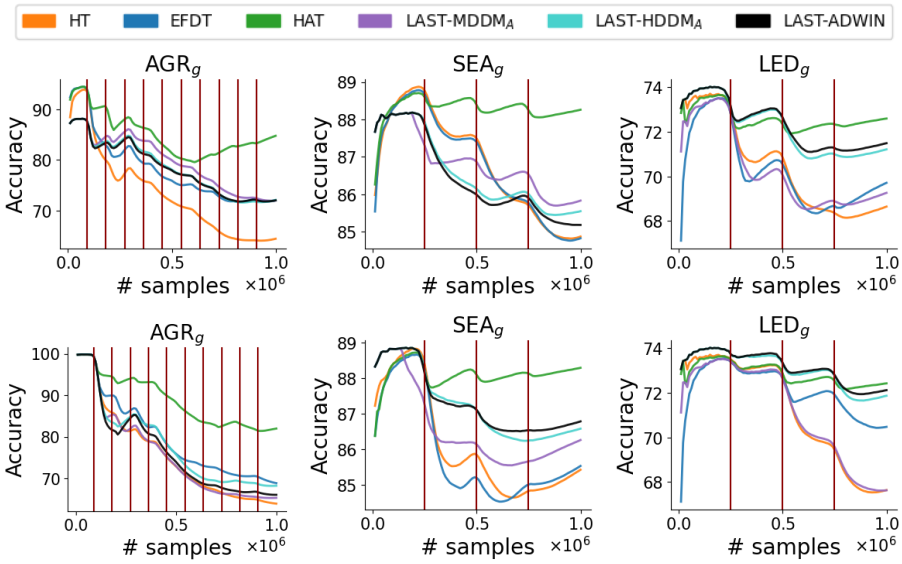

Fig. 19: Accuracy (%) of decision trees in datasets that simulate Reocurring gradual changes.

In datasets with incremental changes (Figures 20 and 21), EFDT and LAST with MDDM_A presented the best results in the RBF dataset with slow change, while LAST with detector HDDM_A and ADWIN obtained the worst results, and LAST, independently of the detector, obtained the best results in the HYPER dataset with slow change. In the RBF and HYPER datasets with moderate change, LAST, regardless of the detector, had superior results compared to HT, EFDT, and HAT. HT and EFDT were the only methods with performance decay. In the RBF dataset with fast change, LAST with detectors HDDM_A and MDDM_A obtained the best results, while HAT presented better results compared to LAST with ADWIN from instance 400000. HT and EFDT obtained the worst results throughout the stream. In the HYPER dataset with fast change, HAT and EFDT obtained the best results, while HT and LAST obtained the worst.

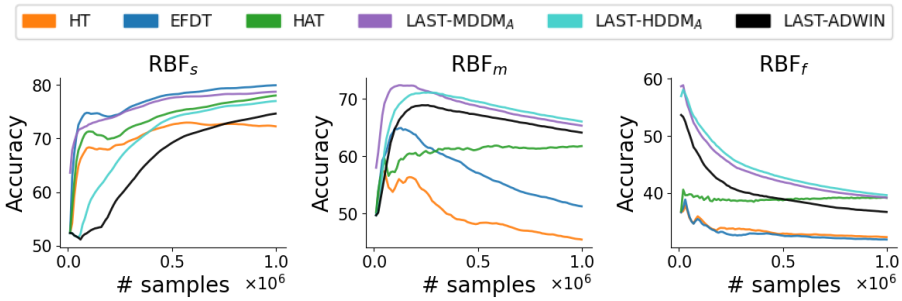

Fig. 20: Accuracy (%) of decision trees in the RBF dataset, which simulates incremental changes.

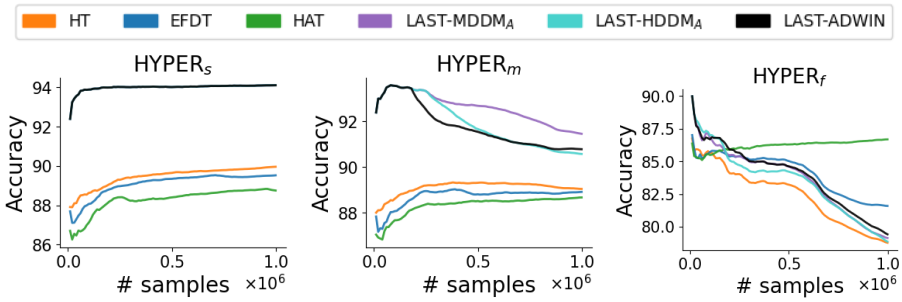

Fig. 21: Accuracy (%) of decision trees in the HYPER dataset, which simulates incremental changes.

In the INSECTS datasets (Figure 22), it is evident the disparity of performance between HAT results in real-world datasets with concept drift and synthetic data that simulate concept drift. Concept drifts simulated in the literature are usually used to

simulate addictive changes (addition or removal of relevant features in LED, or addition of concepts in AGR) or similar to the original (change in the parameter θ in SEA). The changes in the INSECTS datasets are the combination of real and virtual drifts, where the change in temperature affects the measurement of the insect wing beat by the sensor, resulting in changes in both $P_t(X)$ and $P_t(y|X)$, being more challenging than the concept drifts simulated in the literature. Therefore, HAT performance disparity in synthetic and real-world data shows that the reevaluation mechanism of HAT is the best for dealing with simulated changes. However, INSECTS depict a scenario in which HAT could not react to concept drifts. To the best of our knowledge, no study depicts this, even in works focused only on the concept drift problem such as [16].

In the INSECTS datasets with abrupt change, EFDT started with the results, until concept drift affected its performance, ending the stream with inferior performance to all LAST's with different detectors. LAST, with MDDM$_A$, obtained the best results, and LAST with detectors ADWIN and HDDM$_A$ had the second-best results. HAT had a performance decay (the second worst, while HT had the worst) after the first concept drift.

In the INSECTS datasets with gradual change, LAST, regardless of the detector used, had the lowest performance decay after the concept drift. Before the change, EFDT was the tree with the best performance, but after the change, it approached all LASTs with different detectors. HT and HAT presented the worst performance throughout the stream.

In the INSECTS dataset with incremental change, HT and HAT obtained the worst results after instance 7000, while other methods did not show performance decay. LAST with MDDM$_A$ and ADWIN presented the best results.

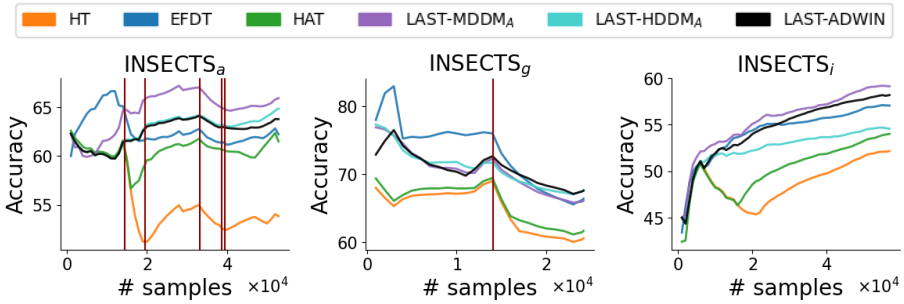

Fig. 22: Accuracy (%) of decision trees in the INSECTS dataset variants.

As an answer to **RQ4**, LAST presents superior results in datasets with abrupt changes compared to HT and EFDT. While in gradual changes, LAST with the selected detectors presented a lower difference in performance compared to HT and EFDT. Detectors designed to deal better with gradual changes, such as EDDM, presented higher accuracy. In datasets with incremental changes, LAST performs better in moderate changes. In other scenarios of incremental change, the best performance

varies among methods. We also pointed out HAT disparity in performance and reaction to concept drift between real-world and synthetic data. HAT splitting reevaluation mechanism is ideal for the additive concept drift changes simulated in common synthetic data for drift evaluation in the literature. However, we depict a case where HAT fails to react to concept drift in a challenging real-world scenario.

5 Conclusion

This paper provides a general discussion that attests to LAST performance in predictive quality and computational cost and depicts important characteristics of both Hoeffding-based trees and LAST. Section 4.2 presented a benchmark of online decision trees. LAST presented the best results among decision trees, regardless of the detector. A trade-off in predictive quality and computational cost was observed between methods that apply stricter concept drift detection constraints through statistical bounds and those that do not. Section 4.3 presented how splitting the tree affects its predictive quality. Hoeffding-based trees are mostly invariant to the tree performance. Splitting the tree in performance decay is why LAST obtained superior results compared to Hoeffding-based trees. In some cases, EFDT has shown moments where it could split the tree before a performance decay was observed; however, the number of occasions was less significant compared to the LAST reaction to performance decay through splitting. Section 4.4 shows the behavior of decision trees by varying the number of features and classes. EFDT softer Hoeffding bound has shown a higher number of nodes throughout the stream. Methods with reevaluation of splits, such as EFDT and HAT, have shown the worst results in memory usage, while LAST and HT presented similar results in CPU-time and RAM-hours. Section 4.4 presented the effect of concept drift in the performance of online decision trees. LAST with the selected detectors obtained superior results in datasets with abrupt change, while similar results were obtained in datasets with gradual change. The selection of the change detector in LAST is important in this case since detectors designed to deal better with gradual changes, such as EDDM, obtained superior results to HT and EFDT. We also discussed the disparity in the performance of HAT in real-world and synthetic datasets. HAT obtained good results since the HAT reevaluation mechanism deals well with changes in $P(y|X)$ of additive concepts and prunes the tree where performance decay is observed. Meanwhile, it did not obtain superior results in a real-world challenging scenario involving changes in both $P(X)$ and $P(y|X)$.

In future works, we aim to apply LAST to ensembles and benchmark the current state-of-the-art ensemble learning in streaming scenarios. LAST might contribute to additional adaptability to base learners of an ensemble. Due to the high sampling of instances, more changes in the distribution of data might happen, and this might influence LAST which monitors the class distribution purity (LAST_D) to create a diverse ensemble. Also, other scenarios could be interesting to adapt LAST, such as regression, delayed-labeling setting, active learning, semi-supervised learning, and many others.

References

[1] Ditzler, G., Roveri, M., Alippi, C., Polikar, R.: Learning in nonstationary environments: A survey. IEEE Computational Intelligence Magazine **10**(4), 12–25 (2015) https://doi.org/10.1109/MCI.2015.2471196

[2] Costa, V.G., Pedreira, C.E.: Recent advances in decision trees: an updated survey. Artif. Intell. Rev. **56**(5), 4765–4800 (2022) https://doi.org/10.1007/s10462-022-10275-5

[3] Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '00, pp. 71–80. Association for Computing Machinery, New York, NY, USA (2000). https://doi.org/10.1145/347090.347107 . https://doi.org/10.1145/347090.347107

[4] Bifet, A., Gavaldà, R.: Adaptive learning from evolving data streams. In: Adams, N.M., Robardet, C., Siebes, A., Boulicaut, J.-F. (eds.) Advances in Intelligent Data Analysis VIII, pp. 249–260. Springer, Berlin, Heidelberg (2009)

[5] Manapragada, C., Webb, G.I., Salehi, M.: Extremely fast decision tree. In: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. KDD '18, pp. 1953–1962. Association for Computing Machinery, New York, NY, USA (2018). https://doi.org/10.1145/3219819.3220005

[6] Krawczyk, B., Minku, L.L., Gama, J., Stefanowski, J., Woźniak, M.: Ensemble learning for data stream analysis: A survey, vol. 37, pp. 132–156 (2017). https://doi.org/10.1016/j.inffus.2017.02.004 . https://www.sciencedirect.com/science/article/pii/S1566253516302329

[7] Breiman, L.: Classification and regression trees. Wadsworth Statistics, Wadsworth, Belmont, CA, (1984)

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

[9] Pfahringer, B., Holmes, G., Kirkby, R.: New options for hoeffding trees. In: Orgun, M.A., Thornton, J. (eds.) AI 2007: Advances in Artificial Intelligence, pp. 90–99. Springer, Berlin, Heidelberg (2007)

[10] Jaworski, M., Duda, P., Rutkowski, L.: New splitting criteria for decision trees in stationary data streams. IEEE Transactions on Neural Networks and Learning Systems **29**(6), 2516–2529 (2018) https://doi.org/10.1109/TNNLS.2017.2698204

[11] De Rosa, R., Cesa-Bianchi, N.: Splitting with confidence in decision trees with

application to stream mining. In: 2015 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2015). https://doi.org/10.1109/IJCNN.2015.7280392

[12] Matuszyk, P., Krempl, G., Spiliopoulou, M.: Correcting the usage of the hoeffding inequality in stream mining. In: Tucker, A., Höppner, F., Siebes, A., Swift, S. (eds.) Advances in Intelligent Data Analysis XII, pp. 298–309. Springer, Berlin, Heidelberg (2013)

[13] Assis, D.N., Barddal, J.P., Enembreck, F.: Just change on change: Adaptive splitting time for decision trees in data stream classification. In: Proceedings of the Annual ACM Symposium on Applied Computing, SAC 2024 (2024)

[14] Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., Zhang, G.: Learning under concept drift: A review. IEEE transactions on knowledge and data engineering **31**(12), 2346–2363 (2018)

[15] Gomes, H.M., Bifet, A., Read, J., Barddal, J.P., Enembreck, F., Pfharinger, B., Holmes, G., Abdessalem, T.: Adaptive random forests for evolving data stream classification. Machine Learning **106**, 1469–1495 (2017)

[16] Aguiar, G.J., Cano, A.: A comprehensive analysis of concept drift locality in data streams. Knowledge-Based Systems **289**, 111535 (2024) https://doi.org/10.1016/j.knosys.2024.111535

[17] Gama, J.a., Rocha, R., Medas, P.: Accurate decision trees for mining high-speed data streams. KDD '03, pp. 523–528. Association for Computing Machinery, New York, NY, USA (2003). https://doi.org/10.1145/956750.956813 . https://doi.org/10.1145/956750.956813

[18] Holmes, G., Kirkby, R., Pfahringer, B.: Stress-testing hoeffding trees. In: Jorge, A.M., Torgo, L., Brazdil, P., Camacho, R., Gama, J. (eds.) Knowledge Discovery in Databases: PKDD 2005, pp. 495–502. Springer, Berlin, Heidelberg (2005)

[19] Buntine, W.: Learning classification trees. Statistics and Computing **2**, 63–73 (1992)

[20] Kohavi, R., Kunz, C.: Option decision trees with majority votes. Proceedings of the Fourteenth International Conference on Machine Learning (1998)

[21] Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing, vol. 7 (2007). https://doi.org/10.1137/1.9781611972771.42

[22] Heyden, M., Gomes, H.M., Fouché, E., Pfahringer, B., Böhm, K.: Leveraging plasticity in incremental decision trees. In: Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2024, Vilnius, Lithuania, September 9–13, 2024, Proceedings, Part V, pp. 38–54.

Springer, Berlin, Heidelberg (2024). https://doi.org/10.1007/978-3-031-70362-1_3

[23] Montiel, J., Halford, M., Mastelini, S.M., Bolmier, G., Sourty, R., Vaysse, R., Zouitine, A., Gomes, H.M., Read, J., Abdessalem, T., Bifet, A.: River: machine learning for streaming data in python. J. Mach. Learn. Res. **22**(1) (2021)

[24] Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., Seidl, T.: Moa: Massive online analysis, a framework for stream classification and clustering. In: Proceedings of the First Workshop on Applications of Pattern Analysis, pp. 44–50 (2010). PMLR

[25] Gama, J., Sebastiao, R., Rodrigues, P.P.: On evaluating stream learning algorithms. Machine Learning **90**, 317–346 (2013)

[26] Gama, J., Medas, P., Castillo, G., Rodrigues, P.: Learning with drift detection. In: Bazzan, A.L.C., Labidi, S. (eds.) Advances in Artificial Intelligence – SBIA 2004, pp. 286–295. Springer, Berlin, Heidelberg (2004)

[27] Baena-Garcıa, M., Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavalda, R., Morales-Bueno, R.: Early drift detection method. In: Fourth International Workshop on Knowledge Discovery from Data Streams, vol. 6, pp. 77–86 (2006). Citeseer

[28] Frías-Blanco, I., Campo-Ávila, J.d., Ramos-Jiménez, G., Morales-Bueno, R., Ortiz-Díaz, A., Caballero-Mota, Y.: Online and non-parametric drift detection methods based on hoeffding's bounds. IEEE Transactions on Knowledge and Data Engineering **27**(3), 810–823 (2015) https://doi.org/10.1109/TKDE.2014.2345382

[29] Barros, R.S.M., Cabral, D.R.L., Gonçalves, P.M., Santos, S.G.T.C.: Rddm: Reactive drift detection method. Expert Systems with Applications **90**, 344–355 (2017) https://doi.org/10.1016/j.eswa.2017.08.023

[30] Pesaranghader, A., Viktor, H.L., Paquet, E.: Mcdiarmid drift detection methods for evolving data streams. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–9 (2018). https://doi.org/10.1109/IJCNN.2018.8489260

[31] Souza, V.M., Reis, D.M., Maletzke, A.G., Batista, G.E.: Challenges in benchmarking stream learning algorithms with real-world data. Data Mining and Knowledge Discovery **34**, 1805–1858 (2020)

[32] García, S., Herrera, F.: An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. Journal of Machine Learning Research **9**(89), 2677–2694 (2008)

[33] Benavoli, A., Corani, G., Mangili, F.: Should we really use post-hoc tests based on mean-ranks? Journal of Machine Learning Research **17**(5), 1–10 (2016)

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

[35] Manapragada, C., Webb, G.I., Salehi, M., Bifet, A.: Emergent and Unspecified Behaviors in Streaming Decision Trees (2020). https://arxiv.org/abs/2010.08199

[36] Bengio, Y., Delalleau, O., Simard, C.: Decision trees do not generalize to new variations. Computational Intelligence **26**(4), 449–467 (2010) https://doi.org/10.1111/j.1467-8640.2010.00366.x