Adaptive Options for Decision Trees in Evolving Data Stream Classification

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

Programa de Pós-Graduação em Informática (PPGIa), Pontifícia Universidade Católica do Paraná. Curitiba, Paraná, Brazil {daniel.nassis, jean.barddal, fabricio}@ppgia.pucpr.br

Abstract. Decision trees are fundamental components of data stream mining frameworks and pipelines. However, their inherent instability - where small variations in training data can lead to significant structural changes- has motivated research into methods that either (i) mitigate this instability or (ii) exploit it for improved performance. Option trees provide an alternative approach to instability reduction by allowing non-leaf nodes to have multiple subtrees as child nodes. This enables instances to traverse multiple paths within a single decision tree structure, offering greater processing time and memory efficiency compared to ensemble methods—key advantages for streaming data mining, where data arrives continuously and potentially without bounds. This paper introduces LASTO, an algorithm with adaptive mechanisms for splitting and dynamically adding option nodes. Our primary contribution lies in the option node addition mechanism, where change detectors monitor branch performance and introduce option nodes when a decline in predictive quality is observed. An option node is only added if the split gain surpasses that of the previous split, ensuring its necessity and effectiveness. Experimental results demonstrate that LASTO achieves statistically significant differences in predictive performance while maintaining computational efficiency comparable to state-of-the-art decision trees for data stream classification.

Keywords: Decision Tree · Data Stream · Option Nodes

1 Introduction

Streaming data is integral to today's evolving digital landscape, and mining its concepts can reveal valuable insights. Researchers and practitioners in data stream mining focus on developing machine learning algorithms capable of efficiently processing high-speed, continuous, and potentially infinite data. These algorithms must provide high predictive quality and ensure anytime response, fast processing, and low memory consumption to prevent excessive instance storage. Otherwise, memory limitations or the need to discard instances could overwhelm the system and lead to failure [1].

Another fundamental challenge in streaming data mining is its evolving nature. Unlike traditional machine learning settings, the assumption that data

distributions remain stable over time does not hold. This phenomenon, known as concept drift [2], refers to changes in the probability distribution of data over time. Concept drift can severely impact predictive performance, making it essential for machine learning models to detect and adapt to these changes promptly to maintain reliable learning outcomes. Hoeffding Trees [3] offer an efficient approach to stream classification, resembling batch decision tree algorithms such as C4.5 [4] and CART [5]. These trees continuously learn and predict from incoming instances, performing periodic split evaluations based on the Hoeffding Theorem [6]. This theorem enables incremental learning by distributing the computational cost of evaluating splits at leaf nodes across the data stream. As a result, Hoeffding Trees are among the most widely used and efficient methods for mining streaming data. Several studies, such as [7] and [8], have extended Hoeffding Trees, demonstrating improved predictive performance over the standard approach. Among these, we highlight the Local Adaptive Streaming Tree (LAST) [9], which addresses a key limitation of Hoeffding-based trees: their static split evaluation, which does not account for the evolving state of the tree over time. LAST introduces change detectors [2] at leaf nodes to monitor their statistics dynamically and determine optimal split moments. Experimental results have shown that LAST not only achieves competitive predictive performance against state-of-the-art decision trees but also improves processing efficiency.

Decision trees, however, suffer from limited lookahead ability, making them inherently unstable. Since they make splitting decisions based only on local statistics, they may not always select the best long-term splits [10]. Ensemble methods mitigate this instability by combining multiple decision trees into a single, stronger learner [11]. While ensembles are state-of-the-art in many applications, they introduce significant computational costs due to the need to train and maintain multiple base models—making them less suitable for data stream mining systems with strict memory and time constraints. An alternative to ensembles is Option Trees [12] [13], which reduce decision tree instability while maintaining a single tree structure. Option Trees introduce option nodes, which allow instances to follow multiple decision paths simultaneously. This design enhances predictive robustness without the overhead of training multiple trees. In [10], the authors extended Hoeffding Trees to incorporate option nodes. In this paper, we propose Local Adaptive Streaming Tree with Options (LASTO), a novel approach that integrates adaptivity in both the splitting mechanism and the addition of option nodes. Using change detection algorithms, LASTO continuously monitors node statistics to dynamically determine when to introduce new option nodes, further enhancing model stability and predictive performance.

This paper is structured as follows: Section 2 discusses Hoeffding-based Trees and LAST. Section 3 introduces our proposed method. Section 4 presents experimental results. Finally, Section 5 concludes the paper and outlines future research directions.

2 Related Works on Decision Trees for Data Streams

In this section, we bring forward existing works on decision trees focusing on Hoeffding Trees and the Local Adaptive Streaming Tree (LAST).

2.1 Hoeffding Trees

Hoeffding Trees [3] are incremental and online decision trees designed for classification problems. The tree structure is periodically updated with split attempts when the number of samples observed in a leaf node achieves a user-given parameter called *grace period* (GP). For instance, if GP=50, a split attempt will occur when a leaf node observes $50, 100, 150, 200, \ldots$ samples until a split ensues.

A split occurs if it passes the Hoeffding-bound constraint. Given a level of confidence δ (user-given), a split occurs if:

$$\Delta G(X_a) - \Delta G(X_b) \geq \sqrt{\frac{R^2 \log(\frac{1}{\delta})}{2n}} \tag{1}$$

where $\Delta G(\cdot)$ is the impurity measure applied, (such as gini index [5] or information gain[4]), X_a is the feature that maximizes $\Delta G(\cdot)$, X_b is the feature that presents second best $\Delta G(\cdot)$ value, R is the range of $\Delta G(\cdot)$ function and n is the number of samples in the leaf nodes before splitting.

Figure 1 illustrates the training process of Hoeffding Trees, where ϵ is the Hoeffding bound.

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

In [14], the authors observed that Naive Bayes could enhance predictive quality at leaf nodes. Authors in [15] noticed that selecting Naive Bayes or majority class strategy prediction according to the method with the highest accuracy at the leaf could further enhance Hoeffding Trees, which is commonly used.

2.2 Hoeffding Adaptive Tree

Hoeffding Adaptive Tree (HAT) [7] extend the Hoeffding Tree algorithm by adding the ADaptive WINdowing (ADWIN) [16] change detection algorithm to each non-terminal node. ADWIN [16] is a widely used change detector that maintains a window W of recent observations and attests that no change is present in the window if the mean value of subwindows of W has a similar value according to Hoeffding's theorem. If ADWIN flags a change in the accuracy of the instances that traverse the tree, a new subtree is created and replaces the node and its branch if the subtree is more accurate.

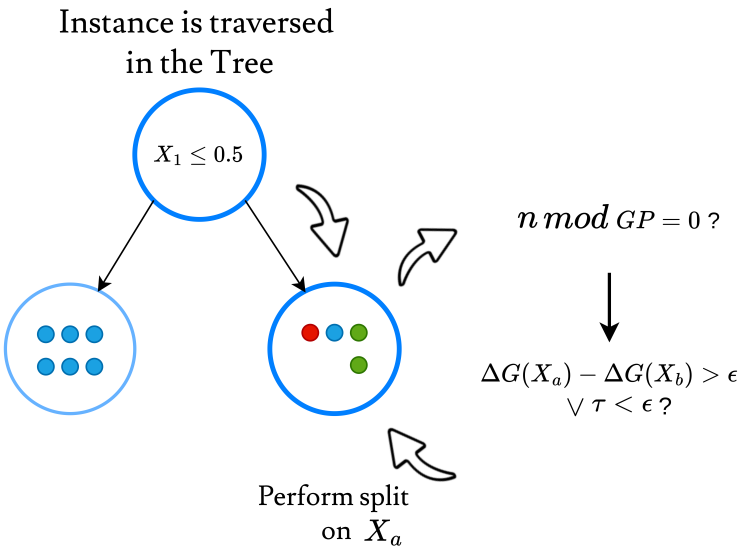

Fig. 1. Hoeffding Tree training process for an instance

2.3 Extremely Fast Decision Tree

In [8], the authors propose the Extremely Fast Decision Tree (EFDT). The first difference between EFDT and Hoeffding Trees is that EFDT applies a softer split constraint. 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 $\Delta G(X_a) - \Delta G(X_\emptyset) \geq \epsilon \equiv \Delta G(X_a) \geq \epsilon$, thus resulting in deeper trees.

Similar to HAT, EFDT has a split reevaluation mechanism. This process is similar to the split attempt. A split reevaluation is performed at each GP_2 (user-given) sample traversed in non-terminal nodes. Given X_c, the feature used for splitting at the non-terminal node before the reevaluation and $\Delta G(X_c)$ the impurity at the non-terminal node, the sub-tree and its child nodes are substituted for a newly evaluated split on X_a if:

$$\Delta G(X_a) - \Delta G(X_c) \geq \epsilon \tag{2}$$

In other words, if a new split is more beneficial than the previous one done at the terminal node, the new split replaces the sub-tree and its child nodes.

2.4 Local Adaptive Streaming Trees

The trees described earlier perform a static and periodic evaluation of splits. A split attempt will only occur at every GP samples observed at a leaf node, but

a change in the purity or accuracy of the leaf node may occur earlier, and the tree would be unable to anticipate it. Even when small changes occur in a leaf node, the greedy evaluation of attributes and their values that compose the best split will still be performed and might not result in significant changes in the tree and consume more processing time.

Depicting these points, the authors in [9] propose the Local Adaptive Streaming Tree (LAST) algorithm. At each leaf node, LAST maintains a change detector [2] that constantly monitors leaf node statistics to determine ideal moments for splitting. Change detectors are often tailored to handle binomial distributions, i.e., error rates. If a change detector triggers a change, a split will ensue if $\Delta G(X_a) > 0.0$, the softest split constraint possible, which means change detectors control how the tree grows. Since change detectors constantly monitor new upcoming instances, they track how the stream evolves on an instance basis rather than in chunks.

Figure 2 illustrates the training process of LAST.

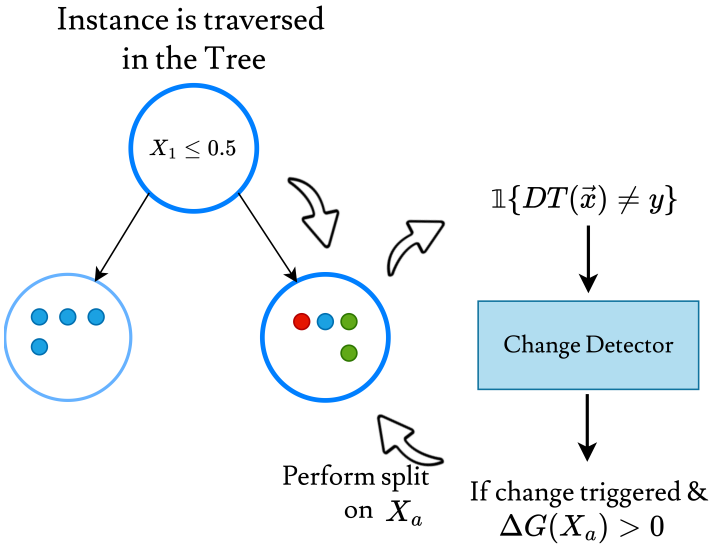

Fig. 2. LAST training process for an instance

2.5 Hoeffding Option Trees

Option nodes let an instance traverse more than one decision path. If an instance reaches an option node, it traverses all the child node paths. For example, if the instance reaches the option node in the tree in Figure 4, it will follow all paths to the child nodes $\{X_3 \leq 3.7, X_2 \leq 9\}$.

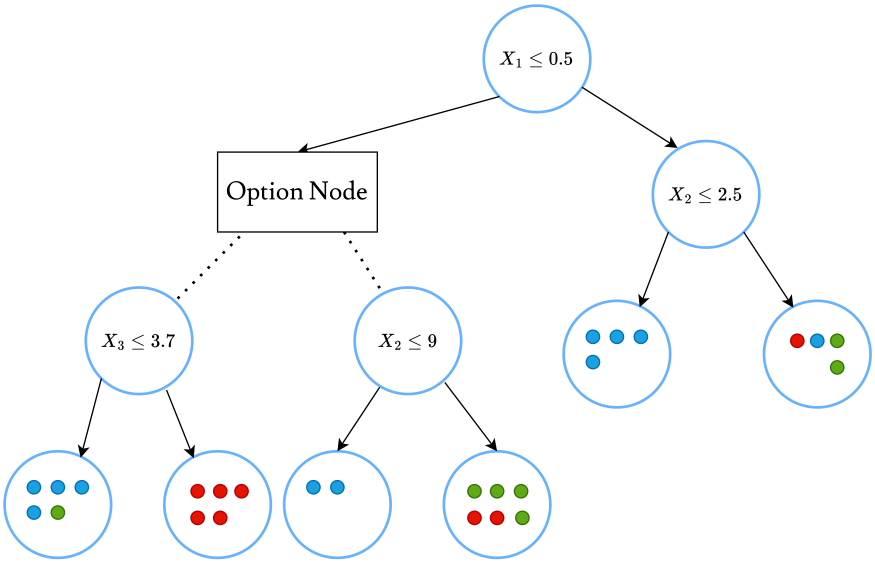

Fig. 3. An Option Tree

In the batch setting, option nodes are created in the training phase with restrictions, for instance, the maximum number of child nodes in option nodes [13]. For voting, since an instance can arrive in more than one leaf node, it is possible to perform majority voting, as in [13], or weighted voting, as in [12], such that the probabilities of each class at each leaf node are summed.

The authors in [10] proposed constraining the maximum number of child nodes and applying weighted voting instead of a simple majority voting scheme. The authors also propose adding child nodes for option nodes that split on features not previously used in the option node children.

Since the tree induction is incremental, adding child nodes to option nodes is also incremental. The authors use a similar approach to the splitting phase. Given a feature X_{max}, the feature with a maximum value of $\Delta G(\cdot)$ in the option node and X_{cand} the candidate feature for splitting in a new child node, a new child node split in X_{cand} is added if pass the constraint in equation 3.

$$\Delta G(X_{cand}) - \Delta G(X_{max}) \geq \sqrt{\frac{R^2 \log(\frac{1}{\delta'})}{2n}} \tag{3}$$

In this case, the authors apply a softer confidence $1-\delta' = 0.955$ since the one used in the splitting [3] is $\delta = 10^{-8}$ with confidence $(1-\delta) = 0.99\ldots$, avoiding the creation of few option nodes.

3 Local Adaptive Streaming Tree with Options (LASTO)

In this section, we introduce our proposal called *Local Adaptive Streaming Tree with Options* (LASTO). Compared to traditional Hoeffding Option Trees [10], our approach presents two distinctive characteristics: (i) the adaptive addition of option nodes and (ii) the determination of split attempt moments via adaptive leaf statistics monitoring following the work of [9].

As depicted in [9], the periodic split attempt cannot anticipate changes in the accuracy of the leaf node between two split attempts, and even when little change occurred in the leaf node, the greedy evaluation of attributes and their values that compose the best split will still be performed and might not result in a significant tree change and consume more processing time. The same follows for adding option nodes in non-leaf nodes in [10]. Hoeffding Option Tree cannot anticipate decays in performance between two periodic evaluations for adding new option nodes, and the periodic review of split nodes is costly since multiple nodes will perform split evaluations constantly. In contrast, our algorithm performs split evaluations only in moments where a decay in performance is observable according to the change detector.

Algorithm 1 presents LASTO pseudocode. Lines 1-19 present the LAST algorithm's adaptive splitting mechanism performed at leaf nodes. Lines 19-33 show the adaptive addition of new option nodes for non-terminal nodes of the tree. For the addition of option nodes, since the change detector determines the moment of node addition to the option node and applies a hard constraint for detecting a change in accuracy, we use a softer constraint for the addition of the option node (Algorithm 1, line 23). In this case, the gain of a new split must result in a value superior to the maximum gain of the nodes in the option node, justifying the addition of a new node to the option node.

Figure 4 illustrates the LASTO training process, highlighting the paper's main contribution. Specifically, the figure demonstrates the addition of option nodes when a change in accuracy is detected in the subsequent nodes under a decision node. Additionally, it incorporates the information gain constraint, ensuring that an option node is added only if its information gain exceeds the maximum gain among existing option nodes. In the case of Figure 4, this condition is represented as $\Delta G(X_a) > \Delta G(X_5)$, since there is only one option node in the decision node $\{X_5 \leq 3.2\}$.

Unlike in [10], as the change detectors take into account the performance of the tree, parts of the tree where the performance is decaying, and the split decision at these parts turn out to be not the best option in the long term (instability), the addition of option nodes can mitigate the instability of the tree by aggregating the predictions with more options nodes and decision paths.

One of the method's main advantages is that the user does not have to specify the *Grace Period*, τ, δ, and δ' hyperparameters. These are crucial hyperparameters for the algorithm's performance, and the optimal value of these parameters can differ in multiple scenarios. Change detectors can also have hyperparameters, but some are well-established regarding change detection, and some have no hyperparameters.

Algorithm 1 LASTO

Input: S: a data stream, X: a set of attributes, $\Delta G(\cdot)$: a split evaluation function, *maxOptions*: the maximum number of options reachable by a single example, ψ: Change detector used in nodes.

for each sample $(\boldsymbol{x}, y) \in S$ **do**
  Sort $\boldsymbol{x}$ into option nodes L using LASTO
  for all option nodes l of the set L **do**
    Update l statistics using $(\boldsymbol{x}, y)$
    $n_l \leftarrow n_l + 1$
    Update l_ψ with $\mathbb{1}\{DT(\boldsymbol{x}) \neq y\}$
    if l_ψ detected change $\wedge$ $\neg$(l contains samples from only one class) **then**
      if l has no children **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 $(\Delta G(X_a) > 0.0)$ **then**
          Add nodes below l that split on X_a
          for each leaf node l_i from splitting on X_a **do**
            Let $n_{l_i} \leftarrow 0$
            $l_{i\psi} \leftarrow \psi$
          end for
        end if
      else
        if l.optionCount $<$ *maxOptions* **then**
          Compute $\Delta G(X_i)$ for existing splits and (non-used) attributes
          Let X_{max} be existing child split with highest ΔG
          Let X_{cand} be (non-used) attribute with highest ΔG
          if $\Delta G(X_{cand}) > \Delta G(X_{max})$ **then**
            Add an additional child option to l that splits on X_{cand}
            for each leaf node l_i from splitting on X_{cand} **do**
              Let $n_{l_i} \leftarrow 0$
              $l_{i\psi} \leftarrow \psi$
            end for
          end if
        else
          Remove attribute statistics stored at l
        end if
      end if
    end if
  end for
end for

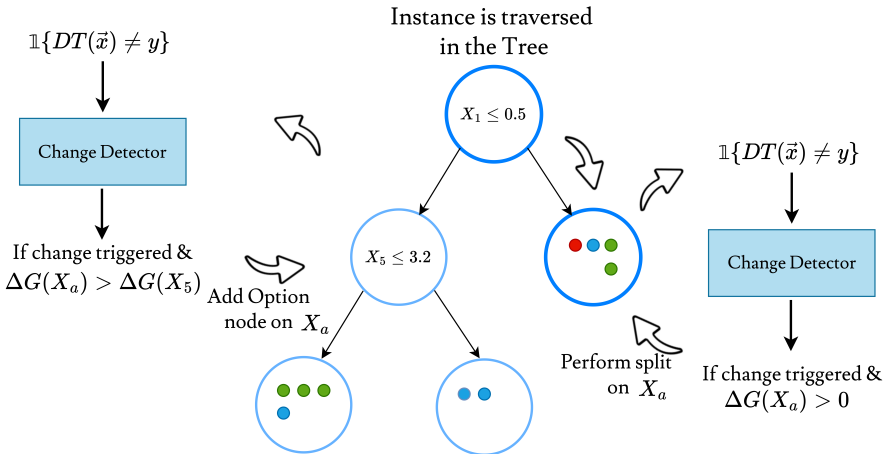

Fig. 4. LASTO training process for an instance

As in [10], we maintain the weighted voting scheme if an instance traverses more than one path and set the maximum number of option nodes to five.

4 Experiments and Results

This section introduces the experimental protocol adopted, followed by the results obtained and discussion.

4.1 Experimental protocol

All algorithms were implemented in the Massive Online Analysis (MOA) framework [17]. All experiments were done in an Intel(R) Xeon(R) CPU E5649 @ 2.53GHz with 32 GB of RAM.

The code of the proposal is available in [1] for the MOA framework.

All the Hoeffding Tree-based algorithms evaluated had default hyperparameters from MOA (*Grace Period* = 200, level of confidence = 10^{-7}, impurity measure is information gain). LAST and LASTO had ADWIN [16] as change detectors with default hyperparameters.

We assess the predictive performance using accuracy computed in a prequential validation strategy, where every instance is used first for testing and then for training. Additionally, we measured computational resource usage in terms of CPU-Time (in seconds), RAM-Hours (GB/h), and tree size (number of nodes). CPU-Time reports the full experiment duration, while RAM-Hours and tree size reflect the peak values observed throughout the stream.

[1] https://sites.google.com/view/lasto-paper

Experiments were held with 21 datasets reported in Table 1, with twelve real-world and nine synthetic datasets. The synthetic datasets and parameters used are discussed as follows.

LED [5] This generator produces 24 boolean features, while 17 are irrelevant. Each feature has a 10% of being inverted, simulating noise. We simulated three abrupt (LED_a) and gradual (LED_g) drifts.

SEA [19] 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 θ values.

AGRAWAL [18] This generator has six nominal 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 [17] This generator produces ten features and 5 class values. Data is generated based on the radial basis function (RBF). Centroids are generated randomly and mapped with a standard deviation value, a weight, and a class label. In this dataset, incremental drifts are simulated by continuously changing the centroids' position. The parameters used were 50 centroids at a speed change of 10^{-4} (moderate, RBF_m) and 10^{-3} (fast, RBF_f).

HYPER [20] 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 set up with 10 features and a magnitude of change of 10^{-3}.

The real-world datasets used were Outdoor, Rialto, Airlines, CovType, Nomao, Poker, NOAA, and three versions of the INSECTS dataset with abrupt, gradual, and incremental drifts, respectively. All real-world datasets were collected from [21].

4.2 Discussion

Table 2 presents the prequential accuracy of the decision trees evaluated, where bold values indicate the best result per dataset, and Figure 5 provides a critical distance plot of a pairwise one-sided Wilcoxon signed-rank tests with $\alpha = 0.1$ and form cliques using the Holm correction for multiple testing as performed in [22, 23]. LASTO has been shown to improve the results of LAST, as it presented the best ranking overall and statistical difference to all methods. LASTO presented the best ranking in real-world datasets and the second-best ranking in synthetic datasets. Hoeffding Adaptive Trees (HAT) could present the best results in synthetic datasets but could not outperform EFDT in real-world datasets and show no statistical difference to EFDT.

Figure 6 shows a comparison between LASTO and HAT across the evaluated datasets in this work in a scatter plot, while Figure 7 presents a comparison between LASTO and LAST. LASTO had 14 wins against HAT, and the difference

Table 1. Description of the evaluated datasets.

Dataset	# examples	# features	# classes	Majority class (%)
LED_a	1,000,000	24	10	10.28
LED_g	1,000,000	24	10	10.28
SEA_a	1,000,000	3	2	59.91
SEA_g	1,000,000	3	2	59.91
AGR_a	1,000,000	9	2	52.83
AGR_g	1,000,000	9	2	52.83
RBF_m	1,000,000	10	5	30.01
RBF_f	1,000,000	10	5	30.01
HYPER	1,000,000	10	2	50
Outdoor	4,000	21	40	4.11
Rialto	82,250	27	10	10
Airlines	539,383	7	2	55.47
CoverType	581,012	54	7	48.75
Nomao	34,465	119	2	71.44
Poker	829,201	10	10	47.78
NOAA	18,158	8	2	69.74
$INSECTS_a$	52,848	33	6	16.07
$INSECTS_i$	57,018	33	6	11.56
$INSECTS_g$	24,150	33	6	15.76
LADPU	22,950	96	10	10
Asfault	8,066	62	5	55.59

in accuracy is more noticeable than LASTO against LAST. LASTO had 17 wins against LAST.

Figures 8-11 present the accuracy throughout time for the datasets LED_a, LED_g, $INSECTS_a$ and $INSECTS_g$, respectively. In the $INSECTS_a$ dataset, HAT showed a significant decrease in accuracy, while LAST and LASTO achieved stabler and higher accuracy. In the third drift of the dataset, LAST had an accuracy decrease bigger than LASTO, and LASTO ended as the most accurate classifier. In the $INSECTS_g$ dataset, HAT presented an accuracy lower than LAST and LASTO, but after the drift, HAT had an accuracy decrease greater than LAST and LASTO. In the LED_a dataset, all methods reacted similarly to drifts, but HAT presented lower accuracy compared to LAST and LASTO. In the LED_g dataset, the methods presented similar behavior, and HAT presented lower accuracy compared to LAST and LASTO until the third drift occurred, and LAST had a big drop in accuracy, while HAT and LASTO remained stable.

Table 3 shows the tree size of decision trees. LASTO presented the worst tree size ranking but presented a lower tree size ranking compared to HT and EFDT in synthetic datasets. In some real-world datasets, LASTO presents smaller tree sizes and higher accuracy than EFDT, such as in Asphalt and $INSECTS_i$. In synthetic data, with the exception of the RBF *dataset*, LASTO had a smaller tree size while achieving higher accuracy. RBF is a dataset that incremental changes occur, such that the detector flags changes recurrently because a change includes multiple intermediary concepts, affecting both LAST and LASTO high complexity in this dataset. Regularization pre-prunining techniques to avoid high complexity [24] or simply a stricter splitting condition that guarantees the split with maximized gain is indeed better than the other possible splits in LAST and LASTO could come to good use.

Figures 12 and 13 present box plots for CPU-Time (in seconds and log scale), and the methods were ordered by the median value. LASTO presented a higher median and upper quartile compared to other methods. In real-world data, LASTO also presented a higher lower quartile greater than other methods, while in synthetic data, HAT was the only method that presented a greater lower quartile compared to LASTO. No more than one clique was identified in a pairwise Wilcoxon signed-rank test with a Holm correction. One must be careful in analyzing log-scaled box plots, as values are small, and slight changes are still feasible as efficient learners for mining data streams.

Figures 14 and 15 present box plots for RAM-Hours (in GB/h and log scale), and the methods were ordered by the median value. In real-world data, LASTO presented the highest lower quartile, median, and upper quartile. In synthetic data, LASTO presented lower quartile, median, and upper quartiles that were greater than HOT and HT. No more than one clique was identified in a pairwise Wilcoxon signed-rank test with a Holm correction.

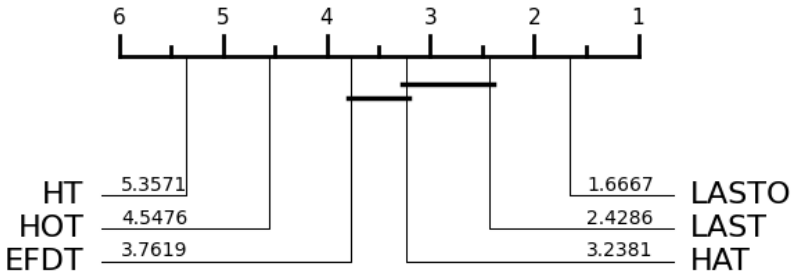

Fig. 5. Critical distance on accuracy with Wilcoxon signed-rank test and $\alpha = 0.01$.

5 Conclusions

In this work, we propose the *Local Adaptive Streaming Tree with Options (LASTO)* algorithm. LASTO extends the LAST algorithm to allow the creation of option nodes throughout the stream, and the main contribution of this paper is the adaptive creation of option nodes through change detection algorithms. LASTO further leverages the results of LAST and outperforms state-of-the-art decision tree algorithms. LASTO presented higher computational costs in tree size ranks in real-world datasets and lower quartiles, medians, and upper quartiles in CPU-Time and RAM-Hours than most methods, yet no statistical difference in computational cost is observed.

In future works, we plan to propose a tree with an adaptive split reevaluation, as in HAT, that can replace branches with decreasing accuracy in the tree with

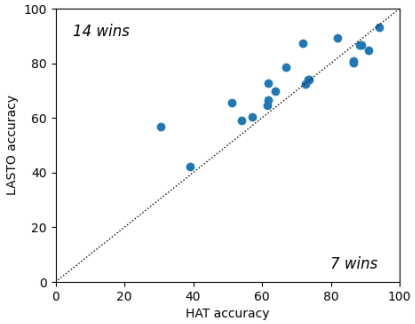

Fig. 6. Comparison of LASTO and HAT accuracy.

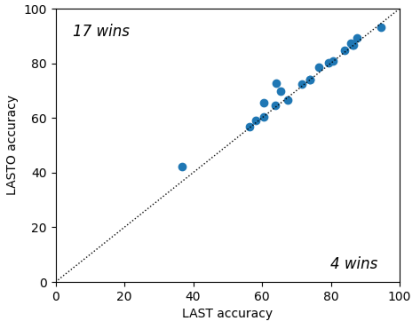

Fig. 7. Comparison of LASTO and LAST accuracy.

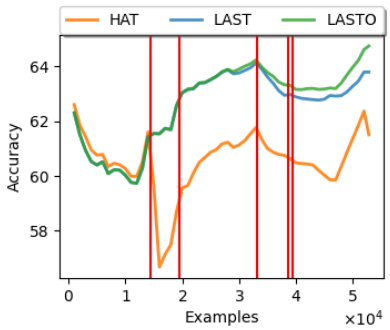

Fig. 8. Accuracy over time in the INSECTS$_a$ dataset.

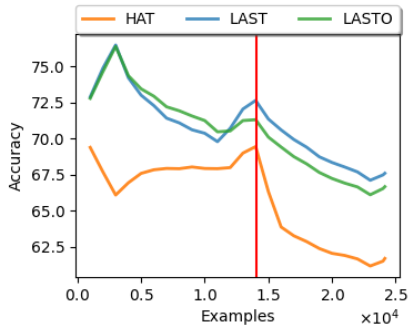

Fig. 9. Accuracy over time in the INSECTS$_g$ dataset.

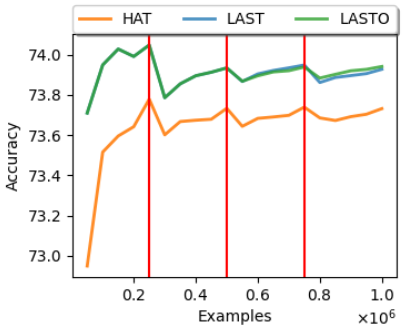

Fig. 10. Accuracy over time in the LED$_a$ dataset.

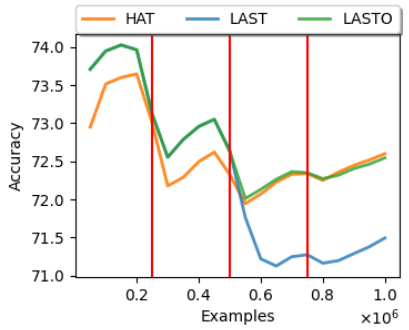

Fig. 11. Accuracy over time in the LED$_g$ dataset.

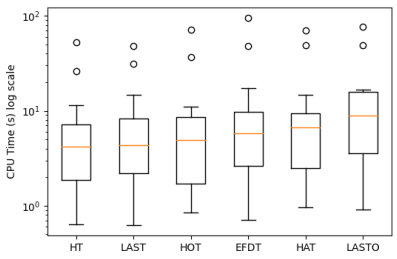

Fig. 12. CPU-Time (seconds, log scale) in real-world datasets.

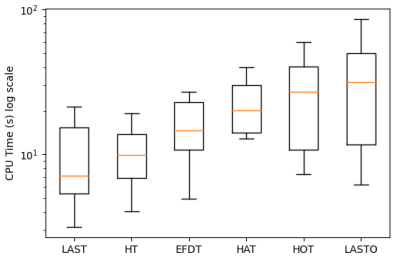

Fig. 13. CPU-Time (seconds, log scale) in synthetic datasets.

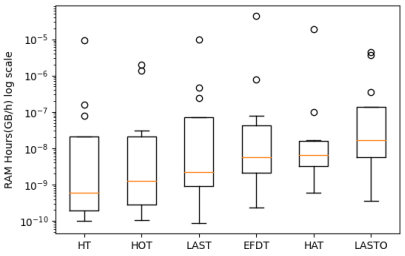

Fig. 14. RAM-Hours (GB/h, log scale) in real-world datasets.

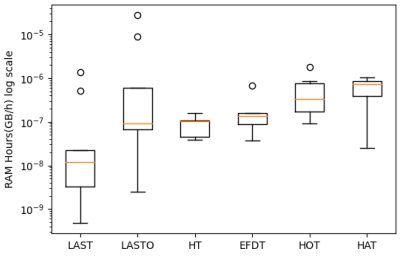

Fig. 15. RAM-Hours (GB/h, log scale) in synthetic datasets.

Table 2. Prequential Accuracy of decision trees

Data Stream	HT	EFDT	HAT	LAST	HOT	LASTO
LED_a	69.03	69.87	73.73	73.93	70.50	**73.94**
LED_g	68.65	69.72	**72.60**	71.49	69.59	72.55
SEA_a	86.42	86.41	**88.81**	86.61	86.42	86.58
SEA_g	86.42	86.37	**88.51**	86.38	86.43	86.67
AGR_a	81.05	82.87	**91.05**	83.94	81.09	84.88
AGR_g	77.37	80.09	**86.53**	80.58	77.50	80.97
RBF_m	45.49	51.27	61.75	64.11	56.55	**72.60**
RBF_f	32.29	31.87	39.16	36.70	33.94	**42.15**
HYPER	78.77	81.59	**86.69**	79.41	79.72	80.10
Outdoor	57.33	59.58	57.27	**60.40**	57.33	60.35
Rialto	31.35	**57.74**	30.62	56.33	27.14	56.71
Airlines	65.08	65.27	63.81	65.52	64.77	**69.83**
CovType	80.31	84.67	81.89	87.52	84.92	**89.27**
Nomao	92.13	93.93	93.97	**94.68**	93.04	93.34
Poker	76.07	76.60	66.87	76.40	76.23	**78.73**
NOAA	73.43	73.23	73.53	73.92	73.64	**74.09**
$INSECTS_a$	53.83	62.24	61.50	63.79	55.95	**64.74**
$INSECTS_i$	52.16	57.06	54.01	58.19	52.16	**59.02**
$INSECTS_g$	60.61	66.42	61.70	**67.61**	60.61	66.68
LADPU	51.20	59.78	51.25	60.63	51.20	**65.42**
Asfault	71.85	83.61	71.86	85.88	71.85	**87.30**
Avg. $Rank_{Synth}$	5.56	4.67	**1.56**	3.11	4.22	1.89
Avg. $Rank_{Real}$	5.21	3.08	4.50	1.92	4.79	**1.50**
Avg. Rank	5.36	3.76	3.24	2.43	4.55	**1.67**

newer ones, and an adaptive splitting strategy, as in LAST, that can react to changes in the leaf nodes accuracy by splitting the tree. We also plan to adapt LAST and LASTO to regression problems.

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

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. 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(9-10):1469–1495, June 2017.
2. Lu,J., Liu,A., Dong,F., Gu,F., Gama, J. and Zhang, G., "Learning under Concept Drift: A Review," in IEEE Transactions on Knowledge and Data Engineering, vol. 31, no. 12, pp. 2346-2363, 1 Dec. 2019, doi: 10.1109/TKDE.2018.2876857.
3. Domingos, P., Hulten, G. 2000. Mining high-speed data streams. In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '00). Association for Computing Machinery, New York, NY, USA, 71–80. https://doi.org/10.1145/347090.347107
4. Quinlan, J. R. 1992. C4.5: Programs for Machine. Morgan Kaufmann Publishers, 340 Pine Street, 6th Floor San Francisco, CA 94104 USA.
5. Breiman, L. 1984. Classification and Regression Trees. Wadsworth Statistics, Wadsworth, Belmont, CA.

Table 3. Tree size (# nodes) of decision trees

Data Stream	HT	EFDT	HAT	LAST	HOT	LASTO
LED_a	229	278	37	**15**	598	37
LED_g	221	299	78	**29**	634	56
SEA_a	753	357	541	**7**	828	19
SEA_g	743	350	1049	**11**	822	46
AGR_a	1056	966	157	**53**	1429	246
AGR_g	1190	623	1220	**187**	1563	843
RBF_m	219	1112	46	3277	726	16066
RBF_f	139	170	82	1455	290	6022
HYPER	1087	795	618	**225**	3566	1084
Outdoor	**1**	17	**1**	11	**1**	19
Rialto	**9**	164	9	185	26	914
Airlines	8582	15146	91376	8873	**1208**	1840
CovType	339	893	168	941	1392	2738
Nomao	34	31	8	36	98	110
Poker	297	543	3	683	1484	3392
NOAA	13	15	1	**7**	28	22
$INSECTS_a$	9	69	1	35	22	118
$INSECTS_i$	**7**	65	11	11	10	56
$INSECTS_g$	**3**	32	1	25	4	72
LADPU	**1**	42	**1**	35	**1**	132
Asfault	**1**	23	**1**	5	**1**	20
Avg. $Rank_{Synth}$	3.89	3.56	3.06	**1.89**	5.33	3.28
Avg. $Rank_{Real}$	**1.88**	4.62	2.29	3.62	3.17	5.42
Avg. Rank	2.74	4.17	**2.62**	2.88	4.10	4.50

6. Hoeffding, W., Probability inequalities for sums of bounded random variables. The collected works of Wassily Hoeffding, Springer, p. 409–426, 1963.
7. Bifet, A., Gavaldà, R. "Adaptive learning from evolving data streams." In International Symposium on Intelligent Data Analysis, pp. 249-260. Springer, Berlin, Heidelberg, 2009.
8. Manapragada, C., Webb, G., Salehi, M. Extremely Fast Decision Tree. In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '18). ACM, New York, NY, USA, 1953-1962. DOI: https://doi.org/10.1145/3219819.3220005
9. Assis, D. N., Barddal, J P., Enembreck, F.. 2024. Just Change on Change: Adaptive Splitting Time for Decision Trees in Data Stream Classification. In Proceedings of ACM SAC Conference (SAC'24). ACM, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/3605098.3635899
10. Pfahringer,B. , Holmes, G., Kirkby,R. 2007. New Options for Hoeffding Trees. In: Orgun, M.A., Thornton, J. (eds) AI 2007: Advances in Artificial Intelligence. AI 2007. Lecture Notes in Computer Science, vol 4830. Springer, Berlin, Heidelberg.
11. Gomes, H.M, Barddal, J. P., Enembreck, F. and Bifet, A. 2017. A Survey on Ensemble Learning for Data Stream Classification. ACM Comput. Surv. 50, 2, Article 23 (March 2018), 36 pages. https://doi.org/10.1145/3054925
12. Buntine,W., Learning classification trees. Stat Comput 2, 63–73 (1992). https://doi.org/10.1007/BF01889584
13. Kohavi, R., Kunz, C. 1997. Option Decision Trees with Majority Votes. In Proceedings of the Fourteenth International Conference on Machine Learning (ICML '97). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 161–169.
14. Gama, J., Rocha, R., Medas, P. 2003. Accurate decision trees for mining high-speed data streams. In Proceedings of the ninth ACM SIGKDD international conference

on Knowledge discovery and data mining (KDD '03). Association for Computing Machinery, New York, NY, USA, 523–528. https://doi.org/10.1145/956750.956813
15. Holmes, G., Kirkby,R., Pfahringer, B. (2005). Stress-Testing Hoeffding Trees. Knowledge Discovery in Databases: PKDD 2005. PKDD 2005. Lecture Notes in Computer Science, vol 3721. Springer, Berlin, Heidelberg.
16. Bifet, A., Gavaldà, R. 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
17. 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. volume 11 of Proceedings of Machine Learning Research, pages 44–50, Cumberland Lodge, Windsor, UK, 01–03 Sep 2010. PMLR
18. R. Agrawal, Imielinski, T. , Swami, A. "Database mining: a performance perspective," in IEEE Transactions on Knowledge and Data Engineering, vol. 5, no. 6, pp. 914-925, Dec. 1993, doi: 10.1109/69.250074.
19. W. Street, N. and Kim, Y. 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 (KDD '01). Association for Computing Machinery, New York, NY, USA, 377–382.
20. Hulten, G., Spencer, L., Domingos, P. 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
21. Souza, V. M., dos Reis, D. M., Maletzke, A. G., Batista, G. E. Challenges in benchmarking streaming learning algorithms with real-world data, Data Mining and K1nowledge Discovery 34 (2020) 1805– 1858.
22. García, S., Herrera, F. (2008) An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. J Mach Learn Res (2008) 9:2677–2694
23. Benavoli, A., Corani, G., Mangili, F. (2016) Should we really use post-hoc tests based on mean-ranks? J Mach Learn (2016) Res 17:1–10
24. Jean Paul Barddal and Fabricio Enembreck. 2019. Learning Regularized Hoeffding Trees from Data Streams. In Proceedings of the Annual ACM Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 08-12, 2019.