Learning Regularized Hoeffding Trees from Data Streams

Jean Paul Barddal
PPGIA – Pontifícia Universidade Católica do Paraná (PUC-PR)
Curitiba, Brazil
jean.barddal@ppgia.pucpr.br

Fabrício Enembreck
PPGIA – Pontifícia Universidade Católica do Paraná (PUC-PR)
Curitiba, Brazil
fabricio@ppgia.pucpr.br

ABSTRACT

Learning from data streams is a hot topic in machine learning that targets the learning and update of predictive models as data becomes available for both training and query. Due to their simplicity and convincing results in a multitude of applications, Hoeffding Trees are, by far, the most widely used family of methods for learning decision trees from streaming data. Despite the aforementioned positive characteristics, Hoeffding Trees tend to continuously grow in terms of nodes as new data becomes available, i.e., they eventually split on all features available, and multiple times on the same feature; thus leading to unnecessary complexity. With this behavior, Hoeffding Trees lose the ability to be human-understandable and computationally efficient. To tackle these issues, we propose a regularization scheme for Hoeffding Trees that (i) uses a penalty factor to control the gain obtained by creating a new split node using a feature that has not been used thus far; and (ii) uses information from previous splits in the current branch to determine whether the gain observed indeed justifies a new split. The proposed scheme is combined with both standard and adaptive variants of Hoeffding Trees. Experiments using real-world, stationary and drifting synthetic data show that the proposed method prevents both original and adaptive Hoeffding Trees from unnecessarily growing while maintaining impressive accuracy rates. As a by-product of the regularization process, significant improvements in processing time, model complexity, and memory consumption have also been observed, thus showing the effectiveness of the proposed regularization scheme.

CCS CONCEPTS

• **Computing methodologies → Supervised learning by classification**; **Online learning settings**;

KEYWORDS

Data Stream Mining; Decision Tree; Concept Drift; Regularization

ACM Reference format:
Jean Paul Barddal and Fabrício Enembreck. 2019. Learning Regularized Hoeffding Trees from Data Streams. In *Proceedings of The 34th ACM/SIGAPP Symposium on Applied Computing, Limassol, Cyprus, April 8–12, 2019 (SAC '19),* 8 pages.
https://doi.org/10.1145/3297280.3297334

1 INTRODUCTION

The growth rates of data acquisition and storage have gathered the effort from both researchers and practitioners towards the efficient analysis and knowledge extraction from these humongous datasets. One important trait of several computational systems nowadays is that data becomes available sequentially over time, in the form of a potentially unbounded data stream. Targeting learning classification models from data streams, different approaches have been created mimicking and adapting techniques from batch scenarios. For instance, bayesian, logistic regression based on stochastic gradient descent, and decision trees models are exemplars of families of data stream classification learners. Due to their simplicity and convincing results, decision trees and their ensembles are widely used approaches for learning from data streams, and Hoeffding Trees are the main exemplar of this family of methods [14]. Hoeffding Trees are an elegant, efficient and robust approach that learns decision trees using constant time per instance and has theoretical guarantees that the convergence between the decision trees learned in streaming and batch fashions basically depend on the sample size[1] used during the evaluation of features during the split.

Despite the aforementioned positive characteristics, the original Hoeffding Trees have two important drawbacks: (i) they assume that the data distribution is stationary, and (ii) they continuously grow in terms of nodes as new data becomes available, regardless of (i). As we discuss in Section 3, strategies for tackling (i) do exist, but even such methods either still fail to address (ii); or are ensemble-based methods that were developed targeting accuracy rates and not model readability [6] or are post-pruning techniques that depend on a multitude of hyper-parameters that are domain dependent [26]. Additionally, as a result of (ii), Hoeffding Trees tend to overfit to data, and they lose the important characteristic of being "white-boxes" in the sense that they become too complex and are no longer human-understandable.

In this paper, we propose a regularization scheme for Hoeffding Trees with the goal of preventing them from splitting - and consequently growing - unnecessarily as new data becomes available. Regularization is a process that discourages learning algorithms from unnecessary complexity. It has many flavors depending on the machine learning scheme being adopted. For instance, a famous regularization scheme is the LASSO [23], where the loss function of a linear learner is penalized according to a parameter λ that determines how much we wish to punish our model given increases

[1]This parameter is later referred to as the grace period parameter.

in its complexity. Conversely, regularization in decision trees may take place using different approaches, such as: (i) limiting the maximum depth of the tree, (ii) bagging more than a single tree, or even (iii) setting a stricter stopping criterion (such as a minimum gain function value) to avoid unnecessary splits. Our regularization approach is inspired in some of the aforementioned strategies and is divided in two parts:

(1) The use of a penalty factor ω to control the gain obtained by creating a new split node on the decision tree with a feature that has not been used thus far; and
(2) The use of information from previous splits in the current tree branch to determine whether the gain observed in a leaf indeed justifies a new split.

Together, we show that these two simple strategies prevent Hoeffding Trees from growing indefinitely while using a small subset of features that fits the concept to be learned.

This paper is divided as follows. Section 2 details data stream classification and its main challenges. Section 3 reviews Hoeffding Tree-based classifiers that will be later extended in Section 4 with the addition of the proposed heuristics for regularization. The proposed regularization scheme is then compared to the original Hoeffding Trees in Section 5 in both synthetic data streams and real-world data. Finally, Section 6 concludes this paper.

2 DATA STREAM CLASSIFICATION

In this paper, we target the data stream classification task. Formally, we denote S to be a potentially unbounded data stream providing instances $i^t = (\vec{x}^{\,t}, y^t)$ in the $(\vec{x}^1, y^1), \ldots, (\vec{x}^t, y^t), \ldots, (\vec{x}^{\,\infty}, y^{\infty})$ form. Each instance $(\vec{x}^t, y^t)$ drawn at a timestamp t is a realization from an input space X and and outcome space Y, such that the former is called the feature set, and the latter the class set. To denote the i-th feature from a feature set X, we will adopt the X_i notation.

Given $\mathcal{S}$, the goal behind the classification task is to learn and update a model $f : X \rightarrow Y$ over time. Updates on f can be either incremental, if the underlying patterns obtained from incoming instances adhere to the current concept; or decremental, for example, when a concept drift occurs. Generally speaking, the underlying concept C of a stream is a set of prior probabilities of the classes and class-conditional probability density functions [21]:

$$C = \bigcup_{y_i \in Y} \{(P[y_i], P[\vec{x}|y_i])\} \tag{1}$$

Given $\mathcal{S}$, instances will be labeled according to the current concept C^t. If between two timestamps t_i and $t_j > t_i$ it follows that $C^{t_i} \neq C^{t_j}$, then we have a concept drift [16]. Another important categorization for concept drifts regards their length: if $C^{t_i} \neq C^{t_i+1}$ the drift is said to be abrupt, while if $C^{t_i} \neq C^{t_i+\Delta}$ with $\Delta > 1$ occurs, the drift is called gradual. In this paper, we synthesize drifts using the sigmoidal approach proposed in the Massive Online Analysis (MOA) framework [9]. For different types of drift formulations, we refer the reader to the works of [15] and [24].

3 HOEFFDING TREES

Decision trees are a popular choice for learning prediction models in batch settings as they are simple, robust, and "white-boxes" in the sense that they can be easily understood. Decision trees are learned recursively with the replacement of leaves with split nodes, starting at the root. The definition of which attribute will be used in a split node is chosen by comparing all available features and choosing the best according to a heuristic function J. In practice, different realizations of J exist, such as the Gini Index, Conditional Entropy, and Information Gain. In this work, we adopt the Information Gain metric as it is widely used in both batch and streaming settings and achieves interesting results in a variety of scenarios. The information gain provided by a random variable A with respect to another variable B is given by:

$$IG(A; B) = H(A) - H(A|B)$$

where $H(A) = -\sum_{a \in A} P[A = a] \log_2 P[A = a]$ is the entropy, and $H(A|B) = \sum_{b \in B} H(A|B = b)$ is the conditional entropy of A given that the value of B is known. The splitting process is repeated on top of a set of training examples that are stored in main memory, and as a result, classical decision trees are limited to learning from this limited set of instances and are not tailored to evolve over time.

By definition, the assumption that the entire dataset is available for training does not hold in streaming scenarios, and thus, authors in [14] have proposed a swift method to learn decision trees from streaming data. Hoeffding Trees relax this constraint by comparing which feature is the most appropriate, according to J, on top of a small data sample. To determine how big this sample should be, still in [14], authors proposed the use of the *Hoeffding bound*. Assuming an heuristic function J with range R, the Hoeffding bound states that with probability $(1 - \delta)$, the true mean of J is at least $(\bar{J} - \epsilon)$, where ϵ is the bound calculated following Equation 2.

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

The rationale behind Hoeffding Trees is that, with high probability, the data distribution observed in a sample with size n adheres to the population distribution, which is expected to be infinite in streaming scenarios. In practical terms, a Hoeffding Tree will attempt an split after n instances are observed at one of its leaves. Assuming that the goal is to maximize[2] J, and that X_a is the best-ranked feature in terms of J and X_b the second best, then a split will be performed on X_a if $\Delta J = J(X_a, Y) - J(X_b; Y) \geq \epsilon$. As a result of empirical results, it has been noticed in [14] that reasonably small values of n, e.g., $n = 200$, achieve interesting results, and the same value has been adopted in frameworks of the area, i.e. the Massive Online Analysis (MOA) framework [9], which has also been used for the implementation of the proposed method. Also, it is important to highlight that Hoeffding Trees, by default, possess a pre-pruning step, which analyzes a 'null' attribute, which is the same as not splitting the node. As a result, a split will be made *iff* with confidence $(1 - \delta)$, the best split found is better w.r.t. J than not splitting.

One important shortcoming of conventional Hoeffding Trees is that they work under the assumption that the underlying distribution of the arriving data is stationary. As described in Section 2, this

[2]In practice, depending on the metric J being used, we should target its minimization instead. For instance, in CART-based trees, our goal would be to minimize the Gini Impurity metric instead of maximizing it, and as a result, the process should be adapted.

assumption does not hold in a variety of scenarios, and thus, trees should be able to update their branches according to changes in the arriving data. One important milestone in adaptive tree learning for data streams was Concept-adapting Very Fast Decision Trees (CVFDT) [18], yet, it depends on three distinct user-given window sizes to monitor, detect, and adapt to changes. Naturally, the tuning of such parameters is an unwanted task in streaming scenarios as the behavior of changes may also change over time.

With the same goal, Hoeffding Trees have been later extended in [8], where each split node uses an ADWIN drift detector [7] that monitors the error rate of that specific node. Whenever the error rate of that node significantly changes, the split node and its subtrees are replaced by a leaf, which starts the re-learning process. Hoeffding Adaptive Trees have been investigated in many domains, but we highlight feature drifting scenarios, which are cases where the subset of features that is relevant to the learning task changes over time [3, 25]. Whenever a feature drift occurs, the error rate of those features that become irrelevant rapidly increase, and these are consequently flagged by ADWIN, which result in smooth adaptations to drifts. Finally, it is also worthy to mention that Hoeffding Adaptive Trees have been extended in [4] where their leaves have their predictions weighted by adaptive entropy computations, also with the goal of overcoming feature drifts.

Another trend that deserves attention are tree-based ensembles [2, 10], as they reach relevant accuracy rates in a variety of applications. Closer to our approach, it is worth to cite the Ensemble of Restricted Hoeffding Trees [6], in which Hoeffding Trees are limited to a specific height, yet, their readability are jeopardized as the number of trees is reasonably high (in the scale of hundreds). Also related to our approach is the work of [26], in which authors target both accuracy improvements while performing post-pruning in Hoeffding Trees, yet, their approach relies on a multitude of user-given parameters that are domain dependent.

4 REGULARIZATION IN HOEFFDING TREES

In this section, we present our proposed regularization scheme for Hoeffding Trees. Our proposal was designed with two goals: (i) to prevent Hoeffding Trees from splitting on features that have already been used in the current branch unless the gain observed is greater than the gains observed earlier; and (ii) to prevent Hoeffding Trees from using features that have not been used thus far unless the gain obtained is significant.

In batch learning settings, regularization in decision trees may occur using different approaches, being the following the most common: (i) limiting the maximum depth of the tree, (ii) bagging more than a single tree, or even (iii) setting a stricter stopping criterion (such as a minimum gain function value) to avoid unnecessary splits. Our proposal is inspired in the aforementioned strategies and is divided in two parts that tackle the aforementioned items. The following subsections describe each of these parts individually and how they are embedded within the Hoeffding Tree learning.

4.1 Avoiding the selection of unused features

The first part of the proposed regularization scheme is to penalize selecting a new feature for splitting when its gain J is similar to the features used in previous splits. This scheme is similar to the one introduced in [13], where authors explore regularization in random forests and boosted trees models for batch feature selection. During the split analysis in a leaf node l, we assume F to be the list of features used in the previous split nodes that go from l to the tree's root. To avoid the selection of a new feature, we introduce an user-given penalty parameter $0 \leq \omega \leq 1$ that impacts J for $X_j \notin F$, as described by Equation 3:

$$J^*(X_i, Y, F, \omega) = \begin{cases} \omega \times J(X_i; Y) & \text{if } X_i \notin F \\ J(X_i; Y) & \text{otherwise} \end{cases} \tag{3}$$

In practice, the regularization process will only select a new feature to split on unless the gain provided by it is substantially larger than the maximum gain obtained with a feature that has already been selected, as described by Equation 4:

$$\max_{X_i \notin F} J^*(X_i, Y, F, \omega) \geq \max_{X_j \in F} J^*(X_j, Y, F, \omega) \tag{4}$$

As observed in [13], it is expected that several values of ω to present similar behavior when applied to different applications, yet, we devote Section 5.1 to discuss the impact of different values of ω.

4.2 Analyzing previous splits during a leaf split

The second part of the proposed regularization process analyzes whether the best-ranked feature chosen to split on in a leaf node provides meaningful gains compared to the gains observed in previous split nodes of the decision tree. Here, we denote (F, M) to be a pair of lists, where F are the features that the tree has split on that go from the tree's root until the current leaf l that reached its grace period n, and M to be the respective J^* values of the splits we have in F. Given the information contained in (F, M), the rationale behind our regularization process is to avoid a split if the gain computed for a feature X_α is lower then the maximum gain also computed for X_α in a previous split. Even though this heuristic is an approximation, since the data distribution observed at l may differ from the data distribution in previous splits nodes, we work under the assumption that the data distribution is stationary, and thus, if the gain observed in a leaf node l is not superior to the gain observed earlier, a new split should be considered as an unnecessary complexity being added to the model. More formally, assuming X_α to be the best-ranked feature according to J^* in a leaf node l, a split will only occur if the Hoeffding bound condition is met (as previously explained in Equation 2 and later revisited in Section 4.3) and if the condition given Equation 5 holds.

$$\Psi(X_\alpha, F, M) = \max_{(X_\psi, M_\psi) \in (F, M)} \begin{cases} M_\psi & \text{if } X_\alpha = X_\psi \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

In practical terms, a feature will only be used in a split if the gain provided by it is superior to the gains observed by the same feature in previous splits in the current branch.

4.3 Hoeffing tree learning with regularization

Given the formulation of our proposed regularization scheme, we devote this section to show how they are embedded within Hoeffding Tree learning. In Algorithm 1 we present an overview of the Hoeffding Tree learning process combined with the proposed regularization scheme. This pseudocode details the process for both

```
input  : a data stream $S$, the grace period $n$, and the
         regularization parameter $\omega$.
output : be ready to provide the decision tree $HT$ at any time.
1  Let $HT$ be a tree with a single leaf node;
2  foreach $i^t = (\vec{x}^t, y^t) \in S$ do
3      Sort $i^t$ into a leaf $l$ using $HT$, while $(F, M)$ is the set of
         features and their respective heuristic values used in the
         split nodes that go from the root to $l$;
4      Increment $l$'s counters for features and classes with $i^t$;
5      if $l$'s class distribution is not pure and the number of
         instances observed at $l$ is greater than $n$ then
6          $\forall X_i \in X$, compute $J^*(X_i, Y, F, \omega)$;
7          Let $X_\alpha$ be the best-ranked feature in terms of $J^*$;
8          Let $X_\beta$ be the second best-ranked feature in terms of
             $J^*$;
9          if $J^*(X_\alpha) - J^*(X_\beta) \geq \epsilon$ and $J^*(X_\alpha) > \Psi(X_\alpha, F, M)$
             then
10             Replace the leaf node $l$ with a split node with a
                 test on $X_\alpha$;
11         end
12     end
13 end
```

Algorithm 1: Hoeffding Tree learning with the proposed regularization process. For details on the implementation of counters and statistics, the reader is referred to the original paper on Hoeffding Trees [14] as the same process holds.

standard and adaptive Hoeffding Trees, as the general process is the same with the exception of the drift detector application, for which the reader is referred to [8]. The pseudocode reported here follows the original code reported in [14], as counters, statistics and other implementation details are maintained. The core of the Hoeffding Tree learning process resides in the loop given by lines 2-13, in which training instances arrive. In line 3, each instance i^t is traversed from the root until a leaf l is reached, and during this process, the feature set F that contains the features used in the split nodes of that specific branch are stored and their respective gain values (governed by J^*) are also stored. Next, in line 4, the counters and statistics for computing J^*, e.g., class distribution and feature-class counters, are updated at l. With the arrival of instances the grace period n will be reached and if the class distribution at l is not pure (test perform in line 5), i.e., biased towards a single class, the Hoeffding Tree will attempt an split. Next, all features X_i are evaluated according to J^* (Equation 3), and the two best-ranked features, X_α and X_β, have their gain differences compared against the Hoeffding bound. If the difference between their gains exceeds the Hoeffding bound, and the gain is greater than the maximum gain observed by the same feature in previous split nodes (stored in (F, M)), and verified with the Ψ function earlier introduced in Equation 5, then the leaf node l is replaced by a split node with a test on X_α.

Table 1: Details of synthetic and real-world experiments conducted.

Experiment	Number of Instances	Number of Relevant Features	Total Number of Features	Type	Reference
AGR	1,000,000	4	500	Synthetic	[1]
BG1	1,000,000	3	500	Synthetic	[17]
SEA	1,000,000	2	500	Synthetic	[22]
COV	581,012	N/A	55	Real	[11]
SPAM	9,324	N/A	39,917	Real	[20]

5 ANALYSIS

In this section, we compare the original Hoeffding Tree [14] and its adaptive variant [8] both with and without the proposed regularization scheme. In our analysis, we report accuracy rates, processing time (in seconds) and memory usage (measured in GB-Hour) computed using a Prequential test-then-train validation scheme [16]. On top of that, we also report the number of nodes (both split and leaf nodes) in the trees, thus highlighting how standard implementations of Hoeffding Trees grow indefinitely and why regularization is beneficial. We divided this analysis in 4 subsections: (i) a discussion on the impact of ω in terms of accuracy rates and number of selected features in both real-world and synthetic data; (ii) a comparison of Hoeffding Trees with and without the proposed regularization scheme in stationary synthetic streams; (iii) the same as the latter item, but using drifting data; and finally, (iv) an analysis on real-world data. In the following sections, we use **HT** to represent the original Hoeffding Tree, **HAT** the Hoeffding Adaptive Tree, and the same names followed by the suffix **-REG** are the same classifiers using the proposed regularization process.

An overview of the experiments conducted is given in Table 1. We use the AGRAWAL (AGR) [1], BG1 [17]; and SEA [22] generators and use the protocol proposed in [3] so that irrelevant features are appended in each experiment. In practical terms, a feature is deemed as irrelevant if the concept that is labelling instances does not use it, and as a result, the correlation between each of these features and the class tends to zero [3]. Also, for the sake of brevity, the concept classification functions of each of these synthetic data generators are omitted and the reader is referred to the original papers and to the Massive Online Analysis (MOA) framework[3] for details. In total, each synthetic experiment reported in this section contain 1 million instances and 500 features, where only a handful of them are relevant, as also depicted in Table 1. Regarding real-world data, two widely used datasets, namely Forest Covertype (COV) [11] and Spam Corpus (SPAM) [20], were used. Each of these experiments has a different number of features and instances, and thus, will allow a comprehensive overview of how regularization acts on streams with different characteristics. Regarding the parameters of Hoeffding Trees, all the parameters have been configured according to the default values used in the MOA framework, i.e. the heuristic function J is the Information Gain and the grace period size $n = 200$. Also, the code used to reproduce the experiments listed in this paper are available at https://github.com/jpbarddal/moa-reght and all the results obtained were obtained in an 2.9GHz Intel i7-based iMac with 16 GB of RAM. Finally, all of the results have been tested for

[3]The Massive Online Analysis (MOA) framework is available at https://moa.cms.waikato.ac.nz and its source code is available at https://github.com/waikato/moa.

statistical confidence with the help of Friedman and Nemenyi tests according to the protocol given in [12] using a 99% confidence level.

5.1 The impact of ω

In this section, we evaluate different values of ω and how they impact Hoeffding Trees in terms of accuracy rates and number of features selected. As reported in Equation 3, smaller values of ω result in higher penalties for features that have not been used in the current tree branch, yet, it is unclear how different values of ω will impact overall accuracy rates in different concepts. To analyze such impact, we test values of ω within $[0.1; 0.9]$, using a step of 0.1. For the sake of brevity, we only report the results obtained for the standard Hoeffding Tree as no differences have been observed between it and its adaptive variant. The results obtained are shown in Figures 1, 2 and 3, where we have the AGR, BG1 and SEA experiments with no concept drifts. In these figures, we observe that the accuracy rates do not change drastically according to changes in ω. In practice, most of the accuracy curves are overlapping during the entire stream. One exception is the AGR experiment, where $\omega = 0.8$ and $\omega = 0.9$ yield slightly higher results, but that in average, are approximately 1% higher.

Conversely, the results obtained in terms of number of selected features are consistent with the expectations, as the number of selected features increases with smaller ω values. Naturally, according to the Minimum Description Length [5] and Occam's Razor principles, we should minimize model complexity if it does not jeopardize model efficiency. The combination of results obtained here in terms of accuracy w.r.t. ω variations are consistent with what has been observed in [13], and thus, we follow the same conservative approach used by authors, as $\omega = 0.5$ will be used in the following experiments. Yet, a disclaimer for the use of the proposed regularization scheme may require an optimization of ω in future applications. Also, as we will show in the following sections, even if we follow a more conservative approach, i.e., slightly higher values of ω, the impact in computational resources, i.e., number of number of nodes, memory consumption and processing time, will be significant.

5.2 Synthetic Stationary Data

In this step of the analysis, we use the same synthetic data streams used in the previous section, i.e., AGR, BG1 and SEA, again without concept drifts. Here, we aim at stressing Hoeffding Trees and check how the number of nodes and number of features selected in the tree evolve during the progress of each stream and how the proposed regularization scheme impacts the results in terms of accuracy, processing time, memory consumption, number of tree nodes, and number of selected features.

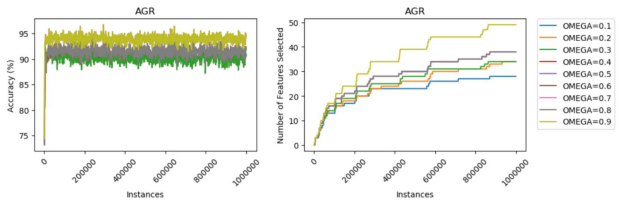

Figure 1: Results obtained with different ω values in the AGR experiment using the HT-REG learner.

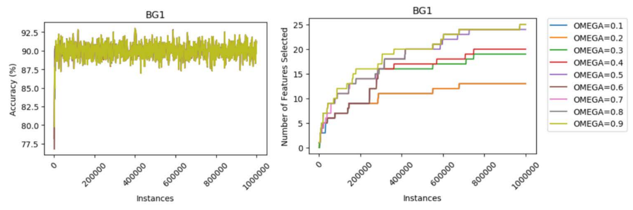

Figure 2: Results obtained with different ω values in the BG1 experiment using the HT-REG learner.

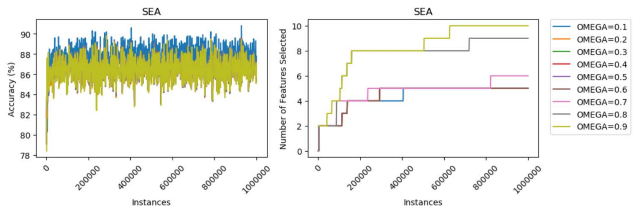

Figure 3: Results obtained with different ω values in the SEA experiment using the HT-REG learner.

Table 2: Accuracy (%) results obtained in stationary synthetic experiments.

Experiment	HT	HT-REG	HAT	HAT-REG
AGR	**94.95**	93.86	**94.29**	93.88
BG1	89.93	**89.99**	89.89	**89.99**
SEA	**89.05**	88.48	**89.13**	88.71

First, we analyze whether the adoption of the proposed regularization scheme puts the tree learning process in risk by checking the overall accuracy rates of both Hoeffing Tree (HT) and Hoeffding Adaptive Tree (HAT) learners. The results obtained are reported in Table 2, where we observe that the accuracy rates of both learners are similar regardless of the adoption of the proposed regularization strategy as the biggest deviation observed was of 1.09% in the AGR experiment between HT and HT-REG.

After observing that the proposed regularization scheme is able to maintain consistent accuracy rates, it is important to verify whether it is significantly improving the model complexity in terms of processing time, memory consumption and tree structure. We now focus on processing time rates, shown in Table 3, where the gains obtained with the regularization scheme are high. For instance, the smaller improvements observed are for HAT in the SEA experiment, which became 6 times faster, while the biggest improvements are for HT in the AGR stream, which became 18 times faster; both corroborated by the combination of Friedman and Nemenyi's tests.

Following the trend observed for processing time rates, the gains obtained for memory consumption can be observed in Table 4. Again, the gains observed here vary from the HAT being 7 times

Table 3: CPU Time (s) results obtained in stationary synthetic experiments.

Experiment	HT	HT-REG	HAT	HAT-REG
AGR	4548.62	**244.48**	4531.19	**269.25**
BG1	2189.56	**214.53**	2917.88	**304.88**
SEA	4292.11	**479.76**	6203.82	**967.64**

Table 4: RAM-Hours (GB-Hour) results obtained in stationary synthetic experiments.

Experiment	HT	HT-REG	HAT	HAT-REG
AGR	7.13×10^{-2}	$\mathbf{4.06 \times 10^{-5}}$	7.12×10^{-2}	$\mathbf{4.57 \times 10^{-5}}$
BG1	1.66×10^{-2}	$\mathbf{9.01 \times 10^{-5}}$	2.98×10^{-2}	$\mathbf{2.30 \times 10^{-4}}$
SEA	6.88×10^{-2}	$\mathbf{3.13 \times 10^{-4}}$	1.33×10^{-2}	$\mathbf{1.84 \times 10^{-3}}$

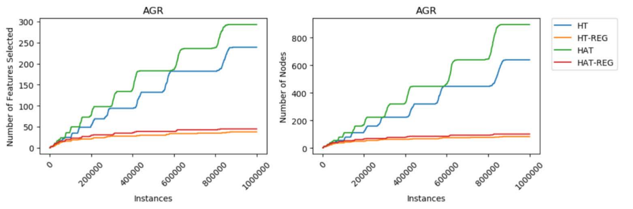

Figure 4: Details about the Hoeffding Tree models in the AGR experiment with no concept drift.

lighter in the SEA experiment to the HT being 1756 times more lightweighted in the AGR experiment, therefore showing significant improvements according to Friedman and Nemenyi's tests.

The biggest reason behind the improvements in processing time and memory consumption rates resides in the tree structure. In Figures 4 through 6, we report the number of features selected and the number of tree nodes in both HT and HAT learners with and without the regularization scheme. The behavior observed across all experiments are quite similar, as the number of features selected and the number of tree nodes in both HT and HAT learners rapidly grow with the arrival of training instances. In all cases, we highlight that more than 200 features have been selected, while in practice, only a handful of them are actually relevant to the concept to be learned (see details in Table 1), thus showing that the Hoeffding Trees tend to overfit to data. Similarly, the number of nodes also grows quickly, reaching nearly 900 nodes after 1 million instances, thus rendering the models learned not human-understandable. Taking into account the results obtained with the regularization process, we can observe that both the number of features selected and tree nodes are much smaller than those observed in the original tree implementations without regularization. In practice, the combination of the results shown here show that the proposed regularization scheme benefits Hoeffding Trees in stationary environments in terms of model complexity, processing time, memory consumption, while at the cost of limited accuracy rates.

5.3 Synthetic Drifting Data

In this section, we repeat the experiments conducted previously, but with one important difference: all streams now contain one gradual

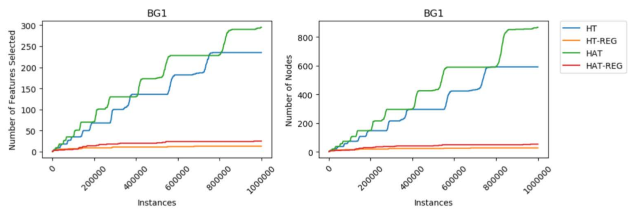

Figure 5: Details about the Hoeffding Tree models in the BG1 experiment with no concept drift.

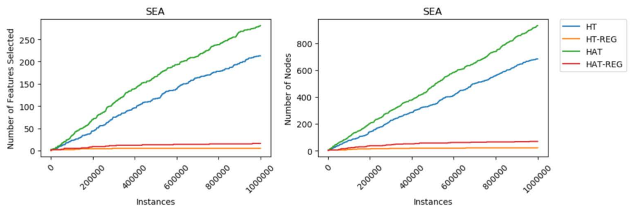

Figure 6: Details about the Hoeffding Tree models in the SEA experiment with no concept drift.

Table 5: Accuracy (%) results obtained in drifting synthetic experiments.

Experiment	HT	HT-REG	HAT	HAT-REG
AGR	**87.56**	83.60	**91.63**	90.59
BG1	84.90	**87.93**	89.25	**89.40**
SEA	85.21	**85.98**	**88.23**	87.75

Table 6: CPU Time (s) results obtained in drifting synthetic experiments.

Experiment	HT	HT-REG	HAT	HAT-REG
AGR	3745.07	**206.71**	2331.09	**570.30**
BG1	1290.60	**266.59**	962.80	**332.30**
SEA	2826.98	**421.92**	2604.54	**866.15**

concept drift ($\Delta = 50,000$) located at the middle of the stream. In Figures 7 through 9, we report the results obtained for all the synthetic experiments, again focusing on the accuracy rates, number of tree nodes, and number of features selected in each tree model. Similarly as before, we start by conducting an analysis of the accuracy rates obtained with and without the proposed regularization. The results are shown in Table 5, where similar behavior is observed as the proposed regularization scheme renders interesting accuracy rates compared to the original tree models. The improvements obtained for processing time and memory consumption, reported in Tables 6 and 7, again show important gains that show how beneficial the regularization process is due to the simplified tree models that are learned. Such simplification can again be observed in the results depicted in Figured 7 through 9, where the simplification in terms of features selected and tree nodes are again significant according to Friedman and Nemenyi's tests.

Table 7: RAM-Hours (GB-Hour) results obtained in drifting synthetic experiments.

Experiment	HT	HT-REG	HAT	HAT-REG
AGR	8.86×10^{-2}	$\mathbf{8.30 \times 10^{-5}}$	3.00×10^{-2}	$\mathbf{1.35 \times 10^{-3}}$
BG1	1.11×10^{-2}	$\mathbf{4.16 \times 10^{-4}}$	4.98×10^{-3}	$\mathbf{6.06 \times 10^{-4}}$
SEA	5.17×10^{-2}	$\mathbf{6.70 \times 10^{-4}}$	3.57×10^{-2}	$\mathbf{2.03 \times 10^{-3}}$

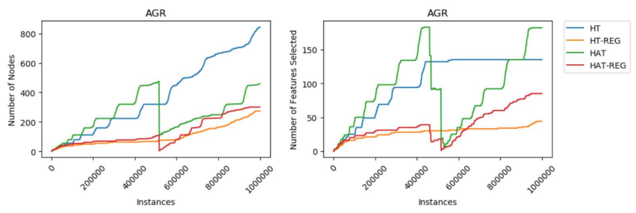

Figure 7: Results for the AGR experiment with one concept drift.

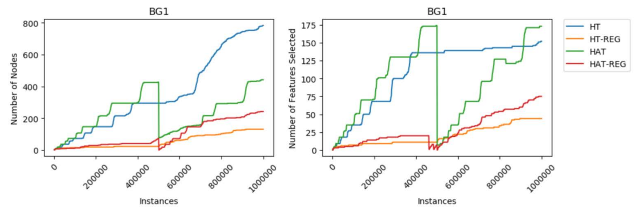

Figure 8: Results for the BG1 experiment with one concept drift.

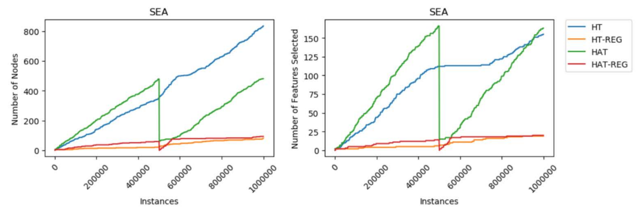

Figure 9: Results for the SEA experiment with one concept drift.

5.4 Real-world Data

In this final section, we evaluate the Hoeffding Tree models using real-world data. Following the protocol adopted for synthetic experiments, the results obtained in terms of accuracy, depicted in Table 8, by the tree models with and without regularization are similar, while the results for processing time and memory consumption rates are again interesting (see Tables 9 and 10). We note, however, that the improvements obtained here are not as significant as those observed for synthetic experiments since the real-world datasets are relatively small in terms of instances. As observed in the previous sections, the growth of the number of tree nodes is associated with the number of instances that the tree is trained on, and thus, the regularization process does not seem to affect the computational resources rates too highly.

Table 8: Accuracy (%) results obtained in real-world datasets.

Experiment	HT	HT-REG	HAT	HAT-REG
COV	80.34	**80.40**	81.92	**83.45**
SPAM	**80.35**	80.12	**85.21**	84.77

Table 9: CPU Time (s) results obtained in real-world datasets.

Experiment	HT	HT-REG	HAT	HAT-REG
COV	29.10	**15.85**	21.92	**20.44**
SPAM	286.23	**245.52**	334.17	**317.90**

Table 10: RAM-Hours (GB-Hour) results obtained in real-world datasets.

Experiment	HT	HT-REG	HAT	HAT-REG
COV	1.84×10^{-5}	$\mathbf{1.39 \times 10^{-6}}$	5.96×10^{-7}	$\mathbf{5.69 \times 10^{-8}}$
SPAM	3.30×10^{-3}	$\mathbf{2.96 \times 10^{-3}}$	5.43×10^{-3}	$\mathbf{3.75 \times 10^{-3}}$

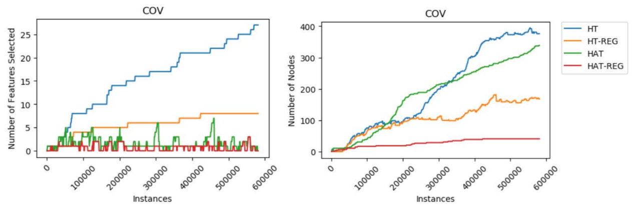

Figure 10: Details about the Hoeffding Tree models in the COV experiment.

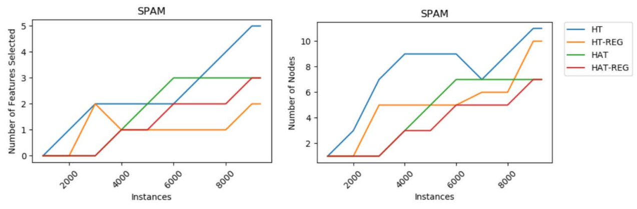

Figure 11: Details about the Hoeffding Tree models in the SPAM experiment.

However, it is still important to analyze the number of selected features and tree nodes, which are shown in Figures 10 and 11. Regarding the Forest Covertype (COV) experiment (Figure 10), we observe that both the number of features selected and tree nodes rapidly grow with the number of training instances, thus following the behavior observed in previous experiments. A more discrete behavior is observed for the SPAM experiment (Figure 11), where the regularization process is still able to reduce the tree size and number of features selected, but in a smaller scale.

6 CONCLUSION

Hoeffding Trees are a family of efficient and accurate classification models for data streams. Despite Hoeffding Trees being fast and producing interesting results in a variety of scenarios, they are still prone to branching indefinitely as new data becomes available for training. In this paper, we conducted an analysis on the number of nodes of Hoeffding Trees and on the number of features selected on these splits in different scenarios. This analysis highlighted their propensity to overfit as the number of splits on these trees rapidly grow, and that many irrelevant features are selected as new data becomes available. Next, we proposed a regularization scheme that prevents Hoeffding Trees from unnecessarily growing. The first penalizes features that the tree has not split on already according to user-given ω factor, while the second utilizes information of previous splits to determine whether a new split is necessary. Experiments using both real-world and synthetic data showed that the proposed method prevents both original and adaptive Hoeffding Trees from unnecessarily growing while maintaining interesting accuracy rates. As a by-product of the regularization process, improvements in processing time, model complexity, and memory consumption have also been observed, while at the expense of bounded accuracy decreases. As future works, we cite the following: (i) the application of the proposed regularization scheme for regression trees [19], and (ii) an analysis of the impact of the proposed regularization scheme in ensembles of Hoeffding Trees in terms of accuracy rates and computational resources.

ACKNOWLEDGEMENTS

The authors thank the anonymous referees that provided meaningful considerations on the initial manuscript and thoughts on future works to continue this research. This research was funded by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) since the main author was an awardee of a Programa de Suporte à Pós-Graduação de Instituições Comunitárias de Ensino Superior (PROSUC) scholarship.

REFERENCES

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

[2] Geoffrey Holmes Albert Bifet, Eibe Frank and Bernhard Pfahringer (Eds.). 2010. *Accurate Ensembles for Data Streams: Combining Restricted Hoeffding Trees using Stacking*. JMLR Proceedings, Vol. 13. JMLR.org.

[3] Jean Paul Barddal, Heitor Murilo Gomes, FabrÃŋcio Enembreck, and Bernhard Pfahringer. 2017. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software* 127 (2017), 278 – 294. https://doi.org/10.1016/j.jss.2016.07.005

[4] Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, Bernhard Pfahringer, and Albert Bifet. 2016. On Dynamic Feature Weighting for Feature Drifting Data Streams. In *ECML/PKDD'16 (Lecture Notes in Computer Science)*. Springer.

[5] Andrew R Barron, Jorma Rissanen, and Bin Yu. 1998. The Minimum Description Length Principle in Coding and Modeling. *IEEE Trans. Inf. Theory* 44, 6 (1998), 2743–2760. http://dblp.uni-trier.de/rec/bibtex/journals/tit/BarronRY98

[6] Albert Bifet, Eibe Frank, Geoff Holmes, and Bernhard Pfahringer. 2012. Ensembles of Restricted Hoeffding Trees. *ACM Trans. Intell. Syst. Technol.* 3, 2, Article 30 (Feb. 2012), 20 pages. https://doi.org/10.1145/2089094.2089106

[7] Albert Bifet and Ricard Gavaldà. 2007. Learning from time-changing data with adaptive windowing. In *In SIAM International Conference on Data Mining*.

[8] Albert Bifet and Ricard Gavaldà. 2009. *Adaptive Learning from Evolving Data Streams*. Springer Berlin Heidelberg, Berlin, Heidelberg, 249–260. https://doi.org/10.1007/978-3-642-03915-7_22

[9] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. MOA: Massive Online Analysis. *The Journal of Machine Learning Research* 11 (2010), 1601–1604.

[10] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. 2010. Leveraging Bagging for Evolving Data Streams. In *Machine Learning and Knowledge Discovery in Databases*, JosÃl' Luis BalcÃązar, Francesco Bonchi, Aristides Gionis, and MichÃĺle Sebag (Eds.). Lecture Notes in Computer Science, Vol. 6321. Springer Berlin Heidelberg, 135–150. https://doi.org/10.1007/978-3-642-15880-3_15

[11] Jock A. Blackard and Denis J. Dean. 1999. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture* 24, 3 (1999), 131 – 151. https://doi.org/10.1016/S0168-1699(99)00046-0

[12] Janez Demsar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.* 7 (Dec. 2006), 1–30. http://dl.acm.org/citation.cfm?id=1248547.1248548

[13] Houtao Deng and G. Runger. 2012. Feature selection via regularized trees. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*. 1–8. https://doi.org/10.1109/IJCNN.2012.6252640

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

[15] Joao Gama. 2010. *Knowledge Discovery from Data Streams* (1st ed.). Chapman & Hall/CRC.

[16] J. Gama and P. Rodrigues. 2009. Issues in evaluation of stream learning algorithms. In *Proc. of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM SIGKDD, 329–338.

[17] Mark A. Hall and Lloyd A. Smith. 1999. Feature Selection for Machine Learning: Comparing a Correlation-based Filter Approach to the Wrapper. (1999).

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

[19] Elena Ikonomovska, João Gama, Bernard Zenko, and Saso Dzeroski. 2011. Speeding-Up Hoeffding-Based Regression Trees With Options. In *ICML*. 537–544.

[20] Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. 2006. Dynamic Feature Space and Incremental Feature Selection for the Classification of Textual Data Streams. In *in ECML/PKDD-2006 International Workshop on Knowledge Discovery from Data Streams. 2006*. Springer Verlag, 107.

[21] Hai-Long Nguyen, Yew-Kwong Woon, Wee-Keong Ng, and Li Wan. 2012. Heterogeneous Ensemble for Feature Drifts in Data Streams. In *Advances in Knowledge Discovery and Data Mining*, Pang-Ning Tan, Sanjay Chawla, ChinKuan Ho, and James Bailey (Eds.). Lecture Notes in Computer Science, Vol. 7302. Springer Berlin Heidelberg, 1–12. https://doi.org/10.1007/978-3-642-30220-6_1

[22] W. Nick Street and Y. Kim. 2001. A streaming ensemble algorithm (SEA) for large-classification. In *Proc. of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM SIGKDD, 377–382.

[23] Robert Tibshirani. 1996. Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society. Series B (Methodological)* 58, 1 (1996), 267–288. http://www.jstor.org/stable/2346178

[24] Geoffrey I. Webb, Loong Kuan Lee, Bart Goethals, and François Petitjean. 2018. Analyzing concept drift and shift from sample data. *Data Mining and Knowledge Discovery* (12 Mar 2018). https://doi.org/10.1007/s10618-018-0554-1

[25] Gerhard Widmer and Miroslav Kubat. 1996. Learning in the Presence of Concept Drift and Hidden Contexts. *Mach. Learn.* 23, 1 (April 1996), 69–101. https://doi.org/10.1023/A:1018046501280

[26] H. Yang and S. Fong. 2011. Optimized very fast decision tree with balanced classification accuracy and compact tree size. In *The 3rd International Conference on Data Mining and Intelligent Information Technology Applications*. 57–64.