Boosting Decision Stumps for Dynamic Feature Selection on Data Streams

Jean Paul Barddal, Fabrício Enembreck

Graduate Program in Informatics (PPGIa) – Pontifícia Universidade Católica do Paraná

Heitor Murilo Gomes, Albert Bifet

INFRES, Institut Mines-Télécom, Télécom ParisTech, Université Paris-Saclay, Paris, France

Bernhard Pfahringer

Department of Computer Science – University of Waikato

Abstract

Feature selection targets the identification of which features of a dataset are relevant to the learning task. It is also widely known and used to improve computation times, reduce computation requirements, and to decrease the impact of the curse of dimensionality and enhancing the generalization rates of classifiers. In data streams, classifiers shall benefit from all the items above, but more importantly, from the fact that the relevant subset of features may drift over time. In this paper, we propose a novel dynamic feature selection method for data streams called Adaptive Boosting for Feature Selection (ABFS). ABFS chains decision stumps and drift detectors, and as a result, identifies which features are relevant to the learning task as the stream progresses with reasonable success. In addition to our proposed algorithm, we bring feature selection-specific metrics from batch learning to streaming scenarios. Next, we evaluate ABFS according to these metrics in both synthetic and real-world scenarios. As a result, ABFS improves the classification rates of different types of learners and eventually enhances computational re-

Email addresses: {jean.barddal, fabricio}@ppgia.pucpr.br (Jean Paul Barddal, Fabrício Enembreck), {heitor.gomes, abifet}@telecom-paristech.fr (Heitor Murilo Gomes, Albert Bifet), bernhard@cs.waikato.ac.nz (Bernhard Pfahringer)

sources usage.

Keywords: Data Stream Mining, Feature Selection, Concept Drift, Feature Drift

1. Introduction

Feature selection is an essential part of a machine learning pipeline. The central goal of this task is to identify and retain the subset of features of a dataset that is relevant to the learning task. Despite its benefits to reduce computation time by focusing the model training only on a subset of features, feature selection can have an even bigger impact in diminishing the curse of dimensionality. This characteristic can enhance the performance of predictive models. Comparing the feature selection studies from the last 20 years, the number of dimensions (features) has rapidly grown. For instance, the studies of [1, 2] tackled datasets described with an average of 40 features. In the age of Big Data, the number of features has grown tremendously, where hundreds, thousands [3] or even millions of them are observed in specific domains [4], and these are called *high-dimensional* scenarios.

Traditional feature selection techniques are tailored to be part of the pre-processing step of the batch knowledge discovery process. Nevertheless, a variety of data mining applications are not static, as data often arrive in the form of potentially infinite sequences of data, the so-called *data streams*. Learning from data streams exhibits not only the challenges from traditional learning schemes, e.g., missing values and class imbalance; but also concept drifts. A concept drift occurs when the data distribution changes, possibly impacting the relationship of features, their values, and the target variable [5].

Recently, studies on data stream mining shed light on the fact that certain types of drift affect the importance of features over time. Scenarios where features become, or cease to be, relevant to the learning task are called *feature drifting data streams*, and these are the target of this paper.

In this paper, we propose a novel dynamic feature selection method for data streams called Adaptive Boosting for Streaming Feature Selection. Our proposal is tailored to tackle feature drifting high-dimensionality scenarios, thus allowing classifiers to learn from a reduced number of features. This method adopts a boosting scheme inspired by the work of [6] with decision stumps to dynamically identify which features are relevant to the current

concept of a data stream. Drift detectors are used to flag drifts and enable quick response to changes in the importance of features. We evaluate our proposal in a variety of streaming scenarios, and also with different types of learners. Additionally, we show how metrics from batch feature selection can be ported to streaming scenarios. These metrics, along with the code for our proposed method, are made available for the community[1] as part of the Massive Online Analysis (MOA) software [7].

This paper is divided as follows. Section 2 introduces the data stream classification and feature drifts. Section 3 reviews related work on feature analysis in data streams. Section 4 introduces our proposal, namely *Adaptive Boosting for Feature Selection* (ABFS), while Section 5 conducts an analysis on how feature selection-specific evaluation metrics can be used in streaming scenarios. Next, in Section 6, the proposed method is evaluated against different base learners, showing its efficacy and efficiency in a variety of data streams and dimensions. Finally, Section 7 concludes this paper.

2. Problem Statement

In this paper, we target the classification task for inductive learning from data streams. More formally, let $\mathcal{S}$ to be a data stream providing instances $(\vec{x}^1, y^1)$, ..., $(\vec{x}^t, y^t)$ as $t \to \infty$. We also denote $\vec{x}^t$ to be a d-dimensional feature vector belonging to a feature set $\mathcal{F} = \bigcup_{i=1}^{d} \{f_i\}$, and $Y = \bigcup_{i=1}^{c} \{y_i\}$ to be the set of c possible class labels. In data stream classification, our goal is to continuously learn and update a model $h : \vec{x} \to y$ that maps features and their values to class labels as new data becomes available.

One of the most important traits of data streams is that their underlying distribution may change over time. As a result, these changes affect the concept to be learned; a phenomenon referred to as *concept drift*. In this paper, we target a specific type of concept drift that occurs when features become, or cease to be, relevant to the learning task. In the seminal work of [5], this type of drift was referred to as *contextual concept drift*, while more recent works [8, 9] call it *feature drift*. In contrast to conventional concept drifts, where changes occur in the relationship between values or ranges of

[1]The code for our implementation of ABFS, data generators, evaluation metrics, and scripts to reproduce the experiments shown in this paper are available at `https://github.com/jpbarddal/moa-abfs`.

variables and the class, feature drifts occur whenever a subset of features becomes or ceases to be relevant for the current concept to be learned.

Formally, we divide features in two types: relevant and irrelevant according to the work of [8]. Assuming $S_i = \mathcal{F} \setminus \{f_i\}$, a feature f_i is deemed **relevant** *iff* Equation 1 holds.

$$\exists S'_i \subset S_i, \text{ such that } P[Y|f_i, S'_i] \neq P[Y|S'_i] \tag{1}$$

Otherwise, the feature f_i is said **irrelevant**. In practice, if a feature that is statistically relevant is removed from a feature set, it will reduce overall prediction power since (i) it is strongly correlated with the class; or (ii) it belongs to a subset of features that is strongly correlated with the class [10].

According to the previous definitions, if a feature that is statistically relevant is removed from a feature set, it will reduce overall prediction power. This definition encompasses two possibilities for a feature to be statistically significant: (i) it alone is strongly correlated with the class; or (ii) it forms a feature subset with other features and this subset is strongly correlated with the class [11, 10].

In streaming scenarios, changes in the relevant subset of features force the learning process to adapt its model accordingly [12]. Given a feature space $\mathcal{F}$ at a timestamp t_j, we denote $\mathcal{F}^*_{t_j}$ as its ground-truth relevant subset of features such that $\forall f_i \in \mathcal{F}^*_{t_j}$ the aforementioned definition of relevance holds. Therefore, a feature drift occurs if between two timestamps t_j and t_k we find that $\mathcal{F}^*_{t_j} \neq \mathcal{F}^*_{t_k}$.

To overcome this type of drift, a classifier must identify these relevance changes, and either (i) discard and learn a new model with the newly relevant features, or (ii) adapt its current model to relevance drifts [12].

3. Related Work

Finding a compact subset of relevant features is a widely tackled problem of batch learning, yet, the same does not hold for streaming scenarios. At this point, it is relevant to disclaim that this is different from *online feature selection*, which is the task that targets the identification of the best subset of features in a very high-dimensional space (hundreds of thousands or millions of dimensions), which is a typical problem of big data [13, 14]. Although both tasks' objectives overlap in the sense that both tackle the issue of feature selection, streaming feature selection receives as input a stream of features

(not instances), and their inclusion in the model is performed sequentially, without observing future features [15]. In contrast, our goal in this paper is to dynamically select features in streaming scenarios where new instances become available over time and the original feature set is static.

Recently, the works of [16] and [17] surveyed and evaluated different approaches to tackle this problem. According to these studies, incremental decision trees [18] and its variants [19] are the best performing approaches. Decision trees can be regarded as feature selection processes since they continuously select the feature that maximizes a quality metric during the branching step. For instance, the Hoeffding Tree [18] collects statistics about incoming data, and periodically, according to a grace period parameter, determine which feature should be used to split the tree and create even more specific leaf nodes. One of the significant drawbacks of the conventional Hoeffding Tree learner is that it is purely incremental as it does not check if any of the previous splits are still accurate. To overcome this issue, the Hoeffding Adaptive Tree [19] uses the ADWIN drift detector [20] inside decision nodes to monitor the internal error rates of the tree, and re-learn branches if needed.

Another recent work that focused on performing feature selection is the Heterogeneous Ensemble with Feature Drift for Data Streams (HEFT-Stream) [12]. HEFT-Stream incorporates traditional feature selection into a heterogeneous ensemble to adapt to different types of concept and feature drifts. HEFT-Stream adopts a modification of the Fast Correlation-Based Filter (FCBF) algorithm so it dynamically updates the selected relevant feature subset of a data stream. The main shortcoming of HEFT-Stream is that is processes the data stream using mini-batches, and the determination of the size of such batches is left to the user. To perform the scoring of features over sliding windows, the work of [8] proposes a dynamic feature weighting scheme for the streaming versions of Naive Bayes and k-Nearest Neighbors. Weights are computed with sliding window formulas for the Symmetrical Uncertainty scoring operator [21] and are used in the prediction step of the learners mentioned above. The accuracy gains are noticeable despite the expense of reasonable processing time and memory usage. Finally, the proposed weighting scheme was used at the leaves of the Hoeffding Adaptive Tree to improve its prediction rates, again with a computational overhead.

Another relevant approach is given in the work of [22], where authors proposed an unsupervised approach for feature ranking, and posterior selection, based on Frequent Directions. In practice, their proposal operates in a streaming by constructing and maintaining a sketch matrix that shrinks

the original data in orthogonal vectors. Even though the results obtained in terms of feature selection are interesting, authors work under the assumption that the number of features to be selected is known *a priori* and that the user is able to provide this number correctly.

At this point, it is worthy to notice that measuring the importance of features on streaming regression scenarios has also gained traction in the work of [23], yet, no feature selection proposals are provided. And finally, it is also relevant to mention that our main goal in this paper is to introduce a feature selection method for data streams that is not highly dependent of a user-given window size. This is relevant since, in practice, any traditional feature selection method can be applied to data streams by partitioning the arriving data in chunks of size n. At the end of each chunk, traditional feature selection methods could then be applied to the n previously instances gathered, thus resulting in a feature set that would be used to build a classifier to predict the upcoming n instances. Naturally, the biggest issue here is how to determine an appropriate value for n, and this problem is referred to as the plasticity-stability dilemma. While short windows reflect the current data distribution and ensure fast adaptation to drifts (plasticity), they usually worsen the performance of the system in stable areas. Conversely, larger windows give better performance in stable periods (stability), however, these imply in slower reaction to drifts [24].

4. Proposed Method

In this section, we propose a novel method based on Online Boosting [6] and drift detectors to identify which features are relevant for the classification task on data streams. We start this section with an introduction to Boosting methods for both batch and stream learning settings. These methods are at the core of the proposed method as Boosting allows feature interactions to be swiftly identified in data streams. The next part describes the proposed method and discusses its functioning. At this point, it is important to highlight that even though the proposed method has its foundations borrowed from [6], it is not tailored for classification as its inner weak learners are **not** used during predictions; and no convergence with batch learning is reported for the same reason.

4.1. Preliminaries on boosting

In machine learning, Boosting is a family of meta-learning methods that target the construction of a strong learner by combining multiple weak learners that, by definition, are slightly better than random guessing.

The most widely used and known implementation of Boosting is AdaBoost [25], and multiple variants of it were proposed throughout the years [26, 27]. In AdaBoost, a set of weak learners H is trained over a series of rounds $t = 1, \ldots, T$. During each iteration, a new weak learner $h_t : \vec{x} \rightarrow y$ is trained over the dataset $(\vec{x}^1, y^1), \ldots, (\vec{x}^n, y^n)$ taking into account a distribution of weights D_t for these instances. In the first round, it is assumed that all instances have the same weight, i.e. $D_1(i) = \frac{1}{n}$. The error of a weak learner in the t-th round is the sum of the weights of misclassified instances, as shown in Equation 2.

$$\epsilon_t = Pr_{i \sim D_t}\left[h_t(\vec{x}^i) \neq y^i\right] = \sum_{\forall i, h_t(\vec{x}^i) \neq y^i} D_t(i) \tag{2}$$

In each of the following rounds, the weights $D_t(i)$ are updated according to a parameter α_t, which is calculated according to Equation 3.

$$\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \epsilon_t}{\epsilon_t}\right) \tag{3}$$

In practice, α_t quantifies how "important" h_t is, as $\alpha_t \geq 0$ if $\epsilon_t \leq 1/2$ and that α_t increases with the decrease of ϵ_t. According to α_t, the weight distribution can be updated following Equation 4, where $Z_t = \sum_i^n D_t(i)$ is a normalization factor to guarantee that D_t is a distribution.

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(\vec{x}^i) = y^i \\ e^{\alpha_t} & \text{if } h_t(\vec{x}^i) \neq y^i \end{cases} \tag{4}$$

With this update, the weights of correctly classified instances will decrease, while misclassified instances will increase. As a result, this process highlights *hard-to-classify* instances for future rounds.

Finally, predictions can be extracted from the final "strong" learner as depicted in Equation 5[2].

[2]The original prediction scheme presented in [28] focuses only on binary classification tasks and has been extended here to account for multi-class problems. Details about the

$$H(\vec{x}) = \arg\max_{y_i \in Y} \left(\sum_{t=1}^{T} \begin{cases} \alpha_t h_t(\vec{x}) & \text{if } h_t(\vec{x}) = y_i \\ 0 & \text{if } h_t(\vec{x}) \neq y_i \end{cases} \right) \tag{5}$$

Even though AdaBoost is iterative, it works under the assumption that all instances of the dataset are available at all times so that re-weighting occurs. Naturally, this is an assumption that does not hold in streaming scenarios, as each instance should be processed and discarded right after. Targeting the development of boosting techniques for data streams, different approaches for classification [6, 29, 30] and regression [31] tasks have been developed over the years.

In this work, we follow a similar framework proposed in [6], called OzaBoost. OzaBoost was tailored to be an approximation of AdaBoost for data streams. Contrary to AdaBoost, where the number of rounds T determines the ensemble size, OzaBoost has a predefined number of weak learners M and counters for correctly $(\lambda_i^c, 1 < i < M)$ and incorrectly $(\lambda_i^e, 1 < i < M)$ classified instances which are updated as new instances are processed. For each instance $(\vec{x}^t, y^t)$ drawn from the data stream S, a weight $\lambda = 1$ is set. The instance is then traversed along the weak learners $h_1, \ldots, h_M$ sequentially. For each h_i, the instance is tested to check whether if it is correctly classified or not, i.e. $h_i(\vec{x}^t) = y^t$ or not; and as a result, the counters λ_i^c and λ_i^e are incremented with λ (Equations 6 and 7, respectively). Next, the value of λ is incremented or decremented following Equation 8. This is the same procedure adopted by AdaBoost in Equations 4 and 5, except the normalization factor Z_t, which cannot be used in data streams as past instances have been discarded.

$$\lambda_i^c \leftarrow \lambda_i^c + \lambda; \tag{6}$$

$$\lambda_i^e \leftarrow \lambda_i^e + \lambda; \tag{7}$$

$$\lambda \leftarrow \lambda \times \begin{cases} \frac{\lambda_i^c + \lambda_i^e}{2\lambda_i^c} & \text{if } h_i(\vec{x}) = y^t \\ \frac{\lambda_i^c + \lambda_i^e}{2\lambda_i^e} & \text{if } h_i(\vec{x}) \neq y^t \end{cases} \tag{8}$$

usage of AdaBoost in binary classification tasks and proofs on its error bounds can also be found in the same paper.

4.2. Adaptive Boosting for Feature Selection

The properties of boosting have been investigated to improve classification rates, but also as a proxy for feature selection in batch scenarios. For instance, the work of [32] uses gradient boosted regression trees to select features. Also, related to our approach, authors in both [33] and [34] proposed different boosting techniques that use decision stumps to select features in batch scenarios.

We now propose a novel method based on Boosting to dynamically select features in streaming scenarios hereafter referred to as *Adaptive Boosting for Feature Selection* (ABFS). At this point, it is important to disclaim that the term *Adaptive* used here stands for the fact that the proposed method incrementally selects features as the stream is processed, but it is also able to detect feature drifts and adapt to them on the fly.

ABFS combines decision stumps and drift detectors to perform dynamic feature selection. Decision stumps are light-weighted, incremental, easy to implement and understand, but more importantly, an elegant approach to identify which feature maximizes a purity criterion and selects a feature accordingly. Below, we describe each of these components individually and later how they are chained together to allow dynamic feature selection in data streams.

4.2.1. Decision stumps

The decision stump implementation used here is the core unit of incremental decision trees, e.g., Hoeffding Trees [18], and receive as input three parameters: a selection threshold θ, a grace period gp and a purity metric $\Omega(\cdot)$ that we wish to maximize, e.g. Information Gain and Gini Index. By definition, a decision stump ds gathers statistics on the arriving data until the grace period gp is reached. After that, all features $f_i \in \mathcal{F}$ are evaluated according to a criterion $\Omega(\cdot)$. Let f_α and f_β be the two best-ranked features according to Ω. As proposed in [18], a decision stump will split on f_α if $\Omega(f_\alpha) - \Omega(f_\beta) > \epsilon$, where ϵ is the Hoeffding bound [35], given by Equation 9, and R is the range of Ω.

$$\epsilon = \sqrt{\frac{R^2 \ln\left(\frac{1}{\delta}\right)}{2 \times gp}} \tag{9}$$

As in [18], the Hoeffding bound is used to approximate how many samples are required to achieve the optimal selection of a feature that would occur if

the entire data stream was observed. As a result, with probability $(1 - \delta)$, it is statistically valid that f_α is the best feature to be selected [18]. It is also possible that during this evaluation, two or more features show similar Ω values, and thus, the number of examples required to decide between them with high confidence may grow indefinitely. Naturally, this is unwanted, as the gain obtained by either makes little difference, i.e., such features are likely to be redundant; and thus, the proposed implementation for decision stumps follows the Hoeffding Tree protocol where a parameter τ is used for tie-breaking [18]. Following the definition of [18] and [7], the tie-breaking parameter was set as $\tau = 0.05$ for all of the experiments conducted in this paper. In practice, the decision stump will select the best-ranked feature f_α if $\tau > \Omega(f_\alpha) - \Omega(f_\beta) > \epsilon$.

In the proposed method, the decision stump is extended in two aspects. First, a decision stump will select the most appropriate feature f_α from a subset of features that have not been previously selected by other decision stumps. The idea on using boosting with decision stumps is that by traversing each instance across all the boosting units, instances that are hard to classify will be highlighted and will force the decision stump that is about to split to select a feature that better separates such samples. And second, the best-ranked feature f_α will only be selected if $\Omega(f_\alpha) > \theta$, which is a user-given threshold. Naturally, the definition of a selection threshold θ depends on the data domain being worked on, and different values are evaluated in Section 6.

4.2.2. Drift detectors

A drift detector is a statistical method that observes a data sequence and upon on its distribution, flags the occurrence of significant changes. In data streams, most of the drift detectors are used to monitor the error rates of a classifier. In this work, we denote ψ to be a drift detector that receives as input a value of 1 if $h(\vec{x}^t) \neq y^t$, or 0 otherwise. Evidently, different realizations of ψ exist, e.g. ADWIN [19], HDDM-A and HDDM-W [36]; and the impact of different techniques are also assessed in Section 6.

ADWIN is, by far, the most popular choice for drift detection. It keeps a variable-length window of recently seen items, with the property that the window has the maximal length statistically consistent with the hypothesis "there has been no change in the average value inside the window". The ADWIN change detector is parameter- and assumption-free in sense that it automatically detects and adapts to the current rate of change. In the average

case, the cost of processing each instance by ADWIN is instantaneous, while in the worst case it can be of $O(\log W)$, where W is the size of the sliding window maintained in memory.

More recently, the authors in [36] proposed two variants of the Hoeffding Drift Detection Method (HDDM) detector: HDDM-A and HDDM-W. Both the former and the latter are similar to ECDD in the sense that they use moving averages to detect drifts, yet, only the latter uses an exponentially weighted procedure to provide higher importance to most recent data. In both cases, the moving averages are compared to flag concept drifts based on the misclassification rates of a classifier, where the Hoeffding Bound (see Equation 9) is used to set an upper bound to the accepted level of difference between them. In contrast to ADWIN, the complexity for both HDDM-A and HDDM-W is of $O(1)$ in the worst case.

4.2.3. Chaining decision stumps and drift detectors in a boosting scheme

The rationale behind ABFS is that boosting gives more weight to instances that are hard to classify. By intuition, these instances are either (i) located at the decision boundaries of classes, or (ii) are noise. If we work under the assumption that the labels of incoming instances are trustworthy (not noisy), decision stumps will be able to select the most important features according to these hard-to-classify instances as they naturally account for these weights during the feature selection process. In ABFS, each decision stump will be responsible for finding the feature that maximizes the merit function Ω without observing features that have been selected previously.

Since ABFS was tailored for feature selection in classification scenarios and not for actually training classifiers, we will adopt a slightly different notation from the boosting schemes presented earlier. ABFS is composed of a dynamic set of boosting units U such that each unit $u_i \in U$ is a 4-tuple in the $(ds_i, \lambda_i^c, \lambda_i^e, \psi_i)$ form, where ds_i is a decision stump, λ_i^c and λ_i^e are counters for correctly and incorrectly classified instances by ds_i and ψ_i is a drift detector. The functioning of ABFS is detailed in Algorithm 1 and is divided into 3 steps: initialization, training, and selection.

In the initialization step (lines 1-4 of Algorithm 1), ABFS instantiates both the set of boosting units U and the subset of selected features $\mathcal{F}'$ as empty lists, a candidate decision stump $ds_{candidate}$ that will gather statistics about incoming data to determine which feature to split on and select.

During the training step (lines 5-31 of Algorithm 1), ABFS updates its internal structures according to the arrival of an instance $(\vec{x}^t, y^t)$. First, the

instance weight λ and an index to store the first layer that detects a drift i_{drift} are initialized. Next, the arriving instance is sequentially traversed along all of the boosting units in U. In each boosting unit $u_i = (ds_i, \lambda_i^c, \lambda_i^e, \psi_i)$, it is verified if the decision stump is able to correctly predict the class label ($ds_i(\vec{x}^t) = y^t$), or not ($ds_i(\vec{x}) \neq y^t$). Here, AdaBoost's weighting strategy is followed (Equations 6, 7 and 8), where higher values of λ will be associated with instances that are hard to classify. It is also important to highlight that after the classification of the instance in a unit u_i, the selected feature used in its decision stump ds_i is removed from $\vec{x}$ (line 20) so that the candidate decision stump $ds_{candidate}$ is enforced to select a feature that has not been selected already by the decision stumps in U.

In addition to the definition of λ, the drift detector is fed with the classification result (1 represents an error, while 0 represents a correct classification). Therefore, each drift detector is used to keep track of the error distribution of each decision stump. The rationale here is that changes in these distributions work as a proxy to identify when the importance of a feature changes, and thus, upon the flagging of a drift, it becomes necessary to re-start the feature selection process. In practice, if a drift is flagged by ψ_i, its index i will be stored in i_{drift} so that this and the following units are removed, and that the feature selection process can self-adjust upon the new data distribution. Naturally, depending on the drift, it would be possible that multiple units flag drifts, and thus, only the first layer that detects such changes is stored.

If no changes are detected (lines 21 to 26), the candidate decision stump $ds_{candidate}$ is trained[3] with $(\vec{x}^t, y^t)$ assuming a weight λ. With the arrival of multiple instances, the candidate decision stump will reach the grace period gp, and as a result, it will eventually select a new feature f_α according to the process described earlier. When this condition holds, a new boosting unit is instantiated with this decision stump, and it is added to U. A new candidate decision stump is then created to select the next best feature, and the selected subset of features is incremented with f_α.

On the other hand, it is, a feature drift is detected, all boosting units from the index that detected the change until the end of the list are removed (loop described by lines 28 and 29), as a boosting unit u_i affected the creation of its following units $\forall u_j, j \geq i$. Next, and the classifier h is reset to allow

[3]In decision stumps, whenever an instance $(\vec{x}^t, y^t)$ is used for training with a weight λ, it means that the same instance has been observed λ times

Algorithm 1 ABFS pseudocode. We denote h to be a pointer to the classifier, $ds_{candidate}$ a candidate decision stump, $\mathcal{F}'$ the currently selected subset of features, θ a selection threshold used in decision stumps, and U the set of boosting units such that the u_i is the i-th unit and it is composed of a decision stump ds_i, a set of counters for correctly (λ_i^c) and misclassified (λ_i^e) instances, and a drift detector ψ_i.

procedure INITIALIZE(h, $\mathcal{F}, \theta$)
    $U \leftarrow \emptyset$;
    $ds_{candidate} \leftarrow$ **new** $DecisionStump(\theta)$;
    $\mathcal{F}' \leftarrow \emptyset$;
procedure TRAIN($\vec{x}^t, y^t$)
    $\lambda \leftarrow 1$;
    $i_{drift} \leftarrow -1$;
    for $i \leftarrow 1$ to $|U|$ **do**
        if $ds_i(\vec{x}^t) = y^t$ **then**
            $\lambda_i^c \leftarrow \lambda_i^c + \lambda$;
            $\lambda \leftarrow \lambda \times \frac{\lambda_i^c + \lambda_i^e}{2\lambda_i^c}$;
            Update ψ_i with 0;
        else
            $\lambda_i^e \leftarrow \lambda_i^e + \lambda$;
            $\lambda \leftarrow \lambda \times \frac{\lambda_i^c + \lambda_i^e}{2\lambda_i^e}$;
            Update ψ_i with 1;
        if ψ_i flagged a drift **and** $i_{drift} = -1$ **then**
            $i_{drift} \leftarrow i$;
            break;
        Remove from $\vec{x}$ the feature selected at ds_i;
    if $i_{drift} = -1$ **then**
        Train $ds_{candidate}$ with $(\vec{x}^t, y^t)$ assuming a weight λ;
        if $ds_{candidate}$ has selected a feature $f_\alpha \in \mathcal{F}$ **then**
            $U \leftarrow U \cup \{$**new** $BoostingUnit(ds_{candidate})\}$;
            $ds_{candidate} \leftarrow$ **new** $DecisionStump(\theta)$;
            $\mathcal{F}' \leftarrow \mathcal{F}' \cup \{f_\alpha\}$;
    else
        while $|U| > i_{drift}$ **do**
            Remove from $\mathcal{F}'$ the feature selected in $U.last()$;
            $U \leftarrow U \setminus \{U.last()\}$;
        Reset the learner h;
procedure SELECT($\vec{x}$)
    return $\vec{x}$ after selecting the features selected in $\mathcal{F}'$ and dropping the remainder of the features;

faster adaptation to the new concept. In practice, the reset of a classifier stands for the process in which its model is discarded and the learners starts to learn from scratch.

Finally, the last part is the testing step (lines 32 and 33 of Algorithm 1), ABFS filters the arriving instance $\vec{x}$ so that only the features in $\mathcal{F}'$ are selected. This instance can then be passed to the classifier with a reduced dimensionality equals to $|\mathcal{F}'|$.

4.2.4. A note on complexity analysis

The initialization step of ABFS is trivial, as it simply instantiates the required structures, which results in $\mathcal{O}(1)$. Naturally, the most computationally intensive part of ABFS is training step. In practice, the upper bound cost of ABFS is given by the loop described by lines 8 to 20 of Algorithm 1, which basically loops over all the boosting units in U, which has the same cardinality as the subset of selected features $\mathcal{F}'$. Inside this loop, the conditions described by lines 9-12 and 13-16 are mutually exclusive and have the same cost, which is basically described by the cost of the drift detector update, which is $O(\log W)$ for ADWIN and $O(1)$ for HDDM-A and HDDM-W. Next, another important aspect of ABFS occurs in line 22, where the update of a decision stump has a cost of $O(d-U)$, as it needs to loop over all the unselected features. Similarly, the condition described in line 23, where a feature is selected requires $(d-U)$ computations of Information Gain, but these are precomputed as statistics are incrementally updated, but also $(d-U)\log_2(d-U)$ computations so that the features are sorted. Yet, such computation only occurs every gp instances, and thus, the computational cost becomes $O(\frac{(d-U)\log_2(d-U)}{gp})$. On the other hand, if a drift is flagged in line 27, the cost in the worst case occurs with the removal of all features that have been selected, and thus, the cost is of $O(U)$. Given that, the overall cost for ABFS is of $O(U\log W + (d-U) + \frac{(d-U)\log_2(d-U)}{gp})$, which after simplification becomes $O(U\log W + \frac{(d-U)\log_2(d-U)}{gp})$. Also, since the average cost of ADWIN is pessimistic, one could also assume $O(U + (d-U) + \frac{(d-U)\log_2(d-U)}{gp})$ instead. Finally, the selection step is also simple, as it builds a new instance by iterating over the selected subset of features $\mathcal{F}'$, and thus, with a cost of $\mathcal{O}(U)$.

5. Evaluating Feature Selection on Data Streams

There are different factors to account for when evaluating feature selection proposals. Throughout the years, different quantitative measures, such as accuracy and scalability, and subjective ones, such as 'ease of use', have been used to highlight the efficiency of feature selectors [37]. In this paper, we define a quantitative framework to evaluate our proposal and future works on feature selection for data streams. This framework includes (i) accuracy, (ii) processing time, (iii) memory usage, (iv) selection accuracy; and (v) stability metrics. Metrics (i) through (iii) are widely used in the area as they follow traditional data stream evaluation frameworks [38] for the assessment of classifiers. Metrics (iv) and (v) are absent in streaming scenarios as both Selection Accuracy and Stability metrics were developed for batch scenarios. The following sections present these metrics and discuss their computation on data streams. Along the proposed method, the implementation of these metrics is also provided to the data stream mining community as part of the Massive Online Analysis (MOA) software [7].

5.1. Selection Accuracy

Selection Accuracy (SA) is a classifier-independent score that quantifies to what extent a selected subset of features matches the ground-truth relevant ones [39]. Given a feature set $\mathcal{F}$, its relevant subset of features $\mathcal{F}^*$ and the selected set $\mathcal{F}' \subseteq \mathcal{F}$, SA is given by Equation 10, where $\gamma \in [0; 1]$ is a weighting factor and its output is also bounded in $[0; 1]$.

$$SA(\mathcal{F}, \mathcal{F}^*, \mathcal{F}') = \gamma \overbrace{\left(\frac{|\mathcal{F}^* \cap \mathcal{F}'|}{|\mathcal{F}^*|}\right)}^{\text{RRF}} + (1-\gamma) \underbrace{\left(1 - \frac{|(\mathcal{F} \setminus \mathcal{F}^*) \cap \mathcal{F}'|}{|\mathcal{F}| - |\mathcal{F}^*|}\right)}_{\text{CCP}} \tag{10}$$

A score of $SA = 1$ corresponds to a perfect selection, where all the relevant set of features is selected, and no extraneous features are retained; whereas $SA = 0$ represents the opposite, i.e., a selection with no relevant features and all extraneous ones selected. One of the main advantages of SA is that the information on the degree to which a model has been correctly or incorrectly specified is combined into a single value, thus making the comparison between several feature selection proposals clear and classifier-independent.

The computation of SA also requires an appropriate γ. This choice is subjective and depends on how much one wants to favor accuracy over parsimony or vice-versa. Hereafter, the two components that compose the SA formula are referred as *Recall of Relevant Features* (RRF) and *Complement of Complexity Penalty* (CCP). A suitable value for γ should reflect the fact that choosing an extraneous feature is usually better than missing a relevant one, something that can be achieved by selecting γ, such that $\frac{\gamma}{|X^*|} > \frac{1-\gamma}{|X|-|X^*|}$ [39]. On the other hand, γ should not be too large, as that would result in insignificant penalties for unnecessarily complex models. Authors in [37] provided an empirical evaluation of different values of γ and claimed that 0.7 is an appropriate value since it satisfies the condition mentioned above while being sufficiently less than 1 to appropriately penalize unnecessary complexity.

The advantage of computing Selection Accuracy scores is that they express the degree to which a selection model over- or under-specifies. It is important to mention, however, that this metric is affected by the dimensionality of the problem, as the cardinality of the relevant and extraneous feature sets are taken into account directly in the formula.

Evidently, computing a Selection Accuracy score after each instance is unfeasible as its running time scales with the dimensionality d. Therefore, these scores are computed every n instances, where n is the user-given window evaluation size.

5.2. Stability

Another important trait of feature selectors that deserves attention is stability. Stability measures the sensitivity of the feature selection solution given perturbations in input data. The goal is to provide evidence that the selected features are consistent across different data samples. Therefore, stable feature selection algorithms are preferable when compared to those with highly volatile outputs. It is important to highlight that stability, however, does not relate to the performance of the selected features as it indices how *unstable* a feature selection algorithm is w.r.t. perturbations in input data, and not on how *accurate* the selection is.

In batch learning, stability is often measured by repeatedly performing feature selection over k different bootstraps of disjoint folds of a static dataset, leading to a set of feature selection results. Let $\mathcal{F}_i'$ be the subset of features selected over the i^{th} samples of instances extracted from a static dataset. The stability of a feature selection algorithm can be computed by

averaging the similarity coefficient ϕ for each of the possible pairs of $(\mathcal{F}'_i, \mathcal{F}'_j)$ of selected features, as stated in Equation 11.

$$S = \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \phi(\mathcal{F}'_i, \mathcal{F}'_j) \tag{11}$$

Although several similarity metrics (ϕ) for stability do exist, until recently, there has not been an agreement on which one to use [40]. Recently, the work of [41] has provided insights on the main properties a stability measure should possess. First, it should be **fully defined**, as a stability measure should be defined regardless of the selected feature sets and respective lengths. Also, it must have pre-defined **upper and lower bounds** to facilitate the comparison between selectors. The third trait is the relationship called **Deterministic Selection $\Leftrightarrow$ Maximum Stability**: if a selector always selects the same k features, then it should present maximum stability. The converse should also hold, i.e., the stability is maximum only if the selection is deterministic. Finally, it should have **chance correction**, so if the selector is random, its stability should be 0. Even though these traits are rather simple, the analysis conducted in [41] shows that most approaches do not fulfill these criteria. More importantly, in the same work, authors show that the Pearson coefficient overcomes this problem. This coefficient is given by Equation 12, where $d = |\mathcal{F}|$, $r_{i,j} = |\mathcal{F}'_i \cap \mathcal{F}'_j|$, and $v_i = \sqrt{\frac{k_i}{d}\left(1 - \frac{k_i}{d}\right)}$ with $k_i = |\mathcal{F}'_i|$.

$$\phi_{\text{Pearson}}(\mathcal{F}'_i, \mathcal{F}'_j) = \frac{r_{i,j} - \frac{k_i k_j}{d}}{d\, v_i v_j} \tag{12}$$

Finally, the last challenge to be tackled here regards how stability scores can be calculated in streaming scenarios. A naive proposition to select samples of a stream would be to adopt a landmark windowing scheme, where every m instances would be grouped and inputted to a feature selection algorithm. After performing feature selection over n batches, the stability could then be computed. The major drawbacks of such proposal are that it assumes that (i) the feature selection algorithm is not dynamic and that (ii) the underlying data distribution is static since the selected subset of features for each batch is expected to be the same. As discussed in the previous sections of this paper, none of the latter assumptions hold or are preferable, thus, evaluating with landmarks is not reasonable.

To overcome such limitations, we propose to adapt the Prequential Cross-Validation (Preq-CV) scheme presented in [42] for stability computation. Following the original Preq-CV, three different k-fold approaches can be used to evaluate the stability of a feature selector: **cross-validation**, **split-validation** and **bootstrapping**. The first strategy updates $(k-1)$ folds, while the second updates only one of the k folds. Finally, the bootstrapping approach updates each of the k folds using a weight obtained with a Poisson distribution with a parameter $\lambda = 1$. In this scheme, the probability of an instance being used in each fold is approximately two thirds, as $P[x > 0] = 1 - P[x = 0] = 1 - \frac{e^{-1}}{x!} \approx 63\%$ and the same value depicts the intersection of instances used in each pair of folds.

Similarly to Selection Accuracy, calculating a Stability score is computationally intensive as it requires $\frac{k(k-1)}{2}$ pairwise similarity computations, and thus, these are only calculated according to an user-given evaluation window size. Also, even though this score is calculated only every n instances, it is important to notice that the actual feature selection process occurs incrementally, which causes the process to be different from performing batch feature selection and conventional stability computation.

6. Analysis

In this section we analyze the proposed method in light of the evaluation metrics presented in Section 5 in both synthetic and real-world scenarios. First, we report the experimental protocol adopted, followed by the discussion on synthetic experiments, and finally, on real-world datasets.

6.1. Experimental Protocol

In the following sections, ABFS is applied to different classifiers on both synthetic and real-world data. In Table 1 we show the synthetic experiments conducted, including the number of instances, average number of relevant features and number of irrelevant features appended. Regarding synthetic experiments, **AGR** represents the AGRAWAL [43] generator and **AN** is the Asset Negotiation generator [44]. **BG1**, **BG2** and **BG3** are synthetic generators based on binary features proposed in [45] that were recently used to synthetize feature drifts in [17]. Finally, the Random Tree Generator (**RTG**) was used to create more complex concepts (where the number of relevant features is bigger), while **SEA** [46] concepts depend on only 2 features. All of the aforementioned synthetic experiments are reported in different

Table 1: Details on the synthetic experiments conducted. Each of the synthetic experiments has been repeated where drifts were abrupt and gradual.

Experiment	Type	# of Instances	Avg. # of Relevant Features	# of Irrelevant Features Appended	# of Redundant Features
AGR	Synthetic	200,000	3.66*	100/200/500	–
AN	Synthetic	200,000	2	100/200/500	–
BG1	Synthetic	200,000	3	100/200/500	–
BG2	Synthetic	200,000	3	100/200/500	–
BG3	Synthetic	200,000	3	100/200/500	–
RTG	Synthetic	200,000	6	100/200/500	15/30/40
SEA	Synthetic	200,000	2	100/200/500	–

* - The AGR experiment has three concepts, such that the first has 4 relevant features, the second only 3, and the last another 4.

variants. First, all synthetic data streams have 2 equally distributed drifts along the stream, i.e. each occurring at 66,666 and 133,333 instances, and each of these drifts is gradual with a window of 10,000 instances. As a result, drifting regions of synthetic experiments are located at 61,666-71,666 and 128,333-138,333 [7]. Second, a different number of irrelevant features were appended (100, 200 and 500), such that half are numeric and the other half categorical. The proposed strategy to append irrelevant features is to increment the attribute set $\mathcal{F}$ of a data stream with numeric or categorical attributes. In the first case, values for a numeric attribute are sampled from a uniform distribution bounded in $[0; 1]$, with no regard to the instance outcome. The procedure for categorical features is similar, where new irrelevant attributes possess m different values, such that m is a user-given value and the probability of each partition being used in an instance equals $\frac{1}{m}$. Also, the RTG experiment was changed so that redundant features were also added. Redundant features are synthesized by copying the value of another feature with 95% probability, while the remainder 5% result in a value drawn from a uniform distribution of the other possible values for that specific feature.

Regarding real-world datasets, depicted in Table 2, it is impossible to tell whether and when drifts occur. Nevertheless, five different datasets were still used to verify how ABFS behaves when applied to real-world datasets with a reasonable number of features. The first is the Forest Covertype [47] dataset (**COVTYPE**), which is widely used to evaluate data stream learning algorithms. This dataset represents the problem of determining the forest covertype given characteristics (features) of forest areas. Another dataset used was the Internet Advertisements (**IADS**) [48], which targets the classification of whether images on a website are advertisements or not. Next,

Table 2: Details of the real-world datasets used during the experiments.

Experiment	Type	# of Instances	# of Features	Feature Types	Reference
COVTYPE	Real-world	581,012	54	Mixed	[47]
IADS	Real-world	3,279	1,558	Numeric	[48]
NOMAO	Real-world	34,465	118	Mixed	[49]
PAMAP2	Real-world	1,942,872	52	Numeric	[50]
SPAM	Real-world	9,324	39,917	Binary	[51]

the NOMAO dataset (**NOMAO**) [49] was introduced during the ECML-PKDD'12 challenge as part of a deduplication task for determining whether two spots should have their data merged or not. The Physical Activity Monitoring dataset (**PAMAP2**) contains data of 18 different physical activities performed by 9 subjects wearing 3 inertial measurement units and a heart rate monitor [50]. The goal of this dataset is to determine which activity each subject is performing over time, such as walking, cycling, playing soccer, and so forth. Another traditional dataset is the Spam Corpus (**SPAM**), which is the result of a text mining process of an e-mail dissemination system which targeted initially the determination of whether each e-mail was spam or not [51]. It is also important to highlight that other datasets that are commonly used in data stream studies, e.g., Pokerhand and Electricity; were not used here due to the small number of features or because traditional feature selection has already been applied before these datasets were made publicly available.

The latter experiments are used to benchmark ABFS with different types of classifiers. In this testbed, we verify how ABFS works in conjunction with Naive Bayes, k-Nearest Neighbors (kNN), Hoeffding Tree [18], and Hoeffding Adaptive Tree [19] classifiers. All of the classifiers parameters' were set following the default values used in the Massive Online Analysis framework, except for the window size in kNN, which was set to 500 to make it viable as larger window sizes impact on larger processing times as the number of distance computations per instance grows according to the number of instances buffered. The parameters for ABFS will be discussed in Sections 6.2 and 6.3 as multiple combinations have been empirically tested, and as the characteristics of synthetic and real-world experiments strongly vary, different sets of parameters have been adopted. Also, due to the lack of techniques that dynamically select features during the processing of data streams, we compare our method against a theoretical upper bound hereafter referred to

as the "oracle"[4], which always selects the relevant features and ignores the irrelevant ones resulting in $SA = 1$. Also, every time a change in the relevant subset of features is detected, the classifier is reset. We refer to this selector as ORACLE in the following experiments.

Evaluation of the classifiers with and without the proposed method has been conducted regarding accuracy, processing time, and memory consumption. Accuracy is measured following the Prequential test-then-train [38] procedure, processing time is the time that the methods spend in the CPU (in seconds), and memory consumption is given in RAM-Hours, where 1 RAM-Hour corresponds to 1 GB of memory spent in 1 hour of processing. We also use Selection Accuracy and Stability metrics to evaluate ABFS and show how accurate it is during the feature selection process and how stable this method is given perturbations in the input data. All of the above-cited metrics are computed every 5% of the experiment.

All experiments reported in this paper have been coded and conducted on the Massive Online Analysis (MOA) software. The results were obtained in a computer with 40 Intel(R) Xeon(R) CPU E5-2660 v3 2.60GHz cores and with 64 GB of RAM devoted to the experiments. Statistical tests have been conducted with Wilcoxon's test [52], or a combination of Friedman [53] and Nemenyi's [54] hypothesis tests following the protocol of [55], according to the number of hypotheses being tested. In the sections below, the synthetic experiments have been performed 30 times by changing the random seed in the data generation process and by randomly shuffling real-world datasets. The results of the statistical reports are then performed with the average results obtained from these executions and under a 95% confidence level.

6.2. Synthetic experiments

In this section, we show how different classifiers behave with and without ABFS in synthetic experiments. In contrast to real-world datasets, synthetic experiments allow greater flexibility. As depicted in the previous section, we target the dimensionality aspect of data streams, where 100, 200, and 500 features are appended to each of the experiments. The rationale behind this process is to verify how each learner and ABFS behave when noisy features are added to a data stream regarding accuracy, processing time, and memory consumption.

[4]The term *oracle* is borrowed from dynamic selection methods in ensemble learning.

We start this section by investigating how different values for each of the main parameters of ABFS impact final classification accuracy and selection accuracy rates. Our investigation targets the parameters and values detailed below, whereas each one will be analyzed individually w.r.t. classification accuracy and selection accuracy metrics, and finally, the best parametrization will be chosen as the default one. In practice, this analysis will be conducted across all experiments, meaning that we are trying to find a good parametrization that works reasonably well across different datasets, which is different from tuning our method to each experiment individually. The parameters analyzed are as follows:

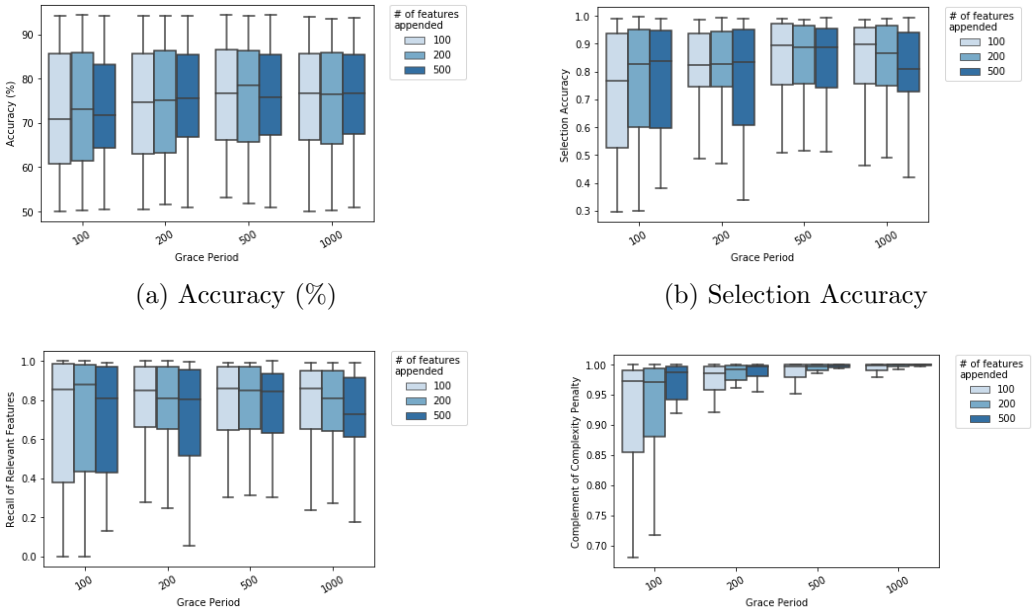
- **Grace period (gp)**: This parameter controls how "fast" the candidate decision stump will attempt to select a feature. Smaller values of gp allow the decision stump to branch quicker, yet, the sample distribution obtained during this grace period is expected to be less precise compared to the samples obtained with greater grace periods. The values of 100, 200, 500 and 1,000 were tested for this parameter.

- **Selection threshold (θ)**: This parameter determines the minimum value of Ω so that a feature is selected. In practice, if the candidate decision stump determines that f_α is the most appropriate feature to be selected, it will only select it if $\Omega(f_\alpha) \geq \theta$. Three different values were tested for this parameter: 0.01, 0.05 and 0.1.

- **Drift detector (ψ)**: this parameter determines which type of drift detector is used in each boosting unit. Three different competitive methods have been tested here, namely ADWIN [19], HDDM-A and HDDM-W [36].

As a result, 36 different configurations for ABFS were tested in association with Naive Bayes, KNN, Hoeffding Tree and Hoeffding Adaptive Tree classifiers, culminating in a total of 144 configurations per stream, which were then repeated 30 times by changing the random seed of the experiments. Below, we use box-plots to report the results obtained across different classifiers, streams, and parameter values.

In Figure 1 we see the results obtained by different grace period values across experiments grouped by the number of irrelevant features appended. Even though no clear difference is observable across different grace period concerning accuracy (Figure 1a), the highest results are obtained when the



(a) Accuracy (%)

(b) Selection Accuracy

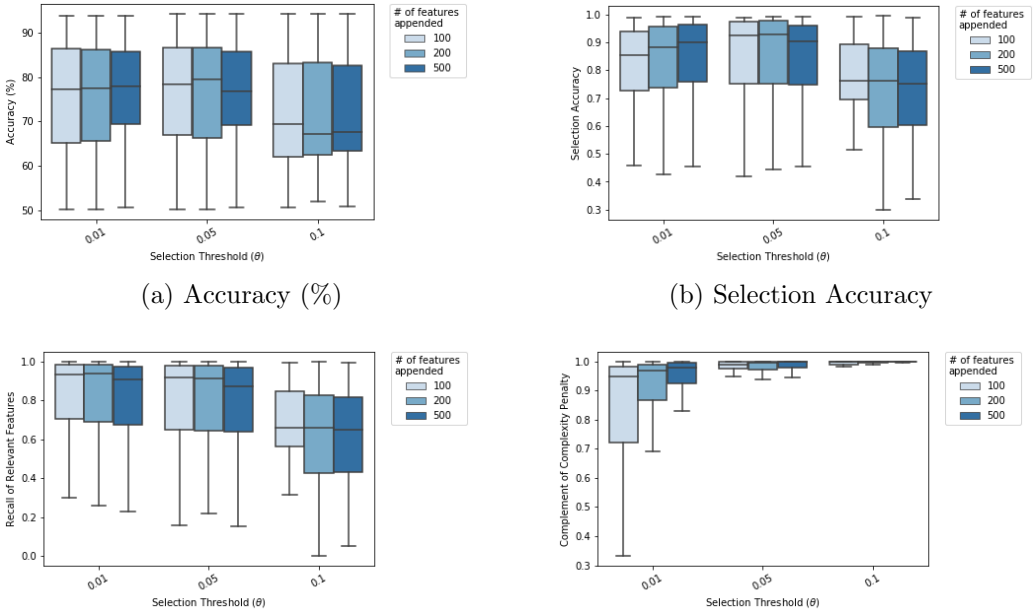
(c) Recall of Relevant Features

(d) Complement of Complexity Penalty

Figure 1: Results obtained across different grace period values.

grace period is set to either 500 or 1000, showing that higher grace periods are preferable. Nevertheless, the results observed in Figures 1b, 1c and 1d show the results for Selection Accuracy and its components, where $gp = 500$ is the most stable and preferred value regardless of the experiment dimensionality in terms of Selection Accuracy and Recall of Relevant Features.

Naturally, an important aspect here is the high variance observed in the results, as the rates go from 50% up to 90% or more. This high variance occurs mainly because of the BG3 and RTG experiments. If we analyze the classification and selection accuracy rates, depicted in Figures 5a and 5b, respectively; we observe that these experiments result in rates that are much lower than the rest. The explanation is that these concepts are much more complex than the others, as BG3 is a XOR-like classification problem [45], and RTG has complex interactions between the features [17]. In practice, the Selection Accuracy rates obtained in the BG3 and RTG experiments are below the expected baseline of 0.7. This is relevant since if one selects all features, it would incur in a SA baseline of 0.7, regardless of the selection of the extraneous features since $\gamma = 0.7$.



(a) Accuracy (%)

(b) Selection Accuracy

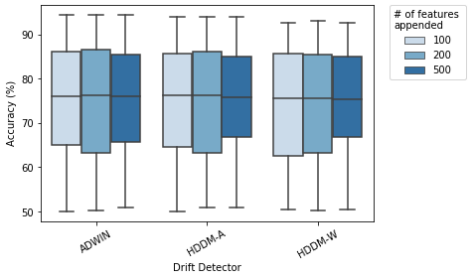
(c) Recall of Relevant Features

(d) Complement of Complexity Penalty

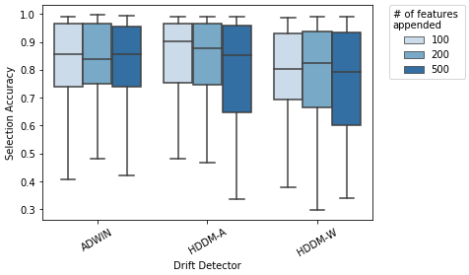
Figure 2: Results obtained across different selection threshold (θ) values.

In Figure 2 we conduct a similar analysis for the selection threshold (θ) parameter. From the accuracy results shown in Figure 2a, the three threshold values behave similarly in terms of variance, yet smaller values, i.e., 0.01 and 0.05, show higher accuracy rates. When analyzing Selection Accuracy rates (Figure 2b) and its components (Figures 2c and 2d), we observe a trade-off between θ and the accuracy of the selection process. In practice, higher threshold values are more 'selective' as less irrelevant features are selected (higher Complement of Complexity Penalty rates), while it misses the relevant ones (lower Recall of Relevant Features values). Overall, both $\theta = 0.01$ and $\theta = 0.05$ seem reasonable as they are able to correctly identify relevant features in all the tested dimensionalities (Figure 2c), while reasonably ignoring the irrelevant ones (Figure 2d).

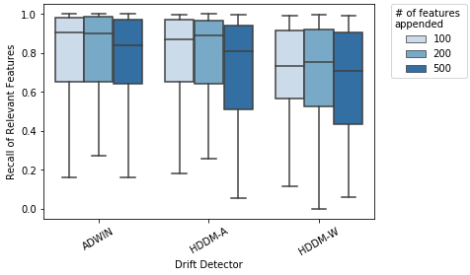
Finally, the results for different drift detectors are reported in Figure 3. Regarding classification accuracy, depicted in Figure 3a, the use of different drift detectors barely impact the overall results regardless of the dimensionality of the experiments. Yet, when analyzing the results for Selection Accuracy and its components (Figures 3b through 3d), we observe that HDDM-A is



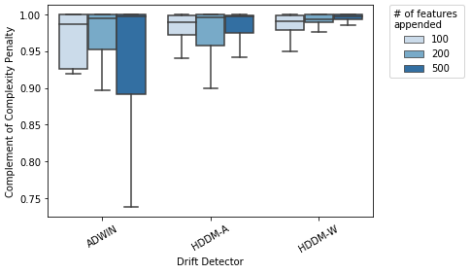
(a) Accuracy (%)



(b) Selection Accuracy



(c) Recall of Relevant Features

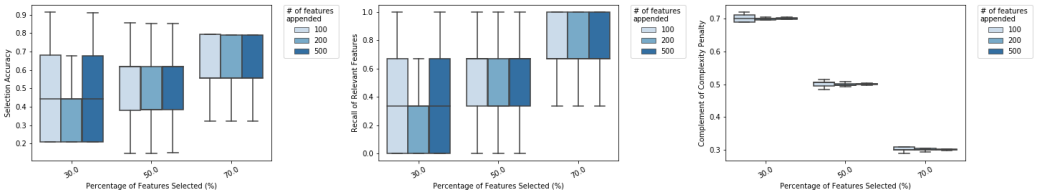


(d) Complement of Complexity Penalty

Figure 3: Results obtained across different drift detectors.

slightly better in overall Selection Accuracy rates, while experiments with ADWIN are better at retaining the relevant features, and HDDM-W is the best performing in terms of ignoring the irrelevant features. It is important to note that even though these drift detectors are not part of the feature selection process, they indirectly impact the entire process, as they may flag drifts at different moments, which cause the feature selection process adapt itself at different regions of the stream. As a result, ABFS becomes more or less precise according to each of the metrics mentioned above depending on the drift detector being used.

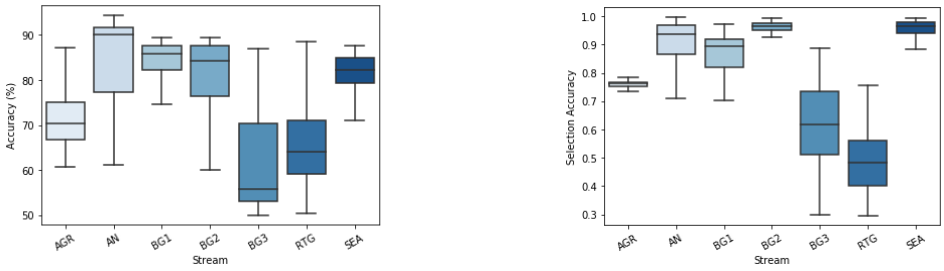
To determine whether these results are reasonable, we also report in Figure 4 the results obtained by a random feature selection process. These results were obtained across different 30 executions, such as the remainder of the experiments. In this figure, we report the selection accuracy rates and its components across different proportions where different proportions of the features available in the dataset are randomly selected. From these results, we observe that the Selection Accuracy rates obtained are quite volatile,



(a) Selection Accuracy (b) Recall of Relevant Features (c) Complement of Complexity Penalty

Figure 4: Results obtained by ABFS and a random feature selection algorithm.

mostly since the recall of relevant features highly varies, whereas the rates for the complement of complexity penalty are stable since the number of irrelevant features is much higher than the relevant ones. When these results are compared to the results given in Figures 1, 2, and 3, it becomes evident that the results are significantly better than random guessing for feature selection on all of the components of Selection Accuracy computation.



(a) Accuracy (%) (b) Selection Accuracy (%)

Figure 5: Classification and selection accuracy rates obtained per experiment.

Naturally, since the goal of classification is to achieve the highest classification rates possible, we show in Figure 6 the 10 best-ranked configurations of ABFS. In this figure, we corroborate the values identified in the previous analyses, as the best performing parametrization, in average, for ABFS in synthetic experiments was ($gp = 500, \theta = 0.01, \psi = \text{ADWIN}$), and this configuration is assumed for comparisons against the base learners and the ORACLE feature selector. We highlight, however, that this configuration is not the optimal one for each of the experiments conducted, and thus, these parameters' values must not be assumed to be the result of a tuning process.

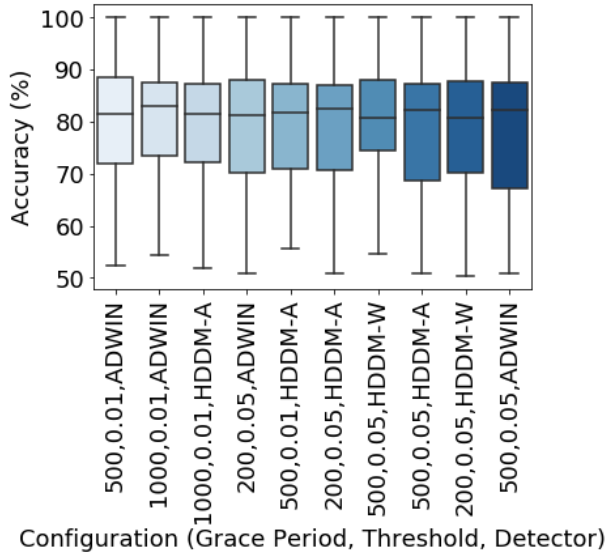
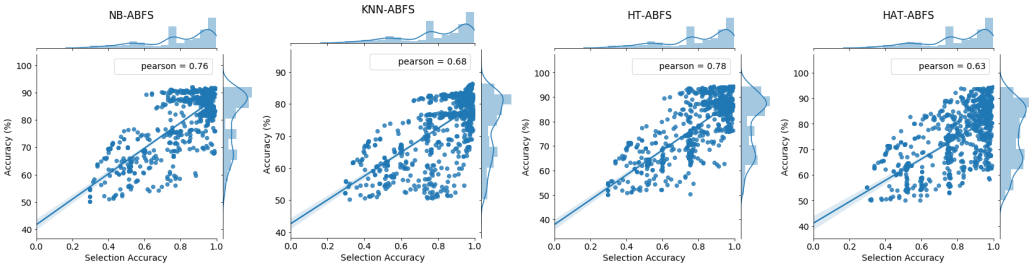


Figure 6: Accuracy rates (%) obtained across the 10-best ranked ABFS configurations in synthetic experiments.

Accuracy rates. The accuracy rates obtained by the classifiers without feature selection, with the ORACLE selector and ABFS are reported in Tables 3, 4 and 5. Focusing on the accuracy rates obtained in experiments with 100 irrelevant features, we observe that ABFS can improve the classification rates of the NB, KNN and HT classifiers in all scenarios. In average, the improvements for NB, KNN and HT classifiers are of 7.67%, 11.95%, and 4.64%, respectively. On the other hand, the combination of ABFS with the HAT classifier results in accuracy decreases in most scenarios with an average of -5.76%, which shows that combining two adaptive approaches that concomitantly select features jeopardizes the learning process. The comparison of the ABFS results against the ORACLE show that there is still room for improvements and other feature selection methods for data streams since the ORACLE feature selector overcomes ABFS in 1.46% for the NB classifier, 8.18% for KNN, 0.03% for HT, and 5.83% for HAT.

In Figure 7 we report the relationship between Selection Accuracy and classification accuracy rates obtained by different learners in the different experiments. From this visualization, we see that: (i) different learners benefit differently when fed with the same subset of features, (ii) there is an interesting relationship between achieving higher selection accuracy rates and classification accuracy, and finally (iii) that most of the results obtained by ABFS are located in regions of high Selection Accuracy and classification accuracy rates, thus showing the efficacy of the proposed method.

The results obtained in experiments with 200 and 500 irrelevant features,



(a) Naive Bayes (b) K-Nearest Neighbors (c) Hoeffding Tree (d) Hoeffding Adaptive Tree

Figure 7: Relationship between Selection Accuracy and classification accuracy rates across different classifiers with ABFS. The results plotted in this figure report the rates obtained with different ABFS configurations and stream dimensionalities (100, 200, and 500).

reported in Tables 3 and 4, follow the same behavior as noticed in Table 3, where NB, KNN, and HT classifiers benefit from ABFS, while HAT has its accuracy rates decreased. As an important disclaimer, we highlight that the experiments using 100, 200, and 500 features are not the same, as each one is created with a different concept generator scheme, and thus, the ORACLE results differ. Analyzing the results in quantitative terms, accuracy changes of 8.25% and 7.99% for NB, 11.11% and 9.24% for KNN, 5.11% and 5.21% for HT, -3.19% and -2.35% for HAT, are observed in experiments with 200 and 500 irrelevant features, respectively. Similarly as before, the ORACLE method overcomes ABFS in 1.15% for the NB classifier, 10.68% for KNN, 0.03% for HT, and 3.66% for HAT, when focusing on the experiments with 200 irrelevant features. The rates obtained with 500 irrelevant features are also similar, with 1.18% for NB, 15.60% for KNN, 0.03% for HT, and 3.00% for HAT.

Table 3: Average accuracy (%) obtained by different classifiers and feature selection methods in experiments with 100 irrelevant features. Results in bold highlight the best accuracy rates per classifier and underlined results are the best across learners and selectors.

Experiment	NB	NB-ORACLE	NB-ABFS	KNN	KNN-ORACLE	KNN-ABFS	HT	HT-ORACLE	HT-ABFS	HAT	HAT-ORACLE	HAT-ABFS
AGR	67.27	**77.13**	76.10	50.62	**85.38**	73.30	77.38	**85.95**	85.90	91.15	**91.19**	81.38
AN	81.52	**92.71**	91.20	64.57	**75.59**	70.05	92.92	**93.66**	93.63	94.33	**94.36**	93.61
BG1	80.17	**88.78**	86.96	70.94	**81.48**	76.92	86.41	**88.82**	88.79	89.09	**89.17**	89.13
BG2	74.11	**89.51**	88.56	57.63	**83.18**	75.98	79.91	**88.58**	88.55	88.06	**88.08**	84.76
BG3	55.84	**61.91**	60.94	53.11	**75.26**	65.46	70.46	**72.18**	72.17	85.61	**85.63**	67.31
RTG	59.22	**66.83**	65.93	54.57	**68.07**	55.64	66.09	**74.53**	74.48	88.56	**88.59**	80.01
SEA	79.15	**84.35**	81.33	59.14	**82.52**	76.90	84.05	**86.25**	86.22	86.41	**86.68**	86.66

Table 4: Average accuracy (%) obtained by different classifiers and feature selection methods in experiments with 200 irrelevant features. Results in bold highlight the best accuracy rates per classifier and underlined results are the best across learners and selectors.

Experiment	NB	NB-ORACLE	NB-ABFS	KNN	KNN-ORACLE	KNN-ABFS	HT	HT-ORACLE	HT-ABFS	HAT	HAT-ORACLE	HAT-ABFS
AGR	67.19	**76.82**	75.67	50.55	**85.38**	61.62	77.53	**86.10**	86.06	91.07	**91.12**	81.39
AN	81.58	**92.22**	91.21	61.15	**75.75**	69.67	92.53	**93.88**	93.85	94.36	**94.41**	93.12
BG1	79.72	**87.89**	86.81	65.87	**81.56**	75.38	85.87	**88.81**	88.76	89.11	**89.23**	89.22
BG2	74.11	**90.09**	89.09	55.41	**83.00**	69.01	79.01	**89.13**	89.11	88.02	**89.17**	89.12
BG3	55.47	**61.52**	60.52	51.90	**75.32**	65.29	68.56	**71.12**	71.11	86.22	**86.24**	78.70
RTG	59.22	**68.02**	66.18	55.78	**66.47**	57.66	63.25	**70.00**	69.98	91.01	**91.04**	84.07
SEA	79.04	**85.63**	84.63	56.79	**82.51**	76.58	82.52	**86.23**	86.18	84.69	**86.60**	86.56

Table 5: Average accuracy (%) obtained by different classifiers and feature selection methods in experiments with 500 irrelevant features. Results in bold highlight the best accuracy rates per classifier and underlined results are the best across learners and selectors.

Experiment	NB	NB-ORACLE	NB-ABFS	KNN	KNN-ORACLE	KNN-ABFS	HT	HT-ORACLE	HT-ABFS	HAT	HAT-ORACLE	HAT-ABFS
AGR	66.97	**76.12**	75.78	50.41	**85.38**	52.49	75.60	**85.07**	85.02	90.80	**90.85**	81.44
AN	81.56	**92.33**	91.31	56.99	**75.75**	70.12	92.74	**93.61**	93.58	94.39	**94.40**	93.00
BG1	79.49	**87.98**	86.81	60.18	**81.43**	73.53	85.84	**88.74**	88.71	89.04	**89.29**	89.25
BG2	73.84	**89.89**	88.19	53.28	**83.18**	63.93	77.57	**85.84**	85.79	87.50	**88.50**	88.48
BG3	55.85	**60.12**	59.72	50.85	**75.20**	54.61	67.51	**71.94**	71.91	85.46	**85.48**	83.22
RTG	68.08	**77.67**	76.59	58.84	**74.99**	62.50	70.17	**77.83**	77.81	89.93	**89.96**	82.14
SEA	79.05	**84.95**	82.39	53.90	**82.43**	71.99	82.28	**85.36**	85.34	81.81	**85.02**	84.98

Computational resources. In addition to the comparisons conducted in terms of classification accuracy and selection accuracy, it is also important to verify if the introduction of ABFS in the data stream classification process is not computationally prohibitive, or in the best case scenario, improves the processing time and memory consumption rates of learners. For the sake of brevity, we only compare the computational resources required for the biggest experiments, i.e., those with 500 irrelevant features, as they are the most computationally intensive. In Table 6, we report the processing times obtained by classifiers both with and without ABFS. From these results, we observe that the introduction of ABFS impacts different learners differently. For instance, NB has its processing times significantly improved in all scenarios, while KNN has the opposite behavior. Regarding KNN, such processing time decreases are expected as the complexity of computing distances between instances with reduced dimensionality are faster than computing distances with the entire set of features. It is also worthy to highlight that even decision trees have their processing times decreased in a handful of scenarios.

Similarly, the results obtained for memory consumption are reported in Table 7. Regarding the NB and HAT classifiers, the introduction of ABFS introduces significant overheads in memory consumption rates, while KNN highly benefits from it, as the buffered instances are stored in reduced dimensionality. Next, the results for the HT classifier show that in most cases ABFS does introduce a relatively small overhead, yet, some improvements are also observed. The overheads observed for memory consumption are expected since all learners (with the exception of KNN) still allocate memory assuming the existence and availability of the original feature set, but is trained only on the selected ones. This is an implementation gap that should be further examined in future implementations.

Finally, it is important to highlight that when analyzing the computational resource metrics mentioned above, the technology in which the method is implemented on is important. As the implementation of ABFS evaluated here has been performed on the Massive Online Analysis framework, it is important to highlight that when a classifier is fed with an instance for training, it still loops over all the original feature set $\mathcal{F}$ and not only over the selected subset $\mathcal{F}'$. As a result, the overall processing times are expected to be incremented, but this behavior may change if the base learners allow sparse data representations. Similarly, the NB, HT and HAT classifiers still instantiate data structures for each of the original features in $\mathcal{F}$ and not only for $\mathcal{F}'$,

Table 6: Average processing time (s) obtained by different classifiers and feature selection methods in experiments with 500 irrelevant features. Results in bold highlight the smallest times per classifier and underlined results are the best across learners and selectors.

Experiment	NB	NB-ABFS	KNN	KNN-ABFS	HT	HT-ABFS	HAT	HAT-ABFS
AGR	**32.03**	58.81	818.80	**525.66**	178.27	**98.85**	**165.19**	215.49
AN	**69.53**	117.76	1827.45	**1023.79**	392.34	**267.95**	395.43	**367.15**
BG1	**13.08**	51.10	612.49	**570.14**	**44.27**	75.90	**45.93**	80.55
BG2	**11.55**	30.86	606.38	**387.84**	**45.24**	52.23	**51.92**	59.97
BG3	**12.78**	38.11	596.88	**437.59**	**43.52**	45.00	**50.33**	70.55
RTG	**52.23**	71.95	698.87	**422.32**	187.92	**148.03**	145.52	**117.78**
SEA	**27.01**	28.86	656.02	**259.64**	85.78	**53.86**	195.08	**79.92**

Table 7: Average RAM-Hours (GB-Hour) obtained by different classifiers and feature selection methods in experiments with 500 irrelevant features. Results in bold highlight the smallest memory consumption rates per classifier and underlined results are the best across learners and selectors.

Experiment	NB	NB-ABFS	KNN	KNN-ABFS	HT	HT-ABFS	HAT	HAT-ABFS
AGR	$\mathbf{1.76 \times 10^{-6}}$	7.82×10^{-4}	6.63×10^{-3}	$\mathbf{1.23 \times 10^{-4}}$	$\mathbf{1.21 \times 10^{-3}}$	1.45×10^{-3}	$\mathbf{4.05 \times 10^{-4}}$	3.68×10^{-3}
AN	$\mathbf{7.87 \times 10^{-6}}$	8.33×10^{-4}	7.04×10^{-3}	$\mathbf{5.32 \times 10^{-4}}$	4.73×10^{-3}	$\mathbf{2.60 \times 10^{-3}}$	$\mathbf{2.65 \times 10^{-3}}$	4.53×10^{-3}
BG1	$\mathbf{6.03 \times 10^{-7}}$	2.40×10^{-4}	2.65×10^{-3}	$\mathbf{9.01 \times 10^{-5}}$	$\mathbf{7.84 \times 10^{-5}}$	4.21×10^{-4}	$\mathbf{3.69 \times 10^{-5}}$	4.04×10^{-4}
BG2	$\mathbf{5.32 \times 10^{-7}}$	8.40×10^{-5}	1.04×10^{-3}	$\mathbf{8.92 \times 10^{-5}}$	$\mathbf{8.53 \times 10^{-5}}$	1.71×10^{-4}	$\mathbf{5.14 \times 10^{-5}}$	1.81×10^{-4}
BG3	$\mathbf{5.90 \times 10^{-7}}$	3.36×10^{-4}	3.77×10^{-3}	$\mathbf{8.80 \times 10^{-5}}$	$\mathbf{7.46 \times 10^{-5}}$	4.21×10^{-4}	$\mathbf{4.47 \times 10^{-5}}$	6.88×10^{-4}
RTG	$\mathbf{2.97 \times 10^{-6}}$	2.69×10^{-4}	1.62×10^{-3}	$\mathbf{1.03 \times 10^{-4}}$	2.24×10^{-3}	$\mathbf{1.32 \times 10^{-3}}$	$\mathbf{4.15 \times 10^{-4}}$	6.58×10^{-4}
SEA	$\mathbf{1.61 \times 10^{-6}}$	5.89×10^{-4}	5.16×10^{-3}	$\mathbf{9.53 \times 10^{-5}}$	$\mathbf{3.13 \times 10^{-4}}$	1.18×10^{-3}	$\mathbf{1.03 \times 10^{-3}}$	1.95×10^{-3}

and as a result, the introduction of ABFS negatively impacts the memory consumption rates of these learners.

Stability. Determining how ABFS behaves when fed with different inputs of data is another important trait that must be analyzed. In Figure 8 we report the stability rates obtained by ABFS across synthetic experiments, following bootstrap-, split- and cross-validation schemes in a 10-fold validation environment. At first, it is important to highlight that the stability rates achieved by ABFS vary according to the experiment conducted, but more importantly, according to the validation process adopted. Naturally, the highest stability rates are achieved using the cross-validation scheme, as 9 out of the 10 folds are updated with the arrival of each instance, thus making the selection process much more uniform across the folds. The same rationale can be applied to explain the rates obtained by the boostrap-validation experiments, as each instance is used to update the feature selection process allocated in each fold approximately 66% of the times. Finally, the results obtained with the split-validation process are the lowest, as only 1 out of the 10 folds are updated with the arrival of each instance. We also highlight at this point that it is hard to tell how 'stable' ABFS is due to the lack of

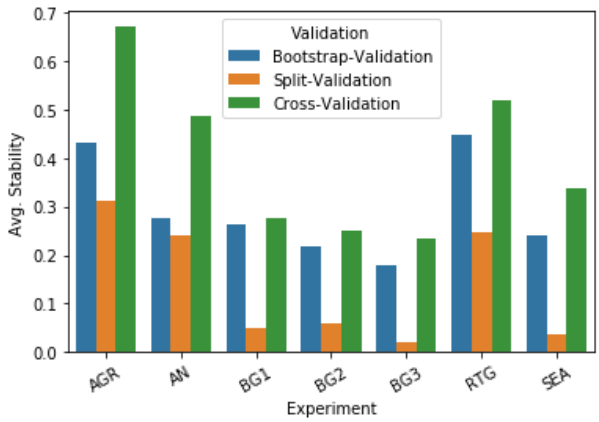
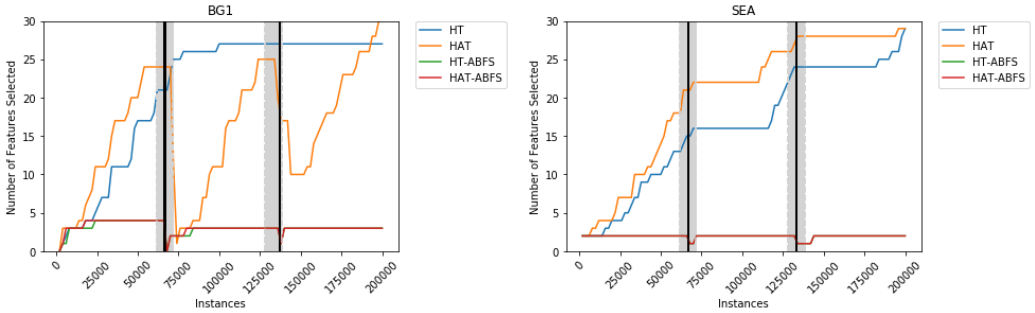


Figure 8: Stability results obtained for synthetic data streams.

competing techniques, and as a result, the results reported here may serve as baselines for future works on the area.

Number of selected features. To finalize the discussion on synthetic experiments, we highlight two examples on the number of selected features over the processing of streams. In Figure 9 we show the number of features that were selected by HT and HAT classifiers with and without ABFS in BG1 and SEA experiments. We target these experiments as these are cases where the overall accuracy of tree-based learners has improved with ABFS. In Figure 9a we observe that the number of features used by the Hoeffding Tree (HT) classifier continuously increases, while the Hoeffding Adaptive Tree (HAT) can discard features when drifts occur, which are the areas highlighted in the plot. It is important to remember that in this experiment, only 3 features are relevant, and thus, both HT and HAT are rapidly growing and selecting features as new instances become available. In contrast to this behavior, we observe that the same classifiers with ABFS selects up to 4 features and quickly flags and adapts to drifts, which are marked as a vertical line in the plot. A different behavior is observed in Figure 9b, where HAT has the same behavior of a conventional incremental HT, as the number of selected features continuously increases, showing the HAT is unable to discard features that become irrelevant after drifts. Again, ABFS shows a limited number of selected features, which result in much smaller decision trees, thus improving their readability and understandability.

ABFS on scenarios with a high number of relevant features.



(a) BG1 (b) SEA

Figure 9: Number of features selected and used by decision tree models with and without ABFS in experiments with 500 irrelevant features. Grayed areas are drifting regions and vertical black lines depict the moments where drifts have been flagged by ABFS. The drift moments for HT-ABFS and HAT-ABFS match as ABFS is classifier independent.

The experiments conducted and discussed during this section show a small number of relevant features. Therefore, it becomes of interest to determine how ABFS behaves when confronted with data stream scenarios where a relatively high number of features is required for classification. To achieve this, we conducted a variation of the RTG experiment, where 100 relevant features out of the 500 available are relevant. The results obtained by the 3 best-ranked ABFS configurations presented in Figure 6 are reported in Table 8, whereas the accuracy rates obtained by classifiers using all the 500 features available are given in Table 9.

First, it is important to note the selection accuracy rates obtained by ABFS, which are competitive with the results obtained in the previous experiments, only a smaller number of features was relevant. This, accompanied by the number of features selected, shows that ABFS is able to scale to scenarios where more features are required for the classification task. In terms of classification accuracy rates, and comparing the rates shown in Tables 8 and 9, we are able to see that ABFS continues to improve NB and KNN learning schemes, whereas decision trees marginally benefit from the features selected by ABFS or even have their accuracy rates prejudiced.

6.3. Real-world datasets

As conducted in the synthetic experiments, the different configurations of ABFS were ranked across all the real-world experiments according to the

Table 8: Results obtained by different configurations of ABFS and base learners in the RTG experiment with 100 relevant features. Classification accuracy improvements compared to the results obtained by the same classifier when using all features are reported in bold.

ABFS Configuration			Classifier	Avg. Accuracy (%)	Avg. SA	Avg. RRF	Avg. CUCP	Avg. # of features selected
ψ	gp	θ						
ADWIN	500	0.01	NB	**64.13%**	0.735	0.66	0.91	102
ADWIN	500	0.01	KNN	**58.58%**				
ADWIN	500	0.01	HT	**66.89%**				
ADWIN	500	0.01	HAT	68.11%				
ADWIN	1000	0.01	NB	**62.93%**	0.516	0.36	0.88	84
ADWIN	1000	0.01	KNN	**58.52%**				
ADWIN	1000	0.01	HT	**65.49%**				
ADWIN	1000	0.01	HAT	66.61%				
HDDM-A	1000	0.01	NB	**60.54%**	0.569	0.41	0.94	65
HDDM-A	1000	0.01	KNN	**58.48%**				
HDDM-A	1000	0.01	HT	60.97%				
HDDM-A	1000	0.01	HAT	62.17%				

Table 9: Results obtained by different classifiers in the RTG experiment with 100 relevant features.

Classifier	Avg. Accuracy (%)
NB	58.46%
KNN	51.83%
HT	61.22%
HAT	73.53%

accuracy rates obtained. The 10 best configurations among the 36 tested are reported in Figure 10 with the accuracy results. In contrast to what was observed for synthetic experiments, smaller grace periods combined with the ADWIN drift detector dominate the top positions, and as a result, we select $gp = 100$, $\theta = 0.05$ and ADWIN as the default configuration for real-world experiments.

In Table 10 we compare the average accuracy rates obtained by different classifiers with and without ABFS across the 30 executions performed. Here, we note that the NB and HT classifiers benefit from ABFS in all experiments (at least marginally, as we note in COVTYPE), whereas they match or improve for KNN. The observed increases are relevant as they broaden 1.61% to 19.45% for NB and up to 7.10% for HT. Similarly, the results for HAT show no difference for IADS, while a significant decrease of 4.66% is observed in NOMAO, another decrease for COVTYPE of 1.86%, and an increase of 1.47% for SPAM. Following the outcome of the Wilcoxon test, both NB and HT classifiers are significantly improved regarding accuracy in these scenarios, whereas the remainder are not significantly affected.

In Table 11 we compare the processing times of the classifiers with and

Table 10: Average accuracy rates (%) obtained by different classifiers and ABFS in real-world experiments. Results in bold are the highest accuracy rates obtained per classifier type.

Experiment	NB	ABFS-NB	KNN	ABFS-KNN	HT	ABFS-HT	HAT	ABFS-HAT
COVTYPE	70.41	**83.20**	81.23	**84.64**	83.77	**84.31**	**82.75**	80.89
IADS	80.55	**100.00**	**100.00**	**100.00**	92.90	**100.00**	82.80	**84.00**
NOMAO	83.36	**93.52**	94.10	**94.43**	91.43	**94.55**	**93.67**	89.01
PAMAP2	97.11	**98.72**	**99.91**	**99.91**	97.66	**98.85**	86.72	**87.91**
SPAM	75.78	**87.76**	86.15	**94.14**	83.49	**88.81**	83.45	**84.92**

without ABFS. Here, we observe similar behavior to what has been observed for synthetic data, where the processing time rates of all classifiers have increased, except for KNN. Here, the Wilcoxon test showed that ABFS significantly improves the KNN running times, while NB is worsened, and the results obtained for the remainder of the classifiers are rendered inconclusive. The memory consumption results, depicted in Table 12, show that ABFS also introduces overheads to all classifiers. This is a similar behavior to the one observed in the previous section, as the actual classification models still allocates memory to keep track of statistics about all the original features, even though they only update those for the selected ones. One exception worthy

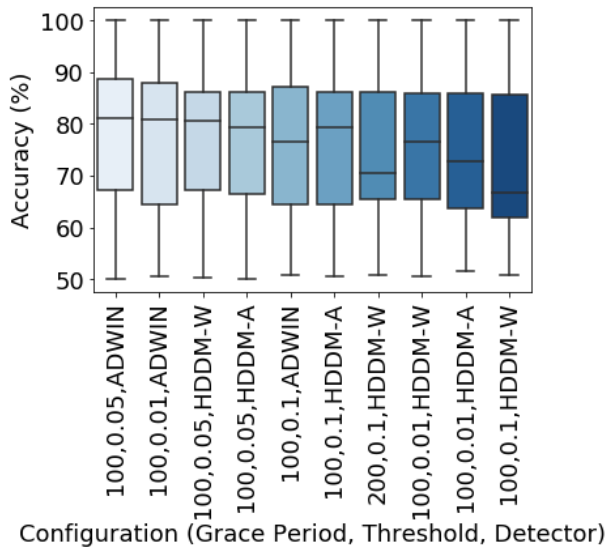


Figure 10: Accuracy rates (%) obtained across the 10-best ranked ABFS configurations in real-world experiments.

Table 11: Average processing time (s) rates obtained by different classifiers and ABFS in real-world experiments. Results in bold are the smaller rates obtained per classifier type.

Experiment	NB	ABFS-NB	KNN	ABFS-KNN	HT	ABFS-HT	HAT	ABFS-HAT
COVTYPE	**8.71**	10.45	82.16	**52.96**	**11.02**	15.62	**15.42**	20.91
IADS	**4.06**	9.74	48.82	**28.11**	**5.89**	10.91	**6.78**	12.52
NOMAO	**3.42**	7.81	33.50	**22.96**	**4.95**	10.01	**7.14**	11.19
PAMAP2	**13.32**	82.37	114.42	**81.46**	**14.97**	89.71	**13.15**	89.74
SPAM	**563.71**	586.39	8074.61	**3062.64**	**686.41**	716.89	**739.11**	1277.18

Table 12: Average RAM-Hours (GB-Hour) rates obtained by different classifiers and ABFS in real-world experiments. Results in bold are the smaller rates obtained per classifier type.

Experiment	NB	ABFS-NB	KNN	ABFS-KNN	HT	ABFS-HT	HAT	ABFS-HAT
COVTYPE	$\mathbf{1.17 \times 10^{-7}}$	4.29×10^{-6}	$\mathbf{7.97 \times 10^{-6}}$	2.01×10^{-5}	$\mathbf{1.06 \times 10^{-6}}$	5.49×10^{-6}	$\mathbf{5.04 \times 10^{-7}}$	6.97×10^{-6}
IADS	$\mathbf{7.78 \times 10^{-7}}$	7.25×10^{-4}	$\mathbf{1.48 \times 10^{-4}}$	1.73×10^{-3}	$\mathbf{1.87 \times 10^{-6}}$	7.95×10^{-4}	2.62×10^{-6}	$\mathbf{6.16 \times 10^{-9}}$
NOMAO	$\mathbf{5.59 \times 10^{-8}}$	2.80×10^{-5}	$\mathbf{6.38 \times 10^{-6}}$	5.64×10^{-5}	$\mathbf{7.32 \times 10^{-7}}$	3.02×10^{-5}	$\mathbf{5.70 \times 10^{-7}}$	3.43×10^{-5}
PAMAP2	$\mathbf{1.71 \times 10^{-7}}$	9.11×10^{-6}	7.34×10^{-6}	$\mathbf{1.20 \times 10^{-6}}$	9.54×10^{-7}	$\mathbf{2.90 \times 10^{-7}}$	2.90×10^{-7}	$\mathbf{2.88 \times 10^{-7}}$
SPAM	$\mathbf{4.09 \times 10^{-3}}$	9.15×10^{-2}	6.32×10^{-1}	$\mathbf{4.82 \times 10^{-1}}$	$\mathbf{8.51 \times 10^{-3}}$	1.26×10^{-1}	$\mathbf{1.28 \times 10^{-2}}$	2.22×10^{-1}

to mention is that memory consumption of HAT in the IADS experiment, which has significantly decreased, while the accuracy rate was maintained. Again, the Wilcoxon test was used, and its outcomes show that both NB and HT are significantly penalized when combined with ABFS, while the remainder of the classifiers is not.

Finally, we highlight the improvements observed in the SPAM experiment, which are important as it is the experiment with the highest dimensionality. In this experiment, all classifiers have their accuracy rates significantly improved (Table 10), while their processing time and memory consumption rates decreased (Tables 11 and 12). To understand the impact of ABFS in the SPAM experiment, we show in Figure 11 the average number of features selected by ABFS and used by decision tree-based classifiers. Here, we observe that out of the nearly 40 thousand features, and only 16 were used by the HAT alone, while the maximum number of features used by the same classifier with ABFS was 5. A similar behavior can be observed for the HT classifier, which used 10 features, while its version with ABFS used only 5. These results are particularly interesting as it shows that despite the fact that decision trees select a small subset of features to build its predictive model, they can still be further simplified so that their models are smaller and achieve higher generalization rates.

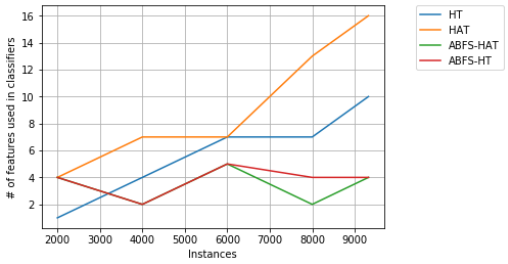


Figure 11: Number of features selected and used during the SPAM experiment.

7. Conclusion

In this paper, we introduced ABFS, an adaptive boosting-based feature selection algorithm for data streams. ABFS includes strategies to select features over data streams and to detect and adapt to concept drifts. The proposed method is classifier-independent, and results show that ABFS can improve all types of classifiers in different scenarios. Despite performing interesting cuts to the dimensionality of data streams, ABFS still increases the processing time and memory consumption of Bayesian and decision tree-based types of learners. An important exception is the KNN classifier, in which the results show that both processing times and memory consumption rates are improved.

In addition to the proposed method, we expect that the contributions on feature selection evaluation and the framework added to the Massive Online Analysis software to help in the assessment and comparison of future works in the area. Feature selection-specific metrics, such as Selection Accuracy and Stability have been introduced to streaming scenarios, and the results reported here can be assumed as baselines in future works of the area.

As future works, we highlight the following:

- **Feature selection based on decision trees and random forests:** generally speaking, decision trees can be seen as a feature selection process are they iteratively select features to maximize some 'purity' metric. Nevertheless, as observed in some of the experiments conducted in this paper, original Hoeffding Trees are prone to overfitting, as they tend to (i) branch over irrelevant features, and (ii) they are unable to prune poor performing branches (except the Hoeffding Adaptive Tree). For instance, the works of [56, 57] are two of many studies where only the k-best features used on the first branches of decision

trees are used and improve the classification rates of different learners. Also, if the dimensionality of the stream is too big, another important investigation would be to perform feature selection based on Adaptive Random Forests [58], as previously conducted on batch scenarios on the works of [59, 60].

- **Distributed feature selection:** feature selection and dimensionality reduction are of the utmost importance when the number of features in a data stream grows up to thousands or millions. Therefore, proposing feature selection techniques that can be paralleled and scaled up to these extreme scenarios is another important gap to be pursued.

- **Feature selection on semi-supervised, unsupervised and delayed labelling learning schemes:** in real-world scenarios the assumption that all instances are labeled is unlikely to hold. Thus, it is important to tailor feature selection techniques that can accurately select features with very few or even no labels at all (for clustering scenarios). Closely related to this topic, feature selection methods should also be able to handle delayed labelling scenarios, where the labels of instances become available, but after a delay of n instances.

Acknowledgments

This project was financially supported by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES) through the *Programa de Suporte à Pós-Graduação de Instituições de Ensino Particulares* (PROSUP) program for Doctorate students. Part of this research has also been conducted with a grant for visiting researchers also provided by CAPES under grant number 88881.131817/2016-01. Finally, we truthfully thank the reviewers for the constructive comments that yielded relevant changes in our initial manuscript and how the assessment of the methods was done.

References

[1] A. L. Blum, P. Langley, Selection of relevant features and examples in machine learning, Artif. Intell. 97 (1-2) (1997) 245–271. doi:10.1016/S0004-3702(97)00063-5.
URL http://dx.doi.org/10.1016/S0004-3702(97)00063-5

[2] D. W. Opitz, Feature selection for ensembles, in: Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence, AAAI '99/IAAI '99, American Association for Artificial Intelligence, Menlo Park, CA, USA, 1999, pp. 379–384.
URL http://dl.acm.org/citation.cfm?id=315149.315328

[3] Y. Yang, J. O. Pedersen, A comparative study on feature selection in text categorization, in: Proceedings of the Fourteenth International Conference on Machine Learning, ICML '97, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997, pp. 412–420.
URL http://dl.acm.org/citation.cfm?id=645526.657137

[4] M. L. Bermingham, R. Pong-Wong, A. Spiliopoulou, C. Hayward, I. Rudan, H. Campbell, A. F. Wright, J. F. Wilson, F. Agakov, P. Navarro, C. S. Haley, Application of high-dimensional feature selection: evaluation for genomic prediction in man, Scientific Reports 5.
URL http://dx.doi.org/10.1038/srep10312

[5] G. Widmer, M. Kubat, Learning in the presence of concept drift and hidden contexts, Mach. Learn. 23 (1) (1996) 69–101. doi:10.1023/A:1018046501280.

[6] N. Oza, Online bagging and boosting, in: Systems, Man and Cybernetics, 2005 IEEE International Conference on, Vol. 3, 2005, pp. 2340–2345 Vol. 3. doi:10.1109/ICSMC.2005.1571498.

[7] A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer, MOA: Massive online analysis, The Journal of Machine Learning Research 11 (2010) 1601–1604.

[8] J. P. Barddal, H. M. Gomes, F. Enembreck, B. Pfahringer, A. Bifet, On dynamic feature weighting for feature drifting data streams, in: ECML/PKDD'16, Lecture Notes in Computer Science, Springer, 2016.

[9] P.-N. Tan, M. Steinbach, V. Kumar, Introduction to Data Mining, (Second Edition), Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

[10] Z. Zhao, F. Morstatter, S. Sharma, S. Alelyani, A. Anand, H. Liu, Advancing feature selection research, ASU feature selection repository (2010) 1–28.

[11] G. Chandrashekar, F. Sahin, A survey on feature selection methods, Computers & Electrical Engineering 40 (1) (2014) 16 – 28, 40th-year commemorative issue. doi:http://dx.doi.org/10.1016/j.compeleceng.2013.11.024.
URL http://www.sciencedirect.com/science/article/pii/S0045790613003066

[12] H.-L. Nguyen, Y.-K. Woon, W.-K. Ng, L. Wan, Heterogeneous ensemble for feature drifts in data streams, in: P.-N. Tan, S. Chawla, C. Ho, J. Bailey (Eds.), Advances in Knowledge Discovery and Data Mining, Vol. 7302 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 1–12. doi:10.1007/978-3-642-30220-6_1.
URL http://dx.doi.org/10.1007/978-3-642-30220-6_1

[13] S. C. H. Hoi, J. Wang, P. Zhao, R. Jin, Online feature selection for mining big data, in: Proceedings of the 1st International Workshop on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications, BigMine '12, ACM, New York, NY, USA, 2012, pp. 93–100. doi:10.1145/2351316.2351329.
URL http://doi.acm.org/10.1145/2351316.2351329

[14] J. Wang, P. Zhao, S. C. H. Hoi, R. Jin, Online feature selection and its applications, IEEE Transactions on Knowledge and Data Engineering 26 (3) (2014) 698–710. doi:10.1109/TKDE.2013.32.

[15] D. Zhou, J. Huang, B. Schölkopf, Learning from labeled and unlabeled data on a directed graph, in: Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005, 2005, pp. 1036–1043. doi:10.1145/1102351.1102482.
URL http://doi.acm.org/10.1145/1102351.1102482

[16] J. P. Barddal, H. M. Gomes, A. de Souza Britto Jr., F. Enembreck, A benchmark of classifiers on feature drifting data streams, in: ICPR'16, Lecture Notes in Computer Science, Springer, 2016.

[17] J. P. Barddal, H. M. Gomes, F. Enembreck, B. Pfahringer, A survey on feature drift adaptation: Definition, benchmark, challenges and future directions, Journal of Systems and Software 127 (2017) 278 – 294. doi:https://doi.org/10.1016/j.jss.2016.07.005.

[18] P. Domingos, G. Hulten, 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, 2000, pp. 71–80. doi:10.1145/347090.347107.
URL http://doi.acm.org/10.1145/347090.347107

[19] A. Bifet, R. Gavaldà, Adaptive Learning from Evolving Data Streams, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 249–260. doi:10.1007/978-3-642-03915-7_22.

[20] A. Bifet, R. Gavaldà, Learning from time-changing data with adaptive windowing, in: In SIAM International Conference on Data Mining, 2007.

[21] I. H. Witten, E. Frank, Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[22] H. Huang, S. Yoo, S. P. Kasiviswanathan, Unsupervised feature selection on data streams, in: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, CIKM '15, ACM, New York, NY, USA, 2015, pp. 1031–1040. doi:10.1145/2806416.2806521.

[23] J. Duarte, J. Gama, Feature ranking in hoeffding algorithms for regression, in: Proceedings of the Symposium on Applied Computing, SAC '17, ACM, New York, NY, USA, 2017, pp. 836–841. doi:10.1145/3019612.3019670.
URL http://doi.acm.org/10.1145/3019612.3019670

[24] J. A. Silva, E. R. Faria, R. C. Barros, E. R. Hruschka, A. C. P. L. F. d. Carvalho, J. Gama, Data stream clustering: A survey, ACM Comput. Surv. 46 (1) (2013) 13:1–13:31. doi:10.1145/2522968.2522981.

[25] R. E. Schapire, The strength of weak learnability, Mach. Learn. 5 (2) (1990) 197–227. doi:10.1023/A:1022648800760.
URL https://doi.org/10.1023/A:1022648800760

[26] R. Appel, P. Perona, A simple multi-class boosting framework with theoretical guarantees and empirical proficiency, in: D. Precup, Y. W. Teh (Eds.), Proceedings of the 34th International Conference on Machine Learning, Vol. 70 of Proceedings of Machine Learning Research, PMLR, International Convention Centre, Sydney, Australia, 2017, pp. 186–194.
URL http://proceedings.mlr.press/v70/appel17a.html

[27] Q. Miao, Y. Cao, G. Xia, M. Gong, J. Liu, J. Song, Rboost: Label noise-robust boosting algorithm based on a nonconvex loss function and the numerically stable base learners, IEEE Transactions on Neural Networks and Learning Systems 27 (11) (2016) 2216–2228. doi:10.1109/TNNLS.2015.2475750.

[28] R. E. Schapire, A brief introduction to boosting, in: Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, pp. 1401–1406.
URL http://dl.acm.org/citation.cfm?id=1624312.1624417

[29] R. Pelossof, M. Jones, I. Vovsha, C. Rudin, Online coordinate boosting, in: 2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops, 2009, pp. 1354–1361. doi:10.1109/ICCVW.2009.5457454.

[30] B. Wang, J. Pineau, Online bagging and boosting for imbalanced data streams, IEEE Transactions on Knowledge and Data Engineering 28 (12) (2016) 3353–3366. doi:10.1109/TKDE.2016.2609424.

[31] H. Hu, W. Sun, A. Venkatraman, M. Hebert, J. A. Bagnell, Gradient boosting on stochastic data streams, CoRR abs/1703.00377. arXiv:1703.00377.
URL http://arxiv.org/abs/1703.00377

[32] Z. Xu, G. Huang, K. Q. Weinberger, A. X. Zheng, Gradient boosted feature selection, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, ACM,

New York, NY, USA, 2014, pp. 522–531. doi:10.1145/2623330.2623635.
URL http://doi.acm.org/10.1145/2623330.2623635

[33] S. Das, Filters, wrappers and a boosting-based hybrid for feature selection, in: Proceedings of the Eighteenth International Conference on Machine Learning, ICML '01, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001, pp. 74–81.
URL http://dl.acm.org/citation.cfm?id=645530.658297

[34] Q. Miao, Y. Cao, J. Song, J. Liu, Y. Quan, Boostfs: A boosting-based irrelevant feature selection algorithm, IJPRAI 29 (7). doi:10.1142/S0218001415510118.
URL http://dx.doi.org/10.1142/S0218001415510118

[35] W. Hoeffding, Probability inequalities for sums of bounded random variables, Journal of the American Statistical Association 58 (301) (1963) 13–30.
URL http://www.jstor.org/stable/2282952?

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

[37] S. Galelli, G. B. Humphrey, H. R. Maier, A. Castelletti, G. C. Dandy, M. S. Gibbs, An evaluation framework for input variable selection algorithms for environmental data-driven models, Environmental Modelling & Software 62 (2014) 33 – 51. doi:http://dx.doi.org/10.1016/j.envsoft.2014.08.015.

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

[39] L. Molina, L. Belanche, A. Nebot, Feature selection algorithms: a survey and experimental evaluation, in: Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on, 2002, pp. 306–313. doi:10.1109/ICDM.2002.1183917.

[40] L. I. Kuncheva, A stability index for feature selection, in: Proceedings of the 25th Conference on Proceedings of the 25th IASTED International Multi-Conference: Artificial Intelligence and Applications, AIAP'07, ACTA Press, Anaheim, CA, USA, 2007, pp. 390–395.
URL http://dl.acm.org/citation.cfm?id=1295303.1295370

[41] S. Nogueira, G. Brown, Measuring the stability of feature selection, in: Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part II, 2016, pp. 442–457. doi:10.1007/978-3-319-46227-1_28.

[42] A. Bifet, G. de Francisci Morales, J. Read, G. Holmes, B. Pfahringer, Efficient online evaluation of big data stream classifiers, in: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '15, ACM, New York, NY, USA, 2015, pp. 59–68. doi:10.1145/2783258.2783372.
URL http://doi.acm.org/10.1145/2783258.2783372

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

[44] F. Enembreck, B. C. Ávila, E. E. Scalabrin, J.-P. A. Barthès, Learning drifting negotiations., Applied Artificial Intelligence 21 (9) (2007) 861–881.
URL http://dblp.uni-trier.de/db/journals/aai/aai21.html#EnembreckASB07

[45] M. A. Hall, Correlation-based feature selection for discrete and numeric class machine learning, in: Proceedings of the Seventeenth International Conference on Machine Learning, ICML '00, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000, pp. 359–366.
URL http://dl.acm.org/citation.cfm?id=645529.657793

[46] W. N. Street, Y. Kim, 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, 2001, pp. 377–382.

[47] J. A. Blackard, D. J. Dean, 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. doi:https://doi.org/10.1016/S0168-1699(99)00046-0.
URL http://www.sciencedirect.com/science/article/pii/S0168169999000460

[48] N. Kushmerick, Learning to remove internet advertisements, in: Proceedings of the third annual conference on Autonomous Agents, ACM, 1999, pp. 175–181.

[49] L. Candillier, V. Lemaire, Design and analysis of the nomao challenge active learning in the real-world, in: Proceedings of the ALRA: Active Learning in Real-world Applications, Workshop ECML-PKDD, 2012.

[50] A. Reiss, D. Stricker, Introducing a new benchmarked dataset for activity monitoring, in: Proceedings of the 2012 16th Annual International Symposium on Wearable Computers (ISWC), ISWC '12, IEEE Computer Society, Washington, DC, USA, 2012, pp. 108–109. doi:10.1109/ISWC.2012.13.
URL https://doi.org/10.1109/ISWC.2012.13

[51] I. Katakis, G. Tsoumakas, I. Vlahavas, 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, 2006, p. 107.

[52] F. Wilcoxon, Individual Comparisons by Ranking Methods, Biometrics Bulletin 1 (6) (1945) 80–83. doi:10.2307/3001968.
URL http://dx.doi.org/10.2307/3001968

[53] M. Friedman, The Use of Ranks to Avoid the Assumption of Normality Implicit in the Analysis of Variance, Journal of the American Statistical Association 32 (200) (1937) 675–701. doi:10.2307/2279372.
URL http://dx.doi.org/10.2307/2279372

[54] P. Nemenyi, Distribution-free multiple comparisons, Ph.D. thesis, New Jersey, USA (1963).

[55] J. Demsar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
URL http://dl.acm.org/citation.cfm?id=1248547.1248548

[56] V. Sugumaran, V. Muralidharan, K. Ramachandran, Feature selection using decision tree and classification through proximal support vector machine for fault diagnostics of roller bearing, Mechanical Systems and Signal Processing 21 (2) (2007) 930 – 942. doi:https://doi.org/10.1016/j.ymssp.2006.05.004.
URL http://www.sciencedirect.com/science/article/pii/S0888327006001142

[57] N. M. Tahir, A. Hussain, S. A. Samad, K. A. Ishak, R. A. Halim, Feature selection for classification using decision tree, in: 2006 4th Student Conference on Research and Development, 2006, pp. 99–102. doi:10.1109/SCORED.2006.4339317.

[58] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, T. Abdessalem, Adaptive random forests for evolving data stream classification, Machine Learning 106 (9) (2017) 1469–1495. doi:10.1007/s10994-017-5642-8.
URL https://doi.org/10.1007/s10994-017-5642-8

[59] A. P. Cassidy, F. A. Deviney, Calculating feature importance in data streams with concept drift using online random forest, in: 2014 IEEE International Conference on Big Data (Big Data), 2014, pp. 23–28. doi:10.1109/BigData.2014.7004352.

[60] A. Verikas, A. Gelzinis, M. Bacauskiene, Mining data with random forests: A survey and results of new tests, Pattern Recognition 44 (2) (2011) 330–349.