2018 IEEE International Conference on Data Mining

Enhancing Very Fast Decision Trees with Local Split-Time Predictions

Viktor Losing*†, Heiko Wersing† and Barbara Hammer*
*Bielefeld University, Universitätsstr. 25, 33615 Bielefeld
†HONDA Research Institute Europe, Carl-Legien-Str. 30, 63073 Offenbach am Main

***Abstract*—An increasing number of industrial areas recognize the opportunities of Big Data, requiring highly efficient algorithms which enable real-time processing to reduce the burden of data storage and maintenance. Decision trees are extremely fast, highly accurate and easy to use in practice. Merging multiple decision trees to an ensemble leads to one of the most powerful machine learning methods. The Very Fast Decision Tree is the state-of-the-art incremental decision tree induction algorithm, capable of learning from massive data streams. It is successful due to its theoretical guarantees based on the Hoeffding bound as well as its competitive performance in terms of classification accuracy and time / space efficiency. In this paper, we increase the efficiency even further by replacing its global splitting scheme, which periodically tries to split every $n_{\min}$ examples. Instead, we utilize local statistics to predict the split-time, thus, avoiding unnecessary split-attempts, usually dominating the computational cost. Concretely, we use the class distributions of previous split-attempts to approximate the minimum number of examples until the Hoeffding bound is met. This cautious approach yields by design a low delay and reduces the number of split-attempts at the same time. We extensively evaluate our method using common stream-learning benchmarks also considering non-stationary environments. The experiments confirm a substantially reduced run-time without a loss in classification performance.**

***Index Terms*—Decision trees, data streams, supervised classification, concept drift**

I. Introduction

More and more fields are facing the challenges of Big Data, where the mere storage and maintenance of the huge amount of information becomes very costly. YouTube currently has 300 hours of video being uploaded every minute, and it is projected to grow up to 1,700 hours per minute by 2025. In astronomy, the Australian Square Kilometer Array Pathfinder (ASKAP) project acquires 7.5 terabytes/second of sample image data and is expected to increase to 750 terabytes/second by 2025 [1]. The field of genomics is predicted to be the biggest challenge in Big Data [2] in the future.

Additionally, a multitude of devices generates an increasing amount of streaming-data. The main driving force behind this development is the Internet of Things (IOT) technology, where the idea is to provide all-time connected systems, exchanging all sorts of data in real-time to provide adaptive functions in smart-home or smart-city environments [3]. Devices continuously stream data to cloud-services, providing answers to queries and personalized assistance. Hence, efficient real-time processing and analysis is becoming extremely important not only as a way to reduce the burden of storage, but also to enable companies to be up-to-date with current trends and demands of their customers.

Numerous state-of-the-art classification algorithms for data streams are based on decision trees due to their high efficiency and accuracy [4]. The most prominent is the Very Fast Decision Tree (VFDT) [5], an incremental, anytime decision tree induction algorithm, capable of learning from massive data streams. Relying on the Hoeffding bound, the VFDT provides a justification to choose the best split-attribute based on a small sample of data, therefore, being able to construct trees similar to those of classical batch algorithms such as C4.5 [6] or CART [7]. As mentioned in [5], the most time consuming step of the algorithm is to evaluate after each example whether enough evidence has been gathered to determine the best attribute for a split. To reduce the computational load of these *split-attempts* the VFDT incorporates the hyperparameter $n_{\min}$, controlling the number of examples a leaf has to accumulate before a new split-attempt is performed. Even though this regular approach reduces the computational complexity, it also leads to a *split-delay* since more examples are accumulated than the strict minimum. Consequently, the tree growth is hampered, often resulting in a lower classification performance.

In this paper, we tackle this issue and increase the efficiency of the VFDT. Instead of the periodic split-attempts, we predict the local *split-time*, i.e. the number of examples a leaf needs to accumulate before a split is performed, avoiding unnecessary attempts when splits are unlikely. Concretely, we use the class distributions of previous split-attempts to approximate the minimum time until the Hoeffding bound is met. This cautious approach splits by design with a low delay, but still substantially reduces the number of split-attempts. Furthermore, it increases the robustness of the algorithm because the run-time dependence on the setting of $n_{\min}$ is reduced.

In recently published work [8], the authors evaluate their split-time prediction method solely based on the classification accuracy, which is not only imprecise but can even be misleading in case of overfitting. Instead, we formally define the term *split-delay*, which exactly measures the delay caused by the split-time prediction. On this basis, we perform a detailed evaluation including an analysis of the trade-off between split-delay and the number of split-attempts, the execution time and a statistical assessment of the resulting classification accuracy. Thereby, we use commonly applied data stream learning benchmarks. The experiments show that our approach significantly reduces the run-time without a loss in performance.

978-1-5386-9159-5/18/$31.00 ©2018 IEEE
DOI 10.1109/ICDM.2018.00044

II. Framework

Our focus is on data stream classification under supervised learning. The aim in supervised classification is to predict a target variable $y \in \{1, \dots, c\}$ given feature vector $\mathbf{x} \in \mathcal{X}_1 \times \dots \times \mathcal{X}_d$ where the domains are either real-valued $\mathcal{X}_i = \mathbb{R}$ or discrete $\mathcal{X}_j = \{a_1^j, \dots, a_{n_j}^j\}$ with $n_j \geq 2$ possible values. Learning algorithms applied on data streams are usually evaluated in the on-line learning setting. A potentially infinite sequence $S = (s_1, s_2, \dots, s_t \dots)$ of tuples $s_i = (\mathbf{x}_i, y_i)$ arrives one after another. As t represents the current time stamp, the learning objective is to predict the corresponding label y_t for a given input x_t, which is supposed to be unknown. The prediction $\hat{y}_t = h_{t-1}(\mathbf{x}_t)$ is done according to the previously learned model h_{t-1}. Afterward, the true label is revealed and the loss $\mathcal{L}(\hat{y}_t, y_t)$ determined. Before proceeding with the next example, the applied learning algorithm generates a new model h_t on the basis of the current tuple s_t and the previous model h_{t-1}: $h_t = \text{train}(h_{t-1}, s_t)$. The Interleaved Test-Train error for a sequence up to the current time t is given by:

$$E(S) = \frac{1}{t} \sum_{i=1}^{t} \mathbb{1}(h_{i-1}(x_i) \neq y_i).$$

Algorithms applied to streaming data face the challenge of anytime model adaption, one pass learning and the examples have to be processed in incoming order, often violating the assumption of sample independence.

III. Incremental decision tree induction

As usual, decision trees consist of trees where each interior node is labeled by an attribute $j \in \{1, \dots, d\}$. It has n_j children in case of a discrete domain $\mathcal{X}_j$ where children are labeled according to the n_j values of $\mathcal{X}_j$. Leaves are assigned a distribution over the class labels $\{1, \dots, c\}$. A given data point x is assigned to a leaf node and its class probability according to the match of the attributes of x and the values of the tree's nodes, starting from the root. Since the inference of optimum decision trees is NP-hard, greedy approaches typically iteratively grow a tree based on impurity measures of the current nodes such as the entropy or the Gini index. Unlike batch scenarios, where the best split-attributes are based on the whole training data, incremental decision tree induction constructs a tree subsequently based on a small sample of the data and generates splits on the fly. Algorithm 1 shows the pseudocode of the basic algorithm. Thereby, the impurity measure per leaf l can be given as the classical information gain, for example: For a set M of labeled data pairs (x, y), the entropy is defined as $H(M) = -\sum_{j=1}^{c} p_j \; log_2 p_j$ where p_j is the probability of class j in M given by its relative frequency. Every discrete-valued attribute j induces a split into n_j leaves according to the possible attribute values $a_1^j, \dots a_{n_j}^j$. The weighted entropy of a split of M along j is given by

$$H_j(M) = \sum_{k=1}^{n_j} \frac{|M(a_k^j)|}{|M|} H(M(a_k^j)),$$

where $M(a_k^j) := \{(x, y) \in M \mid x_j = a_k^j\}$, yielding the information gain $G_j(M) = H(M) - H_j(M)$. The attribute providing the maximum gain is chosen for the split. In case of binary splits, we alternatively denote the weighted entropy in terms of both sets $M(a_1^j)$ and $M(a_2^j)$

$$H_j(M(a_1^j), M(a_2^j)) = \frac{|M(a_1^j)| H(M(a_1^j)) + |M(a_2^j)| H(M(a_2^j))}{|M(a_1^j)| + |M(a_2^j)|}. \quad (1)$$

Algorithm 1 incremental-tree()

```
Let Tr be a tree with a single leaf $l_1$ (the root)
Let $S(l_1)$ be the sufficient statistics $l_1$ collects
Let $C(l_1) := (\frac{1}{c}, \dots, \frac{1}{c})$ be the initial class distribution of $l_1$
for all time steps $t$ do
    assign $(x_t, y_t)$ to leaf $l_i$ according to matching attributes
    update $S(l_i)$ with $(x_t, y_t)$
    update $C(l_i)$ with $y_t$
    if criterion to split is fulfilled(*) then
        choose best split attribute $j$ of $l_i$ depending
          on impurity($S(l_i)$)
        split $l_i$ along all discrete attribute values of $j$
    end if
end for
```

Please note, that incremental decision trees usually do not store the raw samples (x_t, y_t) in the leaf. Instead, they derive sufficient statistics in the form of class-conditional attribute-value occurrences, enabling the calculation of the information gain with a reduced time and space complexity.

IV. VFDT

The VFDT is an incremental decision tree induction algorithm. The moment when a new split is performed is crucial with respect to learning-speed, tree complexity and overfitting. The VFDT splits a leaf only if enough evidence has been seen to guarantee that the chosen split-attribute is the best with a predefined probability. This mathematically sound approach distinguishes it from other incremental learning trees, which heuristically decide to split based on a predefined number of examples [9] or a predefined impurity-threshold [10]. The VFDT relies on the Hoeffding bound, therefore it is also known under the name Hoeffding tree. Given a continuous random variable r of range R and its observed mean $\bar{r}$ after n independent observations, the Hoeffding bound states that with probability of $1 - \delta$ the true mean of r is at least $\bar{r} - \epsilon$ where

$$\epsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}}.$$

Concretely, the VFDT determines the two attributes j_1, j_2 with the highest information gain G_{j_1}, G_{j_2}. If their gain difference $\Delta G = G_{j_1} - G_{j_2}$ is larger than ϵ, then G_{j_1} is indeed the best split-attribute with probability of $1 - \delta$ and a split is performed. The VFDT also performs a split if too many examples are necessary to decide on the best attribute. These so called tie-splits occur when several attributes have a similar gain and are necessary to avoid stunted tree growth which can lead

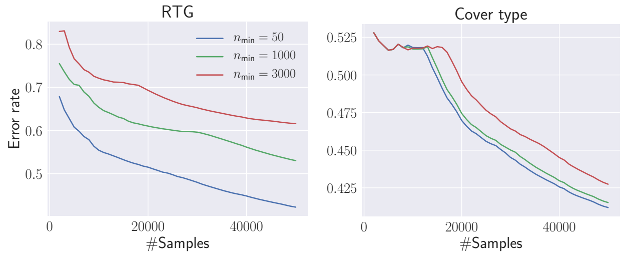

Fig. 1. The error-rate typically increases with higher n_{min} values, due to a slower tree growth. The gap with respect to the error rate varies depending on the data set.

to a performance degradation, particularly at the early stages of learning [11]. The number of examples required for a tie-split is controlled by a parameter τ which is set in terms of the Hoeffding bound. Formally, a tie-split is performed when $\Delta G < \epsilon < \tau$. In other words, a tie-split is performed after $\frac{R^2 ln(1/\delta)}{2\tau^2}$ examples. Concretely, using the default parameter setting ($\tau = 0.05$, $\delta = 1e - 7$) and assuming a range of $R = 1$ a tie-split will be done after 3224 examples, which is rather a small amount particularly considering Big Data tasks. Tie-splits are supposed to occur rarely but tend to happen very often in practice and can even be harmful because they sometimes cause imbalanced trees [11]. The split-criterion (*) (see Algorithm 1) of the VFDT is given as pseudocode in Algorithm 2.

Algorithm 2 vfdt-split-criterion()

```
if $|S(l_i)| \mod n_{\text{min}} = 0$ then
    determine the attributes $j_1, j_2$ with highest gain $G_{j_1}$, $G_{j_2}$
    return $(\Delta G > \epsilon$ or $\Delta G < \tau)$
else
    return false
end if
```

A. Trade-off: split-delay vs. the number split-attempts

Ideally, the VFDT would test after each example whether the condition $\Delta G > \epsilon$ is fulfilled. However, as mentioned in the original paper [5] the necessary recomputation of all information gains is a very time-consuming , and therefore, not applicable in practice. To reduce the load, the authors introduced the hyperparameter n_{min} to define the number of examples a leaf has to accumulate before a new *split-attempt* is done. Even though this periodic approach reduces the computational complexity for tree construction by a factor of n_{min}, it comes at the cost of an effectively slower tree growth and usually a lower performance, since more examples are accumulated than the strict minimum [12]. Figure 1 shows some examples where the mentioned relation can be observed. We refer to the number of surplus examples as *split-delay*. The setting of n_{min} controls a trade-off between computational complexity and classification performance, or more precisely, between the number of split-attempts and the split-delay.
In the following we fromally define the term split-delay. Given a leaf l, its true minimum split-time $\hat{t}_l$ and a split-time prediction function F which uses some leaf-specific data D_l to estimate the local split-time. D_l is obtained from the previously unsuccessful split-attempt and depends on the corresponding time, however we omit this detail in favor of a clear notation. Whenever the predicted time $F(D_l)$ is passed, a new split-attempt is performed. This process is repeated until l is split. The split-delay is given as

$$d_l = \sum_{i=1}^{b} F_i(D_l) - \hat{t}_l,$$

where b is the number of required split-attempts and $F_1 = n_{\text{min}}$ is predefined. Hence, the initial split-attempt is always performed after n_{min} examples and the split-time prediction is only used if no split initially occurs. Particularly, the split-delay of nodes splitted at the initial attempt ($\hat{t}_l <= n_{\text{min}}$) is the same for all split-time prediction functions. Please note, that tie-splits have by definition no split-delay, since leaves are always splitted as soon as the maximum number of examples is accumulated.
We determine the true minimum split-time $\hat{t}_l$ by using the VFDT with $n_{\text{min}} = 1$, in other word it tries to split after each example, and has therefore by definition no split-delay. Concretely, we construct a reference tree which is associated with its set of L split-nodes at the end of learning. The average split-delay of a F is simply given as $\frac{1}{|L|} \sum_{l \in L} d_l$[1].

V. Split-time Prediction

The goal of split-time prediction is to minimize the split-delay and the number of split-attempts. In the VFDT algorithm, the parameter n_{min} globally controls the time interval between split-attempts without utilizing any information of the processed data. Split-time predictions use the local statistics of the leaves to increase the efficiency. The VFDT requires i.i.d. data for the Hoeffding bound to be valid. In such cases, the central limit theorem states that the more examples are accumulated within a leaf the more precise is the estimation of the true local class distribution, the two best attributes and the corresponding difference in information gain ΔG. Using such information, split-attempts can be avoided when successful split-attempts are unlikely or can be encouraged in the contrary case. These guarantees are not given in the case of concept drift, however, past or at least recent data has still some predictive value in these scenarios which in addition benefit from local adaptation.
Split-time predictions are only performed when the initial split-attempt fails (after n_{min} examples). From then on, we continuously predict the local split-time. We bound the maximum predicted split-time with the number of examples required for a tie-split $n_{\text{max}} = \frac{R^2 ln(1/\delta)}{2\tau^2}$. Algorithm 3 shows the pseudocode for constructing a VFDT with split-time prediction and the corresponding splitting-criterion is given in Algorithm 4.

[1] The prediction function F is continuously applied during the construction of the reference tree to measure the split-delay.

Algorithm 3 vfdt-stp()

```
Let Tr be a tree with a single leaf $l_1$ (the root)
Let $S(l_1)$ be the sufficient statistics $l_1$ collects
Let $C(l_1) := (\frac{1}{c}, \dots, \frac{1}{c})$ be the initial class distribution of $l_1$
Let pt$(l_1) := -1$ be the predicted split-time for $l_1$
for all time steps $t$ do
    assign $(x_t, y_t)$ to leaf $l_i$ according to matching attributes
    update $S(l_i)$ with $(x_t, y_t)$
    update $C(l_i)$ with $y_t$
    if vfdt-split-time-prediction-criterion() then
        choose best split attribute $j$ of $l_i$ depending
          on impurity($S(l_i)$)
        split $l_i$ along all discrete attribute values of $j$
    end if
end for
```

Algorithm 4 vfdt-stp-split-criterion()

```
if $|S(l_i)|$ mod $n_{\text{min}} = 0$ and pt$(l_i) = -1$ or $|S(l_i)| =$ pt$(l_i)$ then
    determine the attributes $j_1, j_2$ with highest gain $G_{j_1}$, $G_{j_2}$
    if $(\Delta G > \epsilon$ or $\Delta G < \tau)$ then
        return true
    else
        pt$(l_i)$ :=min(split-time-prediction(), $n_{\text{max}}$)
        return false
    end if
else
    return false
end if
```

A. Proposed method: One-sided minimum (OSM)

On the basis of the class-distributions resulting from the currently best split, we approximate the minimum amount of required additional examples until the Hoeffding bound is met. In other words, assuming that the attributes j_1 and j_2 deliver the highest gain G_{j_1} and G_{j_2}, we estimate how many examples are necessary to sufficiently reduce G_{j_1} such that ΔG gets large enough. The estimation assumes that the class distribution resulting from the best split remains similar and that G_{j_2} is approximately constant.

More specifically, the weighted entropy of the currently best split j_1 is given by $H_{j_1}(M)$. For simplicity, we consider only binary splits (the approach can be extended to multiway-splits in a straight-forward way) and use $H_{j_1}(M(a_1^{j_1}), M(a_2^{j_1}))$ (see equation 1) as alternative to denote the weighted entropy.

How many examples need to be added such that this weighted entropy becomes sufficiently small? The entropy is reduced most if examples are added to the most frequent class. For the weighted entropy, an additional degree of freedom is the choice between $M(a_1^{j_1})$ or $M(a_2^{j_1})$. An exact solution how to best distribute m novel samples needs to calculate 2^m possible solutions, a greedy one, which incrementally adds examples to the class distribution minimizing the weighted entropy the most, requires $2m$ recalculations of the entropy. Instead, we propose to initially determine the lowest of both entropies $M(a_1^{j_1})$, $M(a_2^{j_1})$ and to incrementally add examples to this set until ΔG is large enough. We argue in the appendix, that this approximation yields most time the same solution as the greedy search. The returned values are equal or larger than the exact solution solution which is sufficient for our purpose, since predicting the minimum is already a cautious approach. The naive implementation requires m entropy calculations which is further reduced on the basis of Brent's method [13], a root-finding algorithm, combining the bisection method, the secant method and inverse quadratic interpolation. In contrast to Newton's method, it does not require derivatives and converges superlinearly for well-behaved functions. Formally, we search a root of the function

$$f(x) = H_{j_2}(M) - H(E(M(a_1^{j_1}), x), M(a_2^{j_1})) - \sqrt{\frac{R^2 ln(1/\delta)}{2|M| + x}} \quad (2)$$

in the range $(0, n_{max}]$, where $E(M, x)$ is a function which increases the most frequent class of M by x samples. Thus, m is given as root of $f(x)$. In advance, we verify that $f(0)$ and $f(n_{max})$ have different signs to ensure a solution exists in the desired interval. In our experiments, the solution was usually found in few iterations (on average 12.8).

This strategy is very cautious and predicts low values leading to unnecessary split-attempts when a split gets likely. Hence, we bound the minimum prediction by n_{min}. The final prediction is given by

$$F_{\text{OSM}} \begin{cases} n_{max} & \text{if } f(0)f(n_{max}) > 0 \\ max(n_{\text{min}}, m) & \text{otherwise.} \end{cases}$$

The complexity of our prediction function is dominated by the entropy calculations and therefore given as $\mathcal{O}(c)$. The *OSM* approach is in general pessimistic in its predictions, highly valuing a low split-delay in favor of reduced split-attempts. However, the split-attempts can be simply reduced by scaling its predictions for tasks where low split-delay are of secondary importance.

VI. Related work

Local split-time predictions have been only recently proposed in [8]. The authors assume that ΔG remains constant between the two best split attributes and simply predict the number of necessary examples given by the Hoeffding bound

$$F_{\text{CGD}} = min\left(\frac{R^2 ln(1/\delta)}{2\Delta G^2}, n_{max}\right).$$

They showed that this intuitive and simple approach drastically reduces the run-time but also diminishes the accuracy of the VFDT. In case of an increased ΔG, this method distinctly overestimates the split-time because the number of examples depends quadratically on ΔG. Our approach is more precise, since it incorporates more information for its prediction, namely the resulting class-distributions of the currently best split. Furthermore, its cautious predictions result in a substantially lower split-delay, providing the same classification performance as the original VFDT.

A lot of work has been done to improve the VFDT in different aspects. It was adapted to specifically deal with noisy datasets [14] as well as to one-class scenarios [15]. Holmes et al. pointed out that tie-splits, occurring when the best attribute is not determined within a maximum number of examples, are done very frequently and sometimes lead to skewed trees. They mitigated this issue by increasing stepwise the maximum number of required examples after each tie-split.

Gama et al. incorporated Naive Bayes [16] classifiers within the leaves to gain more classification power [17], initially proposed in [18]. However, Holmes et al. showed for various tasks that the initially more accurate Naive Bayes leaves are outperformed by majority voting in the long run. They applied a simple adaptive strategy, choosing the classification method which was more accurate in the past, to get the best of both worlds [19]. Bifet et al. used the Perceptron [20] algorithm instead to reduce the computational load while maintaining the classification gain [21].

Even though the Hoeffding bound assumes i.i.d. data, a lot of publications adapted the VFDT for streams incorporating changes in the distribution. The first was the Concept-Adapting VFDT [22] of Hulten et al. which grows alternative subtrees as soon as there is evidence that previously chosen split-attributes are becoming inappropriate and replaces them when they are less accurate than the alternative ones. Bifet and Gavalda generalized this approach and used explicit drift detection to initiate the growth of sub-trees and were able to achieve better results [23]. The VFDT was also applied in ensemble methods, a very popular approach to deal with non-stationarity. In [24], the authors insert a drift detection algorithm in each single VFDT and reset the classifier when the detection fires. The resulting method called Leveraging Bagging performed well on numerous datasets.

Publications tackling the efficiency of the VFDT focused so far rather on the implementation instead of the core algorithm. A parallelized version of the VFDT (Vertical Hoeffding Tree [25]) performs the attribute-specific tasks such as collecting statistics in the nodes as well as calculating the gain of corresponding splits in a distributed way. The gained speed-up scales with the number of attributes and the amount of applied CPU cores. Recently, an optimized but sequential C++ implementation was introduced in [26], able to gain a distinct speed-up in comparison to the versions available in Very Fast Machine Learning (VFML), the original framework by Domingos and Hulten, and Massive Online Analysis (MOA) [27], a well established open source framework for data stream mining.

Our enhancement of the VFDT is independent from already published work, hence, it can be integrated in any of them, providing the same benefits as it does for the original algorithm. In particular, it can be combined with already optimized or parallel implementations such as [24], [26] to increase the efficiency even further.

A. Datasets

We used large data streams consisting of well known artificial and real-world benchmarks. Table I shows their main characteristics and we briefly describe them in the following.

TABLE I
VARIOUS CHARACTERISTICS OF THE CONSIDERED DATASETS. SOME REAL-WORLD DATASETS ARE SORTED (POKER) OR SHUFFLED (MNIST8M) AND THEREFORE IT IS CLEAR WHETHER THEY CONTAIN CONCEPT DRIFT OR NOT. HOWEVER, FOR OTHER TASKS SUCH AS HIGGS OR AIRLINE IT IS UNCLEAR.

Dataset	#Samples	#Feat.	#Class	Type	Concept drift
RTG	5M	200	25	artificial	No
RBF	10M	100	50	artificial	No
LEDDrift	10M	24	10	artificial	Yes
Rialto	82K	27	10	real-world	Yes
Airline	539K	7	2	real-world	?
Cover type	581K	54	7	real-world	?
Poker	829K	10	10	real-world	Yes
MNIST8M	8.1M	782	10	real-world	No
HIGGS	11M	28	2	real-world	?

Radial basis function (RBF) Gaussian distributions with random initial positions, weights and standard deviations are generated in d-dimensional space. The weight controls the partitioning of the examples among the Gaussians. We used the RBF generator in MOA (100 dim., 50 classes, 100 centroids). This dataset is i.i.d.

Random Tree Generator (RTG) The RTG in MOA constructs a decision tree by randomly splitting along the attributes as well as assigning random classes to each leaf. Numeric and nominal attributes are supported and the tree depth can be predefined. Instances are generated by uniform sampling along each attribute. Initially proposed in [5], it is regarded as an easy task for decision trees (Parameters in MOA: 100 numeric dim., 100 nominal dim., 25 classes). This dataset is i.i.d.

LED-Drift Generator (LEDDrift) This generator yields instances with 24 boolean features with 17 of them being irrelevant. The remaining features corresponds to segments of a seven-segment LED display. The goal is to predict the digit displayed on the LED display, where each feature has a 10% chance of being inverted. Drift is generated by swapping the relevant features with irrelevant ones. We used the LEDDrift generator in MOA (7 drifting dimensions, 10% noise).

Rialto Bridge Timelapse Ten of the colorful buildings next to the famous Rialto bridge in Venice are encoded in a normalized 27-dimensional RGB histogram [28]. The images were obtained from time-lapse videos captured by a webcam with fixed position. The recordings cover 20 consecutive days during may-june 2016. Continuously changing weather and lighting conditions affect the representation, generating natural concept drift.

Airline The Airline data set was inspired by the regression data set from Ikonomovska[1]. The task is to predict whether a given flight will be delayed or not based on seven attributes encoding various information on the scheduled departure. This dataset is often used to evaluate concept drift classifier.

Forest Cover Type In this task, cartographic variables such as elevation, slope, soil type, ... of 30 × 30 meter cells are mapped to different forest cover types [29].

[1] https://kt.ijs.si/elena_ikonomovska/data.html

TABLE II
THE EVALUATED APPROACHES AND THEIR ABBREVIATIONS.

Abbr.	Method
VFDT	The standard VFDT with periodic split-attempts
CGD	VFDT with prediction based on the gain difference [8]
OSM	VFDT with prediction based on the class distribution (proposal)

Only forests with minimal human-caused disturbances were used, so that the cover types are rather based on ecological processes. It is often used as a benchmark for drift algorithms [30]–[32].

Poker One million randomly drawn poker hands are represented by five cards each encoded with its suit and rank. The class is the poker hand itself such as one pair, full house and so forth. The original form [33] has no drift, since the poker hands are static and instances are randomly generated. However, we used the version presented in [30], containing virtual drift via sorting the instances by rank and suit.

MNIST-8M Loosli et al. used in [34] pseudo-random deformations and translations to extended the well known MNIST database [35] to eight million instances. The ten handwritten digits are encoded in 782 binary features. The dataset is already shuffled in its original publication and therefore contains no concept drift.

HIGGS This task consists of eleven million simulated particle collisions and was initially published in [36]. The goal of this binary classification problem is to distinguish between a signal process producing Higgs bosons and a background process. The data consist of low-level kinematic features recorded as well as some derived high-level indicators.

VII. EXPERIMENTS

We rely on MOA as framework for our experiments and integrated all split-time prediction in its implementation of the VFDT. Our code as well as links to all datasets are available at GitHub[1]. We use the majority class in the leaves for classification and the default parameters of the VFDT ($n_{\text{min}} = 200, \tau = 0.05, \delta = 1e - 7$) to provide comparable results to other publications. We investigate how our method performs in stationary and non-stationary environments by performing two experiments for each dataset. One uses the original order of the data, whereas the other shuffles the data to ensure stationarity. The datasets RBF, RTG and MNIST8M are already shuffled in their original form, therefore, we report their result only in the shuffled experiments. Table II lists the algorithms that are used in the experiments. The split-delay of *CGD* is too large in comparison to the other methods[2], therefore, we neglect its uncompetitive results for the most part due to space limitations and in benefit of reasonably scaled plots. However, we always report its average performances and include it in the crucial analysis of the classification error.

TABLE III
THE NUMBER OF SPLIT-ATTEMPTS AS WELL AS THE AVG. SPLIT-DELAYS OF *VFDT* AND *OSM*. THE RESULTS FOR THE SHUFFLED DATA ARE GIVEN AT THE TOP, WHEREAS THOSE USING THE ORIGINAL ORDER ARE AT THE BOTTOM. *OSM* REQUIRES ALWAYS FEWER ATTEMPTS AND PROVIDES SIMILAR SPLIT-DELAYS. *CGD* HAS THE SMALLEST AMOUNT OF ATTEMPTS BUT ALSO AN INTOLERABLE HIGH SPLIT DELAY (ON AVERAGE 701 ATTEMPTS, 7675 SPLIT-DELAY [$n_{\text{MIN}} = 200$]).

	$n_{min} = 50$				$n_{min} = 200$			
Dataset	#attempts		∅delay		#attempts		∅delay	
(shuffled)	*VFDT*	*OSM*	*VFDT*	*OSM*	*VFDT*	*OSM*	*VFDT*	*OSM*
RTG	69900	**13883**	**18**	29	17102	**7660**	**94**	98
RBF	170487	**1921**	**22**	28	42702	**1640**	**118**	125
LEDDrift	199933	**11327**	23	**20**	49897	**9687**	**111**	113
Rialto	1644	**273**	45	**45**	409	**235**	90	**50**
Airline	8904	**6665**	15	**13**	1821	**1811**	103	**103**
Cover type	11613	**1390**	32	**26**	2900	**1049**	86	**55**
Poker	16570	**1732**	**26**	27	4138	**1382**	**92**	96
MNIST8M	161975	**13283**	**20**	21	40420	**11859**	**104**	107
HIGGS	68966	**24725**	**24**	56	17564	**14889**	**103**	118
∅	78888	**8355**	**25**	**29**	**19661**	**5579**	100	**96**

	$n_{min} = 50$				$n_{min} = 200$			
Dataset	#attempts		∅delay		#attempts		∅delay	
(original)	*VFDT*	*OSM*	*VFDT*	*OSM*	*VFDT*	*OSM*	*VFDT*	*OSM*
LEDDrift	199927	**11154**	23	**21**	49891	**9528**	**111**	113
Rialto	1643	**243**	25	**5**	409	**211**	120	**120**
Airline	8467	**6631**	5	**5**	1618	**1618**	98	**98**
Cover type	11028	**1703**	**21**	1113	2727	**1098**	**100**	934
Poker	16133	**2662**	**20**	132	3968	**1750**	**93**	155
HIGGS	219624	**74730**	**22**	42	55992	**46559**	**88**	97
∅	76137	**16187**	**19**	219	19101	**10127**	**102**	252

A. Split-delay versus split-attempts

In the following, we investigate the trade-off between the split-delay and the number of split-attempts achieved by the *VFDT* and our approach *OSM*. We generate Pareto fronts between the split-delay and the number of split-attempts by varying the parameter n_{min} in the range of $[50, 500]$ [3]. Figure 2 shows the corresponding results for various datasets, whereas Table III lists the values for all dataset with $n_{\text{min}} \in \{50, 200\}$.

The *VFDT* requires clearly more attempts particularly for low n_{min} values. Its average split-delay converges as expected around $\frac{n_{\text{min}}}{2}$. Even though it is able to achieve an arbitrary low split-delay by definition, the computational cost increases drastically. In case of stationarity, the Pareto fronts of *OSM* are always more optimal than those of *VFDT*. *OSM* achieves a similar split-day with substantially fewer split-attempts. The discrepancy is especially high for low n_{min} values, where the number of split-attempts performed by *OSM* is hardly affected. The split-delay smoothly adapts with the corresponding n_{min} value. This is confirmed by the average results given in Table III. The split-delay of *OSM* is similar or even lower (for $n_{\text{min}} = 200$), but the number of split-attempts is drastically

[1] https://github.com/ICDM2018Submission/VFDT-split-time-prediction

[2] *CGD* has an average split-delay of 7600, *OSM* has a delay of 96.

[3] Please note, that *OSM* still uses n_{min} as minimum prediction bound.

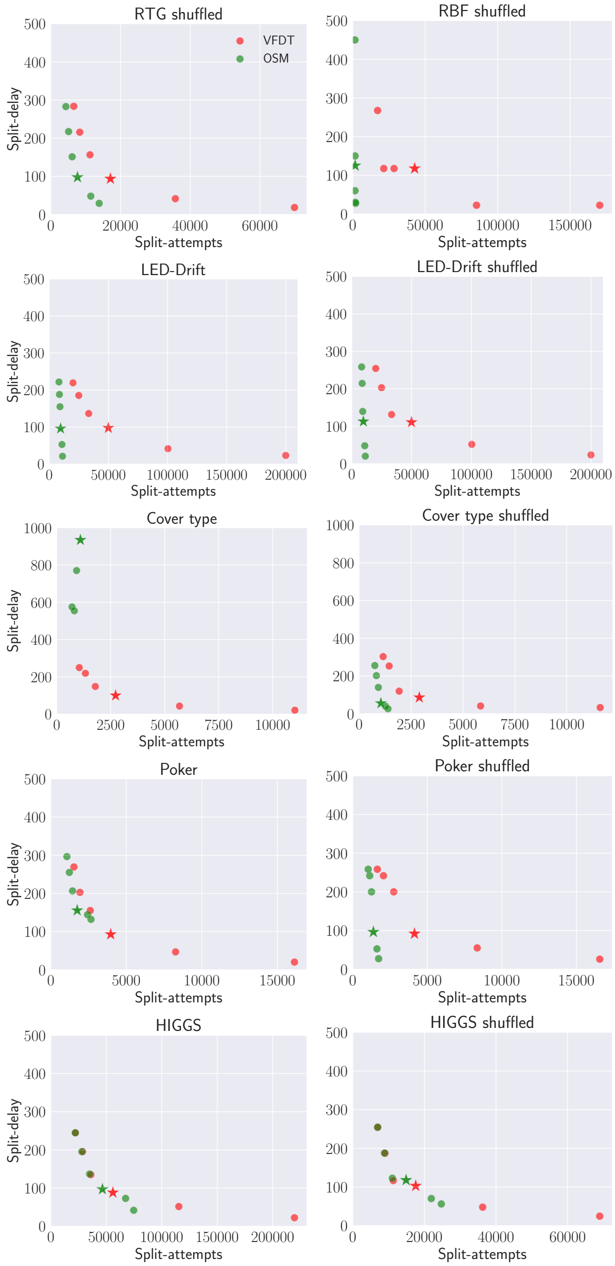

Fig. 2. Trade-off between the number of split-attempts and the split-delay resulting from varying n_{min} in $[50, 500]$. A star highlights results which are achieved with the default setting of $n_{\text{min}} = 200$. *OSM* also provides better results for data with concept drift (original order) with the exception of *Cover type*.

TABLE IV

CHARACTERISTICS OF THE SPLIT-TIME DISTRIBUTIONS OF VFDT ($n_{\text{MIN}} = 200$). THE AVERAGE SPLIT-TIME VARIES DUE TO THE DIFFICULTY OF THE TASK, THE NUMBER OF CLASSES AS WELL AS THE AMOUNT OF NOISE / OVERLAP.

Dataset (original)	∅ split-time	bound-splits	tie-splits
RTG	3.2K	508	5
RBF	78.1K	2	102
LEDDrift	30.1K	41	180
Rialto	19.7K	2	1
Airline	1.9K	27	27
Cover type	19.1K	6	16
Poker	18.6K	12	21
MNIST8M	34.1K	13	168
HIGGS	3.2K	50	2546

reduced.

OSM also performs well in non-stationary environments with the exception of *Cover type*, where the split-delay is higher in comparison to *VFDT*. In contrast to all other datasets, it is not able to reduce the split-delay with lower values of n_{min}. In this particularly case, the best split-attribute drastically changes, violating the assumptions of the approach. Hence, one weak point of the method is that it rarely overestimates the split-time in non-stationary environments.

The relative reduction of split-attempts by *OSM* varies depending on the dataset, mainly caused by the corresponding split-time distribution. A high average split-time increases the leeway for the prediction algorithms to gain an advantage by predicting times far into the future. Table IV shows the split-time distributions as well as the number of bound- and tie-splits. The correlation between the reduction in split-attempts and the average split-time is quite pronounced, e.g. RBF versus HIGGS. The split-time depends on the data structure itself and the contained amount of noise / overlap. A high amount of noise requires more examples to confidently determine the best split-attribute, whereas splits are generated quickly for data that can easily be discriminated along a few dimensions. Datasets with a high proportion of bound-splits and a low average split-time as RTG are rather easy because the best split-attributes are quickly determined. The opposite is the case for datasets with a high percentage of tie-splits such as RBF and HIGGS. However, the fact that RBF and HIGGS have still very different split-times is explained by the different amount of classes within the datasets. The number of examples the algorithm waits until a tie-split is performed depends on the number of classes within the leaf[1]. Hence, the split-time of tie-splits correlates with the number of classes in the corresponding dataset, making split-time predictions particularly appealing for multi-class classification tasks.

The high proportion of tie-splits, which are supposed to rarely happen, for HIGGS and MNIST cast doubt on the guarantee of the VFDT to construct similar trees as batch algorithms.

[1] The range of the entropy depends on the number of classes ($R = \log c$).

TABLE V

TOP: AVERAGE ERROR RATE (%) AND CORRESPONDING STANDARD DEVIATIONS OBTAINED FROM 100 REPETITIONS, EACH TIME RESHUFFLING THE DATA. THE BEST RESULTS ARE MARKED IN BOLD. RESULTS WHICH ARE SIGNIFICANTLY WORSE ARE UNDERLINED (p-VALUE < 0.05).
BOTTOM: ERROR RATES OF THE APPROACHES ACHIEVED WITH THE ORIGINAL ORDER OF THE DATA. THE SIGNIFICANCE TEST IS NOT POSSIBLE FOR THE DRIFT-SETTING, SINCE VFDT IS A DETERMINISTIC ALGORITHM AND RESHUFFLING THE DATA REMOVES THE DRIFT.

Dataset (shuffled)	*VFDT*	*CGD*	*OSM*
RTG	15.32(0.2)	$\underline{17.69(0.5)}$	**15.30(0.2)**
RBF	19.26(0.5)	19.34(0.5)	**19.23(0.5)**
LEDDrift	27.35(0.1)	$\underline{27.46(0.1)}$	**27.34(0.1)**
Rialto	84.26(0.3)	$\underline{86.11(0.6)}$	**84.26(0.3)**
Airline	39.57(0.3)	39.57(0.3)	**39.54(0.3)**
Cover type	33.80(0.2)	$\underline{34.14(0.4)}$	**33.79(0.3)**
Poker	38.34(0.3)	38.32(0.3)	**38.26(0.4)**
MNIST8M	40.53(0.5)	40.78(0.6)	**40.53(0.5)**
HIGGS	30.63(0.2)	30.63(0.2)	**30.59(0.2)**
∅	36.75	37.67	**36.73**

Dataset (original)	*VFDT*	*CGD*	*OSM*
LEDDrift	27.17	27.24	**27.17**
Rialto	**86.04**	86.36	86.18
Airline	38.76	38.78	**38.72**
Cover type	**30.27**	32.61	31.32
Poker	**31.39**	41.2	31.69
HIGGS	**30.3**	30.58	30.67
∅	**40.66**	42.8	40.96

B. Error rates

The shuffling of the data enables us to test for significant error rate differences between the approaches. Welch's t-test [37] was used, a two-sample location test allowing different variances between the samples. We performed 100 repetitions, each time reshuffling the data. Table V shows at the top the average error rates with corresponding standard deviations. Our split-time prediction achieves the best results for all shuffled datasets and is significantly better than *CGD* on half of them, underlining that *OSM* does not reduce the classification performance. The *CGD* approach performs significantly worse due to its frequent and severe overestimations of the split-time. We are not able to test for significance in the experiments with original order of the data. The deterministic results are given at the bottom of Table V. Our method *OSM* delivers also in this case similar results to *VFDT*, proving its robustness in non-stationary environments. The increased split-delay for the Cover type data is only slightly affecting its error rate. The *CGD* method performs worse for each task, particularly in the case of the Poker data. The temporal course of the error rate is depicted in Figure 3 for some datasets. In case of non-shuffled data, *CGD* is performing worse than the normal VFDT, whereas *OSM* continuously delivers a similar classification performance.

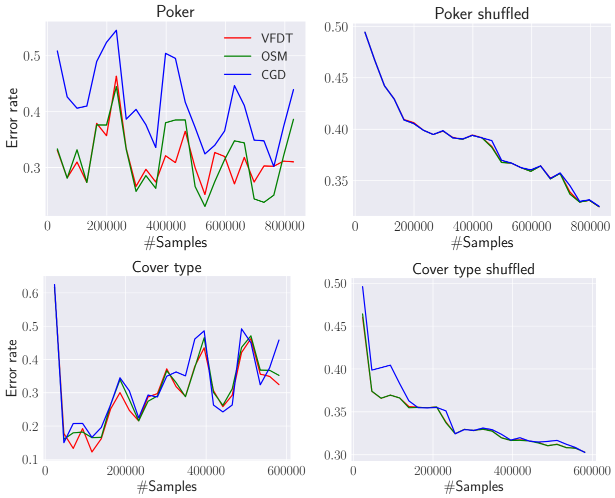

Fig. 3. The error course of all methods for two exemplary datasets using the shuffled as well as the original order.

TABLE VI

THE ATTEMPT-TIMES AS WELL AS CORRESPONDING RELATIVE SPEEDUPS ACHIEVED BY *OSM*. PLEASE NOTE, THAT ONLY THE EFFICIENCY OF THIS PART OF THE ALGORITHM HAS BEEN IMPROVED. THE SPEEDUP DRASTICALLY INCREASES WITH SMALLER n_{min} VALUES BECAUSE *OSM* IS LESS DEPENDENT ON THIS PARAMETER, USING IT ONLY AS MINIMUM PREDICTION BOUND. *CGD* HAS THE LOWEST AVERAGE ATTEMPT TIME OF 2.5 SECONDS AND A SPEEDUP OF 41 ($n_{min} = 200$).

	$n_{min} = 50$			$n_{\text{min}} = 200$		
Dataset	Attempt time (s)		Speedup (×)	Attempt time (s)		Speedup (×)
(original)	*VFDT*	*OSM*	*OSM*	*VFDT*	*OSM*	*OSM*
RTG	590.87	**127.55**	4.63	153.57	**80.53**	1.91
RBF	374.19	**6.12**	61.14	92.87	**5.48**	16.95
LEDDrift	19.67	**1.37**	14.36	5	**1.05**	4.76
Rialto	0.62	**0.11**	5.64	0.15	**0.07**	2.14
Airline	1.98	**1.71**	1.16	0.49	**0.47**	1.04
Cover type	1.63	**0.21**	7.76	0.45	**0.19**	2.37
Poker	1.25	**0.18**	6.94	0.35	**0.13**	2.69
MNIST8M	2504.54	**205.31**	12.2	644.87	**191.69**	3.36
HIGGS	50.62	**18.19**	2.78	14.26	**11.58**	1.23
∅	393.93	**40.08**	12.96	101.33	**32.35**	4.05

C. Computation time

The run-time was measured on a cluster with 256 GB RAM and 40 cores, each with 2.6 GHz. We present only the time used by the VFDT algorithm, excluding processes such as reading the data from the hard drive, parsing the file and so forth. Table VI depicts the time spend on the split-attempts as well as corresponding speedups of *OSM*. Our approach clearly reduces the attempt time for most datasets. The speedup is particularly high for the RBF and MNIST8M and is in general drastically increased for $n_{min} = 50$, highlighting the fact that *OSM* substantially scales better for low n_{min} values.
Table VII shows the total time of the approaches and corresponding speedups achieved by our split-time prediction method. *OSM* reduces the overall run-time for all datasets, in some cases as RBF and MNIST8M quite drastically. Figure 4 shows the exponentially growing total speedup for very low

TABLE VII

The runt-time of the whole learning algorithm as well as corresponding relative speedups achieved by *OSM*. The speedup is limited, since the run-time of a sub-part is reduced. However, a clear run-time reduction is achieved most of the times, particularly in case of $n_{min} = 50$. *CGD* has on average the lowest total time of 106 seconds and a speedup of 1.97 ($n_{min} = 200$).

	$n_{min} = 50$			$n_{min} = 200$		
Dataset	Total time (s)		Speedup (×)	Total time (s)		Speedup (×)
(original)	*VFDT*	*OSM*	*OSM*	*VFDT*	*OSM*	*OSM*
RTG	706.47	**246.80**	2.86	277.28	**206.56**	1.34
RBF	495.12	**114.30**	4.33	199.09	**91.71**	2.17
LEDDrift	35.18	**16.72**	2.10	19.4	**14.93**	1.30
Rialto	0.72	**0.21**	3.43	0.24	**0.16**	1.50
Airline	3.83	**3.49**	1.10	2.3	**2.29**	1.00
Cover type	3.40	**1.58**	2.15	2.01	**1.41**	1.43
Poker	1.93	**0.79**	2.44	0.95	**0.71**	1.34
MNIST8M	3097.27	**812.52**	3.81	1255.6	**794.97**	1.58
HIGGS	98.79	**65.5**	1.51	67.39	**61.92**	1.09
∅	493.63	**140.21**	2.64	202.7	**130.52**	1.42

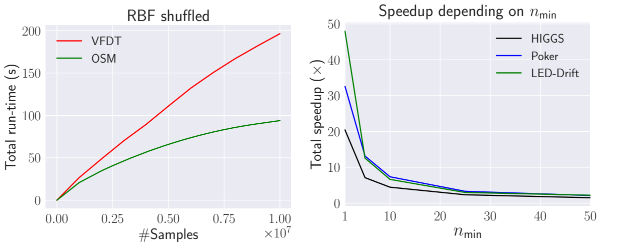

Fig. 4. On the left: The course of the total run-time for the RBF dataset. The run-time is not increasing linearly because the accuracy increases over time, reducing the frequency of the splits ($n_{min} = 200$). On the right: The total speedup of *OSM* increases drastically for low n_{min} values.

n_{min} values on the right. The development of the run-time depending on the number of instances for the RBF dataset can be seen on the left of Figure 4. It can be seen that the speedup would be even higher for more instances.

Our approach optimizes only the efficiency of split-attempts, hence the potential overall-speedup is limited by the relative time of the split-attempts in comparison to the total time of the algorithm. Figure 5 illustrates on the right that the relative time increases with more classes[1], which is another reason why split-time predictions are particularly valuable for multi-class classification tasks.

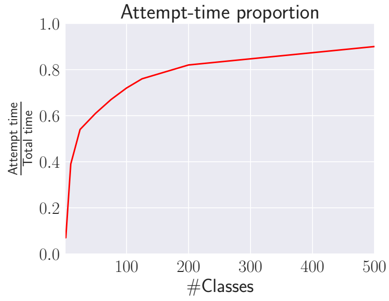

Fig. 5. The attempt-time in comparison to the total depending on the number of classes in the RBF task. The relative proportion clearly increases with more classes in the dataset, making split-time prediction particularly valuable for multi-class problems.

VIII. Conclusion

In this paper, we improved the efficiency of the VFDT by replacing its periodic split-attempt scheme. Instead, we use the local statistics within a leaf to accurately predict its split-time. On the basis of the resulting class-distributions from the best split, we approximate the minimum amount of required additional examples until the Hoeffding bound is met. The yielded approximations are for the most part equivalent to those of a greedy approach, but are obtained at a substantially reduced computational cost. We formally introduced the term *split-delay* as a precise measurement for evaluating split-time predictions, instead of the potentially misleading classification performance. In an extensive evaluation, we empirically showed a distinct speed-up of our approach in comparison to the default VFDT, particularly for multi-class classification tasks. Our method delivers the same classification performance as VFDT while being significantly better for various datasets than a recently published method. Another advantage is the reduced dependence of the run-time on the parameter n_{min}. The approach can easily be integrated in parallelized implementations of the VFDT to increase the efficiency even further.

Appendix

In the following, we show that *OSM* yields for the most part equivalent estimates as the greedy search. To keep the notation clear we define $M_1 = M(a_1^{j_1})$ and $M_2 = M(a_2^{j_1})$. Adding one example to a class distribution has a dwindling effect on the class frequencies and consequently on the entropy. Hence, treating both single entropies as constants, we can determine the partial derivatives with respect to the size of each class distribution

$$\frac{\partial H(M_1, M_2)}{\partial |M_1|} = \frac{|M_2|(H(M_1) - H(M_2))}{(|M_1| + |M_2|)^2}$$
$$\frac{\partial H(M_1, M_2)}{\partial |M_2|} = \frac{|M_1|(H(M_2) - H(M_1))}{(|M_1| + |M_2|)^2}.$$

The sign of the gradient depends only on the difference between the single entropies $H(M_1) - H(M_2)$ and vice versa. Therefore, if the entropies were constant, the greedy search would yield the same solution as *OSM*, always adding examples to the class distribution having initially the lower entropy.

Taking the reduction of the entropies into account, we determine under which circumstances the solutions deviate. Assume $H(M_1) < H(M_2)$ and we iteratively add examples exclusively to M_1. Suppose we increase the number of M_1's

[1] The run-time of split-attempts is dominated by the determination of the resulting class distributions for considered splits, requiring an iteration through all class-dependent feature distributions.

most frequent class by one, leading to the new distribution M_1' and the corresponding entropy $H(M_1') = H(M_1) + \delta_1$, where $\delta_1 < 0$. Analogously for M_2 we get M_2' and $H(M_2') = H(M_2) + \delta_2$. The denominator of $H(M_1, M_2)$ is unaffected by the choice, thus our solution deviates from the greedy search when

$$\begin{aligned} |M_2|\delta_2 + H(M_2') &< |M_1|\delta_1 + H(M_1') \\ (|M_2|+1)\delta_2 + H(M_2) &< (|M_1|+1)\delta_1 + H(M_1) \\ (|M_2|+1)\delta_2 &< (|M_1|+1)\delta_1 + H(M_1) - H(M_2) \end{aligned}$$

$(|M_2|+1)\delta_2$ has not only to be smaller than $(|M_1|+1)\delta_1$, but it additionally has to make up for the entropy difference $H(M_1) - H(M_2)$, which is negative since we assume that $H(M_1) < H(M_2)$. Since δ_k decreases with increasing size of the distribution, the term $|M(a_k^{j_1})|\delta_k$ remains approximately constant. Hence, the only likely scenario that our method deviates from the greedy search is when initially $H(M_1) \approx H(M_2)$. Please note, that the greedy search unlikely switches between M_1 and M_2 because the entropy difference increases when additional examples are distributed.

References

[1] R. Spencer, "The square kilometre array: The ultimate challenge for processing big data," in *IET Seminar on Data Analytics 2013: Deriving Intelligence and Value from Big Data*, Dec 2013.

[2] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson, "Big data: Astronomical or genomical?" *PLOS Biology*, no. 7, 07 2015.

[3] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, Fourthquarter 2015.

[4] V. Losing, B. Hammer, and H. Wersing, "Incremental on-line learning: A review and comparison of state of the art algorithms," *Neurocomputing*, vol. 275, 2018.

[5] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proc. of the Sixth SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.

[6] J. R. Quinlan, *C4.5: Programs for Machine Learning*, 1993.

[7] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, "Classification and regression trees," 1993.

[8] E. Garca-Martn, "Extraction and energy efficient processing of streaming data," 2017.

[9] A. Saffari, C. Leistner, J. Santner, M. Godec, and H. Bischof, "On-line random forests," in *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, Sept 2009.

[10] A. Wang, G. Wan, Z. Cheng, and S. Li, "An incremental extremely random forest classifier for online learning and tracking," in *2009 16th IEEE International Conference on Image Processing (ICIP)*, Nov 2009.

[11] G. Holmes, K. Richard, and B. Pfahringer, "Tie-breaking in hoeffding trees," in *Proc. of the Workshop W6 The Second International Workshop on Knowledge Discovery in Data Streams*, 2005.

[12] R. B. Kirkby, "Improving hoeffding trees," Ph.D. dissertation, The University of Waikato, 2007.

[13] R. Brent, *Algorithms for Minimization Without Derivatives*, ser. Dover Books on Mathematics, 1973.

[14] H. Yang and S. Fong, "Optimized very fast decision tree with balanced classification accuracy and compact tree size," in *The 3rd International Conference on Data Mining and Intelligent Information Technology Applications*, Oct 2011.

[15] C. Li, Y. Zhang, and X. Li, "Ocvfdt: One-class very fast decision tree for one-class classification of data streams," in *Proc. of the Third International Workshop on Knowledge Discovery from Sensor Data*, ser. SensorKDD '09, 2009.

[16] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine Learning*, vol. 29, no. 2, Nov 1997.

[17] J. Gama, R. Rocha, and P. Medas, "Accurate decision trees for mining high-speed data streams," in *Proc. of the Ninth SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, 2003.

[18] R. Kohavi, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid," in *Proc. of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96, 1996.

[19] G. Holmes, R. Kirkby, and B. Pfahringer, "Stress-testing hoeffding trees," in *Knowledge Discovery in Databases: PKDD 2005: 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, Porto, Portugal, October 3-7, 2005. Proc.*, 2005.

[20] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, 1958.

[21] A. Bifet, G. Holmes, B. Pfahringer, and E. Frank, "Fast perceptron decision tree learning from evolving data streams," in *Advances in Knowledge Discovery and Data Mining: 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proc.. Part II*, 2010.

[22] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proc. of the Seventh SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001.

[23] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *Advances in Intelligent Data Analysis VIII: 8th International Symposium on Intelligent Data Analysis, IDA 2009, Lyon, France, August 31 - September 2, 2009. Proc.*, 2009.

[24] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Machine Learning and Knowledge Discovery in Databases*, 2010.

[25] N. Kourtellis, G. D. F. Morales, A. Bifet, and A. Murdopo, "Vht: Vertical hoeffding tree," in *2016 IEEE International Conference on Big Data (Big Data)*, Dec 2016.

[26] A. Bifet, J. Zhang, W. Fan, C. He, J. Zhang, J. Qian, G. Holmes, and B. Pfahringer, "Extremely fast decision tree mining for evolving data streams," in *Proc. of the 23rd SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17, 2017.

[27] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *JMLR*, vol. 11, 2010.

[28] V. Losing, B. Hammer, and H. Wersing, "Knn classifier with self adjusting memory for heterogeneous concept drift," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, Dec 2016.

[29] J. A. Blackard, "Comparison of neural networks and discriminant analysis in predicting forest cover types," Ph.D. dissertation, 1998, aAI9921979.

[30] A. Bifet, B. Pfahringer, J. Read, and G. Holmes, "Efficient data stream classification via probabilistic adaptive windows," in *Proc. of the 28th Annual Symposium on Applied Computing*, ser. SAC '13, 2013.

[31] J. Gama, R. Rocha, and P. Medas, "Accurate decision trees for mining high-speed data streams," in *Proc. of the Ninth SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

[32] N. C. Oza and S. Russell, "Experimental comparisons of online and batch versions of bagging and boosting," in *Proc. of the Seventh SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2001.

[33] M. Lichman, "UCI machine learning repository," 2013.

[34] G. Loosli, S. Canu, and L. Bottou, "Training invariant support vector machines using selective sampling," in *Large Scale Kernel Machines*, L. Bottou, O. Chapelle, D. DeCoste, and J. Weston, Eds., 2007.

[35] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, Nov 1998.

[36] P. Baldi, P. Sadowski, and D. Whiteson, "Searching for Exotic Particles in High-Energy Physics with Deep Learning," *Nature Commun.*, vol. 5, 2014.

[37] B. L. Welch, "The generalization ofstudent's' problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1/2, 1947.