Dynamic Model Tree for Interpretable Data Stream Learning

Johannes Haug  
University of Tuebingen  
Tuebingen, Germany  
johannes-christian.haug@uni-tuebingen.de

Klaus Broelemann  
Schufa Holding AG  
Wiesbaden, Germany  
klaus.broelemann@schufa.de

Gjergji Kasneci  
University of Tuebingen  
Tuebingen, Germany  
gjergji.kasneci@uni-tuebingen.de

Abstract—Data streams are ubiquitous in modern business and society. In practice, data streams may evolve over time and cannot be stored indefinitely. Effective and transparent machine learning on data streams is thus often challenging. Hoeffding Trees have emerged as a state-of-the-art for online predictive modelling. They are easy to train and provide meaningful convergence guarantees under a stationary process. Yet, at the same time, Hoeffding Trees often require heuristic and costly extensions to adjust to distributional change, which may considerably impair their interpretability. In this work, we revisit Model Trees for machine learning in evolving data streams. Model Trees are able to maintain more flexible and locally robust representations of the active data concept, making them a natural fit for data stream applications. Our novel framework, called Dynamic Model Tree, satisfies desirable consistency and minimality properties. In experiments with synthetic and real-world tabular streaming data sets, we show that the proposed framework can drastically reduce the number of splits required by existing incremental decision trees. At the same time, our framework often outperforms state-of-the-art models in terms of predictive quality — especially when concept drift is involved. Dynamic Model Trees are thus a powerful online learning framework that contributes to more lightweight and interpretable machine learning in data streams.

Index Terms—machine learning, data stream, model tree, concept drift, interpretability

I. INTRODUCTION

Large-scale data streams are integral to most modern web-based applications such as online credit scoring, e-commerce or social media. Accordingly, the demand for powerful streaming machine learning models has increased. In practice, streaming or online learning models have to cope with limited hardware capacity and drifts of the data generating concept. Efficient, accurate and interpretable machine learning for evolving data streams is thus a major challenge.

Unlike traditional batch learning models, online learning models are updated incrementally. In this way, online learning models can be trained without the entire data set being available in main memory. Consequently, online learning models enable machine learning in practical applications that generate a potentially unlimited amount of data, e.g., large sensor systems or credit card transactions.

Online learning models usually have to cope with limited hardware capacity and drifts of the data generating concept. Changing customer preferences or emerging social media trends are prominent examples of such concept drift. In the worst case, concept drift may render previously learned concepts obsolete.

Accordingly, online learning models must provide discriminative predictions and adjust to concept drift, while reducing overall resource consumption. In addition, much attention has recently been paid to the interpretability of machine learning models [1], [2]. In particular, high-stakes applications and regulations (e.g., the EU General Data Protection Regulation (GDPR)) require models to be interpretable. For example, if a model is used to predict the risk of recidivism or the probability of a loan default, it can be crucial to be able to describe the model in understandable terms. However, compared to other domains such as image recognition [3], relatively little attention has been paid to the interpretability of machine learning models in evolving data streams. As one of the first works, we therefore briefly outline important aspects of interpretable online learning below.

A. On “Interpretability” in Evolving Data Streams

In general, we distinguish between post-hoc explainability and intrinsic interpretability [4]. The former concerns dedicated methods, e.g., local feature attributions [5]–[8], that allow to explain complex (black-box) models. Conversely, we speak of intrinsic interpretability when the internal mechanics of the predictive model are inherently understandable to a human.

Interpretability has varying domain-specific definitions [2] and cannot be measured in a standardized way [4]. Hence, interpretability is often represented by heuristic measures such as model size or complexity [9]. Intuitively, it is easier for humans to attribute meaning to individual model parameters when complexity is low. That is, the less complex a model is, the easier it is to interpret. For example, linear models and decision trees are typically considered highly interpretable. Specifically, the interpretability of linear models can be linked to their sparsity, i.e., the number of nonzero parameters. Similarly, the interpretability of decision trees can be quantified by the number of split nodes or the depth of the tree [10].

Since online learning models are incrementally updated, the parameters and model complexity can change between time steps. Therefore, in order to achieve interpretable online learning, we argue that it is not sufficient to deliver low complexity at each individual time step. Rather, changes in model complexity must also be comprehensible to humans.
Fig. 1: Hoeffding Tree vs. Model Tree: A Hoeffding Tree (HT, red) aims for maximum node purity regarding the target (here we assume a binary target, i.e. white, grey circles). A Model Tree (MT, green), on the other hand, trains simple (linear) models at every leaf node (with parameters $\theta_t$), and aims to minimize the estimated loss. Above, the dotted lines represent binary splits of the corresponding trees, and the dashed, green lines indicate the linear functions of the Model Tree. Model Trees are capable to maintain more flexible representations of the active data concept than existing frameworks like the Hoeffding Tree (see Figure 1). In particular, Model Trees can represent linear relationships with only a few splits. Hence, Owing to the simple models, Model Trees are able to apply a less rigid separation of observations in the leaf nodes. In comparison to batch-trained decision trees, Hoeffding Trees suffer practical limitations. For example, the way in which Hoeffding’s inequality and heuristic purity measures are used within the framework has been repeatedly questioned [15]–[17].

C. Model Trees As An Alternative to Hoeffding Trees

In this work, we revisit Model Trees as an alternative to Hoeffding Trees [18]–[21]. Model Trees have much in common with regular decision trees, but contain simple predictive models in place of each (leaf) node. Hence, similar to an ensemble, Model Trees are a collection of weak learners that are combined in a structured way through a set of binary decisions. However, unlike Hierarchical Mixtures of Experts, Model Trees use only a single feature to split at each inner node. Accordingly, Model Trees preserve much of the simplicity of a regular decision tree.

Owing to the simple models, Model Trees are able to apply a less rigid separation of observations in the leaf nodes. In this way, Model Trees are generally more flexible regarding the active data concept than existing frameworks like the Hoeffding Tree (see Figure 1). In particular, Model Trees can represent linear relationships with only a few splits. Hence, Model Trees can usually achieve high predictive quality while using a simple and robust representation.

Replacing regular leaf nodes with simple models in an otherwise unmodified tree increases complexity. However, Model Trees often remain extremely shallow and thus interpretable, as we show in experiments. In addition, unlike Hoeffding Trees, Model Trees allow feature weights for different subgroups to be extracted directly from the simple models. In comparison to majority weighting schemes, this can be an advantage for local feature-based explanations.
D. Our Contribution

In this paper, we introduce a novel online learning framework called Dynamic Model Tree. We show that the simple models of a Model Tree can be leveraged to define node-specific gain functions. These gain functions guarantee sensitivity and minimality properties, which contribute to more intuitive and interpretable online learning. Compared to existing state-of-the-art methods such as Hoeffding Trees or earlier incremental Model Trees [20], [21], Dynamic Model Trees adapt to concept drift by design. In particular, the proposed framework does not rely on Hoeffding’s inequality, heuristic purity measures or explicit concept drift detection mechanisms. Consequently, the Dynamic Model Tree eliminates some of the most fundamental weaknesses of existing online decision trees.

In summary, the contributions of this work are as follows:

- We specify valuable properties related to the consistency and minimality of incremental decision trees (Section III). Combined, these properties lead to more interpretable online learning as described above.
- We introduce the Dynamic Model Tree framework (Section IV). In particular, we define generic gain functions that guarantee the above-mentioned properties and can be efficiently approximated via gradients [19].
- We propose an effective implementation of the Dynamic Model Tree that uses Generalized Linear Models and the negative log-likelihood loss (Section V).
- We evaluate the Dynamic Model Tree on multiple synthetic and real-world tabular data sets with different types of concept drift (Section VI). While maintaining high efficiency, our implementation often outperforms existing classifiers in terms of predictive quality and complexity.

II. Related Work

Incremental decision trees are a powerful class of online predictors. In the following, we briefly outline state-of-the-art algorithms based on the Hoeffding Tree, along with their limitations. Moreover, we discuss previous works on incremental Model Trees. For more information about online learning, we refer to recent surveys [22]–[24].

A. Variations of the Hoeffding Tree

The Very Fast Decision Tree (VFDT) is the first and basic implementation of a Hoeffding Tree [11]. As mentioned above, it has practical limitations. In particular, the VFDT assumes that a relatively small set of past and current observations is representative of all future observations – a misconception under realistic streaming conditions. Accordingly, the VFDT grows indefinitely and does not revisit old split decisions, which can impair its interpretability.

Most of the limitations of the basic VFDT can be overcome, e.g. by using regularization [25], different probabilistic inequalities [15], gain measures [16], [17] or tricks in the implementation [26]. To increase the predictive performance under concept drift, the Hoeffding Tree may also be augmented with adaptation strategies like alternate tree growth [12], sliding windows [13] or a dynamic replacement of inner nodes [14]. In addition, ensembles of Hoeffding Trees, e.g. Bagging or Boosting [27], [28], can increase the predictive performance of the basic models at the cost of higher overall complexity.

The Hoeffding Tree has gained popularity due to its rigorous convergence guarantees, efficiency, extensibility and accessibility via packages like MOA [29] or scikit-multiflow [30]. However, the inherent limitations of the basic architecture may ultimately leave users in doubt about the reliability of the Hoeffding Tree. Hence, we argue that a different framework is needed, which offers a similar level of efficiency and extensibility, but is more flexible and interpretable in a dynamic online environment.

B. Incremental Model Trees

Hoeffding Trees have been augmented with simple models, such as Naïve Bayes [31] and Perceptrons [32]. Such extensions often provide considerable improvements in predictive performance compared to majority-weighted leaves. Surprisingly, however, the more general family of Model Trees has received only little attention in online learning scenarios. Notable exceptions include the work by [20], which is aimed at stationary applications, and the FIMT-DD model [21]. FIMT-DD was introduced as a solution for online regression tasks. Similar to Hoeffding Trees, FIMT-DD applies Hoeffding’s inequality to split at the inner nodes. Specifically, FIMT-DD aims to find the split that gives the largest reduction in the standard deviation of the target variable. To avoid infinite growth, FIMT-DD employs explicit concept drift detection via the Page-Hinkley test and offers various adaptation strategies. The FIMT-DD model and the Dynamic Model Tree proposed in this work have fundamental differences, which we outline in Section IV.
and, for simplicity, let \( L(\Omega_t) \) be the shorthand notation for \( L(\Theta_{\Omega_t}, Y_{\Omega_t}, X_{\Omega_t}) \), which denotes the estimated loss of an incremental decision tree with respect to \( \Omega_t \). As before, we assume that the parameters \( \Theta_{\Omega_t} \) are given by the context.

**Property 1** (Consistency with Parent Splits). Suppose we perform a split at time step \( t \). Let \( L_{\text{C}}(\Omega_t) \) be the new estimated loss after the split. An incremental decision tree algorithm is consistent with parent splits regarding the set \( \Omega_t \), if \( L_{\text{C}}(\Omega_t) \leq L(\Omega_t) \).

Accordingly, we must avoid splits that would increase the estimated loss. This property primarily concerns the predictive quality of the obtained tree and is a common objective. Additionally, by choosing an adequate loss function that approximates the active data concept (which we discuss in Section \( V \), Property 1 enables interpretable split decisions.

With the goal of low model complexity, i.e., high interpretability, we add a second property:

**Property 2** (Model Minimality). Suppose there exists a subtree of the incremental decision tree at time step \( t \), whose loss is denoted by \( L_{\text{alt}}(\Omega_t) \). An incremental decision tree algorithm preserves model minimality regarding the set \( \Omega_t \), if for \( L(\Omega_t) = L_{\text{alt}}(\Omega_t) \) it retains the tree with fewer number of parameters.

Hence, we are bound to replace a complex tree, whenever it contains a simpler subtree that has equal predictive quality regarding \( \Omega_t \). For practical purposes, this means that we have to prune or replace nodes or branches of the tree that no longer improve the estimated loss (since the number of parameters per node is usually fixed). Consequently, Property 2 also implies a mechanism to adapt to concept drift.

**IV. DYNAMIC MODEL TREE**

In this paper, we extend Model Trees to a novel framework for adaptive predictive modelling in dynamic data streams that adheres to the aforementioned properties.

A Dynamic Model Tree is constructed in a similar fashion as regular decision trees. That is, we begin with a single root node and gradually grow and prune the tree over time. Each node of a Dynamic Model Tree can be represented by a set of time indices \( S_t \subseteq \{1, \ldots, t\} \) corresponding to the observations that have reached the node up to time step \( t \). Other than existing Model Trees, a Dynamic Model Tree maintains simple predictive models at both leaf and inner nodes (see Figure 2). These models are used to identify optimal split candidates (i.e., feature-value combinations) and make predictions. Let \( X_{S_t}, Y_{S_t} \) be the observations and labels, and \( \Theta_{S_t} \) the parameters of the simple model at a node corresponding to the time indices in \( S_t \). We aim to find the parameters that minimize a loss function \( L(\cdot) \geq 0 \):

\[
\Theta^*_{S_t} = \arg \min_{\Theta_{S_t}} L(\Theta_{S_t}, Y_{S_t}, X_{S_t})
\]

\[
= \arg \min_{\Theta_{S_t}} \sum_{t \in S_t} L(\theta_t, Y_t, X_t) \quad (1)
\]

**Fig. 2: DMT Nodes**: Both inner and leaf nodes of a Dynamic Model Tree contain simple models \( M \) that are incrementally trained during a subset of time steps \( S_t \) and \( I_t \), respectively. At every time step \( t \), we check at the leaf nodes whether there is a new split candidate with positive gain (3) (green, see also Algorithm 1). Similarly, we check at the inner nodes whether the gains (4) or (5) are positive, i.e., whether we must replace the current split (blue) and thus prune the old branch.

We assume independence between time steps; a simplifying, yet common assumption in data stream learning that has been shown to work well in practice. Accordingly, we can update the parameters \( \theta_t \) independently at every time step using gradient descent. The optimal parameters from the previous time step can be used as prior parameters at time step \( t \). Accordingly, at every time step, we forward incoming observations to a corresponding leaf node, updating each simple model along the path. Once we have updated all relevant simple models, we attempt to grow or prune the Dynamic Model Tree. To this end, we require gain measures that account for the aspired consistency with parent splits and model minimality.

**A. Loss-Based Gain Functions**

Typically, decision tree algorithms aim for maximum node purity with respect to the target variable. For this purpose, split decisions are usually based on heuristic purity measures such as the Information Gain or the Gini index. However, the simple models of a Dynamic Model Tree offer a fundamental advantage in terms of the proposed properties. Instead of relying on heuristic measures, we may directly select the split candidate that reduces the overall loss of our tree. Consequently, any update of the model complexity can be directly linked to a change in the loss, providing better interpretability as described in Section 4.

Suppose we are at a leaf node of the tree. Let \( S_t \) be the corresponding set of time indices observed at this leaf node. Our goal now is to find a new split candidate, i.e., a feature-value pair, to further split the observations. We can represent each split candidate by a set of time indices that would have been passed to the left child \( C_l \subseteq S_t \) and the right child \( C_r = S_t \setminus C_l \). For the sake of illustration, we assume binary splits. However, our exposition can readily be extended to non-binary trees. Our goal is to select the split candidate that maximizes the improvement of the current loss:

\[
C^*_t = \arg \max_{C_t} G_{S_t, C_t}, \quad \text{with} \quad G_{S_t, C_t} = L(\Theta_{S_t}, Y_{S_t}, X_{S_t}) - L(\Theta_{C_t}, Y_{C_t}, X_{C_t}) \quad (3)
\]
With (3), the proof of consistency to parent splits is almost trivial:

**Lemma 1.** Every new split with a gain $G_{S_t,C_t} \geq 0$ due to (4) implies consistency with parent splits (Property 1).

**Proof.** The loss of a Dynamic Model Tree at step $t$ corresponds to the sum of losses at each leaf node, i.e.
$$L(\Omega_t) = \sum_{J_t \subseteq I_t} L(\Theta_{J_t}, Y_{J_t}, X_{J_t}),$$
where every set $J_t$ represents a leaf node. Suppose there exists an inner node $S_t$, such that $G_{S_t,C_t} \geq 0$ for some split candidate $C_t$. A split on $C_t$ corresponds to a new loss $L_C(\Omega_t) = L(\Omega_t) - G_{S_t,C_t}$, which implies $L_C(\Omega_t) \leq L(\Omega_t)$. □

To satisfy model minimality (Property 2), we also need to evaluate existing splits of the Dynamic Model Tree. Specifically, we may replace an existing inner node with either a new split candidate or a leaf node. In both cases, we would prune the old branch (subtree). Suppose there is a subtree whose root corresponds to an inner node of the original tree. As before, we represent this inner node by a set of time indices $I_t$. Likewise, each leaf node of the subtree is represented by a set $J_t$, such that the union of all $J_t$ is equal to $I_t$. We then try to find an alternate split candidate (represented by $C_t \subseteq I_t$, $C_t = I_t \setminus C_t$), i.e. a substitute for the inner node $I_t$, which offers an improvement in terms of the loss:

$$G_{I_t,C_t} = \sum_{J_t \subseteq I_t} \left( L(\Theta_{J_t}, Y_{J_t}, X_{J_t}) - L(\Theta_{C_t}, Y_{C_t}, X_{C_t}) \right).$$

If the gain (4) is positive, we can prune the old subtree and add a new inner node with two new leaf nodes in its place. Alternatively, we may make the current inner node a leaf. To this end, we need to compare the loss at the inner node with the loss of the current subtree. The corresponding gain is

$$G_{I_t} = \sum_{J_t \subseteq I_t} \left( L(\Theta_{J_t}, Y_{J_t}, X_{J_t}) - L(\Theta_{I_t}, Y_{I_t}, X_{I_t}) \right).$$

If both gains (4) and (5) are positive and $G_{I_t,C_t} \geq G_{I_t}$, we apply the second option, replacing the inner node with a leaf node, to obtain the overall smaller tree. Notably, (4) and (5) allow us to maintain the minimality of a Dynamic Model Tree:

**Lemma 2.** Greedy replacement of inner nodes, wherever $G_{I_t,C_t} \geq 0$ due to (4) or $G_{I_t} \geq 0$ due to (5), implies model minimality (Property 2).

**Proof.** Let $I_t$ represent an inner node of the Dynamic Model Tree. There exists a subtree whose root is the inner node $I_t$. We may prune this subtree by replacing the inner node $I_t$ with a different split candidate or a leaf. The gain $G$ corresponds to (4) or (5) respectively. Accordingly, $L_{alt}(\Omega_t) = L(\Omega_t) - G$ is the loss of the potential alternate tree with the subtree replaced. Note that the alternate tree is guaranteed to have an equal or lower number of nodes and, since the number of parameters per node is fixed, an equal or lower number of parameters. Since $L_{alt}(\Omega_t) = L(\Omega_t)$ implies that $G = 0$, by assumption we would replace the Dynamic Model Tree by the alternate tree with the smaller number of parameters. This procedure may be repeated from the bottom to the root of the tree. □

### B. Candidate Loss Approximation

To compare the gains (4) or (5) of different split candidates, we require loss estimates $L(\Theta_{C_t}, Y_{C_t}, X_{C_t})$ and $L(\Theta_{C_t}, Y_{C_t}, X_{C_t})$ for each candidate. However, due to limited resources, we usually cannot train the simple models corresponding to every potential split candidate. For this purpose, we adopt an efficient gradient-based approximation.

The authors in [19] argue that we may warm-start optimizing the parameters of a split candidate $\Theta_{C_t}$ with a single gradient step on the parameters of the current node $\Theta_{S_t}$:

$$\Theta_{C_t} \approx \Theta_{S_t} - \frac{\lambda}{|C_t|} \nabla_{\Theta_{S_t}} L(\Theta_{S_t}, Y_{C_t}, X_{C_t})(6)$$

The first order Taylor polynomial at the point $\Theta_{S_t}$ then gives a good approximation to the loss of the split candidate $L(\Theta_{C_t}, Y_{C_t}, X_{C_t})$. Accordingly, we write

$$L(\Theta, Y_{C_t}, X_{C_t}) \approx L(\Theta, Y_{C_t}, X_{C_t}) + (\Theta - \Theta_{S_t})^T \nabla_{\Theta_{S_t}} L(\Theta_{S_t}, Y_{C_t}, X_{C_t}) + \frac{\lambda}{|C_t|} \| \nabla_{\Theta_{S_t}} L(\Theta_{S_t}, Y_{C_t}, X_{C_t}) \|^2_2.$$

With (7) we can approximate the loss of different split candidates without maintaining corresponding simple models. Moreover, we can reuse the gradient calculated during the optimisation of the parent model, which further increases efficiency. Finally, note that other work has successfully used gradient-based split finding [33].

### C. Basic Algorithm And Complexity

Algorithm 1 depicts the general procedure at a leaf node of the Dynamic Model Tree. For inner nodes, we compute the gain functions (4) and (5) in line 12. In line 19, we then replace the inner node with a new split or a leaf (depending on which gain is greater). Otherwise, the general update procedure is equivalent for both types of nodes. We update the nodes of the tree in a bottom-up fashion.

The time complexity of Algorithm 1 for updating one node without fitting the simple model is $O(m n t + m^2 v c)$, where $c$ is the number of classes, $m$ is the number of features, $n_t$ is the sample size at time step $t$ and $v$ is the maximal number of unique values of a feature. Depending on the choice of simple model, the time complexity might increase. If the maximal number of unique values is large, i.e. $v \gg n_t$, then the first term becomes negligible, leading to a complexity of $O(m^2 v c)$. In practice, decision tree algorithms often reduce computation time by limiting the number of eligible split candidates. This can be particularly important when we deal with large numbers of (continuous) features. We propose a simple method in Section V.

The memory complexity per node of the Dynamic Model Tree is $O(m^2 v c)$. As before, the memory requirements of the
Dynamic Model Tree scale with the number of split candidates considered.

D. Differences Between DMT and Earlier Methods

Dynamic Model Trees differ clearly from earlier work. A major difference lies in the way Dynamic Model Trees handle concept drift. While purity-based adaptation strategies usually require dedicated drift detection models to identify concept drift, a Dynamic Model Tree does not. In fact, adaptation to concept drift is automatically handled via the proposed gain functions. As a consequence, Dynamic Model Trees only have few hyperparameters that need to be optimized, while providing a similar level of flexibility as earlier works.

FIMT-DD is one of the most popular existing Model Tree frameworks for data streams [21]. In the following, we briefly highlight key differences between FIMT-DD and the Dynamic Model Tree. Like a Hoeffding Tree, FIMT-DD relies on a purity measure (Standard Deviation Reduction) and Hoeffding’s inequality to compare split candidates. That is, in FIMT-DD, “the process of learning linear models in the leaves will not explicitly reduce the size of the (...) tree” [21]. In addition, FIMT-DD requires a dedicated concept drift detection method (Page Hinkley) to adapt to change. As mentioned before, the Dynamic Model Tree neither requires a heuristic measure nor a separate concept drift detection model.

Other than FIMT-DD, the Dynamic Model Tree continues to update the simple models at the inner nodes even after splitting. This may increase the computation time, but allows us to compute the loss concerning the active concept on different hierarchies. In this way, the proposed framework can effectively identify and adjust to global and local concept drift.

E. Limitations

Typically, incremental decision trees like VFT or FIMT-DD [21] primarily occupy memory for saving statistics in the leaf nodes. Dynamic Model Trees also require memory to store statistics for every inner node. For example, while VFDT occupies $O(lmvc)$ memory, a Dynamic Model Tree requires $O((l+i)m^2vc)$, where $l$ and $i$ are the number of leaf and inner nodes, $m$ is the number of features, $v$ is the maximal number of unique values per feature and $c$ is the number of classes. However, Dynamic Model Trees usually remain shallow due to the model minimality property, which reduces the overall computational gap to other methods.

Likewise, Dynamic Model Trees can have a longer training time per node, depending on the selected simple model type. The choice of appropriate simple models also affects the general performance of the tree. With random initial weights, a simple model may take some time to achieve good predictive quality. However, this mainly affects the root node of the Dynamic Model Tree, since all other simple models are warmed-started with the optimized parameters of the parent node. In addition, if the simple models are non-robust or biased, the split and prune decisions of the proposed framework will suffer. In general, however, inadequate model types can be quickly identified by comparing the predictive error to benchmarks (e.g. the VFDT).

V. IMPLEMENTATION

The Dynamic Model Tree offers a large degree of flexibility. In particular, our framework may be implemented with different simple models and loss functions to account for different applications. For illustration, we propose an effective implementation of the Dynamic Model Tree for binary and multi-class classification.

A. Simple Models

We use logit and multinomial logit models (softmax) to represent binary and categorical target variables, respectively. Both models belong to the family of Generalized Linear Models (GLM) and are widely used in practice due to their efficiency and transparency. We train the simple models by
stochastic gradient descent with a constant learning rate. In the future, one might experiment with different base models, optimization strategies or online feature selection [35].

B. Loss Function

Owing to the proposed gain functions, changes in a Dynamic Model Tree are directly linked to changes in the empirical loss. Although purity-based splits usually also lead to a reduction in error, splits based on a change in loss can be very powerful in terms of interpretability.

To this end, we recall that concept drift between two time steps $t_1$ and $t_2$ corresponds to a change in the active concept, i.e., $P_{t_1}(X,Y) \neq P_{t_2}(X,Y)$. Online learning models need to adjust to concept drift in order to maintain high predictive performance. Accordingly, we are mainly interested in concept drift that shifts the optimal decision boundary. This form of concept drift is called real concept drift and is defined as $P_{t_1}(Y|X) \neq P_{t_2}(Y|X)$ [36]. Since the true distribution $P_t(Y|X)$ is generally unknown, our best approximation of the active data concept is the likelihood $P(Y_t|X_t, \theta_t)$ [34].

In this context, the negative log-likelihood loss changes as a consequence of the additional gain functions to avoid excessive updates. By using the negative log-likelihood loss, we enable a natural threshold in terms of the Akaike Information Criterion: $AIC = 2k - 2\ell(\Theta)$, where $\ell$ is the log-likelihood and $k$ is the number of free (estimated) parameters. The AIC is a popular test statistic for model selection problems. It estimates the relative amount of information lost among competing models. Given two models $i$ and $j$ where $AIC_i \leq AIC_j$, the quantity $\exp((AIC_i - AIC_j)/2)$ is proportional to the relative probability that model $j$ minimizes the estimated information loss. Therefore, if we set a threshold for this quantity, we can control the tolerated probability that model $j$ actually has the minimum AIC instead of model $i$.

We can apply this methodology to our split and prune strategy. For example, when attempting to split, we compare the simple models representing the current node ($S_t$) and the potential split ($C_t$, $\tilde{C}_t$). The corresponding AICs are

$$AIC_{S_t} = 2k_{S_t} - 2(\ell(\Theta_{S_t}, Y_{S_t}, X_{S_t})), \quad (9)$$

$$AIC_{C_t} = 2(k_{C_t} + k_{\tilde{C}_t}) - 2(\ell(\Theta_{C_t}, Y_{C_t}, X_{C_t}) + \ell(\Theta_{\tilde{C}_t}, Y_{\tilde{C}_t}, X_{\tilde{C}_t})), \quad (10)$$

where $k_{S_t}$, $k_{C_t}$, and $k_{\tilde{C}_t}$ denote the numbers of free parameters of the corresponding models. Let $\epsilon \in [0, 1]$ be a user-specified hyperparameter. We apply the following test:

$$\exp((AIC_{C_t} - AIC_{S_t})/2) \leq \epsilon \Leftrightarrow \exp(k_{C_t} + k_{\tilde{C}_t} - \ell(\Theta_{C_t}, Y_{C_t}, X_{C_t}) - \ell(\Theta_{\tilde{C}_t}, Y_{\tilde{C}_t}, X_{\tilde{C}_t})) - k_{S_t} + \ell(\Theta_{S_t}, Y_{S_t}, X_{S_t})) \leq \epsilon \Leftrightarrow \exp(-G_{S_t,c_t}) \leq \frac{\exp(k_{C_t} + k_{\tilde{C}_t} - k_{S_t})}{\exp(k_{C_t} + k_{\tilde{C}_t} - k_{S_t})} \Leftrightarrow -G_{S_t,c_t} \leq \log(\epsilon) - k_{C_t} - k_{\tilde{C}_t} + k_{S_t} \Leftrightarrow G_{S_t,c_t} \geq k_{C_t} + k_{\tilde{C}_t} - k_{S_t} - \log(\epsilon) \quad (11)$$

If we use the same simple model type at every node (e.g. logit models as proposed earlier), then [11] simplifies to $G_{S_t,c_t} \geq k - \log(\epsilon)$. Similarly, we can calculate thresholds for the remaining gain functions, which we omit for brevity. Note that the hyperparameter $\epsilon$ controls the trade-off between quick and robust updates. In this way, we can adjust the sensitivity of the Dynamic Model Tree.

D. Algorithmic Considerations

We implemented the Dynamic Model Tree in Python. Note that the Dynamic Model Tree is able to handle both batch-incremental and instance-incremental online learning. In the following, we discuss important algorithmic details and propose a sensible hyperparameter configuration.

In practice, the number of unique split candidates may grow quickly – in particular for continuous variables. This is a problem that most incremental decision trees have in common. To overcome potential memory overload, our framework may be extended with advanced strategies like Binary Search Trees (see their application in FIMT-DD [21], for example). For illustration, however, we have chosen a simpler technique.

https://github.com/haugjo/dynamic-model-tree
TABLE I: Data sets. We used state-of-the-art tabular streaming data sets with different types of concept drift. TuEyeQ [37], as well as Insects-Abrupt and Insects-Incremental [38] have been obtained from the sources referenced in the respective papers. The remaining real-world data sets have been obtained from https://www.openml.org. We included the original reference wherever available. The synthetic data sets have been generated with scikit-multiflow [30]. Here we also indicate the type of concept drift generated (abrupt or incremental).

| Name                  | #Samples | #Features | #Classes (##Majority) |
|-----------------------|----------|-----------|----------------------|
| Electricity           | 45,312   | 8         | 2 - (26,075)         |
| Airlines              | 539,383  | 7         | 2 - (299,119)        |
| Bank [39]             | 45,211   | 16        | 2 - (39,922)         |
| TuEyeQ [37]           | 15,762   | 76        | 2 - (12,975)         |
| Poker-Hand [40]       | 1,025,000| 10        | 9 - (513,701)        |
| KDDCup [40]           | 494,020  | 41        | 23 - (280,790)       |
| Covertype [40]        | 581,012  | 54        | 7 - (283,301)        |
| Gas [41]              | 13,910   | 128       | 6 - (3,009)          |
| Insects-Abrupt [38]   | 355,275  | 33        | 6 - (101,256)        |
| Insects-Incremental [38]| 452,044 | 33        | 6 - (134,717)        |
| SEA (synthetic, abrupt)| 1,000,000| 3         | 2                    |
| Agrawal (synthetic, incremental) | 1,000,000 | 9 | 2 |
| Hyperplane (synthetic, incremental) | 500,000 | 50 | 2 |

Specifically, we store only a fixed number of statistics corresponding to the candidates with largest estimated gain (we recommend a default value of three times the number of features). At every time step, we allow a fixed percentage of the saved candidate statistics to be replaced by newly observed candidates. This is similar to the VFDT algorithm [11], which drops split candidates that diverge too far from the current maximal gain. We recommend a default replacement rate of 50%, which provided good results throughout all our experiments.

Since we limit the number of split candidates in main memory, we need to approximate the gain of newly observed candidates from the current sample. Note that the initial approximation can be biased if the current batch is not representative of the active concept. Specifically, such initial bias might occur if the batch size is small or the data is very noisy. Once stored, however, the statistics are updated at each successive time step, mitigating any initial bias over time. In addition, a split candidate that was rejected or deleted in the past can be added again in the future, e.g. if its importance has changed after concept drift. In experiments, we obtained good results for this simple approximation scheme.

Additionally, we propose a learning rate of 0.05 to train the binary and multinomial logit models and a threshold of $\epsilon = 10e^{-8}$ for the AIC-based confidence test.

Finally, note that we might be able to improve the efficiency of the Dynamic Model Tree by using parallelization or distributed computation. We leave a detailed discussion of more advanced implementation techniques for future work.

VI. EXPERIMENTS

We evaluated the Dynamic Model Tree in multiple experiments on synthetic and real-world streaming classification data sets. Specifically, we compared the proposed framework to the related Model Tree architecture FIMT-DD [21] and different versions of the Hoeffding Tree. We begin with a description of the experimental setup, including the data sets, related methods and performance measures. Afterwards, we summarize our most important findings.

A. Environment and Evaluation Strategy

All models and experiments were implemented in Python (3.8.5) and run on an AMD Ryzen Threadripper 3960X (24x 3.8GHz) CPU with 128Gb RAM under Ubuntu 18.04. In addition, we used the following packages: numply (1.20.1), pandas (1.2.4), matplotlib (3.4.2), scikit-learn (0.24.2) and scikit-multiflow (0.5.3). We specified a random state to guarantee the reproducibility of all results.

We performed a prequential (test-then-train) evaluation [43], which is the most common evaluation strategy for data stream learning. A disadvantage of data stream evaluations compared to regular batch evaluations is the lack of statistical significance. To be precise, since we cannot alter the order of observations without introducing artificial concept drift, we cannot obtain results for different permutations or samples of the data set. There are approaches where multiple instances of a classifier are trained in parallel [44]. However, they are very computationally intensive. Accordingly, we ask readers to be aware that statistical significance, although being standard in other areas of machine learning, is uncommon in the data stream literature.

At each iteration of the prequential evaluation, we processed a batch of 0.1% of the data. We also examined other batch sizes to ensure that the reported results are representative.

B. Data Sets

Typically, online classifiers are evaluated on tabular data sets. Machine learning with heterogeneous and evolving tabular data is challenging and has recently attracted attention in other areas such as deep learning [45]. In our experiments, we used state-of-the-art tabular streaming data sets, which we briefly describe in the following. We obtained most real-world data sets from https://www.openml.org. A summary of the data sets and their properties can also be found in Table I.

The Electricity data set describes price changes in the Australian New South Wales Electricity Market. The prices are not fixed, but adjust over time to the varying supply and demand. In the Airlines data set, the goal is to predict whether a flight will be delayed, given information about its scheduled departure. The Bank Marketing data set incorporates information about a marketing campaign of a Portuguese bank institute [39]. Here, the goal is to predict whether a customer will subscribe a deposit. Poker-Hand is a popular multiclass classification data set that consists of variables describing different poker hands [40]. Covertype contains information about several forest cover types that need to be distinguished [41]. The KDD Cup 1999 data set was introduced as part of a data mining competition. The data set contains features about network connections that are used for intrusion detection. We
shuffled the KDD data set, because it was initially grouped by class labels. Since KDD does not involve known concept drift, shuffling the data is required to obtain an even distribution of classes over time and enable a fair evaluation.

It is usually difficult to determine the exact period of concept drift in a real-world streaming process. In fact, we cannot access such information for any of the above-mentioned data sets. Two recent exceptions are the TüEyeQ [37] and Insects [38] data collections. From TüEyeQ, we used the indicating Hyperplane Tree to different versions of the Hoeffding Tree. Specifically, we obtained results for the basic VFDT [11] and two of its (range 1). Otherwise, we did not pre-process the data sets.

C. Related Algorithms and Hyperparameters

As mentioned before, we compared the Dynamic Model Tree to different versions of the Hoefding Tree. Specifically, we obtained results for the basic VFDT [11] and two of its extensions, the adaptive Hoefding Tree (HT-Ada) [13] and the Extremely Fast Decision Tree (EFDT) [14]. Unlike VFDT, both extensions contain a mechanism to adapt to concept drift. Since it is generally not possible to optimize hyperparameters in a data stream, we applied the default configurations suggested.
ggested by the corresponding scikit-multiflow implementations. These implementations have been heavily optimized over the years. Since our goal was to compare the originally proposed models, and in order to allow a fairer comparison with our implementation, we disabled some of the optimizations of scikit-multiflow. In particular, we did not use bootstrap sampling in the leaves of the HT-Ada algorithm. Moreover, we used majority voting in the leaf nodes of the Hoeffding Trees. However, to give an indication of the possible improvement introduced by simple predictive models in the leaves of a Hoeffding Tree, we also report the results of a VFDT augmented with adaptive Naïve Bayes models [31]. Finally, to improve the efficiency of the EFDT algorithm, we set the minimum number of observations between re-evaluations to 1,000.

For the sake of completeness, we also looked at two state-of-the-art ensembles of the Hoeffding Tree, an Adaptive Random Forest [42] and a Leveraging Bagging Ensemble [27]. Both ensembles were trained with 3 basic Hoeffding Tree classifiers as weak learners. We configured the weak learners in the same way as the stand-alone VFDT model. Otherwise, we used the default parameters of the ensembles specified in scikit-multiflow.

In addition, we evaluated FIMT-DD [21]. To the best of our knowledge, there is no publicly available Python implementation of a FIMT-DD classification model. Therefore, we implemented the classifier based on the description in the paper. Our implementation uses the second drift adjustment strategy proposed by the authors, i.e., it deletes branches where the Page Hinkley test issues an alert. We used a default learning rate of 0.01 for the simple models and a threshold of 0.01 for the significance test based on Hoeffding’s inequality. Besides, we specified a threshold of 0.05 to break ties between split candidates with similar gain.

Finally, note that we only allowed binary splits in all incremental decision trees. The Dynamic Model Tree was configured in the way described in Section V.

D. Performance Measures

1) Predictive Performance: Classification error and accuracy are common measures for evaluating online classifiers. However, both measures might produce biased results for imbalanced data. As our evaluation incorporates many imbalanced data sets, we report the F1 measure instead. The F1 measure is the harmonic mean of precision and recall and provides reliable results even for strong imbalances.

2) Interpretability/Complexity: Since there is no common measure of interpretability, one usually resorts to heuristics. For example, one can compare the number of parameters in linear models or the number of nodes in decision trees. Unfortunately, in our case there is no clear separation between model families. In particular, a comparison between the complexities of Hoeffding and Model Trees is difficult, as their leaf nodes offer different degrees of expressiveness.

Hence, we consider the number of splits in our evaluation, which we calculated as follows: Each inner node counted as one split. Majority-weighted leaf nodes did not contribute to the total number of splits. Conversely, the leaf classifiers can be considered as another final split of the observations. Accordingly, we counted one more split for binary classifiers and c more splits for multiclass classifiers, where c is the number of classes. Compared to measuring the total number of nodes, the number of splits accounts for the different leaf types of Hoeffding Trees and Model Trees. For completeness, we also report the number of parameters. Specifically, we counted one parameter per inner node corresponding to the split value. We counted leaf nodes as either one (majority class) or m (linear model weights; Naïve Bayes conditional probabilities) additional parameters, where m is the number of features.

In practice, it depends on a given application whether the simple leaf models should be considered as limiting interpretability or not. That is, as mentioned earlier, simple models can offer significant advantages in terms of local feature-based explainability. Accordingly, we generally consider the number of splits to be a more reliable indication of the interpretability of incremental decision trees. Still, instead of giving too much importance to heuristics, one should aim for online learning models that have meaningful interpretability properties, such as those proposed in this paper.

3) Computational Efficiency: Computational efficiency depends on the respective implementation and hardware configuration. As we used both scikit-multiflow and custom implementations, we did not focus on computational efficiency in the experiments. However, for the sake of completeness, we provide the average computation time for one train/test-iteration of each model in Table V.

E. Results

In the following, we discuss our most important findings.

1) Predictive Performance: Table II shows the average F1 measure of all models and data sets. Using simple leaf models instead of majority voting has generally improved the obtained F1 score (see DMT, FIMT-DD and VFDT (NBA)). This advantage is most evident in the Hyperplane data set. The Hyperplane data was generated by rotating a decision hyperplane in multidimensional space. Thus, after a few splits, the observations can be linearly separated sufficiently well by the simple models. Although no model achieved good predictive quality on Poker, the VFDT with Naïve Bayes has a higher average score than the Model Trees, suggesting that a different simple model type may improve the results.

2The FIMT-DD implementation can also be accessed via Github at [https://github.com/haugjo/dynamic-model-tree](https://github.com/haugjo/dynamic-model-tree)
In general, FIMT-DD and the Dynamic Model Tree obtained similar F1 scores. However, our framework outperformed FIMT-DD for Airlines and the synthetic data sets. Looking at the behavior of FIMT-DD over time, its disadvantage can often be attributed to slow growth (e.g., Airlines, Agrawal, SEA) or aggressive pruning (e.g., Agrawal, Hyperplane). Accordingly, a less strict split threshold, different purity measures, and alternative pruning strategies could be explored in the future. Similarly, the Dynamic Model Tree may perform poorly in the first time steps if the random initial weights have not yet converged. This effect is noticeable in the averaged result of Electricity, since it is a relatively small data set. To speed up the initial training of the simple models, one may experiment with dynamic learning rates. As can be seen from the Gas data set, the VFDT and HT-Ada implementations may have difficulty finding optimal split candidates for high-dimensional and continuous feature sets. Both models remained extremely shallow and obtained poor predictive performance. In such cases, where it is difficult or infeasible to find a good split value among all possible candidates, the simple leaf models can provide an advantage. Besides, HT-Ada was not competitive for Covertype and Insects-Abrupt. Here, pruning near the root caused temporary declines of the F1 score. Such behaviour might be avoided by a less aggressive pruning strategy or a more robust drift detection scheme.

Our framework obtained either the best or second best average F1 score for all data sets with known concept drift (TuEyeQ, Insects-Abrupt, Insects-Incremental, SEA, Agrawal and Hyperplane). We depict the detailed results of four data sets in Figure 3. The Dynamic Model Tree often suffers
Fig. 4: Predictive Performance vs. Model Complexity. Above, we compare the F1 measure and the logarithm of the number of splits of each incremental decision tree. The number of splits is an indication of model complexity, which in turn is a common proxy for the interpretability. That is, fewer splits can usually be associated with higher interpretability. Each point corresponds to the average measure of one data set. Detailed results can be found in the Tables [III]and [III]. Ideally, we aim for a large F1 score and a small number of splits, corresponding to a value in the top left quadrant. While achieving competitive F1 scores, the Dynamic Model Tree generally manages to reduce the number of splits compared to the Hoefdding Trees.

only minor performance deterioration after a concept drift. Compared to the other models, our framework usually recovers faster from both abrupt and incremental concept drift. The effect is most notable in the SEA and Insects data set.

In summary, the proposed Dynamic Model Tree (DMT) is among the best performing models for most data sets. In fact, our framework ranks first place on average, even when the more powerful ensemble models are taken into account.

2) Complexity and Interpretability: As described above, we report the number of splits (Table [III]) and the number of parameters (Table [V]) as indicators for the interpretability of a model. Model Trees often maintain a shallower tree structure than Hoefdding Trees. This effect can be attributed to the additional flexibility provided by the simple models. For example, the synthetic Hyperplane and SEA data sets can both be separated by a hyperplane. The Dynamic Model Tree was able to represent these linear relationships with fewer splits than the Hoefdding Trees, while achieving similar or higher predictive quality. The complexities of the Dynamic Model Tree and FIMT-DD often varied. We attribute this effect to the loss-based gains that allow our framework to meet the consistency with parent splits and model minimality properties.

Specifically, while the Dynamic Model Tree will only retain a split, if it is beneficial in terms of the loss, FIMT-DD retains a split as long as the Page Hinkley test does not detect a concept drift. This may lead to overly complex trees that offer only slight or no improvements in terms of the F1 score (see Electricity and Bank). Likewise, if there is no significant difference according to the Hoefdding bound, FIMT-DD does not split a node, even though this might reduce the expected loss (see Airlines and SEA). In addition, FIMT-DD aims to reduce the standard deviation of the target and can therefore obtain leaf nodes that are extremely imbalanced towards one class. While this would be beneficial for majority weighting, it could make training simple (linear) models more difficult. Ultimately, this may reduce the predictive performance of FIMT-DD compared to a Dynamic Model Tree, even though both models have similar complexity (see Agrawal).

The Dynamic Model Tree ranks first for the average number of splits and third for the more conservative number of parameters. Indeed, Figure [V] shows that the complexity of the Dynamic Model Tree typically remains low over time, while other methods such as VFDT produce increasingly larger trees. Besides, the Dynamic Model Tree can adapt to different types of concept drift without drastically changing its complexity.

In general, our results demonstrate that high predictive performance and low complexity need not be mutually exclusive in an evolving data stream. The relationship of predictive performance and complexity is also shown in Figure [IV]. A summary of our experiments is depicted in Table [VI].

TABLE VI: Experiment Summary. We provide a concise summary of our experiments. For more detailed results, please see the remaining tables and plots. We ranked all methods according to four categories. Both predictive performance categories are based on the results in Table [III]. The second category reflects the average performance for the data sets with known concept drift. The complexity and efficiency scores are based on the average results in the Tables [III]and [V].

We used the following methodology: The best and worst models per category have received a score of ++ and – – respectively. The other methods have received a score of + or – depending on whether they were above or below the median.

| Model | Pred. Performance | Pred. For Known Drift | Complexity/Interpretability | Computational Efficiency |
|-------|------------------|-----------------------|-----------------------------|------------------------|
| DMT (ours) | ++ | ++ | ++ | – |
| FIMT-DD | – | – | – | ++ |
| VFDT (MC) | + | – | – | ++ |
| VFDT (NBA) | + | + | – | + |
| HT-Ada | – | – | – | + |
| EFDT | – | + | – | – |
| FIMT-DD | ++ | ++ | ++ | – |

VII. CONCLUSION

In this paper, we introduced the Dynamic Model Tree, a flexible and interpretable framework for machine learning on large-scale evolving data streams. A Dynamic Model Tree adheres to sensible properties that make it a reliable choice even in highly challenging streaming scenarios. Our experiments show that the proposed framework can achieve state-of-the-art performance with a fraction of the complexity of many previous methods. In particular, the Dynamic Model Tree automatically adapts to different types of concept drift, without the need for complex model extensions common in existing frameworks. Accordingly, we hope that our work will support the current trend towards more efficient and interpretable machine learning.
