Extremely Fast Decision Tree

Chaitanya Manapragada
Monash University
Victoria, Australia
chait.m@monash.edu

Geoffrey I. Webb
Monash University
Victoria, Australia
geoff.webb@monash.edu

Mahsa Salehi
Monash University
Victoria, Australia
mahsa.salehi@monash.edu

ABSTRACT
We introduce a novel incremental decision tree learning algorithm, Hoeffding Anytime Tree, that is statistically more efficient than the current state-of-the-art, Hoeffding Tree. We demonstrate that an implementation of Hoeffding Anytime Tree—"Extremely Fast Decision Tree", a minor modification to the MOA implementation of Hoeffding Tree—obtains significantly superior prequential accuracy on most of the largest classification datasets from the UCI repository. Hoeffding Anytime Tree produces the asymptotic batch tree in the limit, is naturally resilient to concept drift, and can be used as a higher accuracy replacement for Hoeffding Tree in most scenarios, at a small additional computational cost.

CCS CONCEPTS
• Computing methodologies → Online learning settings; Classification and regression trees; Machine learning algorithms;

KEYWORDS
Incremental Learning, Decision Trees, Classification

ACM Reference Format:
Chaitanya Manapragada, Geoffrey I. Webb, and Mahsa Salehi. 2018. Extremely Fast Decision Tree. In Proceedings of ACM conference (KDD'18). ACM, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/nnnnnnn.nnmm

1 INTRODUCTION
We present a novel stream learning algorithm, Hoeffding Anytime Tree (HATT). The de facto standard for learning decision trees from streaming data is Hoeffding Tree (HT) [11], which is used as a base for many state-of-the-art drift learners [3, 6, 8, 10, 16, 18, 24]. We improve upon HT by learning more rapidly and guaranteeing convergence to the asymptotic batch decision tree on a stationary distribution.

Our implementation of the Hoeffding Anytime Tree algorithm, the Extremely Fast Decision Tree (EFDT), achieves higher prequential accuracy than the Hoeffding Tree implementation Very Fast Decision Tree (VFDT) on many standard benchmark tasks. HT constructs a tree incrementally, delaying the selection of a split at a node until it is confident it has identified the best split, and never revisiting that decision. In contrast, HATT seeks to select and deploy a split as soon as it is confident the split is useful, and then revisits that decision, replacing the split if it subsequently becomes evident that a better split is available.

The HT strategy is more efficient computationally, but HATT is more efficient statistically, learning more rapidly from a stationary distribution and eventually learning the asymptotic batch tree if the distribution from which the data are drawn is stationary. Further, false acceptances are inevitable, and since HT never revisits decisions, increasingly greater divergence from the asymptotic batch learner results as the tree size increases (Sec. 4).

In Fig. 1.1, we observe VFDT taking longer and longer to learn progressively more difficult concepts obtained by increasing the number of classes. EFDT learns all of the concepts very quickly,
and keeps adjusting for potential overfitting as fresh examples are observed.

In Section 5, we will see that EFDT continues to retain its advantage even 100 million examples in, and that EFDT achieves significantly lower prequential error relative to VFDT on the majority of benchmark datasets we have tested. VFDT only slightly outperforms EFDT on three synthetic physics simulation datasets—Higgs, SUSY, and Hepmass.

### 2 BACKGROUND

Domingos and Hulten presented one of the first algorithms for incrementally constructing a decision tree in their widely acclaimed work, “Mining High-Speed Data Streams” [11]. Their algorithm is the Hoeffding Tree (Table 1), which uses the Hoeffding Bound. For any given potential split, Hoeffding Tree checks whether the difference of averaged information gains of the top two attributes is likely to have a positive mean—if so, the winning attribute may be picked with a degree of confidence, as is described below.

#### Table 1: Hoeffding Tree, Domingos & Hulten (2000)

| Inputs: | $S$ is a sequence of examples, $X$ is a set of discrete attributes, $G_i(.)$ is a split evaluation function, $\delta$ is one minus the desired probability of choosing the correct attribute at any given node. |
|---|---|
| Output: | $HT$ is a decision tree. |

#### Procedure HoeffdingTree $(S, X, G_i, \delta)$

Let $HT$ be a tree with a single leaf $l_1$ (the root).

Let $X_1 = X \cup \{X_g\}$.

Let $G_l(X_g)$ be the $G_i$ obtained by predicting the most frequent class in $S$.

For each class $y_k$:

For each value $x_{ij}$ of each attribute $X_i \in X$

Let $n_{ijk}(l_1) = 0$.

For each example $(x, y)$ in $S$

Sort $(x, y)$ into a leaf $l$ using $HT$.

For each $x_{ij}$ in $x$ such that $X_i \in X_l$

Increment $n_{ijk}(l)$.

Label $l$ with the majority class among the examples seen so far at $l$.

If the examples seen so far at $l$ are not all of the same class, then:

Compute $G_l(X_i)$ for each attribute $X_i \in X_l - \{X_g\}$

using the counts $n_{ijk}(l)$.

Let $X_a$ be the attribute with highest $G_l$.

Let $X_b$ be the attribute with second-highest $G_l$.

Compute $\epsilon$ using Equation 1.

If $G_l(X_a) - G_l(X_b) > \epsilon$ and $X_a \neq X_g$, then:

Replace $l$ by an internal node that splits on $X_a$.

For each branch of the split:

Add a new leaf $l_m$, and let $X_m = X - \{X_g\}$.

Let $G_m(X_g)$ be the $G_i$ obtained by predicting the most frequent class at $l_m$.

For each class $y_k$ and each value $x_{ij}$ of each attribute $X_i \in X_m - \{X_g\}$

Let $n_{ijk}(l_m) = 0$.

Return $HT$.

3 Hoeffding Anytime Tree

If the objective is to build an incremental learner with good predictive power at any given point in the instance stream, it may be desirable to exploit information as it becomes available, building structure that improves on the current state but making subsequent corrections when further alternatives are found to be even better. In scenarios where information distribution among attributes is skewed, with some attributes containing more information than others, such a policy can be highly effective because of the limited cost of rebuilding the tree when replacing a higher-level attribute with a highly informative one. However, where information is more uniformly distributed among attributes, Hoeffding Tree will struggle to split and might have to resort to using a tie-breaking threshold that depends on the number of random variables, while HATT will pick an attribute to begin with and switch when necessary, leading to faster learning.

In this paper, we describe HATT, and provide an instantiation that we denote Extremely Fast Decision Tree (EFDT).

Hoeffding Anytime Tree is equivalent to Hoeffding tree except that it uses the Hoeffding bound to determine whether the merit of splitting on the best attribute exceeds the merit of not having a split, or the merit of the current split attribute. In practice, if no split attribute exists at a node, rather than splitting only when the top candidate split attribute outperforms the second-best candidate, HATT will split when the information gain due to the top candidate split is non-zero with the required level of confidence. At later stages, HATT will split when the difference in information gain between the current top attribute and the current split attribute is non-zero, assuming this is better than having no split. HATT is presented in Algorithm 3.1, Function 3.2, and Function 3.3.

3.1 Convergence

Hoeffding Tree offers guarantees on the expected disagreement from a batch tree trained on an infinite dataset (which is denoted $D^\infty$ in [11], a convention we will follow). “Extensional disagreement” is defined as the probability that a pair of decision trees will
HT at the time HT splits the root node.

Algorithm 3.1: Hoeffding Anytime Tree

Input: S, a sequence of examples. At time t, the observed sequence is \( S^t = ((\hat{x}_1, y_1), (\hat{x}_2, y_2), \ldots, (\hat{x}_t, y_t)) \)

\( X = \{X_1, X_2, \ldots, X_m\} \), a set of m attributes

\( \delta \), the acceptable probability of choosing the wrong split attribute at a given node

\( G() \), a split evaluation function

Result: HATT, the model at time t constructed from having observed sequence \( S^t \).

begin
Let HATT be a tree with a single leaf, the root
Let \( X_1 = X \cup X_0 \)
Let \( G_1(X_0) \) be the G obtained by predicting the most frequent class in S

foreach class \( y_k \) do
foreach value \( x_{ij} \) of each attribute \( X_i \in X \) do
Set counter \( n_{ijk}(\text{root}) = 0 \)
end
end

foreach example \((\hat{x}, y)\) in \( S^t \) do
Sort \((\hat{x}, y)\) into a leaf \( l \) using HATT

foreach node in path (root...l) do
foreach \( x_{ij} \) in \( \hat{x} \) such that \( X_i \in X_{\text{node}} \) do
Increment \( n_{ijk}(\text{node}) \)
if node = l then
| AttemptToSplit(l)
else
| ReEvaluateBestSplit(node)
end
end
end

end

end

Function 3.2: AttemptToSplit(leafNode \( l \))

begin
Label \( l \) with the majority class at \( l \)
if all examples at \( l \) are not of the same class then
Compute \( \bar{G}_l(X_i) \) for each attribute \( X_i \in \{X_0\} \) using the counts \( n_{ijk}(l) \)
Let \( X_a \) be the attribute with the highest \( \bar{G}_l \)
Let \( X_b = X_0 \)
Compute \( \epsilon \) using equation 1
if \( \bar{G}_l(X_a) - \bar{G}_l(X_b) > \epsilon \) and \( X_a \neq X_0 \) then
Replace \( l \) by an internal node that splits on \( X_a \)
for each branch of the split do
Add a new leaf \( l_m \) and let \( X_m = X - X_a \)
Let \( G_m(X_0) \) be the G obtained by predicting the most frequent class at \( l_m \)
for each class \( y_k \) and each value \( x_{ij} \) of each attribute \( X_i \in X_{m} - \{X_0\} \) do
| Let \( n_{ijk}(l_m) = 0 \).
end
end
end

end

Function 3.3: ReEvaluateBestSplit(internalNode \( \text{int} \))

begin
Compute \( \bar{G}_\text{int}(X_i) \) for each attribute \( X_{\text{int}} = \{X_0\} \) using the counts \( n_{ijk}(\text{int}) \)
Let \( X_a \) be the attribute with the highest \( \bar{G}_\text{int} \)
Let \( X_{\text{current}} \) be the current split attribute
Compute \( \epsilon \) using equation 1
if \( \bar{G}_l(X_a) - \bar{G}_l(X_{\text{current}}) > \epsilon \) then
if \( X_a = X_0 \) then
| Replace internal node \( \text{int} \) with a leaf (kills subtree)
else if \( X_a \neq X_{\text{current}} \) then
| Replace \( \text{int} \) with an internal node that splits on \( X_a \)
for each branch of the split do
Add a new leaf \( l_m \) and let \( X_m = X - X_a \)
Let \( G_m(X_0) \) be the G obtained by predicting the most frequent class at \( l_m \)
for each class \( y_k \) and each value \( x_{ij} \) of each attribute \( X_i \in X_{m} - \{X_0\} \) do
| Let \( n_{ijk}(l_m) = 0 \).
end
end
end

end

end

end

produce different predictions for an example, and intensional disagreement that probability that the path of an example will differ on the two trees.

The guarantees state that either form of disagreement is bound by \( \frac{\delta}{P} \), where \( \delta \) is a tolerance level and \( P \) is the leaf probability— the probability that an example will fall into a leaf at a given level. \( P \) is assumed to be constant across all levels for simplicity.

Note that the guarantees will weaken significantly as the depth of the tree increases. While the built trees may have good prequential accuracy in practice on many test data streams, increasing the complexity and size of data streams such that a larger tree is required increases the chance that a wrong split is picked.

On the other hand, HATT converges in probability to the batch decision tree; we prove this below.

For our proofs, we will make the following assumption:

- No two attributes will have identical information gain. This is a simplifying assumption to ensure that we can always split given enough examples, because \( \epsilon \) is monotonically decreasing.

Lemma 3.1. HATT will have the same split attribute at the root as HT at the time HT splits the root node.
We will see that under all these scenarios, HATT will select (or wouldn’t necessarily be equal. The components of Tree will also find may happen to be the current split. That is, it is possible that for HT and HATT. That is, it is possible that for \( t \), both attributes have identical, is, the ranking of the information gains of the potential split at-tion that is unlike \( \Delta G \). We are interested in attaining confidence \( \Delta G \) diverges from zero, because that would imply both attributes do not have equal information gain, and that one of the attributes is the clear winner. Setting \( \mu_\Delta G \) to 0, we want to be confident that \( \Delta G \) differs from zero by at least \( \epsilon \). In other words, we are using a corollary of Hoeffding’s Inequality to state with confidence that our random variable \( \Delta G \) diverges from 0.

In order for this to happen, we need \( \Delta G \) to be greater than \( \epsilon \). \( \Delta G \) is monotonically decreasing, as we can see in equation 1.

Given the same infinite sequence of examples \( S \), both HT and HATT will be presented with the same evidence \( S_i(N_0) \) at the root level node \( N_0 \) for all \( t \) (that is, indefinitely). They will always have an identical value of \( \epsilon \).

If at a specific time \( t \) Hoeffding Tree compares attributes \( X_a \) and \( X_b \), which correspond to the attributes with the highest and second highest information gains \( X^{1,T} \) and \( X^{2,T} \) at time \( T \) respectively, it follows that since \( S_T(N_0)(HT) = S_T(N_0)(HATT) \), that is, since both trees have the same evidence at time \( T \), Hoeffding AnyTime Tree will also find \( X^{1,T} = X_a \). However, HATT will compare \( X_a \) with \( X^T \), the current split attribute. There are four possibilities: \( X^T = X^{1,T}, X^T = X^{2,T}, X^T = X^{i,T}, i > 2 \) or \( X^T \) is the null split. We will see that under all these scenarios, HATT will select (or retain) \( X^{1,T} \).

We need to consider the history of \( \Delta G \), which can be different for HT and HATT. That is, it is possible that for \( t \leq T \), \( \Delta G(HT) \neq \Delta G(HATT) \). This is because while HT always compares \( X^{1,T} \) and \( X^{2,T} \), HATT may compare \( X^{1,T} \) with, say, \( X^{3,T}, X^{4,T} \) or \( X_b \), which may happen to be the current split.

Clearly, at any timestep, \( X^{1,T}(N_0)(HT) = X^{1,T}(N_0)(HATT) \). That is, the ranking of the information gains of the potential split attributes is always the same at the root node for both HT and HATT. It should also be obvious that since the observed sequences are identical, \( G(X^{1,T}(N_0)(HT)) = G(X^{1,T}(N_0)(HATT)) \) the information gains of all of the corresponding attributes at each timestep are equal. So the top split attribute at the root \( X^{1,T}(N_0) \) is always the same for both trees. If we decompose \( \Delta G \) as \( \Delta G_{top} = \Delta G_{bot} \), we will have \( \Delta G_{top}(HT) = \Delta G_{top}(HATT) \), but \( \Delta G_{bot}(HT) \) and \( \Delta G_{bot}(HATT) \) wouldn’t necessarily be equal.

Since at any timestep \( t \) HT will always choose to compare \( G(X^{1,T}) \) and \( G(X^{2,T}) \) while HATT will always compare \( G(X^{1,T}) \) with \( G_{currentSplit} \) where \( G_{currentSplit} \leq G(X^{2,T}) \), we have \( \Delta G_{bot}(HATT) \leq \Delta G_{bot}(HT) \) for all \( t \).

Because we have \( \Delta G_{bot}(HATT) \leq \Delta G_{bot}(HT) \), we will have \( \Delta G^T(HT) \geq \Delta G^T(HATT) \), and \( \Delta G^T(HATT) > \epsilon \) implies \( \Delta G^T(HT) > \epsilon \), which would cause HATT to split on \( X^{1,T} \) if it already does not happen to be the current split attribute simultaneously with HT at time \( T \).

**Lemma 3.2.** The split attribute \( X^{1,T}_{HATT} \) at the root node of HATT converges in probability to the split attribute \( X^{1,T}_{DT^*} \) used at the root node of \( DT^* \). That is, as the number of examples grows large, the probability that HATT will have at the root a split \( X^{1,T}_{HATT} \) that matches the split \( X^{1,T}_{DT^*} \) at the root node of \( DT^* \) goes to 1.

**Proof.** Let \( S \) represent an infinite sequence drawn from a probability space \( (\Omega, \mathcal{F}, P) \), where \((\vec{x}, y) \in \Omega \) constitute our data points. The components of \( x \) take values corresponding to attributes \( X_1, X_2, \ldots X_m \), if we have \( m \) attributes.

In order for this to happen, we need \( \Delta G \) to be greater than \( \epsilon \). \( \Delta G \) is monotonically decreasing, as we can see in equation 1.

Given the same infinite sequence of examples \( S \), both HT and HATT will be presented with the same evidence \( S_T(N_0) \) at the root level node \( N_0 \) for all \( t \) (that is, indefinitely). They will always have an identical value of \( \epsilon \).

If at a specific time \( t \) Hoeffding Tree compares attributes \( X_a \) and \( X_b \), which correspond to the attributes with the highest and second highest information gains \( X^{1,T} \) and \( X^{2,T} \) at time \( T \) respectively, it follows that since \( S_T(N_0)(HT) = S_T(N_0)(HATT) \), that is, since both trees have the same evidence at time \( T \), Hoeffding AnyTime Tree will also find \( X^{1,T} = X_a \). However, HATT will compare \( X_a \) with \( X^T \), the current split attribute. There are four possibilities: \( X^T = X^{1,T}, X^T = X^{2,T}, X^T = X^{i,T}, i > 2 \) or \( X^T \) is the null split. We will see that under all these scenarios, HATT will select (or retain) \( X^{1,T} \).

We need to consider the history of \( \Delta G \), which can be different for HT and HATT. That is, it is possible that for \( t \leq T \), \( \Delta G(HT) \neq \Delta G(HATT) \). This is because while HT always compares \( X^{1,T} \) and \( X^{2,T} \), HATT may compare \( X^{1,T} \) with, say, \( X^{3,T}, X^{4,T} \) or \( X_b \), which may happen to be the current split.

Clearly, at any timestep, \( X^{1,T}(N_0)(HT) = X^{1,T}(N_0)(HATT) \). That is, the ranking of the information gains of the potential split attributes is always the same at the root node for both HT and HATT. It should also be obvious that since the observed sequences are identical, \( G(X^{1,T}(N_0)(HT)) = G(X^{1,T}(N_0)(HATT)) \) the information gains of all of the corresponding attributes at each timestep are equal. So the top split attribute at the root \( X^{1,T}(N_0) \) is always the same for both trees. If we decompose \( \Delta G \) as \( \Delta G_{top} = \Delta G_{bot} \), we will have \( \Delta G_{top}(HT) = \Delta G_{top}(HATT) \), but \( \Delta G_{bot}(HT) \) and \( \Delta G_{bot}(HATT) \) wouldn’t necessarily be equal.

Since at any timestep \( t \) HT will always choose to compare \( G(X^{1,T}) \) and \( G(X^{2,T}) \) while HATT will always compare \( G(X^{1,T}) \) with \( G_{currentSplit} \) where \( G_{currentSplit} \leq G(X^{2,T}) \), we have \( \Delta G_{bot}(HATT) \leq \Delta G_{bot}(HT) \) for all \( t \).

Because we have \( \Delta G_{bot}(HATT) \leq \Delta G_{bot}(HT) \), we will have \( \Delta G^T(HT) \geq \Delta G^T(HATT) \), and \( \Delta G^T(HATT) > \epsilon \) implies \( \Delta G^T(HT) > \epsilon \), which would cause HATT to split on \( X^{1,T} \) if it already does not happen to be the current split attribute simultaneously with HT at time \( T \).

**Lemma 3.3.** Hoeffding AnyTime Tree converges to the asymptotic batch tree in probability.
Proof. From Lemma 3.2, we have that as \( t \to \infty \), \( X_{i \mid L_1}^{HATT} \xrightarrow{P} X_{i \mid L_1}^{DT} \), meaning that though it is possible to see at any individual timestep \( X_{i \mid L_1}^{HATT} \neq X_{i \mid L_1}^{DT} \), we have made convergence in probability in the limit.

Consider immediate subtrees of the root node \( \text{HATT}_1 \) (denoting they are rooted at level 1). In all cases where the root split matches \( X_{i \mid L_1}^{DT} \), the instances observed at the roots of \( \text{HATT}_1 \) will be drawn from the same data distribution that the respective \( \text{DT}_1 \) draws their instances from. Do level 1 split attributes for \( \text{HATT} \), \( X_{i \mid L_1}^{HATT} \) converge to \( X_{i \mid L_1}^{DT} \)?

We can answer this by using the Law of Total Probability. Let us denote the event that for first level split \( i, X_{i \mid L_1}^{HATT} = X_{i \mid L_1}^{DT} \) by \( \text{match}_{i \mid L_1} \). Then we have as \( t \to \infty \):

\[
P(X_{i \mid L_1}^{HATT} = X_{i \mid L_1}^{DT}) = P(\text{match}_{i \mid L_1} \mid \text{match}_{i \mid L_0}) P(\text{match}_{i \mid L_0})
\]

\[
+ P(\text{match}_{i \mid L_1} \mid \neg \text{match}_{i \mid L_0}) P(\neg \text{match}_{i \mid L_0})
\]

We know that \( P(\text{match}_{i \mid L_0}) \to 1 \) and \( P(\neg \text{match}_{i \mid L_0}) \to 0 \) as \( t \to \infty \) from Lemma 3.1. So we obtain \( P(X_{i \mid L_1}^{HATT} = X_{i \mid L_1}^{DT}) = P(\text{match}_{i \mid L_1} \mid \text{match}_{i \mid L_0})^{\infty} \).

Effectively, we end up only having to condition on the event \( \text{match}_{i \mid L_0} \). In other words, we may safely use a subset of the stream where only \( \text{match}_{i \mid L_0} \) has occurred to reason about whether \( X_{i \mid L_1}^{HATT} = X_{i \mid L_1}^{DT} \) as \( t \to \infty \).

Now, we need to show that \( P(\text{match}_{i \mid L_1} \mid \text{match}_{i \mid L_0}) \to 1 \) as \( t \to \infty \) to prove convergence at level 1. This is straightforward. Since we are only considering instances that result in the event \( \text{match}_{i \mid L_0} \) occurring, the conditional distributions at level 1 of \( \text{HATT} \) match the ones at level 1 of \( \text{DT} \). We may extend this argument to any number of levels; thus \( \text{HATT} \) converges in probability to \( \text{DT} \). \(\square\)

3.2 Time and Space Complexity

Space Complexity: On nominal with data with \( d \) attributes, \( v \) values per attribute, and \( c \) classes, \( \text{HATT} \) requires \( O(dvc) \) memory to store node statistics at each node, as does \( \text{HT} \) [11]. Because the number of nodes increases geometrically, there may be a maximum of \( (1-t^d)/(1-v) \) nodes, and so the worst case space complexity is \( O(vd^{-1}dvc) \). Since the worst case space complexity for \( \text{HT} \) is given in terms of the current number of leaves \( l \) as \( O(ldvc) \) [11], we may write the space complexity for \( \text{HATT} \) as \( O(ndvc) \), where \( n \) is the total number of nodes. Note that \( l \) is \( O(n) \), so space complexity is equivalent for \( \text{HATT} \) and \( \text{HT} \).

Time Complexity: There are two primary operations associated with learning for \( \text{HT} \): (i) incorporating a training example by incrementing leaf statistics and (ii) evaluating potential splits at the leaf reached by an example. The same operations are associated with \( \text{HATT} \), but we also increment internal node statistics and evaluate potential splits at internal nodes on the path to the relevant leaf.

At any leaf for \( \text{HT} \) and at any node for \( \text{HATT} \), no more than \( d \) attribute evaluations will have to be considered. Each attribute evaluation at a node requires the computation of \( u \) information gains. Each information gain computation requires \( O(c) \) arithmetic operations, so each split re-evaluation will require \( O(dvc) \) arithmetic operations at each node. As for incorporating an example, each node the example passes through will require \( dvc \) counts updated and thus \( O(dvc) \) associated arithmetic operations. The cost for updating the node statistics for \( \text{HATT} \) is \( O(hdvc) \), where \( h \) is the maximum height of the tree, because up to \( h \) nodes may be traversed by the example, while it is \( O(dvc) \) for \( \text{HT} \), because only one set of statistics needs to be updated. Similarly, the worst-case cost of split evaluation at each timestep is \( O(dvc) \) for \( \text{HT} \) and \( O(hdvc) \) for \( \text{HATT} \), as one leaf and one path respectively have to be evaluated.

4 RELATED WORK

There is a sizable literature that adapts \( \text{HT} \) in sometimes substantial ways [12, 19, 23] that do not, to the best of our knowledge, lead to the same fundamental change in learning premise as does \( \text{HATT} \). [23] and [19] substitute the Hoeffding Test with McDiarmid’s and the “Normal” test respectively; [12] adds support for Naive Bayes at leaves. Methods proposed prior to \( \text{HT} \) are either significantly less tight compared to \( \text{HT} \) in their approximation of a batch tree [14] or unsuitable for noisy streams and prohibitively computationally expensive [26].

The most related other works are techniques that seek to modify a tree through split replacement, usually for concept drift adaptation.

Drift adaptation generally requires explicit forgetting mechanisms in order to update the model so that it is relevant to the most recent data; this usually takes the form of a moving window that forgets older examples or a fading factor that decays the weight of older examples. In addition, when the underlying model is a tree, drift adaptation can involve subtree or split replacement.

Hulten et al [18] follow up on the Hoeffding Tree work with a procedure for drift adaptation (Concept-adapting Very Fast Decision Tree, CVFDT). CVFDT has a moving window that diminishes statistics recorded at a node due to an example that has fallen out of a window at a given time step. The example statistics at each internal node change as the window moves, and existing splits are replaced if the split attribute is no longer the winning attribute and one of a set of alternate subtrees grown by splitting on winning attributes registers greater accuracy.

The idea common to both CVFDT and \( \text{HATT} \) is that of split re-evaluation. However, the circumstances, objectives, and methods are entirely different. CVFDT is explicitly designed for a drifting scenario; \( \text{HATT} \) for a stationary one. CVFDT’s goal is to reduce prequential error for the current window in the expectation that this is the best way to respond to drift; \( \text{HATT} \)’s goal is to reduce prequential error overall for a stationary stream so that it asymptotically approaches that of a batch learner. CVFDT builds and substitutes alternate subtrees; \( \text{HATT} \) does not. CVFDT deliberately employs a range of forgetting mechanisms; \( \text{HATT} \) only forgets as a side effect of replacing splits—when a subtree is discarded, so too are all the historical distributions recorded therein. CVFDT always compares the top attributes, while \( \text{HATT} \) compares with either the current split attribute or the null split.
However, CVFDT is not incompatible with the core idea of Hoeffding Anytime Tree; it would be interesting to examine whether the idea of comparing with the null split or the current split attribute when applied to CVFDT will boost its performance on concept drifting streams. However, that is beyond the scope of this paper.

In order to avoid confusion, we will also mention the Hoeffding Adaptive Tree (HAT) [6]. This method builds a tree that grows alternate subtrees if a subtree is observed to have poorer prequential accuracy on more recent examples, and substitutes an alternate when it has better accuracy than the original subtree. HAT uses an error estimator, such as ADWIN [5] at each node to determine whether the prediction error due to a recent sequence of examples is significantly greater than the prediction error from a longer historical sequence so it can respond to drift. HATT, on the other hand, does not rely on prediction results or error, and does not aim to deliberately replace splits in response to drift.

5 PERFORMANCE

Our EFDT implementation was built by changing the split evaluations of the MOA implementation of VFDT [7]. We compared VFDT and EFDT on all UCI [21] classification data sets with over 200,000 instances that had an obvious classification target variable, did not require text mining, and did not contain missing values (MOA has limited support for handling missing values). To augment this limited collection of large datasets, we also studied performance on the WISDM dataset [20]. In all, we have 12 benchmark datasets with a mixture of numeric and nominal attributes ranging from a few dimensions to hundreds of dimensions.

Many UCI datasets are ordered. VFDT and EFDT are both designed to converge towards the tree that would be learned by a batch learner if the examples in a stream are drawn i.i.d. from a stationary distribution. The ordered UCI datasets do not conform to this scenario, so we also study performance when they are shuffled in order to simulate it. To this end, we shuffled the data 10 times with the Unix shuf utility seeded by a reproducible stream of random bytes [13] to create 10 different streams, averaged our prequential accuracy results over the streams, as well as comparing with performance on the corresponding unshuffled stream.

Our experiments are easily reproducible. Instructions for processing datasets, source code for VFDT and EFDT to be used with MOA, and Python scripts to run the experiments are all available at [https://github.com/chaitanya-m/kdd2018.git].

EFDT attains substantially higher prequential accuracy on most streams (Figs. 5.1 to 5.9) whether shuffled or unshuffled. Where VFDT wins (5.10, 5.11, 5.12) the margin is far smaller than most of the EFDT wins. While EFDT runtime generally exceeds that of VFDT; we find it rarely requires more than double the time and in some cases, when it learns smaller trees, requires less time. We evaluate leaves every 200 timesteps and internal nodes every 2000 timesteps.

Differences in shuffled and unshuffled performance highlight the amount of order that is present in the unshuffled data. The unshuffled Skin dataset contains B,G,R values and a target variable that indicates whether the input corresponds to skin or not. All positive examples are at the start followed by all negative examples; the net effect is that a learner will replace one extremely simple concept with another (Fig. 5.5). When shuffled, it is necessary to learn a more complex decision boundary, affecting performance for both learners.

A different effect is observed with the higher dimensional Fonts dataset (Fig. 5.3). The goal is to predict which of 153 fonts corresponds to a 19x19 greyscale image, with each pixel able to take 255 intensity values. When instances are sorted, by font name alphabetically, each time a new font is encountered VFDT needs to learn the new concept at every leaf of an increasingly complex tree. In contrast, EFDT is able to readjust the model, efficiently discarding outdated splits to achieve an accuracy of around 99.8%, making it a potentially powerful base learner for methods designed for concept drifting scenarios.

The results on the Poker and Forest-Covertype datasets (Figs. 5.2, 5.4) reflect both effects: EFDT performs significantly better on
ordered data, and performance for both learners deteriorates with shuffled data in comparison with unshuffled data.

Every additional level of a decision tree fragments the input space, slowing down tree growth exponentially. A delay in splitting at one level delays the start of collecting information with respect to the splits for the next level. These delays cascade, greatly delaying splitting at deeper levels of the tree.

Thus, we expect HATT to have an advantage over HT in situations where HT considerably delays splits at each level—such as when the difference in information gain between the top attributes at a node is low enough to require a large number of examples in order to overcome the Hoeffding bound, though the information gains themselves happen to be significant. This would lead to a potentially useful split in HT being delayed, and poor performance in the interim.

Conversely, when the differences in information gain between top attributes as well as the information gains themselves are low, it is possible that HATT chooses a split that would require a large number of examples to readjust. However, since we expect this to keep up with VFDT on the whole, the main source of underperformance for EFDT is likely to be an overfitted model making low-level adjustments. Synthetic data from physics simulations available in the UCI repository (Higgs, Hepmass, SUSY) led to such a scenario. Fig. 5.13 shows us that with the MOA tree generator used in Fig. 1.1, even on a 100 million length stream, EFDT’s prequential error is still an order of magnitude lower than that of VFDT.
6 CONCLUSIONS

Hoeffding AnyTime Tree makes a simple change to the current de facto standard for incremental tree learning. The current state-of-the-art Hoeffding Tree aims to only split at a node when it has identified the best possible split and then to never revisit that decision. In contrast HATT aims to split as soon as a useful split is identified, and then to replace that split as soon as a better alternative is identified. Our results demonstrate that this strategy is highly effectively on benchmark datasets.

Our experiments find that HATT has some inbuilt tolerance to concept drift, though it is not specifically designed as a learner for drift. It is easy to conceive of ensemble, forgetting, decay, or subtree replacement approaches built upon HATT to deal with concept drift, along the lines of approaches that have been proposed for HT.

REFERENCES

[1] Pierre Baldi et al. 2014. Searching for exotic particles in high-energy physics with deep learning. Nature communications 5 (2014), 4308.
[2] Pierre Baldi et al. 2016. Parameterized neural networks for high-energy physics. The European Physical Journal C 76, 5 (2016), 235.
[3] R.S.M. de Barros, S.G.T. de Carvalho Santos, and P.M.G. Júnior. 2016. A Boosting-like Online Learning Ensemble. In 2016 International Joint Conference on Neural Networks (IJCNN) 1871–1878.
Dariusz Brzezinski and Jerzy Stefanowski. 2014. Reacting to different types of concept drift: The accuracy updated ensemble algorithm. IEEE Transactions on Neural Networks and Learning Systems 25, 1 (2014), 81–94.

Albert Bifet and Ricard Gavaldà. 2009. Adaptive learning from evolving data streams. In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 71–80.

João Gama, Ricardo Rocha, and Pedro Medas. 2003. Accurate Decision Trees for Mining High-speed Data Streams. In Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD ’03). ACM, New York, NY, USA, 523–528. https://doi.org/10.1145/956750.956813

GNU Coreutils: Random sources. 2018. 2.7–Sources of random data. (2018). Retrieved Jan 05, 2018 from https://www.gnu.org/software/coreutils/manual/html_node/Random-sources.html

Jonathan Gratch. 1996. Sequential inductive learning. In Proceedings of the thirteenth national conference on Artificial intelligence-Volume 1. AAAI Press, 779–786.

Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. Journal of the American statistical association 58, 301 (1963), 13–30.

Stefan Horelganger and Russel Pears. 2007. Use of hoeffding trees in concept based data stream mining. In Information and Automation for Sustainability. 2007. ICAFS 2007. Third International Conference on. IEEE, 57–62.

Ramón Huerta, Mosquero, et al. 2016. Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. Chemometrics and Intelligent Laboratory Systems 157 (2016), 169–176.

Geoff Hulten, Laurie Spencer, and Pedro Domingos. 2001. Mining time-changing data with adaptive windowing. In Proceedings of the 2001 SIAM International Conference on Data Mining. SIAM, 443–448.

Leszek Rutkowski, Lena Pietruczuk, Piotr Duda, and Maciej Jaworski. 2013. Decision trees for mining data streams based on the McDiarmid’s bound. IEEE Transactions on Knowledge and Data Engineering 25, 6 (2013), 108–119.

Silas Santos et al. 2014. Speeding Up Recovery from Concept Drifts. Springer Berlin Heidelberg, Berlin, Heidelberg, 179–194.

Allan Stisen et al. 2015. Smart Devices Are Different: Assessing and Mitigating Mobile Sensing Heterogeneities for Activity Recognition. In Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems (SenSys ’15). ACM, New York, NY, USA, 127–140.

Paul E Utgoff. 1989. Incremental induction of decision trees. Machine learning 4, 2 (1989), 161–186.

Figure 5.11: Hepmass dataset [2, 21]

Figure 5.12: SUSY dataset [1, 21]

Figure 5.13: A longer term view of the experiments from Fig. 1.1 shows us that even 100 million examples in, EFDT maintains a commanding lead on prequential accuracy.