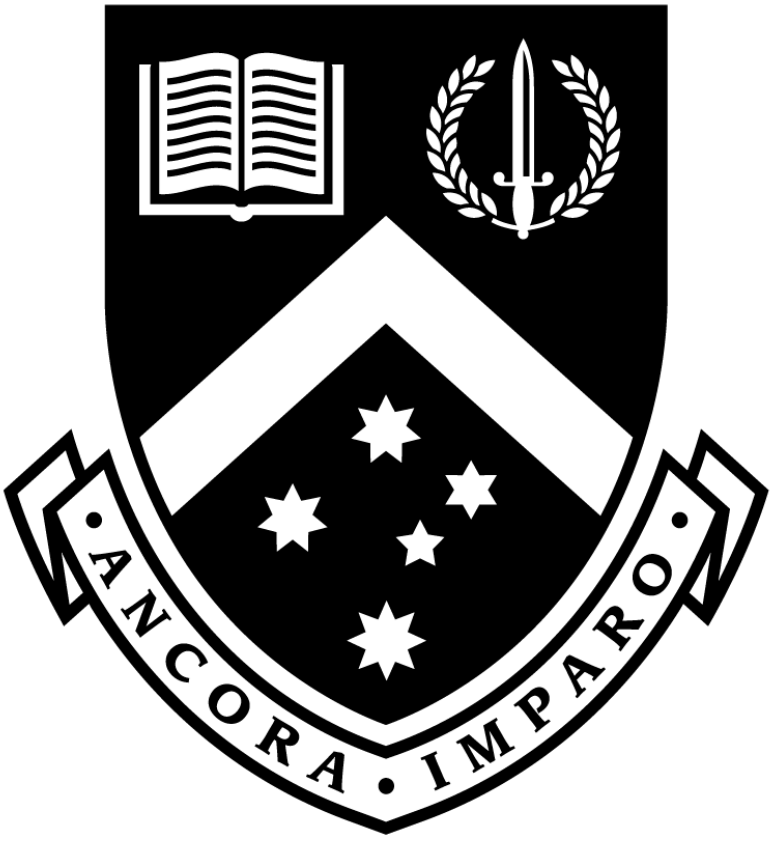
Learning Under Concept Drift

Chaitanya Manapragada

A thesis submitted for the degree of Doctor of
Philosophy at Monash University in 2020

Machine Learning Group
Department of Data Science and Artificial Intelligence
Faculty of Information Technology
Melbourne, Australia

Supervisors
Geoff Webb and Mahsa Salehi



MONASH University

Even the physical “constants” of the universe may be drifting.

Abstract

Change is inexorable, and yet we continue to build learning systems that assume an unchanging world. Machine learning today frequently operates under $i.i.d$ assumptions, more often than not because the assumption of identically and independently distributed data allows us to offer guarantees of learning and other nice properties. Discarding such assumptions places us in a world without learning guarantees where only the empirical success of learning strategies is of utility—the real world.

This is a work that studies how systems for learning may be designed, developed and evaluated in the real, changing world. Experimentation and empiricism are central to this study, but given that scope of experimentation is unbounded, observations from settings that can be theoretically reasoned about are used to aid the development of learning heuristics and hypotheses about learning.

Through thorough experimental analysis, this work demonstrates that complexities in the learning systems we build imply that our understanding and justifications for how those systems work may not match the real reasons for their behavior; that a measured dose of plasticity can improve the error performance of the state-of-the-art online decision tree learner in learning settings with and without changing concepts (concept drift); and that that measured dose of plasticity may also be advantageous in online ensemble learning. These results open a large number of avenues for the future exploration of how best to learn in a changing world and where adaptive strategies fit in.

Concretely, we extricate unspecified features in the state-of-the-art decision tree learners Very Fast Decision Tree (VFDT) and Hoeffding Adaptive Tree; we contribute a novel online decision tree learner Extremely Fast Decision Tree (EFDT) that outperforms VFDT on streams with and without concept drift individually and as part of commonly used ensembling strategies, without significantly increasing the order of time complexity. Further, we indicate that in settings with concept drift, a strategy of deliberate node evisceration would benefit VFDT-based trees; that the unspecified strategy of using rough approximations of information gain is particularly effective in the VFDT-based adaptive tree Hoeffding Adaptive Tree (HAT); and that combining the eager-splitting strategy from EFDT and the subtree substitution mechanism from HAT is highly profitable.

Copyright notice

Declaration

I hereby declare that this thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

This thesis includes (2) original papers published in peer reviewed journals and (2) submitted publications. The core theme of the thesis is "Learning under concept drift". The ideas, development and writing up of all the papers in the thesis were the principal responsibility of myself, the student, working within the Faculty of Information Technology under the supervision of Geoffrey Ian Webb and Mahsa Salehi.

The inclusion of co-authors reflects the fact that the work came from active collaboration between researchers and acknowledges input into team-based research.

In the case of Chapters 3, 4 and 6, and Appendix .1, which are presented as published or submitted research papers, my contribution to the work is detailed in Table 1.

Sections of submitted or published papers have not been renumbered.

Date: May 5, 2021

Table 1: Contribution to Included Publications

Thesis Chapter	Publication Title	Status	Nature and % of student contribution	Nature and % of Co-authors' contributions	Co-author(s), Monash student?
3	Emergent and Unspecified Behaviors in Streaming Decision Trees: Why Online Decision Trees Perform So Well	Submitted to DMKD	**80%** Concept Experimental Design All technical work Writing all drafts Identification of all unspecified and emergent behaviors	Geoff Webb 10% Research Supervision Editing Experimental Design Publication Strategy Mahsa Salehi 5% Research Supervision Editing Albert Bifet 5% Consulting Editing	No
4	Extremely Fast Decision Tree	Published, Proceedings of SIGKDD 2018	**70%** Concept All Experimental Work Writing all drafts All technical work Theoretical Claims and Proofs Experimental Design	Geoff Webb 20% Concept Research Supervision Editing Experimental Design Mahsa Salehi 10% Research Supervision Editing Experimental Design	No
6	An Eager Splitting Strategy for Decision Trees	Submitted to DMKD	**80%** Concept Experimental Design All Experimental Work Writing all drafts All technical work	Geoff Webb 5% Research Supervision Editing Experimental Design Diagram with simple stream example Mahsa Salehi 5% Research Supervision Editing Experimental Design Heitor Gomes 5% Research Supervision Editing Initial Ensemble—Ranking Analysis Albert Bifet 5% Research Supervision Experimental Design Editing	No
Appendix .1	On The Shattering Coefficient of Decision Trees	Published in Expert Systems and Applications	**10%** Finding an issue with the recursive expression of the Shattering Coefficient for decision trees while the work was being prepared	Rodrigo Mello 80% (Main Author) Albert Bifet 10%	No

Preface

I entered academia with the driving motivation of learning how strategies are learnt. I started this work with a desire to gain an intuitive understanding of how equilibria are established in multiagent games and how an individual agent might be able to learn a profitable strategy during the process of gameplay in an unpredictable and ever-changing environment, with the assumption that such agents could be "machines".

This question may be tackled at various levels of abstraction of other agents in the game; one may focus on one-on-one play, as with Google's AlphaGo, or work on derivatives and abstractions of the problem that deal with larger population effects. The nature of PhD projects varies widely, and I picked from the scope of topics available to me the problem of studying concept drift in the hope that it would be a derivative I would find of particular use in developing my intuition. It has served its purpose well.

This work is focused on online learning; little or no mention is made of games. It consistently retains focus on design and analysis of online learning strategies; there are no explicit detours into multiagent gameplay. However, the core thesis of the work—that it is possible to design effective learning strategies for changing environments—is essential to support the notion of agents learning strategies in complex environments. It is this microcosm of the problem space that I have used as a vantage point to ponder the larger problem that has been driving my personal curiosity.

Contents

Chapter 1

Introduction

Should learning be driven by guarantees, or by usefulness?

Real world learning problems inherently have concept drift—that is, processes generating data ("concepts") are always changing in unexpected and unpredictable ways. Human beings learn and update their models of the world in rapidly changing environments well enough to be able to strategize and survive.

It follows that the natural setting for studying machine learning is also one in which concepts are always changing in unexpected and unpredictable ways as data arrive in streams.

However, studying supervised machine learning under such settings does not lend itself to convenient theories about whether a learner can generalize—that is, whether it can predict reasonably well on "out-of-sample" data that have not been used for training the learner. In order to be able to reason about generalizability of a learner, the concept being learned cannot change unexpectedly and unpredictably. It must either not change at all, or must change in a predictable manner.

The inability to reason about generalization should not be a hindrance to studying machine learning under the natural, concept-drifting setting. If the objective of machine learning is generating useful predictions, the measure of progress towards that objective must be practical and relevant to the task at hand. One such measure might be predictive sequential accuracy (prequential accuracy). However, prequential accuracy must not be the only measure by which learners are assessed—it should be a component of a schema that is relevant to the learning task. Similarly, a measure such as generalizability is quite useful as a heuristic component of an evaluation function for a learner even though it assumes stationarity of the data-generating process. However, if it becomes overly important as a yardstick of utility or admissibility, it may push the field onto a trajectory in which learners are designed for stationary streams primarily so that properties relating to generalizability on stationary streams can be proved, as though that were the mark of validity of the learner.

On such a trajectory, the field of machine learning might limit study of adaptive learners that may be difficult to reason about in terms of generalizability on evolving streams in favor of non-adaptive learners for stationary streams, which may be seen as being more "valid". The default response to concept drift then might be to continuously rebuild non-adaptive learners rather than

to develop adaptive ones.

Although theoretical properties—such as divergence of a decision tree from an "ideal batch tree" or a low shattering coefficient—are by nature dependent on strong assumptions (such as the identical-and-independently-distributed-data assumption) that do not hold in the real world, they are highly informative and are of great utility when used as heuristics to evaluate and guide algorithm design. But if satisfaction of such properties is treated as an incontrovertible design principle in machine learning, the end result will be the continued process of algorithms being designed for trivial scenarios purely because "important" properties are provable. Rigor with respect to toy problems would supersede the practical and utilitarian goal of solving real ones.

This work focuses on design, analysis and experimentation with a notable deficiency of strong assumptions. It studies the practical problem of learning in a changing environment, starting with the metalevel of emergent behaviors due to unspecified algorithm implementation details, followed by novel design solutions for online decision trees as independent learners and as part of ensembles.

1.1 Thesis Format

This is a thesis including published works, a format available at Monash University in which the thesis is comprised of works that have been published or have been submitted for publication. Some chapters in such a thesis chapters may comprise published or submitted papers with very brief framing text for continuity.

In this thesis I present one published paper, "Extremely Fast Decision Tree" (KDD 2018) as Chapter 4, and two papers under journal review, "Emergent and Unspecified Behaviors in Streaming Decision Trees" as Chapter 3 and "An Eager Splitting Strategy for Online Decision Trees" as Chapter 6. Chapter 5 on combining the adaptation mechanism from Hoeffding Adaptive Tree and the eager splitting mechanism from Extremely Fast Decision Tree is written as a short regular chapter. "On the Shattering Coefficient of Decision Trees" is provided as an appendix (Appendix .1) to the thesis, as is required for inclusion in a Monash thesis of this form where my contribution is not as lead author.

Brief Introduction, Background and Related Work (Chapter 2) and Conclusions (Chapter 7) chapters provide context and structure. Note that some text in these chapters may overlap with text found in the publications in order to improve readability of the thesis.

1.2 Overview

Predicting the trajectory of events in our world so that we may plan ahead is essential to our survival and central to the human condition.

With enough data, natural tendencies become self evident; for instance, regression towards the mean. However, we may often be interested in tendencies that are more relevant to us in the near term, and these may be different to patterns observed in the long term. The world is always changing. How does one make the most appropriate decisions in our rapidly changing world?

A great deal of additional complexity accrues from humans playing adversarial and cooperative games while learning; it follows that making useful predictions in a changing world would be also be important to someday achieve strong intelligence. How do we make useful predictions under these changing circumstances?

As with most real-world challenges, the problem is difficult to precisely define. For the purposes of making scientific progress and communicating findings unambiguously enough that they are of practical use, I study a reasonably well defined, abstract problem: the problem of supervised classification in a scenario with evolving data streams. I hope that findings and artefacts that result from this study either provide useful indications for solving real-world problems, or do so directly.

1.2.1 Problem statement

How can prequential accuracy be optimized in supervised data stream classification with and without concept drift?

1.2.2 Problems studied

The premise of this project is that learning under concept drift is not well understood, and that a better understanding is needed.

The following questions are addressed in this work:

- **Are theoretical justifications provided for performance of online learning algorithms consistent with the true explanations underlying algorithm behavior? Do unspecified features affect algorithm behavior, and if so, which ones individually or through interactions lead to improvement or deterioration in performance, and how?**

  This theme is present throughout the work. Chapter 3 addresses it directly, diving into the MOA [16] interpretations of the state-of-the-art online decision tree learner, Hoeffding Tree, and an adaptive variant Hoeffding Adaptive Tree, finding significant implementation effects and emergent behaviors due to unspecified features that are not addressed in the theoretical constructs of the algorithms but contribute to algorithm performance in a positive manner that is measurably significant. Chapter 6 notes that the divergence bounds given for the HoeffdingTree are quickly rendered inutile; while the bound on divergence is a nice property, divergence from the ideal batch tree can be very large in practice as it increases with the number of leaves—which increase exponentially—and thus bounding divergence is not as useful a justification for the algorithm's efficacy in practice as effects from many unspecified implementation details are.

- **Can the state-of-the-art online decision tree algorithm, Hoeffding Tree, be improved?**

  We answer this question in the affirmative in Chapter 4 by presenting Hoeffding AnyTime Tree (HATT) [78], an online decision tree that is statistically more efficient than Hoeffding Tree [36], that converges to the ideal batch tree while Hoeffding Tree does not, and achieves higher prequential accuracy on a large set of test streams taken from the UCI Machine Learning repository. HATT, implemented as Extremely Fast Decision Tree (EFDT) minimally increases the order of time complexity as compared to Hoeffding Tree. It achieves its outperformance through a redesign of the split selection mechanism, leading to greater overall plasticity in terms of the stability-plasticity tradeoff [84, 30] while preserving the Hoeffding Test as a rigorous basis for stability.

- **Can the state-of-the-art online decision tree algorithm, Hoeffding Adaptive Tree, be improved?**

  From Sections 2.4.3, 2.4.4, 2.4.5, and 2.4.6, we gather the primary strategies that may be used by individual tree learners to respond to concept drift center around:

  - how tree structure is grown through splitting;
  - and, how tree structure is modified in order to adapt to concept drift

  Our work, Extremely Fast Decision Tree, described in Chapter 6, focuses on splitting eagerly in order to build tree structure efficiently for learning from stationary streams. Meanwhile, Hoeffding Adaptive Tree [11] focuses on adapting tree structure to respond to concept drift.

  In Chapter 5 we study the effects of combining the split mechanism from Extremely Fast Decision Tree with the concept drift adaptation mechanism from Hoeffding Adaptive Tree. The resulting system, which we refer to as Extremely Fast Adaptive Tree (EFAT), outperforms HAT on prequential accuracy on stationary streams with and without concept drift, and on a synthetic concept drift testbench (Chapter 5).

- **Is increasing base learner plasticity in ensembles, and thus ensemble diversity, advantageous for ensemble learning?**

  Replacing Hoeffding Tree with HATT as the base learner in boosting and bagging ensembles leads to measurably significant increases in prequential accuracy in online settings with and without concept drift, as we show in Chapter 6.

1.2.3 The problem of generalization

The generalization question asks: How well will the hypothesis output by my algorithm perform on data that are out-of-sample?

Statistical Learning Theory [83, 115] provides a way of reasoning about generalization in terms of the algorithm bias $\mathcal{F}$—the set of hypotheses produced by the algorithm—and n, the number of data instances; simply put, if an algorithm has a shattering coefficient $\mathcal{N}(\mathcal{F}, n)$ (size of the algorithm bias) that grows polynomially or slower in n, it is possible to bound the probability that any hypothesis in the bias has empirical risk that differs from its true risk by more than some ϵ:

$$P(sup|R(f) - R_{emp}(f)| > \epsilon) \leq 2\mathcal{N}(\mathcal{F}, n)exp(-2n\epsilon^2) \tag{1.1}$$

This is a learning guarantee that bounds the probability of a hypothesis f produced by the algorithm performing very differently on empirical data (risk $R_{emp}(f)$) compared to its expected performance (risk $R(f)$).

Statistical Learning Theory assumes a stationary data generating distribution. There have been attempts to extend the theory to the scenario where distributions change (and thus have concept drift) [72, 33], though with necessarily heavy assumptions on the nature of change in the stream.

Our focus is on experimentally optimizing prequential accuracy; we do not reason about generalization by formally restricting the nature of change of the generating distribution.

1.3 Structure

This manuscript is organized as follows:

- **Chapter 2** provides a brief overview of related work that should be sufficient to read the rest of the thesis.
- **Chapter 3** describes unspecified features and their effects in VFDT and Hoeffding Adaptive Tree.
- **Chapter 4** describes an improved online decision tree base learner, EFDT, for stream data mining (and consequently enables us to study avenues for better concept drift response).
- **Chapter 5** studies Extremely Fast Adaptive Tree (EFAT), obtained by using the revision mechanism from Hoeffding Adaptive Tree with EFDT.
- **Chapter 6** compares EFDT as a base learner in boosting and bagging ensembles against VFDT.
- **Chapter 7** concludes this manuscript with a summary of findings, and briefly touches on directions for future work.

Chapter 2

Background and Related Work

2.1 Overview

The background work referenced in this section and the narrative that accompanies it is intended to clearly and concisely demonstrate the context within which my research was conducted and the role it occupies in advancing the state of knowledge in the field.

The reader might find it useful to first review the techniques used to update models to reflect change occurring in drifting streams. Change detection mechanisms were first used as wrappers that signaled when models should be rebuilt, and eventually made their way into models themselves. Thus we briefly cover extant change detectors. While our aesthetic objective has been to aim for adaptive learning free of explicit change detection, change detection is still indispensable at the current stage of development of the field.

Decision trees are the primary class of learners being studied in the field of concept drift as well as in this work. An overview of online decision trees and their ensembles follows the overview of change detectors.

We then move on to a brief description of test data streams in the literature. This is significant because of the lack of literature offering a standardized testbench. My work comprises by far the largest experimental testbench in online learning.

The literature review mirrors the field and this manuscript in terms of the focus on decision trees; this is still a relatively new subfield of machine learning and a wider approach in terms of model classes is yet to be comprehensively studied. Decision trees are the main object of study in online learning.

We use the Massive Online Analysis platform [16] as the workbench for our experimentation. In the interest of reproducibility and confidence in our results, we used MOA 2016.04 for all of our experimentation.

2.2 Terminology

The notation below is adapted from "Characterizing Concept Drift" (Webb et al, 2016) [120] and "A Survey on Concept Drift Adaptation" (Gama et al, 2013) [44]:

1. **Data stream** A sequence of data instances generated by a process. At time $t \in \mathbb{N}$, we draw

an *example* (or *instance*) (X_t, Y_t) which is a pair consisting of an input vector $X_t \in \mathbb{R}^d$ and a class value $Y_t \in \mathbb{N}$. (X_t, Y_t) is drawn from the joint probability distribution $p^t_{X,Y}$.

2. **Data stream Classification** The task of predicting Y_t given X_t at time t.

3. **Concept** The joint probability distribution at time t, $p^t_{X,Y}$.

4. **Concept Drift** $p^{t_0}_{X,Y} \neq p^{t_1}_{X,Y}$. For two distinct times t_0 and t_1, the joint distribution $p^{t_0}_{X,Y}$ differs from $p^{t_1}_{X,Y}$.

5. **Class Drift** $p^{t_0}_{Y|X} \neq p^{t_1}_{Y|X}$. For two distinct times t_0 and t_1, the posterior distribution of the class variable $p^{t_0}_{Y|X}$ differs from $p^{t_1}_{Y|X}$.

6. **Covariate Drift** $p^{t_0}_{X} \neq p^{t_1}_{X}$. For two distinct times t_0 and t_1, the covariate distribution over non-class variables $p^{t_0}_{X}$ differs from $p^{t_1}_{X}$.

7. **Drift Magnitude** $D(t, u)$. Given some function D that computes distance between concepts, $D(t, u)$ measures the drift in the concept from starting time t to end time u of the concept drift.

8. **Abrupt Drift** $p^{t_0}_{X,Y} \neq p^{t_1}_{X,Y}$ for $t_1 = t_0 + 1$. A concept $p^{t_0}_{X,Y}$ extant at time t_0 is suddenly replaced by a different concept $p^{t_1}_{X,Y}$ at time $t_1 = t_0 + 1$

9. **Gradual Drift** $\forall_{t \in [E_a, S_{a+1} - \nu]} D(t, t + \nu) \leq \mu$ Where E_a is the time the concept a ends, S_{a+1} is the time the concept $a + 1$ starts, μ is the maximum allowed drift magnitude during any time interval ν during the drift phase between the two concepts

10. **Prequential Accuracy** Predictive sequential accuracy [44, 43, 32, 92]. Each example in a data stream is first used for testing, then for training. The error from each test is aggregated and the average error is computed over a window that may span the last M examples, or from the beginning of learning, or may be adjusted using a more complex mechanism (e.g. Prequential-ADWIN.). Prequential accuracy is the most commonly used evaluation measure for performance of classification algorithms on streaming data. Prequential accuracy meshes with the prequential philosophy that stresses the importance of predictive models that generate sequential forecasts over descriptive models that describe the parameters of a system.

Note that the true class value for each example in the stream is made available after the prediction has been made. While there is likely to be some lag between the prediction and the availability of the true class in real scenarios, we study a reduced problem wherein the true value is available as soon as the prediction is made.

2.3 Change Detectors

Since change detection has been a central strategy to construct algorithms that learn under concept drift, I will discuss it in some detail. Change detectors are used to determine when there has been a change in the data-generating distribution, so that corrective measures can be taken in order to keep the model as current as possible. In other words, change detectors serve as a signaling mechanism for when the model must adapt to a new concept. This is particularly relevant in the case of abrupt

drift, when concepts change suddenly; it is also relevant in the case of a gradual transition between concepts by providing an indication of when the cumulative gradual change is large enough that it is profitable to update the model, risking a short term degradation in performance for a potentially longer term improvement obtained from more closely matching the extant concept.

There are two primary approaches to change detection: observing the data distribution to assess whether it changes, and observing classification performance to assess whether it changes. Changes in the model may alter classifications even in the absence of drift in the stream; drift can result in a model's performance improving, thereby preventing drift detection, even though even better performance might be possible if the drift were recognised and the model revised.

2.3.1 ADWIN Family

ADWIN [10] is a change detector with some nice properties; it has a variable self-adjusting window size with resizing decided, once again, using the Hoeffding inequality in order to offer a guarantee with some confidence that old examples are only discarded when a change has actually been detected. ADWIN tracks $1-0$ classification loss to detect change (a misclassification is a "1", and a correct classification is a "0").

ADWIN maintains a growing window of 1-0 losses (obtained from the learner's classification attempts) as examples. It checks after every example has been incorporated whether there is a cut in the window such that the resulting sub-windows w_0 and w_1 have differing measured averages. If the observed population means are different, it signifies that a change has occurred, and discards the old window w_0. Note that this could result in ADWIN signaling a change when a learner is just learning and thus error has fallen. ADWIN is simply a change detector; it is left to the user to decide to ignore detections where error has fallen. If there is no change, then the two sub-windows must share the same true mean μ. It follows (from a variant of the Law of Large Numbers) that the population means of the two windows $\hat{\mu}_{w_0}$ and $\hat{\mu}_{w_1}$ must be very close if both windows are of sufficient size; i.e. $\hat{\mu}_{w_1} - \hat{\mu}_{w_0}$ must be very close to zero with high probability is the sub-windows are sufficiently large. The Hoeffding Inequality gives us a way of deciding whether the two population means have been drawn from the same population with some confidence and tolerance levels. We have $Pr[|\hat{\mu}_{w_1} - \hat{\mu}_{w_1}| - 0 > \epsilon_{cut}] \leq \delta/n$, with δ/n being the greatest probability that ADWIN decides to shrink the window to w_1 and n the number of examples (we correct for the number of tests conducted). Bifet derives an $\epsilon_{cut} = \sqrt{\frac{1}{2m} ln \frac{4m}{\delta}}$; thus, for a given tolerance δ, we can compute what threshold for difference in population means ϵ_{cut} leads to change being signaled. (Note: $m = \frac{1}{\frac{1}{n_0} + \frac{1}{n_1}}$, where n_0 and n_1 are the subwindow sizes.)

In practice, maintaining such a window, and evaluating all possible cuts is computationally inefficient; the same work [10] also proposes ADWIN2, a more efficient derivative of ADWIN that uses buckets to store $1-0$ loss history. The buckets are of size 2^i, and a design parameter, M, determines how many buckets of each size 2^i will be allowed. M is arbitrarily set to a value of 5 in their paper. Instead of evaluating all n possible cut-points, ADWIN2 creates a new bucket of size $2^0 = 1$ for an incoming example. If this causes the number of 1-sized buckets to exceed M, the two oldest buckets of size 1 are concatenated to form a bucket of size 2. If now we have more than M buckets of size 2, we repeat the same process, thus creating larger and larger buckets. When a drift is detected (the buckets are used as cutpoints to test) the oldest bucket is dropped. The number of tests to find a cutpoint is thus reduced from n to $O(\log_2 n)$. That ADWIN2 approximates ADWIN well was experimentally validated by the authors.

2.3.2 SeqDrift Family

A similar pair of approaches, SeqDrift and its successor SeqDrift2, [96], [86] use the Bernstein inequality (as does ADWIN2- this provides tighter bounds). SeqDrift2 takes into account improving drift detection sensitivity and reducing latency by adjusting ϵ_{cut} to be reduced as long as the drift detection false positive rate δ set by the user is not exceeded. Further, it only conducts one pass through the data repository (while ADWIN2 conducts $\log_2 n$ passes, where n is window size), and has a "left" repository, equivalent to the left sub-window in an ADWIN split window, that is implemented as a reservoir, a non-sliding window that contains instances spread over the past: a new instance replaces an older one with some probability so that one has a mixture of instances from different points in the past at any time. The "right" repository simply contains the latest examples. Since the "cut-point" is predefined, they only conduct one test every time an example is seen. Their results show comparable drift detection in terms of number of instances seen post drift before drift is signaled, which they term "delay", and a quicker processing time due to conducting a linear number of hypothesis tests ($O(n)$) with respect to the window size n (ADWIN2 conducts $O(n \log_2 n)$ tests).

2.3.3 Other Change Detectors

Other change detectors proposed specifically to aid learning with concept drift include Drift Detection Method (DDM) [46] and Early Drift Detection Method [4]. These do not use windows; the former tests for a rising mean rate of misclassification, with warning and drift detection levels as parameters, while the latter looks for an increase in the frequency of mis-classifications. Both ADWIN [10] and SeqDrift [86] claim superior performance to such window-free methods based on statistical process control.

Whichever change detector is used, wrapping entire ensembles or internal ensemble members with change detectors implies that models will be rebuilt from scratch every time drift is detected. Useful information from unchanged parts of the model will be discarded. There is a need for a more adaptive system that reduces both the number of base models and reliance on external drift wrappers to achieve both accuracy and performance gains—a motivating factor through the course of my journey.

2.4 Tree Learners

Of the many approaches to inductive learning, Decision Trees are a particularly utile paradigm that store knowledge in an easily interpretable manner. Algorithms that build decision tree models recursively divide the sample space with hyperplane decision boundaries. Each division represents a conditioning of the sample space on a particular set of data attribute values or ranges. The knowledge obtained from a Decision Tree is represented in an elementary form; in the classification case, each path down the tree results in a conditional probability distribution $P(C|X_1 = v_1, X_2 = v_2, ...)$, that is, the probability distribution of the class values given the observations $X_1 = v_1$, $X_2 = v_2, ...$ The regression case may use, for instance, a simple average of observed target values.

Decision Trees are a natural starting point for the study of extending inductive strategies to streaming scenarios; their simplicity allows us to compute model complexity [83], so we may exactly ascertain the state of over/underfitting, while their interpretability allows one to analyse the effects

of proposed strategies from the perspective of model fidelity with respect to a known sequence of generating distributions where an application is specified.

The history of decision trees is conjoined with the development of the field of artificial intelligence. A detailed review of early work on decision trees is found in [64]. At this stage, the nature of work was basic reasoning centred around the notion of "conditional focusing" in order to obtain concepts in the form of decision trees. The binary classification problem is the primary object of discussion in literature of the time; it is noted that a method for constructing decision trees would require, in the first instance, a set of labeled instances; and then, that a series of tests must be applied so as to separate the positive examples from the negative examples. It is also mentioned that the 1950s produced much speculation about how learning and other forms of artificial intelligence may be implemented—and that the 60s might provide concrete artefacts (they did).

A concrete decision tree algorithm was given in 1966 in the form of the Concept Learning System (CLS) in [65], which unfortunately we were unable to access directly. A brief description is provided in [90], which we use as our reference. Concept Learning System aims to construct a decision tree using the cost of classifying an instance as a guide to choose the construction pathway. Two types of cost are considered; the cost of obtaining the value of a given attribute, and the cost of misclassifying the instance. These costs are used to inform a lookahead mechanism wherein the costs of all possible subtrees of upto a fixed depth are computed, and the attribute likely to be least expensive is selected to continue tree construction. Quinlan deems this a minimax-like approach.

The comparison to minimax is a fortunate one, as at this point it is worth noting the primary factor that influenced decision tree development from this starting point. The form of the decision tree is fairly immutable; it is a simple artefact that represents a concept in a manner easily accessible to the human observer. The structure of this artefact is that of a recursively conditioned linear graph of decisions. Any procedure that constructs such a graph is necessarily, obviously, tautologically, a procedure of recursive conditioning. The space of design choices for formulating decision tree procedures is thus limited to two choices: the choice of value system informing the conditioning procedure, and the choice of heuristic the designer considers most appropriate for the value system. For instance, CLS makes minimizing classification cost, defined in a particular way, its primary objective; and the heuristic it uses to assess the expected value of this objective is a limited-depth exploration of the space of potential trees. Just as the core of the minimax strategy is the evaluation heuristic, the core of the decision tree is the split evaluation heuristic.

CLS was followed by the Iterative Dichotomizer (ID3) in 1979 [89]. ID3 differs from CLS in its choice of evaluation objective: the separability of classes. It uses information gain as a heuristic. It was limited in terms of applied usage in its initial iteration as it was, like CLS, only designed to perform binary classification; however, an even greater limitation was the assumption that no training instances would be erroneously or wrongly labelled—ID3 relied on resampling the input space to build a tree, proceeding in steps, assuming that at each stage the current artefact had perfectly classified the input examples, until all training examples had been utilised. ID3 was further studied [91] to assess performance when only a part of the training set was explored (extracting "approximate rules", in the language of the day) .

A more complete system that included pruning to adjust for overfitting, multiclass classification, and a solution for regression, "Classification and Regression Trees" (CART) was proposed in 1984 [21]. CART shared the objective of maximising class purity with ID3; it differed in the heuristic choice of Gini coefficient vis-à-vis information gain for ID3. Consequently, a major improvement of ID3 called C4.5 that also addressed pruning, multiclass classification and regression was released [88]. The ideas embedded in C4.5 and CART form the basis of most modern decision tree learning

systems today designed for both batch and streaming cases.

2.4.1 C4.5

C4.5 was one of the most widely used decision tree algorithms for batch learning. It was designed by Ross Quinlan [88] and uses Information Gain as the heuristic for deciding best splits. Information Gain should tell us what the relative class purity of two given class distributions is (the idea is that more "pure" class distributions contain less information). The class purity of the distribution at a node is compared with the aggregate class purity of the distributions resulting from a split to compute Information Gain. The Information Gain due to several attributes is then compared to find the best one. Simplifying Quinlan's terminology and notation, one can write down information (or entropy) of a class distribution at a node N as $I_N = \sum_{i=1}^{c} -p_i \log p_i$, where p_i is the probability observing class i at node N. If node N were now split on attribute A leading to j child nodes, the information contained in each of the child nodes is $I_{N_A^j} = \sum_{i=1}^{c} -p_i^{N_A^j} \log p_i^{N_A^j}$, and the Information Gain is given by $\sum_j I_{N_A^j} - I_N$. Finding the attribute A that maximises gain is used as a heuristic in C4.5 and similar algorithms to determine a better split.

2.4.2 Early Online Decision Trees

Applying C4.5 to a streaming setting is not straightforward—the main problem is that of anytime prediction. How many examples does one need to see before deciding that one has enough data accumulated to split it into test, validation and training sets and create a model? How frequently does one need to update this model in order to optimize prediction accuracy? Should one use a sequence of sliding windows? Is there a more efficient approach?

One potential solution is to learn in multiple passes with sets of stored instances; this raises the question of what the ideal working instance repository size must be—that is, *how* many instances should be stored at any given time?—a finicky hyperparameter. Such a choice would require a significant space overhead and unduly influence the method's anytime predictions—predictions requested on-demand during the continuous learning process, a standard expectation of online learners. Clearly, in learning from potentially infinite data streams, it is desirable that instances are not stored at all.

A one-pass solution, wherein each example is processed exactly once, is ideal for online settings. Work on scalability of batch learners helped set the foundation for one-pass learning in sequential prediction scenarios.

Bootstrapped Optimistic Algorithm for Tree construction (BOAT) [48] represents a typical attempt at learning from a large database that does not use a predictive sequential setting, by sampling fixed size chunks that are used to bootstrap multiple trees. A "coarse" tree is then extracted, based on the overlapping parts of the bootstrapped trees in terms of split decisions; this tree is further refined to produce a final tree by passing the whole dataset over it. The system is "incremental" in the sense that it can process additional datasets; and it is responsive to drift in that the system detects when a new dataset requires a change in split criterion at a node through a global assessment of split criterion, and causes a rebuild of the subtree rooted at that node. While key ideas that shape later trees are developed in this work, the sizes of the initial bootstrap samples are arbitrarily chosen, and concurrently the notion of anytime prediction is not entertained—there is no automated way of determining how many examples suffice to build a first reliable tree. Further, the focus is on minimising utilisation of main memory; it is assumed that the database D is available

for a corrective step in the algorithm. On the other hand, Hoeffding Tree is truly one-pass, in that it is assumed that an example is seen only once, then discarded. Meanwhile, the RainForest framework [47] introduces the idea of storing attribute-value-class counts at nodes, which we see in Hoeffding Tree as *node statistics*.

2.4.3 Hoeffding Tree

Hoeffding Tree was one of several attempts [102, 113] to provide a one-pass solution, and the first one-pass learner to provide guarantees on deviation of the tree from the batch tree—the hypothetical tree that would be learned if all infinite examples from a stationary distribution were made available at once. Hoeffding Tree uses a statistical test—the Hoeffding Test [36, 59]—to determine the most appropriate time to split and attribute to split on. The Hoeffding Tree work, "Mining High Speed Data Streams", won the KDD Test of Time Award in 2015 [107]. Its success may be attributed to the fact that it provided a one-pass solution, deviation guarantees from the ideal batch tree, and a statistically sound rationale for deciding splits. Hoeffding Tree refers to the theoretical construct proposed in [36], while its implementation is referred to as Very Fast Decision Tree (VFDT).

VFDT is designed for a process generating identical and independently distributed (i.i.d) data. In order to ensure that the tree being built is stable and to ensure that there is a strong rationale for deciding that a split decided upon at a point holds as the stream progresses, VFDT uses concentration inequalities to decide when a potential split attribute has overtaken the remaining potential split attributes at a node in terms of information gain. In particular, the Hoeffding Inequality is used to test the likelihood that the average observed difference in information gain at a node for the two attributes with highest information gain differs from their actual average difference by more than some factor ϵ.

Using notation from [36], consider attributes X_a and X_b that are the top two in terms of Information Gain $G(X)$ at some node N. Say $\bar{G}(X_a) > \bar{G}(X_b)$, i.e. the respective average gains are compared. VFDT does not split node N until it is reasonably certain that there is a winning attribute. To do this, it first computes $\Delta\bar{G}(X) = \bar{G}(X_a) - \bar{G}(X_b)$ at nodes with impure class distributions (once every n_{min} instances, because of the expensive nature of information gain computation). Hoeffding Tree was one of several attempts [102, 113] to provide a one-pass solution, and the first one-pass learner to provide guarantees on deviation of the tree from the batch tree—the hypothetical tree that would be learned if all infinite examples from a stationary distribution were made available at once. We want to test whether the average difference in gain due to the two attributes is non-zero- that would imply one attribute is better than the other. The tolerance ϵ for the deviation of $\Delta\bar{G}$ from 0 is computed from:

$$\epsilon = \sqrt{\frac{R^2 \log(1/\delta)}{2n}} \tag{2.1}$$

where R is the range of our random variables. With confidence $1 - \delta$, after n examples have been seen, the true mean of the random variable is $\Delta\bar{G} \pm \epsilon$. This is derived from the Hoeffding Inequality [59] : for some collection of independent random variables $X_1, X_2, ...X_n$, $Pr[\bar{X} - \mu \geq \epsilon] \leq e^{-2n\epsilon^2}$. The random variable $\bar{X}$ in our case is $\Delta\bar{G}$. $\mu = 0$. And we decide whether to split or not at a node based on whether $\bar{G}(X_a) > \bar{G}(X_b)$.

The authors note that the usage of $n_{min} > 1$ has the effect of implementing a smaller δ than the one specified by the user because more examples are considered than would have otherwise been

considered before deciding a split must be made. They also note than this slows down the building of the node (which is compensated for by reducing the number of Information Gain computations).

VFDT focuses on using as few examples as possible from an *i.i.d* distribution to build a tree. A consequence of this is that as the number of examples grows large, for a fixed confidence level $1-\delta$, ϵ, the tolerance, decreases. So a very small difference in attribute gain ratios $\Delta\bar{G}$ will exceed the monotonically decreasing tolerance ϵ and signal that a split is necessary. Because of the monotonic decrease in ϵ, we should eventually see $\Delta\bar{G} > \epsilon$, but this may not occur before $\Delta\bar{G} < \epsilon < \tau$ occurs, where τ is the tie threshold. The tie threshold exists to resolve a scenario where $\Delta\bar{G} < \epsilon$ holds up splitting a node and thus holds up learning (that is, the top two attributes are so close that it is very difficult to determine a split, and when the threshold ϵ falls below the tie threshold τ, it is likely most effective to simply choose the current best attribute, split on it, and continue learning instead of waiting for $\Delta\bar{G} > \epsilon$ in order to split).

In the edge case where $\Delta\bar{G} > \epsilon$ is achieved exactly when $\epsilon < \tau$ occurs, implying that a winning attribute has been found among the top two just when the tie threshold $\epsilon < \tau$ has been met, both making the split by choosing the top attribute and invoking the tie-breaking mechanism are valid courses of action (the ambiguity is not formally addressed, but both choices will likely lead to the same attribute being picked).

VFDT is not designed to handle concept drift; it assumes a stationary data generating distribution. It does not revise structure that is no longer applicable following a change in distribution. The drift response of VFDT is studied in [40].

2.4.4 CVFDT

The VFDT work was followed up with a method specifically aimed at learning from streams with concept drift, the "Concept-adapting Very Fast Decision Tree" or CVFDT [63]. CVFDT makes a number of important design modifications. One is the introduction of a window so that older examples are forgotten. Forgetting older examples that are no longer relevant to the current concept ensures that they no longer contribute to the distribution at a node and thus to a split decision.

CVFDT maintains instance counts at every node, not just the leaves as does VFDT. CVFDT periodically scans its internal nodes for a possibly better split than the current one; if it finds that at an internal node, other attributes have better overall performance than the current split attribute, it builds a set of alternate trees based on those attributes. Each alternate tree receives a copy of each instance and thus learns; its prediction accuracy is constantly assessed; but alternate trees do not contribute to predictions output by the CVFDT. If and when an alternate tree is found to have better accuracy than the corresponding main subtree at a node, CVFDT swaps in the best alternate tree, replacing the subtree stemming from the current split at the node. This is the key mechanism by which CVFDT adapts to concept drift.

Note that a window as specified by CVFDT will have to be of sufficiently large size n when both the threshold ϵ and confidence level $1-\delta$ are specified according to Equation 2.1 so that there are enough examples to determine whether the attribute with highest information gain is better than the attribute with the second highest information gain. The required window size will vary widely depending on the nature of the stream and thus is a very difficult parameter to set. If the window is too small, alternates will be difficult to establish in the first instance and then difficult to grow in order to be of use; if it is too large, it would adversely impact the primary basis upon which CVFDT responds to concept drift—removing older examples from the window. Further, a window implies a potentially large memory usage, as instances in the window will have to stored

so they may be later unlearned; ideally we would want learners to be one-pass, as with Hoeffding Tree, which does not store instances.

2.4.5 Options for Hoeffding Trees

Hoeffding Option Tree [87] was the first online algorithm to introduce options for online decision trees. It constructs a Hoeffding Tree classifier in the usual manner—but once a split has been made at a node on some attribute using the Hoeffding Test with the usual confidence $1 - \delta$ that the top attribute is more informative than the second best one, the algorithm makes further splits on other attributes at the same node with confidence $1 - \delta'$ that the new split option is better than the best existing option.

Each optional split may lead to the growth of a subtree, as each example is passed down all the splits. Each subtree within the set of options at a node is allowed to predict, through a weighted vote—the individual probability predictions of each class across all the leaves an example reaches are summed.

It is noted in [87] that Hoeffding Option Trees can grow very rapidly. Automated pruning strategies were tried and deemed ineffective, and it is postulated that it is unlikely that automated pruning can obtain satisfactory results. Additional splits are restricted—it is suggested that the number of paths available to an example at each option node be restricted to 5 at most, a point at which experimental accuracy gains were diminished. Further, the δ' parameter is noted as being able to control tree growth—δ' is set in terms of δ through a multiplication factor α, thereby controlling how eagerly optional splits are made.

"Ambiguous CVFDT" or aCVFDT, an option tree based on CVFDT rather than on Hoeffding Tree is proposed in [76]. Unlike with CVFDT, where alternate subtrees being considered as replacements for the original subtree do not vote, optional subtrees are allowed to vote in aCVFDT, as in [87]. The main differences with respect to the strategy in [87] are that only the most accurate or the latest option may be allowed to vote, and that the underlying CVFDT has a moving window of examples, relative to which obsolete options are pruned. It suffers the same issue as does CVFDT with respect to the sufficiency of the window size for determining whether a new option is better than existing ones and consequently growing subtrees.

These works were followed by [66], which focuses on online regression trees where all attributes are numerical. The main change in strategy is that rather than adding optional subtrees one by one after an initial split has been made based on distinguishing the top two attributes using the Hoeffding Test, after a certain minimum number of examples n_{min} is observed at a leaf, optional subtrees are created simultaneously for all candidate splits that are indistinguishable (using the Hoeffding Test) from the best performing attribute in terms of their discriminative power. While this strategy also grows trees rapidly, it introduces the notion of introducing optional subtrees—and thus structure—before a split has been chosen.

Option trees provide an important conceptual basis for designing adaptive decision trees. In different ways, they all reinforce the notion that building more structure is a useful strategy. They provide an interesting direction for research in terms of comparison with ensembles, given they can effectively simulate the combination of multiple trees. However, on account of their rapid growth and potential difficulty to prune without moving windows of examples (which come with their own set of drawbacks), it may also be useful to explore strategies that create very simple models that may then be used with established ensembling methods.

2.4.6 Hoeffding Adaptive Tree

The Hoeffding Adaptive Tree (HAT) [11] places change detectors and error estimators at every internal node of a Hoeffding Tree. When used with ADWIN [10], which serves as both a change detector and error estimator, it is called HAT-ADWIN.

When a change in the concept is signaled at a node by its change detector, an alternate subtree begins to be built at that node, starting as a leaf node. Each instance passes both down the original subtree and the alternate. The method leaves unspecified the issue of whether the alternate is allowed to vote, and implementations can vary. When the alternate subtree has better accuracy than its corresponding original, HAT switches out the alternate for the original subtree.

This resourceful construction of potentially at most a single alternate tree based on inputs from change detectors and error estimators gives HAT an edge over CVFDT and option trees in limiting tree growth; further, it is demonstrated in [11] that HAT has much better performance than CVFDT in terms of prequential accuracy in most concept-drifting settings. As implemented, each alternate may have an alternate, and we study the effect of this and many other unspecified features in Chapter 3.

Note that our method Hoeffding AnyTime Tree (Chapter 4) is abbreviated as HATT to avoid confusion with HAT. While HAT relies on error estimation and change detection to alter tree structure and grow subtrees, HATT—similarly to Hoeffding Tree—relies on discriminatory heuristics alone for modifying tree structure and not on error or change detection. HATT is designed as a simple, elementary replacement for Hoeffding Tree in stationary settings with enough plasticity to be inducted into ensembles as an unstable base learner (thereby boosting ensemble diversity and potentially improving the learning of more subconcepts) in place of Hoeffding Tree, which is a stable learner [51].

2.4.7 OzaBoost

Online boosting (OzaBoost) [85] was proposed as an online version of AdaBoost [38, 39] following the vein of "boosting" weak learners — developing strong learners from combinations of weak ones [101]. AdaBoost is a batch learning ensemble method; a set of base models $h_1, h_2, ...h_m$ is trained in sequence, with each model having half of it's training examples drawn from the ones the previous model misclassified. This will create a series of models with increasingly better results on misclassified examples. Higher-indexed models may overfit to noise, so finding an optimal voting strategy for these models can be of importance (this will be application dependent...). AdaBoost represents steepest functional gradient descent on the chosen hypothesis space [27, 81, 80].

OzaBoost aims to recreate the weighting strategy of AdaBoost by furnishing each incoming example $Pois(\lambda = 1)$ times to the first requisitioned model; then, it adjusts λ so that it mimics having misclassified examples take half the total weight in the input to the next learner. In the MOA implementation, providing multiple copies of an instance is approximated by weighting an instance accordingly.

To simulate misclassified examples making up half of the examples provided to the next model in the ensemble, OzaBoost uses simple multiplicative factors f_m^c and f_m^w for the $m'th$ model depending on whether the example has been correctly or wrongly classified, so that an effectively larger λ is used for wrongly classified examples.

2.4.8 Online SmoothBoost

Online Smooth Boosting [31] is an online version of the smooth boosting algorithm [104] which instead of weighting examples with integer weights as with AdaBoost or OzaBoost uses real weights in $[0, 1]$ for each example, thus providing a rationalised continuous weighting scheme for examples in contrast with the stepped Poisson weighting provided by OzaBoost. One motivation for restricting the weights is to reduce the impact of noisy examples. Smooth online boosting also weights the predictions of each ensemble component in an attempt to mitigate the impact of poorer predictors.

2.4.9 Adaptable Diversity-based Online Boosting (ADOB)

ADOB [99] builds upon OzaBoost by positing that arranging the base learners in increasing order of accuracy will boost the value of λ favorably for the next base learner to enable better learning under conditions of concept drift—because more examples are likely to be misclassified by learners with lower accuracy, and those examples would be prioritized by being assigned half the total weight when fed to the next learner.

The paper specifies a usage wherein the entire ensemble is wrapped in an ADWIN change detector, an alternative ensemble begins to be constructed when a drift warning is issued, and the entire ensemble is replaced with the alternative enemble when drift is confirmed. However, the code provided in MOA defaults to assuming that it is each individual base learner that is wrapped in an ADWIN change detector.

Because the core strategy offered—the rearrangement of learners—was what we were primarily interested in testing, we used ADOB with VFDT and EFDT as base learners without drift detection and thus without having to resolve the ambiguity concerning the text in the paper and the code.

2.4.10 Boosting-Like Online Ensemble (BOLE)

BOLE [8] is an online boosting- like learning framework. It is based on OzaBoost [85]. OzaBoost and it's derivatives have excellent recovery from abrupt concept drift.

BOLE incorporates from the ADOB system the idea of sorting the ensemble members in order of worst performance. BOLE adds to this by allowing base models with lower than 50% accuracy to vote. It also replaces the drift wrapper algorithm; it uses DDM instead of ADWIN in order to achieve quicker drift detection. Again, there is a discrepancy between the publication and the code in terms of whether the change detector wraps the ensemble or individual ensemble components.

2.4.11 OzaBag

Bagging approaches target diversity—that is, a larger bias space of descriptive functions is used for our composite predictor function. A learner with a larger bias space (also confusingly called a "low bias" learner) uses a somewhat different design aesthetic to boosting approaches (which "boost" weak learners to create a strong one): instead of "zero-ing in" on a small portion of the hypothesis space through a process of gradient descent, bagging approaches supply a different subset of training examples to each component, thus simulating a different input space. This results in a larger spread over the predictors generated as a result.

OzaBag creates an ensemble of base models (VFDT trees in MOA, the standard usage in recent literature). Each base model is learned from a different input stream, those streams being random

resamples of the input. This is achieved by assigning each instance a different weight when provided to each base model; the weight is drawn randomly from a Poisson(1) distribution.

OzaBag is based on a simple notion of extending batch bagging to the online setting. In the batch scenario for bagging, let $P(K = k)$ be the probability of an instance in the training set of size N being chosen k times for input to some base model m. [85] argue that because we are considering indefinite streams, we can use the following reasoning to provide $Pois(\lambda = 1)$ copies of the instance to the first base model in the queue for online boosting: $P(K = k) = \binom{N}{k}(\frac{1}{N})^k(1 - \frac{1}{N})^{N-k}$, which is the binomial distribution. And as $N \to \infty$, the distribution of K tends to a $Pois(\lambda = 1)$ distribution—thus providing $Pois(1)$ copies of an instance in the online setting to each base model simulates bagging in OzaBag. Again, in the MOA implementation, providing multiple copies of an instance is approximated by weighting an instance accordingly.

2.4.12 Leveraged Bagging

OzaBag draws the weight for each instance of each ensemble member's input stream from a Poisson(1) distribution; this means that around a third of training examples are discarded, because a weight of 0 is drawn a third of the time with Poisson($\lambda = 1$)! The main contribution of Leveraged Bagging [12] is to draw weights instead from a Poisson(6) distribution, which doesn't discard as many examples. This contributes to diversity of ensemble components without compromising statistical efficiency.

2.4.13 Adaptive Random Forest

Adaptive Random Forest (ARF) [50] is an ensemble of online decision trees that follows on early development in Ultra Fast Forest of Trees [41]. As with regular random forests, the attributes available at each node for spliting are restricted to a random subset of all the available attributes. Just as random forests use bagging to increase diversity within the ensemble, ARF uses online bagging.

Unlike traditional random forests, ARF uses change detectors with each ensemble component to determine whether concept drift is occurring. Each ensemble component has one ADWIN change detector with a high tolerance level to warn of concept drift, and an ADWIN change detector with a very low tolerance level that confirms concept drift. When a warning is signaled on a tree, an alternative tree begins to be constructed. When a concept drift is signaled, the alternative tree replaces the original in the ensemble.

2.5 Testbenches

Experimental studies of online learning algorithms where concept drift is introduced into streams have relied on limited real datasets and on synthetic data streams that are difficult to measure against each other. There is a real need for a comprehensive testbench.

2.5.1 Generators that use a nonstationary distribution

Hyperplane Generator

One of the first synthetic generators to come out was the hyperplane generator, from [63]. This was introduced along with the CVFDT algorithm to compare performance with VFDT, and continues

to be widely used in concept drift studies. It is rather simple: a hyperplane in d dimensions is given by $\sum_{i=1}^{d} w_i x_i = w_0$. We may define as positive examples such that $\sum_{i=1}^{d} w_i x_i \geq w_0$ and as negative examples such that $\sum_{i=1}^{d} w_i x_i < w_0$. Obviously, this only allows us to have two classes. [63] use a slightly more complicated system where they have many parallel hyperplanes, but the points between still only have binary class labels, with labels alternating in between bands. It is noted that in a d-dimensional environment, if all but one dimension were weighted zero, then the only dimension with a weight will contain all the information about the concept; consequently, the weight of a dimension/feature tells us how much it contributes to the concept itself. A larger weight change would correspond to a larger concept change.

The hyperplane generator implemented in MOA is a simpler version that only does a single hyperplane. The hyperplane generator gives us an abstract way of testing learner performance as dimensionality increases. The restriction to binary classes may not allow a direct convenient comparison to multi-class problems. To an extent, it gives us control over the nature and magnitude of drift. Developed further, this does have potential for generating synthetic data for abrupt drifts. For gradual drift, one usually sets the hyperplane in rotatory motion with some angular velocity. This is a somewhat simple notion of gradual drift; consider a more complex scenario where subconcepts are drifting. It is not straightforward, even using multiple parallel hyperplanes, to extend this type of generator to a scenario where you intend to change a conditional dependency for a small subset of attribute-values to study learner responses to subconcept drift.

Random RBF Generator

The radial basis function (RBF) generator [17] consists of centroids with fixed classes and standard deviations assigned to them; the relative weight of each centroid is also the likelihood that it will generate the next data point. In the concept drifting version of the RBF Generator provided by MOA (RBFGeneratorDrift), the centroids are simply given velocities so that their locations are continually changing.

A generator with a measured level of drift: Recurrent AbruptDriftGenerator

A method that allows generation of abrupt drift with specified magnitude in simple categorical instance space is presented in [120]. A full conditional probability table is generated; thus, $p_{Y|X}$, the target distribution, is explicitly modeled. Further, a probability distance measure, the Hellinger, is used to measure the distance between distributions. This allows us to generate a measurable level of drift in settings with increasing dimensionality of the covariate and the class vector. The drift generate may either be covariate drift, or posterior drift ("pure" or "real" concept drift). Note that the size of the table is exponential in the number of input variables; this limits the applicability of the approach to low dimensionality. However, learning problems generated by this generator become significantly harder with small increases in dimensionality, possibly because of the relative independence of the variables.

One change that can be made to better measure changes in the subconcept is to account for the frequency of the covariate $X = x$ conditioned upon. This may be useful because it prevents a large change in the class distribution $p_{Y|X=x}$ conditioned on a rare example $X = x$ leading to observing a large change in the target $p_{Y|X}$. While this remains a direction for future research, it

would provide a good base for a drift generator that would allow a more principled approach for studying the effects of data dimensionality and subconcept drift.

I have added functionality to the AbruptDriftGenerator to generate recurrent abrupt drifts, invoked with the "Recurrent" option. In the charts in our publications, this generator usually appears in the form "recurrent—abrupt—222" in tables, where "222" may be replaced by other three-digit numbers. The first digit signifies the number of classes, the second digit the number of nominal attributes, and the third digit is the number of values each nominal attribute can take.

2.5.2 Meta-generators

Recurrent Concept Drift Stream

Recurrent Concept Drift Stream is provided by MOA [16]. It is parameterized with two distributions that are gradually alternated between. During the first occurrence of drift, the probability that an instance is picked from the alternative concept gradually increases sigmoidally up until the specified central position of the first concept drift change, and then the probability decreases symmetrically up until the end of the drift. A fixed period of stability follows, and then the drift and the stable period recur as many times as specified.

2.5.3 Generators that generate from stationary streams and must be used with a meta-generator

STAGGER

STAGGER concepts [103] as implemented in MOA consist of colored (red, green, blue) shapes (circle, square, triangle) of various sizes (small, medium, large). Three different classification functions are provided for binary classification. We generate concept drift by changing the classification function.

Waveform Generator

Two or three base waves are combined to create three different waveforms for classification [21, 45]. It offers 21 noisy attributes, or 21 attributes and 19 irrelevant attributes to generate data; one of three waveforms is randomly selected for each instance. The concept drifting version of the Waveform Generator provided by MOA (WaveformGeneratorDrift) adds "drift" to a specified number of attributes with respect to the base stream for a given seed—but this still creates a stationary distribution that is merely displaced from the original; therefore, it must be wrapped within a stream such as RecurrentConceptDriftStream in order to provide a nonstationary (concept-drifting) stream.

LED Generator

The LED generator [21, 45] has 24 binary attributes, of which 7 are used to represent a digit from 0 to 9 on an LED screen and thus correspond to the 7 segments in the representation. The digits correspond to 10 classes from 0 to 9. The remaining 17 attributes are irrelevant.

Each data instance consists of 0-1 coding for the 7 segment attributes determining which ones are displayed, while equiprobable random values are output for the 17 irrelevant attributes. At each

step in time, there is a 10% chance that one of the segment attributes flips, thus changing what is shown on the LED screen (this noise implies that a digit may be wrongly represented).

As with WaveformGeneratorDrift, the LEDGeneratorDrift does not by itself generate concept drift—it merely swaps pairs of attributes at the start of the stream and provides a new stationary stream that is displaced from the original for a given seed, so it has to be provided as one of the two distributions in a stream such as RecurrentConceptDriftStream.

Function Generator

Termed AgrawalGenerator in MOA, this generator [2, 45] offers a binary classification problem with six numeric and three categorical attributes, and ten different classification functions to choose from.

SEA Generator

The SEA generator is a synthetic stream written for the purposes of demonstrating the SEA algorithm [110]. It has three attributes, each of which takes values from 0 to 10. Only the first two attributes are relevant. Four concepts are generated by having 4 simple thresholds $\theta_1, \theta_2...$ such that $a1 + a2 \leq \theta_i \rightarrow class1$ else $class2$ within each concept (a concept is thus a division of the dataset with that threshold applied for classification - a decision surface). It is different to the rotating hyperplane system in that the concepts are fixed, there is no rotation, and one switches between concepts.

2.5.4 Limitations of stream generators

Synthetic data streams in use in the field are limited in terms of both rationale for construction as well as cross-comparability. It is not clear what expectations about response to future drift scenarios one might draw from results derived from most available generators—the generality of conclusions that may be drawn is unclear.

2.5.5 Real datasets

Of the real datasets commonly used for published experimental results in the study of concept drift [16], the electricity dataset is relatively small, and neither the forest cover-type nor the poker-hand datasets (also available through MOA and widely used) are naturally time-stamped streams (time sequences). Thus their utility for testing drift detection algorithms is limited.

In order to test with a larger number of datasets with real data, I have chosen the largest classification datasets with streaming data from the UCI Machine Learning repository [37] that have a clear classification objective and do not have missing values—the handling of missing values is a case for future study, as much of my work has involved carefully comparing elementary algorithm mechanisms that interact in unpredictable and complex ways.

As many of the UCI datasets were provided in multiple files—often without an easy or obvious way to concatenate them—using them involved a significant amount of text processing on large files using a suite of tools such as awk, R, and sed in bash and Python scripts to homogenize and concatenate the files and prepare them for use with MOA; the file converter provided by Weka was usually unable to deal with large files. In order to test with homogenised versions of streams without the incidental concept drift due to file boundaries in stitched datasets or concept drift otherwise present in the data, I shuffled the UCI streams, and used both shuffled and unshuffled versions

for experimentation. Shuffled streams were particularly time-intensive to generate and thus were stored on disk. The amount of text processing work that went into just creating my test datasets indicates the general lack of availability of a universal testbench of real data for works in online learning. The difficulty of obtaining a reliable testbench reflects in the literature in the form of a lack of consistency in the real data streams used for comparison of proposed streaming methods.

In this context, I hope that the testbench I provide with this work is a useful resource in terms of standardised evaluation of online learning algorithms. The full UCI testbench makes an appearance in Chapter 6.

Chapter 3

Unspecified Features

> It's not a bug. It's not a feature.
> It's just not specified.

It was not a particularly strenuous task to find the first gap in the literature with regard to understanding learning under concept drift, as with just about the first paper anyone studying stream learning would read—the KDD Test of Time Award 2015 Winner [107] "Mining High-Speed Data Streams" which gave us the Hoeffding Tree (HT) theoretical construct and its implementation Very Fast Decision Tree (VFDT)—I found that my implementation of their HT algorithm did not match the performance of its implementation found in the widely-used MOA software. Despite being designed for stationary distributions and including no mechanism for revising elements once learned, the MOA implementation of Hoeffding Tree responds unexpectedly well to concept drift.

Careful investigation revealed that the reason for the discrepancy was due to a number of open issues with the interpretation of the algorithm. This led me to a productive line of research revealing the manner in which seemingly innocuous design choices in the implementation of a machine learning system can lead to highly significant and potentially overlooked impacts on performance. Simple unspecified features interact in complex ways to significantly influence prequential accuracy, with the interactions being both crucial for algorithm performance and difficult to understand. I identify and study the various unspecified features in the MOA interpretations of Hoeffding Tree and Hoeffding Adaptive Tree—respectively the state-of-the-art learners for stationary and evolving streams—and present my findings in this chapter.

Noname manuscript No.
(will be inserted by the editor)

Emergent and Unspecified Behaviors in Streaming Decision Trees

Why Online Decision Trees Perform So Well

Chaitanya Manapragada · Geoffrey I Webb · Mahsa Salehi · Albert Bifet

Received: date / Accepted: date

Abstract **Keywords** Concept Drift · Hoeffding Tree · Explainability

Hoeffding trees are the state-of-the-art methods in decision tree learning for evolving data streams. Due to their efficiency, these very fast decision trees are used in many real applications where data is generated in real-time. In this work, we extricate explanations for why these streaming decision tree algorithms for stationary and nonstationary streams (HoeffdingTree and HoeffdingAdaptiveTree) work as well as they do. In doing so, we identify thirteen unique unspecified design decisions in both the theoretical constructs and their implementations with substantial and consequential effects on predictive accuracy—design decisions that, without necessarily changing the essence of the algorithms, drive algorithm performance. We begin a larger conversation about explainability not just of the model but also of the processes responsible for an algorithm's success.

1 Introduction

Algorithm design is, for now, mostly a manual, creative process; proposed strategies usually have major components assumed to cause the bulk of a behavior. But are we certain of our understanding of the causal relationships underpinning algorithmic performance?

A learning system may consist of many components, all interacting in complex ways with existing mechanisms and leading to a system that is difficult to comprehend. Further, implementation details are quite often open to interpretation—it would be extremely restricting to formally specify all aspects of implementation. Given a published algorithm, unspecified differences in implementation decisions may lead to—overall—hugely differing behavior and may be best termed *unspecified strategies*.

Minor aspects of theory and implementation may be considered irrelevant to the essence of the algorithm but may silently exert tremendous influence on measured objectives. For instance, we show that though Hoeffding Tree constitutes a major advance in incremental learning due to the introduction of the Hoeffding Test, a minor detail in how node statistics are initialized upon

Chaitanya Manapragada, Geoff Webb, Mahsa Salehi
Monash University, Australia
E-mail: FirstName.LastName@monash.edu

Albert Bifet
University of Waikato, New Zealand
E-mail: abifet@waikato.ac.nz

node creation has a significant role to play in its performance. This "minor detail" pertaining the initialization of node statistics is not considered a key aspect of the algorithm. It is merely a convenience. But it is a convenience that has a strikingly large impact on the performance of HoeffdingTree, as we demonstrate in Section 4.

The complexity of interactions between components of a learning system is such that the most unassuming of algorithm components may interact with other parts of the system to produce unexplained variations of performance. While these are subsumed into the overall aesthetic and direction of the algorithm developer and the algorithm itself, thus resisting straightforward detection, they warrant study and explanation, especially so when effects are unexpectedly significant. It is with this rationale that we study the benchmark MOA (Bifet, Holmes, Kirkby, et al. 2010) implementations of two state-of-the-art stream learning methods: Hoeffding Tree (Domingos and Hulten 2000), and Hoeffding Adaptive Tree (Bifet and Gavaldà 2009). We find that simple design decisions in the algorithms and their respective implementations that are well within the scope of reasonable interpretation lead to substantial and consequential variations in prequential accuracy.

The main contributions of this paper are the following:

1. Identification of unspecified design decisions in Hoeffding Tree and Hoeffding Adaptive Tree
2. A thorough experimental study of how such unspecified features, individually and in combination, influence algorithm behavior and prequential error performance
3. Suggestions for unspecified features that may be worth formally inducting into the algorithms
4. A discriminative testbench that helped us identify differences in behavior and prequential error performance

The outline of this paper is as follows: Section 2 presents Background and Related Work; Section 4 details unspecified features and emergent behaviors due to them in the MOA implementation of Hoeffding Tree; Section 5 details unspecified features and emergent behaviors due to them in the MOA implementation of Hoeffding Adaptive Tree; and finally Section 6 contains our Conclusions.

2 Background and Related Work

2.1 Batch learning origins of Decision Trees

Decision trees were some of the earliest mechanisms identified (Hunt 1962) as highly interpretable models for the *representation and storage* of knowledge. This representation of knowledge was initially in the form of a simple binary tree. All of the data were collected at the "root" vertex of the tree; then, the datapoints were separated into the two child vertices ("nodes") based on a decision pertaining to the value of each datapoint. For instance, with unidimensional data $(x_1, x_2...x_m)$ obtained from the real line observed at the root node N_{root}, the data point x_i might assigned to the child node on the left N_{Left} if less than 0, and to the node on the right N_{Right} if greater than or equal to zero. Further subdivision may be possible in recursive manner. This binary data structure is easily generalised—the data may also be multivariate and nominal (e.g. $\mathbf{x_i} = (Sunny, Cold, Windy)$), and trees are not required to be binary (for instance they may be ternary—with the root having child nodes $N_{Right}, N_{Middle}, N_{Left}$).

Algorithms for the construction of decision trees were consequently proposed; major milestones begin in 1966 with the Concept Learning System by Hunt et al. (Hunt, Marin, and Stone 1966), followed on by Quinlan in 1979 with ID3 (Quinlan 1979; Quinlan 1983; Quinlan 1986), Breiman in 1984 with Classification and Regression Trees (CART) (Breiman et al. 1984), and C4.5 by Quinlan in 1992 (Quinlan 1992).

These algorithms share a common core instrumentation; a split criterion determines how each node, starting with the root, splits data that have filtered to it; child nodes are created, and the tree is thus recursively grown. Early systems assumed data could be perfectly separated, and that data and trees were both binary; however, CART (Breiman et al. 1984) and C4.5 (Quinlan 1992) were robust systems that did not make assumptions of perfect data separability, pruned themselves to avoid overfitting, allowed multivariate and nominal data, and thus were suitable for a wide range of applications.

C4.5 by Ross Quinlan has remained a widely used decision tree algorithm for batch learning—a learning paradigm in which all of the data is available at once. It uses Information Gain as the heuristic for deciding best splits. Information Gain tells us what the relative class purity of two given class distributions is (the idea is that "purer" class distributions contain less information—one needs a shorter message to represent the fact that all instances belong to a class, and a longer one to represent a spread).

As previously mentioned, tree classifiers are grown by "splitting" nodes, starting at an initial node, the root. Each split at a node corresponds to dividing the instance space with decision boundaries that provide an optimal separation of classes based on some convenient measure of separation such as Information Gain or Gini coefficient, creating corresponding child nodes from which the sub-division process may continue.

The class purity of the distribution at a node is compared with the aggregate class purity of the distributions resulting from a split in order to compute Information Gain. The Information Gains due to several attributes being considered at a node are then compared to find the best one. Simplifying Quinlan's terminology and notation from (Quinlan 1992)(p21-22), the information (or entropy) of a class distribution at a node N is $I_N = \sum_{i=1}^{c} -p_i \log p_i$, where p_i is the probability observing class i at node N and c is the number of classes. This characterization encapsulates neatly the notion of a "pure" node having low information—for a one-class distribution, it is easy to observe that information evaluates to 0.

If node N were now split on attribute A leading to j child nodes, the information contained in each of the child nodes is $I_{N_A^j} = \sum_{i=1}^{c} -p_i^{N_A^j} \log p_i^{N_A^j}$, and the Information Gain is given by $\sum_j I_{N_A^j} - I_N$. Finding the attribute A that maximises gain is used as a heuristic in C4.5 and similar algorithms to determine the best split.

Classification and Regression Trees (CART) (Breiman et al. 1984), which use the Gini impurity instead of Information Gain as a splitting heuristic, (Breiman et al. 1984) precede C4.5. In this work, we focus our discussion around Information Gain, though it applies equally to Gini-based trees.

Applying C4.5 or CART to a streaming setting is not straightforward; the main problem is that of anytime prediction. How many examples does one need to see before deciding that one has enough data accumulated to create a reliable model? Do we need separate training, validation and test sets in a streaming scenario? How frequently does one need to update this model in order to optimize prediction accuracy? Should one use a sequence of sliding windows? How does one address concept drift in streams? These are the questions that online decision trees address—a set of questions rather different to those centred around data scarcity that apply to the batch paradigm.

2.2 Online Decision Trees

In learning from potentially infinite data streams, it is imperative that instances are not stored. One potential solution is to learn in multiple passes with sets of stored instances; this raises the question of what the ideal working instance repository size must be—that is, *how* many instances

should be stored at any given time?—a finicky hyperparameter. Such a choice would require a significant space overhead and unduly influence the method's anytime predictions—predictions requested on-demand during the continuous learning process, a standard expectation of online learners.

A one-pass solution, wherein each example is processed exactly once, is ideal for such settings; HoeffdingTree was one of several attempts (Schlimmer and Fisher 1986; Utgoff 1989) to provide a one-pass solution, and the first one-pass learner to provide guarantees on deviation of the tree from the batch tree—the hypothetical tree that would be learned if all infinite examples from a stationary distribution were made available at once. Hoeffding Tree uses a statistical test—the Hoeffding Test (Domingos and Hulten 2000; Hoeffding 1963)—to determine the most appropriate time to split. Its success may be attributed to the fact that it provided both a one-pass solution and deviation guarantees in the same package.

Work on scalability of batch learners also helped set the foundation for one-pass learning in sequential prediction scenarios. Bootstrapped Optimistic Algorithm for Tree construction (BOAT) (Gehrke, Ganti, et al. 1999) represents a typical attempt at learning from a large database that does not use a predictive sequential setting, by sampling fixed size chunks that are used to bootstrap multiple trees. A "coarse" tree is then extracted, based on the overlapping parts of the bootstrapped trees in terms of split decisions; this tree is further refined to produce a final tree by passing the whole dataset over it. The system is "incremental" in the sense that it can process additional datasets; and it is responsive to drift in that the system detects when a new dataset requires a change in split criterion at a node through a global assessment of split criterion, and causes a rebuild of the subtree rooted at that node. While key ideas that shape later trees are developed in this work, the sizes of the initial bootstrap samples are arbitrarily chosen, and concurrently the notion of anytime prediction is not entertained—there is no automated way of determining how many examples suffice to build a first reliable tree. Further, the focus is on minimising utilisation of main memory; it is assumed that the database D is available for a corrective step in the algorithm. On the other hand, Hoeffding Tree is truly one-pass, in that it is assumed that an example is seen only once, then discarded. Meanwhile, the RainForest framework (Gehrke, Ramakrishnan, and Ganti 2000) introduces the idea of storing attribute-value-class counts at nodes, which we see in Hoeffding Tree as *node statistics*.

Hoeffding Tree and Hoeffding Adaptive Tree (HAT) are state-of-the-art one-pass incremental learners for stationary and nonstationary streams respectively; we discuss these methods and briefly outline the primary unspecified effects we observe that influence prequential accuracy performance.

2.2.1 Hoeffding Tree

Hoeffding Tree (Domingos and Hulten 2000) is an online learning strategy that takes as input a stream of instances $(i_1, i_2, ...i_t, i_{t+1}, ...)$ and incrementally builds a decision tree that offers anytime prediction. That is, at any point of time "t", a regression or class value prediction can be made based on the examples in the stream up to that point.

Algorithm 2.1: Hoeffding Tree, Domingos & Hulten (2000) –Reproduced verbatim from original–

Input: S, a sequence of examples,
$\mathbf{X}$, a set of discrete attributes,
G(.), a split evaluation function
δ, one minus the desired probability of choosing the correct attribute at any given node
Output: HT, a decision tree.
begin
  Let HT be a tree with a single leaf l_1 (the *root*).
  Let $\mathbf{X_1} = \mathbf{X} \cup X_\emptyset$.
  Let $\overline{G_1}(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class in S
  foreach *class* y_k **do**
    foreach *value* x_{ij} *of each attribute* $X_i \in \mathbf{X}$ **do**
      Let $n_{ijk}(l_1) = 0$
    end
  end
  foreach *example* $(\mathbf{x}, y)$ *in* S **do**
    Sort $(\mathbf{x}, y)$ into a leaf l using HT
    foreach x_{ij} *in* $\mathbf{x}$ *such that* $X_i \in \mathbf{X}_l$ **do**
      Increment $n_{ijk}(l)$
    end
    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 $\overline{G_l}(X_i)$ for each attribute $X_i \in \mathbf{X}_l - \{X_\emptyset\}$ using the counts $n_{ijk}(l)$
      Let X_a be the attribute with highest $\overline{G_l}$
      Let X_b be the attribute with second-highest $\overline{G_l}$
      Compute ϵ using Equation 1;
      if $\overline{G_l}(X_a) - \overline{G_l}(X_b) > \epsilon$ *and* $X_a \neq X_\emptyset$ **then**
        Replace l by an internal node that splits on X_a **foreach** *branch of the split* **do**
          Add a new leaf l_m and let $\mathbf{X}_m = \mathbf{X} - \{X_\emptyset\}$ Let $\overline{G}_m(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class at l_m **foreach** *class* y_k *and each value* X_{ij} *of each attribute* $X_i \in \mathbf{X}_m - \{X_\emptyset\}]$ **do**
            Let $n_{ijk}(l_m) = 0$
          end
        end
      end
    end
  end
  Return HT
end

The provision of deviation guarantees, and the use of statistical tests (the Hoeffding Test) to make reliable split decisions in a single-pass paradigm made Hoeffding Tree a durable baseline method that won it the KDD Test of Time award in 2015.

Unlike batch decision trees, which process all instances at once, online decision trees need to decide when they are ready to split a node. That is, they need to decide what value of t in the instance stream offers sufficient confidence for a reliable decision boundary to be drawn in input space by splitting a node. Another way of framing this is thus: assuming a stationary distribution, if the entire infinity of instances were filtered down the tree as if in a batch setting, the resulting split is the ideal split—and online decision trees aim to approximate such an ideal split as closely as possible, but because they have to do so in finite time, they need to decide at what time t they have enough confidence that a proposed split for a node matches the "ideal" split.

And so: deciding *when* to split a node is the problem that Hoeffding Tree solves in a principled and reliable manner; the Hoeffding Test is used to determine the likelihood of the true best split—as would be obtained by a hypothetical batch tree able to process all infinite examples at once—varying from the best split being considered by the algorithm at a given time. When a certain preset level of confidence is reached at a node, it is split. Terminals do not have children; they are called "leaf" nodes. Leaf nodes are evaluated for splits that may result in a better model.

In Hoeffding Tree, leaf nodes uniquely serve the purpose of "learning", in the sense of attempting to find a split point based on examples accumulated at the leaf. Once a learning node has been split on, the "split node" that replaces it only serves to filter down examples. Leaf nodes that serve as learning nodes update their statistics—and also make predictions. Learning nodes, in Hoeffding Tree, are always leaf nodes.

Statistics that represent the characteristics of the distribution over data attributes and attribute-values are stored in lieu of the data at each node. These node statistics help determine what child nodes will result in a split on a particular attribute at a learning node. They are used to compute test values for the Hoeffding Test; splits are made when a new division of the space by a certain attribute is ascertained to increase the separation power reliably compared to the next best attribute based on the Hoeffding Test applied to Information Gain, Gini coefficient, or other measure of comparative separation due to the attributes.

The Hoeffding Test uses the Hoeffding bound: for n independent random variables $r_1..r_n$, with range R and mean $\bar{r}$, the Hoeffding bound states that with probability $1-\delta$ the true mean is at least $\bar{r}-\epsilon$ where Domingos and Hulten 2000; Hoeffding 1963:

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

2.2.2 Hoeffding Adaptive Tree

Hoeffding Adaptive Tree (HAT) is an adaptive online decision tree based on Hoeffding Tree. HAT starts growing an alternate subtree when concept drift is detected at a node by ADWIN (Bifet and Gavalda 2007) or another change detector, and replaces the original subtree with an alternate when the error from the alternate is lower than the error from the main subtree. Till date, it remains the best performing single online tree on our expanded concept drift testbench in terms of prequential error; we experimented with several strategies to improve performance, such as allowing alternates to vote as a form of lookahead and weighting examples at subtrees as a form of subtree-level example boosting, only to find that unspecified features already accounted for them to various degrees. In Section 5, we explain what these unspecified features are and how they result in superior performance, isolating strategies that boost prequential accuracy.

3 Experimental Setup

We use the Massive Online Analysis (MOA 2016.04) (Bifet, Holmes, Kirkby, et al. 2010) framework for our experimentation. MOA provides implementations of Hoeffding Tree and Hoeffding Adaptive Tree, and a number of concept drift streams.

We choose a testbench geared towards exposing differences in algorithm behavior in concept-drifting settings. The Hyperplane (Hulten, Spencer, and Domingos 2001) and Radial Basis Function (RBF) (Bifet, Holmes, Pfahringer, et al. 2009) generators have long been used to differentiate the effectiveness of strategies for learning under gradual drift. They generate instances from a naturally evolving concept and allow the parametrization of rate of drift and dimensionality of the stream.

Our Recurrent AbruptDrift generator adds the option of generating recurrent abrupt drifts to the AbruptDrift generator from (Webb et al. 2016). This generator is particularly useful for demonstrating differences in algorithm behavior with respect to increasing dimensionality, as we see in Section 5.2 where a clear difference in prequential accuracy performance is noted between compared strategies with respect to stream dimensionality. The Recurrent AbruptDrift generator models a full conditional probability table for the target distribution $p_{Y|X}$ that grows exponentially in the number of input variables, which are relatively independent.

The Agrawal (Agrawal et al. 1992), LED (Breiman et al. 1984), RandomTree (Bifet, Holmes, Kirkby, et al. 2010), SEA (Street and Kim 2001), STAGGER (Schlimmer and Granger 1986), and Waveform (Breiman et al. 1984) generators are commonly used in studies of concept drift; as they are not based on an inherently evolving distribution, we use the RecurrentConceptDriftStream generator (Bifet, Holmes, Kirkby, et al. 2010) to generate drift between different parametrizations of these streams.

The parametrized synthetic streams are listed in Table 1. The suffix notation in the shorthand "recurrent—abrupt—522" used in tables conveys that that particular stream has 5 classes, 2 nominal attributes, and 2 values per attribute.

Table 1 Synthetic Datasets

	MOA Stream	Shorthand
1	-s (RecurrentConceptDriftStream -x 200000 -y 200000 -z 100 -s (generators.AgrawalGenerator -f 2 -i 2) -d (generators.AgrawalGenerator -f 3 -i 3))	recurrent—agrawal
2	-s (RecurrentConceptDriftStream -x 200000 -y 200000 -z 100 -s (generators.LEDGenerator -i 2) -d (generators.LEDGeneratorDrift -i 3 -d 7))	recurrent—led
3	-s (RecurrentConceptDriftStream -x 200000 -y 200000 -z 100 -s (generators.RandomTreeGenerator -r 1 -i 1) -d (generators.RandomTreeGenerator -r 2 -i 2))	recurrent—randomtree
4	-s (RecurrentConceptDriftStream -x 200000 -y 200000 -z 100 -s (generators.SEAGenerator -f 2 -i 2) -d (generators.SEAGenerator -f 3 -i 3))	recurrent—sea
5	-s (RecurrentConceptDriftStream -x 200000 -y 200000 -z 100 -s (generators.STAGGERGenerator -i 2 -f 2) -d (generators.STAGGERGenerator -i 3 -f 3))	recurrent—stagger
6	-s (RecurrentConceptDriftStream -x 200000 -y 200000 -z 100 -s (generators.WaveformGenerator -i 2 -n) -d (generators.WaveformGeneratorDrift -i 3 -d 40 -n))	recurrent—waveform
7	-s (generators.HyperplaneGenerator -k 10 -t 0.0001 -i 2)	hyperplane—1
8	-s (generators.HyperplaneGenerator -k 10 -t 0.001 -i 2)	hyperplane—2
9	-s (generators.HyperplaneGenerator -k 5 -t 0.0001 -i 2)	hyperplane—3
10	-s (generators.HyperplaneGenerator -k 5 -t 0.001 -i 2)	hyperplane—4
11	-s (generators.RandomRBFGeneratorDrift -s 0.0001 -k 10 -i 2 -r 2)	rbf—drift-1
12	-s (generators.RandomRBFGeneratorDrift -s 0.0001 -k 50 -i 2 -r 2)	rbf—drift-2
13	-s (generators.RandomRBFGeneratorDrift -s 0.001 -k 10 -i 2 -r 2)	rbf—drift-3
14	-s (generators.RandomRBFGeneratorDrift -s 0.001 -k 50 -i 2 -r 2)	rbf—drift-4
15	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 2 -n 2 -v 2 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—222
16	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 3 -n 2 -v 2 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—322
17	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 3 -n 3 -v 2 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—332
18	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 3 -n 3 -v 3 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—333
19	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 3 -n 3 -v 4 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—334
20	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 3 -n 3 -v 5 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—335
21	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 4 -n 2 -v 2 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—422
22	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 4 -n 4 -v 4 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—444
23	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 5 -n 2 -v 2 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—522
24	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 5 -n 5 -v 5 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—555

We use predictive sequential accuracy—prequential accuracy—as our performance measure. This is the cumulative accuracy from a setting where a prediction is offered for each instance

by the learner, and the true value is made available following the prediction. We focus on the classification problem.

All streams generate one million examples. Each stream generator is run with 10 differently initialized random seeds. Because prequential accuracy is a curve that plots error at every timestep, the results are reported thus: a measure of the mean error is obtained by averaging error across seeded runs for each epoch, and then averaging over the epoch-wise error averages. A measure of variance is obtained by computing the variance of the error across the seeded runs for each epoch, then averaging the epoch-wise variances. The error reported for each epoch is itself the average across 1000 examples, which is the default for the evaluator in MOA. The number of leaves, and the CPU time, are simply the averages of the final values across the seeded runs for a given stream.

As a base for our experimentation we use implementations of VFDT and HAT stripped of the unspecified features we have found in the MOA and VFML implementations. All references to "VFDT" or "HAT" in our charts and tables refer to an implementation of the core Hoeffding Tree (Algorithm 2.1) as described in (Domingos and Hulten 2000) and the Hoeffding Adpaptive Tree algorithm (Bifet and Gavaldà 2009) without any of the unspecified features that have entered the implementations of VFDT and HAT in the form of engineering artifacts.

To these base implementations, we have added as options each of the unspecified features we identified in the MOA implementations. In the following sections, we systematically investigate the impact of identified unspecified features on prequential accuracy.

4 Unspecified and Emergent Behaviors in the MOA implementation of Hoeffding Tree

An online strategy designed for concept drift adapts to the stream in order to predict as well as possible as concepts change; an online strategy designed for learning from stationary streams assumes an overall unchanging distribution, and prioritizes minimizing deviation from an ideal batch learner that has access to the entire stream of infinite size at once. A learner designed for a stationary stream can be expected to be stable on the stability-plasticity (Grossberg 1988; Hoens, Polikar, and Chawla 2012) spectrum, as it aims to mitigate a reactive change in the model on exposure to noise so it can capture the unchanging concept.

HoeffdingTree—the state-of-the-art online learning tree that has served as a base procedure for the development of methods for learning from online streams with concept drift—is surprisingly responsive to abrupt drifts, in spite of having been designed for learning from stationary streams and for very high stability. Abrupt drifts are large shifts in concept over a relatively short period of time (Gama, Sebastião, and Rodrigues 2013; Webb et al. 2016)—in our experimental setup, abrupt drifts are considered instantaneous, with the data-generating distribution ("concept") switching instantly to a new one.

Closer inspection reveals several factors promoting a form of response to drift that we call *amnesia*. By *amnesia* we refer to the ability of a learner to update its model to better capture the present state of the stream by diminishing the influence of earlier examples—effectively, forgetting them to some degree.

One of the factors leading to the excellent drift response of Hoeffding Tree in any reasonable implementation is inadvertent inbuilt amnesia—as instances are not stored, and there is no effective way to exactly determine node statistics in child nodes resulting from a split, node statistics are simply initialised to zero for convenience, effectively deleting history upon splitting. We elaborate on this in Section 4.1.

A second factor that is unique to the MOA interpretation of Hoeffding Tree is that it was possible to re-split on a used nominal attribute, again contributing to amnesia when it was most useful. We explain this in greater depth in Section 4.2.

A further simplification, which we explain in Section 4.3, in the MOA implementation—one which contributes to time efficiency without changing time complexity or producing a discernible change in prequential accuracy—is that Information Gains are not averaged; periodic infogain computations are considered good enough approximations—as they indeed turn out to be in practice. This is a significant optimization that could be written into other HoeffdingTree implementations.

Put together, these factors lead to a significance performance gain over a base Very Fast Decision Tree (VFDT), the implementation of the theoretical Hoeffding Tree construct. We use the MOA implementation, though some significant factors are also part of the original VFML implementation. (Section 4.5).

4.1 Inherent Amnesia

HoeffdingTree, as a stream learner, is designed to operate with a low, finite memory requirement. Examples are not stored, but statistics about examples are at each node, "node statistics" that help determine the resultant class distributions when a chosen attribute is split upon. Node statistics are simply representations of the data collected at the node—frequency counts for nominal attribute values, and appropriate statistics such as mean for real attribute values. This allows determination of the best split attribute—one can choose the attribute that results in maximal class separation.

A critical side effect of the usage of node statistics is that freshly created nodes have to start off with *no* node statistics, as there does not appear to be an obviously sound way of redistributing node statistics from the parent without storing examples—one needs the examples in order to be able to determine an appropriate redistribution of node statistics (by filtering examples down the tree).

Though Hoeffding Tree was designed for stationary streams, the consequence of new nodes being created without statistics turns out to be a significant unplanned advantage in the scenario with concept drift; a tree algorithm may "recover" from a concept drift (that is, update its model to be current) simply by splitting, leading to amnesia that erases the previous concept—thus modeling the novel concept better than expected after a concept drift.

For example: in an extreme case, suppose previously that all 1000 examples that reached a leaf belonged to class A. After drift, all examples reaching the leaf belong to class B. Simply by splitting on a now irrelevant attribute, the tree can forget the 1000 examples of class A in the node statistics. A further split may be justified based on the new separable node statistics, and this would render a child with a class distribution derived from these fresh node statistics that classify concept B well.

In order to demonstrate this effect, we built a version of VFDT that does not forget instances, so that the node statistics n_{ijk} in the children are initialized to counts from all the examples i in the stream to date that filter into the freshly created child nodes, instead of being set to zero.

We then set up a simple noise-free test stream with drift in the conditional distribution P(Y|X) with 5 attributes, 5 nominal values per attribute and 5 classes. The stream is from (Webb et al. 2016) and is initialized thus: first, a "starting" distribution is created. A random distribution over the covariates is created by drawing attribute-value probabilities from a Gamma(1,1) function and normalizing them per attribute (every attribute gets a value in every instance); and every possible combination of values of the covariates is assigned a random class by drawing an index

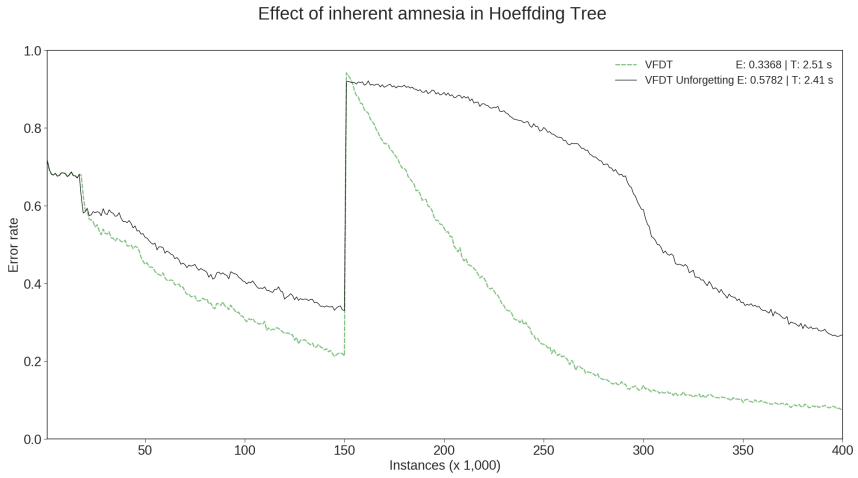


Fig. 1 Plot showing a significant improvement in drift response resulting from amnesia inherent to HoeffdingTree. A single abrupt drift occurs at $t = 150,000$. VFDT shows a far better recovery in prequential error following the drift. 'E' and 'T' are error and time, respectively, averaged every 1000 steps over 10 randomised streams.

from a uniform distribution. Next, a second, "final" distribution after drift is created by randomly changing the class assignment, ensuring we pick a class that is not the one that is already chosen. The proportion of classes to be changed is provided as an input for drift magnitude.

Figure 1 plots the prequential accuracies of unmodified Hoeffding Tree and an "eidetic" Hoeffding Tree that retains instances for initialization of the node statistics n_{ijk} on a synthetic stream with 5 classes, 5 nominal attributes, and 5 values per attribute. An abrupt drift occurs at $t = 150,000$ with the highest possible magnitude of 1.0. The figure shows that due to the amnesia inherent in HoeffdingTree, response to drift in terms of recovery in error rate following the abrupt drift is far quicker than it is without inherent amnesia.

4.2 MOA implementation: Resplitting on attributes

If a nominal attribute that has already been used to split on a decision path is reused to split, we should achieve no advantage at all and only the disadvantage of making the tree larger. Because the value for the previously used nominal attribute has already been decided the first time it was split upon, "splitting" on it again will make no difference whatsoever to the sequence of decisions down the path, and it will in no way discriminate any set of instances accumulated at the node—all instances will contain the same value of the reused attribute! As a result, all examples will pass down the same branch below the split—hence the distribution at the new leaf should be identical to that at the current one.

Thus, when a leaf node evaluates multiple split options, it should normally be the case that a previously used nominal attribute will offer zero information gain—it has already been conditioned upon and hence will all have the same value.

However, VFDT was designed for stationary streams; at least in the MOA implementation, the introduction of concept drift leads to splits occuring when they should not. Previously used attributes are enabled for reuse—that is, the implementation interpretation allows re-"splitting" on the same attribute—with the same previously used attribute-value being assigned to the split

(as the attribute has already taken it's value, and with any other value an instance would end up elsewhere in the tree).

This re-"splitting" behavior has not been found to manifest in drift-free streams. Concept drift triggers it (Table 2 provides an example), and the end result is very interesting—amnesia. In the freshly created leaf node on a resplit on a previously used attribute, node statistics are zeroed as a consequence of the effect described in Section 4.1; this amnesia helps concept drift adaptation, again, by allowing the tree to model a newer concept at any future leaves without the weight of the previous concept as the class distribution used to predict is derived from the parent's node statistics, which have been cleared. The interaction between inherent amnesia and concept-drift triggered "resplitting" leads over all to a better drift response as can be seen in Table 3, which shows an overwhelming win rate on our testbench for VFDT that allows resplitting over our base VFDT that does not (p-value < 0.00001). The number of wins with the resplitting strategy over without it is significant in itself, but further consequence is found in how this interacts with other unspecified features, as demonstrated in Section 4.5.

Table 2 A previously used attribute is "reused", effectively creating a single child with a clean node statistics register, but carrying over the class distribution. This table shows how the behavior can be simulated, and what the results are. The only change to the original code is a clause printing the tree before and after a resplit occurs. The tree as it stands after the first resplit is shown below.

Code used to display behavior

```
// if shouldSplit is True

if(parent != null && splitDecision.splitTest.getAttsTestDependsOn()[0]
        == parent.splitTest.getAttsTestDependsOn()[0]) {
                StringBuilder treeDescription = new StringBuilder();
                getModelDescription(treeDescription, 4);
                System.out.println(treeDescription);
        }
```

MOA command used to trigger behavior

```
java -cp "commons-math3-3.6.1.jar:guava-22.0.jar:moa.jar:mytrees.jar" "moa.DoTask" "EvaluatePrequential -l
trees.HoeffdingTreeOriginal -s (ConceptDriftStream -s (generators.RandomTreeGenerator -o 2 -u 0 -v 2 -d 2 -l 1)
-d (generators.RandomTreeGenerator -r 2 -i 2 -o 2 -u 0 -v 2 -d 2 -l 1) -p 200000 -w 1 -r 4) -i 400000 -f 1000"
```

Resplitting on same attribute behavior (in bold)

```
if [att 1:nominal1] = val 1:value1:
    if [att 2:nominal2] = val 1:value1:
        Leaf [class:class] = <l class 1:class1> weights: 49,391—0
    if [att 2:nominal2] = val 2:value2:
        Leaf [class:class] = <class 2:class2> weights: 0—49,243
if [att 1:nominal1] = val 2:value2:
    if [att 2:nominal2] = val 1:value1:
        if [att 2:nominal2] = val 1:value1:
            Leaf [class:class] = <class 2:class2> weights: 0—208
    if [att 2:nominal2] = val 2:value2:
        Leaf [class:class] = <class 1:class1> weights: 49,411—253
```

Table 3 Performance of the "resplitting" strategy: A previously used attribute is "reused", effectively creating a single child with a clean node statistics register

Streams	VFDT			VFDT with resplitting strategy		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.20846	0.00043	482	**0.19894**	0.00028	788
recurrent—led	**0.33838**	0.00068	54	0.33906	0.00065	56
recurrent—randomtree	0.22404	0.00231	1155	**0.22151**	0.00211	1292
recurrent—sea	0.15251	0.00016	164	**0.15239**	0.00015	360
recurrent—stagger	0.1882	0.00047	23	**0.00699**	5e-05	162
recurrent—waveform	0.19355	0.00171	147	**0.19303**	0.0016	153
hyperplane—1	0.11566	0.00021	285	**0.11455**	0.00019	350
hyperplane—2	0.16785	0.00117	333	**0.16458**	0.00097	395
hyperplane—3	0.1074	0.00013	284	**0.10714**	0.00013	340
hyperplane—4	0.16309	0.00359	316	**0.16005**	0.00301	376
rbf—drift-1	0.11462	0.00053	397	**0.11381**	0.00052	423
rbf—drift-2	0.2858	0.00155	482	**0.28486**	0.00156	491
rbf—drift-3	0.13821	0.00068	365	**0.1377**	0.00066	390
rbf—drift-4	0.40874	0.00141	288	**0.40583**	0.0015	363
recurrent—abrupt—222	0.35403	0.02056	4	**0.00261**	8e-05	17
recurrent—abrupt—322	0.37862	0.01248	4	**0.00325**	7e-05	16
recurrent—abrupt—332	0.3504	0.08913	8	**0.01316**	0.00035	32
recurrent—abrupt—333	0.36505	0.01499	27	**0.0582**	0.00128	134
recurrent—abrupt—334	0.39687	0.00217	64	**0.15073**	0.00525	278
recurrent—abrupt—335	0.39622	0.00319	121	**0.23007**	0.00768	481
recurrent—abrupt—422	0.33416	0.02931	4	**0.00959**	0.00028	15
recurrent—abrupt—444	0.40671	0.00636	235	**0.34738**	0.00684	380
recurrent—abrupt—522	0.3309	0.03999	4	**0.00937**	3e-04	14
recurrent—abrupt—555	0.46461	0.00584	1405	**0.46352**	0.00568	1417
Unique Wins	**1**			**23**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: < 0.00001**	**Confidence Interval: 0.81711 — 1**

As "re-splitting" proves to be a powerful strategy for responding to concept drift, it might be instructive to experiment with variations of initializing/zeroing node statistics when a drift is detected or suspected in adaptive methods designed specifically for concept drift. We provide an example of doing so in Section 4.6.

4.3 Infogain approximations

Hoeffding tree is premised upon offering rationalised splits in streaming scenarios using the Hoeffding Inequality as a test to determine whether a potential split is likely to be the long-term choice in a theoretical batch tree built on an infinite stationary stream. Hoeffding test is applied with the computed prospective infogain differences between the top two attributes due to a potential split at timestep t, ΔG_t, as the random variables.

The standard use of Hoeffding's inequality would involve taking the mean of all computed infogain differences ΔG_t over all timesteps (Domingos and Hulten 2000; Hoeffding 1963). That is, every time the infogain difference ΔG_t is computed, it is fed into the mean value $\overline{\Delta G}_t$ comprising all ΔG_t up until that timestep—the value used in the Hoeffding Test to determine the relative superiority of an attribute. (We use timestep subscripts to clarify the meanings of the random variables. Random variables are often assumed to be "taking values in sequence"; this is a misconception. A random variable is machinery to represent uncertainty about a particular event, and multiple events correspond to different random variables, even if drawn from the same distribution—the random variable does not take "one value after another").

In practice, for the sake of efficiency, infogain is only computed at set intervals, such as once every 200 timesteps. This implies, for example, that over 10 invocations of the $\overline{\Delta G}_t$ computations over $2,000$ timesteps, one would only have $n = 10$ of the random variables $\overline{\Delta G}_t$ to average. Having

fewer $\overline{\Delta G}_t$ to average, would mean a potentially longer wait for VFDT for the Hoeffding Test to become significant.

Both the MOA implementation (Bifet, Holmes, Kirkby, et al. 2010) and the original Very Fast Machine Learning (VFML) implementation (Domingos and Hulten 2000) take n to be the number of examples (not the number of $\overline{\Delta G}_t$ computations), and use the most recently computed ΔG_t divided by n as a proxy for the average $\overline{\Delta G}_t$, at variance with the published algorithm in (Domingos and Hulten 2000).

In the stationary case, this is perfectly reasonable to do. After all, we do expect $\overline{\Delta G}_t$ to converge in the long run, and we do not expect unusually great fluctuations after having observed each instance. Therefore, we benefit from quicker decisions in building tree structure as n grows large quickly in step with the number of examples and not in step with infogain computations.

In the drifting scenario, the larger n implies that the much more variable instantaneous infogain computations would be accepted faster than theoretically expected by the algorithm in situations where drift induces the gap between the best candidate split attribute and the second best one to widen. In contrast, averaging would slow down response as one uses a "fuzzy" average of potentially unhelpful gains over a small n. However, this strategy does not appear to impact VFDT (though we will see in Section 5.7).Table 4 shows us how the unspecified strategy of infogain approximation fares in terms of prequential accuracy—with 11 wins—to 12 when approximation is not used. Thus there is no evidence to support infogain approximation having a conclusive effect (p-value 0.5). Therefore, it may be a valid strategy to continue to use with VFDT for the sake of efficiency even in the drifting scenario.

Table 4 Performance of the infogain approximation strategy

Streams	**VFDT without infogain approximation**			**VFDT with infogain approximation**		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.20846	0.00043	482	**0.20774**	0.00055	466
recurrent—led	**0.33838**	0.00068	54	0.34105	0.00071	48
recurrent—randomtree	0.22404	0.00231	1155	**0.22367**	0.00274	1221
recurrent—sea	0.15251	0.00016	164	**0.15228**	0.00016	163
recurrent—stagger	***0.1882***	0.00047	23	***0.1882***	0.00047	23
recurrent—waveform	**0.19355**	0.00171	147	0.19395	0.00174	147
hyperplane—1	**0.11566**	0.00021	285	0.11576	0.00021	288
hyperplane—2	**0.16785**	0.00117	333	0.1681	0.00119	342
hyperplane—3	**0.1074**	0.00013	284	0.10745	0.00013	286
hyperplane—4	0.16309	0.00359	316	**0.16251**	0.00348	322
rbf—drift-1	0.11462	0.00053	397	**0.11446**	0.00055	404
rbf—drift-2	0.2858	0.00155	482	**0.28565**	0.00147	489
rbf—drift-3	**0.13821**	0.00068	365	0.13867	0.00065	363
rbf—drift-4	**0.40874**	0.00141	288	0.40887	0.00141	287
recurrent—abrupt—222	0.35403	0.02056	4	**0.35402**	0.02053	4
recurrent—abrupt—322	***0.37862***	0.01248	4	***0.37862***	0.01246	4
recurrent—abrupt—332	**0.3504**	0.08913	8	0.35104	0.08916	8
recurrent—abrupt—333	0.36505	0.01499	27	**0.36503**	0.01453	27
recurrent—abrupt—334	0.39687	0.00217	64	**0.39678**	0.00238	64
recurrent—abrupt—335	**0.39622**	0.00319	121	0.39635	0.00277	121
recurrent—abrupt—422	***0.33416***	0.02931	4	***0.33416***	0.02949	4
recurrent—abrupt—444	0.40671	0.00636	235	**0.40632**	0.00827	236
recurrent—abrupt—522	**0.3309**	0.03999	4	0.3312	0.03932	4
recurrent—abrupt—555	0.46461	0.00584	1405	**0.46106**	0.00601	1505
Unique Wins	**10**			**11**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.5**	**Confidence Interval: 0.32811 — 1**

4.4 Using number of instances seen at leaves instead of weight seen at leaves to determine split evaluation

Each instance is assumed to be labeled with a class label i, and a separate count c_i is maintained for each class label at each node, leading to a class distribution.

When a split occurs on a node, child nodes are initialised with a class distribution Y derived from that of the parent node—the parent class counts c_i are distributed among the child nodes at their inception. Thus child leaf nodes are initialised with a class distribution that is useful for making predictions from the point they are created.

Because of the class weight due to initialisation, the class weight $\sum c_i$ for classes i seen at leaves is generally larger than the actual number of instances seen at the leaf. Thus if the summed class weight $\sum c_i$ is used as a proxy for the actual number of instances N seen at the leaf, we will generally end up using a larger value than the actual number of instances seen at the leaf. In other words, the aggregate of the counts of the classes should add up to the number of instances observed (assuming every instance is labeled)—*plus* the counts carried over from the parents at the time of the split.

Now, the application of the Hoeffding Test at a node is based on the number of examples seen at a node. Because $\sum c_i$ is generally larger than N, using $\sum c_i$ for the Hoeffding Test in lieu of N helps increase test confidence faster by sending the signal that a larger number of examples than actually observed have been observed. Because the newly created leaf already has node statistics n_{ijk} set to zero, it should respond more easily to change by splitting. Weighted examples may also arise from streams or learners.

Table 5 shows us how this affects drift streams. The effect is minor and we do not find it particularly significant (note the number of streams with identical prequential error and p-value of 0.38721).

Table 5 Using weightSeen instead of number of instances

Streams	**VFDT with number of instances**			**VFDT with weightSeen**		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.20846	0.00043	482	**0.2078**	0.00045	473
recurrent—led	***0.33838***	0.00068	54	***0.33838***	0.00068	54
recurrent—randomtree	0.22404	0.00231	1155	**0.22403**	0.00229	1155
recurrent—sea	**0.15251**	0.00016	164	0.15253	0.00016	162
recurrent—stagger	***0.1882***	0.00047	23	***0.1882***	0.00047	23
recurrent—waveform	0.19355	0.00171	147	**0.19344**	0.0017	147
hyperplane—1	0.11566	0.00021	285	**0.11565**	0.00021	285
hyperplane—2	**0.16785**	0.00117	333	0.16801	0.00117	332
hyperplane—3	**0.1074**	0.00013	284	0.10744	0.00013	283
hyperplane—4	0.16309	0.00359	316	**0.16287**	0.00358	316
rbf—drift-1	0.11462	0.00053	397	**0.11458**	0.00053	399
rbf—drift-2	**0.2858**	0.00155	482	0.2862	0.00156	475
rbf—drift-3	**0.13821**	0.00068	365	0.13852	0.00068	365
rbf—drift-4	0.40874	0.00141	288	**0.40853**	0.00142	289
recurrent—abrupt—222	***0.35403***	0.02056	4	***0.35403***	0.02056	4
recurrent—abrupt—322	***0.37862***	0.01248	4	***0.37862***	0.01248	4
recurrent—abrupt—332	***0.3504***	0.08913	8	***0.3504***	0.08913	8
recurrent—abrupt—333	***0.36505***	0.01499	27	***0.36505***	0.01499	27
recurrent—abrupt—334	***0.39687***	0.00217	64	***0.39687***	0.00217	64
recurrent—abrupt—335	***0.39622***	0.00319	121	***0.39622***	0.00319	121
recurrent—abrupt—422	***0.33416***	0.02931	4	***0.33416***	0.02931	4
recurrent—abrupt—444	***0.40671***	0.00636	235	***0.40671***	0.00636	235
recurrent—abrupt—522	***0.3309***	0.03999	4	***0.3309***	0.03999	4
recurrent—abrupt—555	***0.46461***	0.00584	1405	***0.46461***	0.00584	1405
Unique Wins	**5**			**7**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.38721**	**Confidence Interval: 0.31524 — 1**

4.5 Putting it all together

Individually, the unspecified strategies from Sections 4.1,4.2, 4.3, 4.4 sometimes lead to significant performance increases, and are sometimes inconclusive; put together, a broad "win" is obtained for our experiments on this broad, standard concept drift testbench. Table 6 shows us the combined winning effect of these unspecified features for VFDT in the case of streams with concept drift (p-value 0.00002).

Table 6 All three unspecified strategies put together

Streams	**VFDT**			**VFDT: unspecified strategies**		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.20846	0.00043	482	**0.1977**	0.00034	822
recurrent—led	**0.33838**	0.00068	54	0.34153	0.00067	49
recurrent—randomtree	0.22404	0.00231	1155	**0.22123**	0.0023	1351
recurrent—sea	0.15251	0.00016	164	**0.15237**	0.00015	361
recurrent—stagger	0.1882	0.00047	23	**0.00699**	5e-05	162
recurrent—waveform	**0.19355**	0.00171	147	0.19366	0.00169	151
hyperplane—1	0.11566	0.00021	285	**0.11464**	0.00019	354
hyperplane—2	0.16785	0.00117	333	**0.16436**	0.001	404
hyperplane—3	0.1074	0.00013	284	**0.10723**	0.00013	342
hyperplane—4	0.16309	0.00359	316	**0.15871**	0.00289	391
rbf—drift-1	0.11462	0.00053	397	**0.11331**	0.00053	434
rbf—drift-2	0.2858	0.00155	482	**0.28356**	0.00147	503
rbf—drift-3	0.13821	0.00068	365	**0.13803**	0.00064	392
rbf—drift-4	0.40874	0.00141	288	**0.40593**	0.0015	361
recurrent—abrupt—222	0.35403	0.02056	4	**0.00263**	8e-05	17
recurrent—abrupt—322	0.37862	0.01248	4	**0.00328**	9e-05	16
recurrent—abrupt—332	0.3504	0.08913	8	**0.01349**	0.00044	32
recurrent—abrupt—333	0.36505	0.01499	27	**0.05837**	0.00124	134
recurrent—abrupt—334	0.39687	0.00217	64	**0.15038**	0.00529	278
recurrent—abrupt—335	0.39622	0.00319	121	**0.23056**	0.00754	480
recurrent—abrupt—422	0.33416	0.02931	4	**0.0096**	0.00027	15
recurrent—abrupt—444	0.40671	0.00636	235	**0.34814**	0.00749	382
recurrent—abrupt—522	0.3309	0.03999	4	**0.00948**	0.00032	14
recurrent—abrupt—555	0.46461	0.00584	1405	**0.4605**	0.00586	1518
Unique Wins	**2**			**22**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics | **p-value: 2e-05** | **Confidence Interval: 0.7602 — 1**

4.6 The Node Evisceration Strategy

Given the success of the unspecified inadvertent strategy of resplitting described in Section 4.2, we experimented with a deliberate policy of eviscerating nodes instead, that is, when a previously used attribute resurfaced as the best, we cleared both node statistics and the class distribution within the node instead of allowing redundant "re-splitting" of the node on a used attribute. Table 7 shows us that this deliberate policy works out to be immensely successful and would be worth trying for users of online decision trees (p-value 0.00002).

Table 7 Clearing node statistics and Class distributions

Streams	VFDT			VFDT with deliberate node clearing		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.20846	0.00043	482	**0.17856**	5e-04	564
recurrent—led	**0.33838**	0.00068	54	0.33916	0.00066	53
recurrent—randomtree	0.22404	0.00231	1155	**0.22004**	0.00209	1196
recurrent—sea	0.15251	0.00016	164	**0.14994**	0.00015	258
recurrent—stagger	0.1882	0.00047	23	**0.00435**	6e-05	23
recurrent—waveform	0.19355	0.00171	147	**0.19323**	0.00171	142
hyperplane—1	0.11566	0.00021	285	**0.11429**	0.00019	298
hyperplane—2	0.16785	0.00117	333	**0.16007**	0.00089	329
hyperplane—3	0.1074	0.00013	284	**0.10719**	0.00013	297
hyperplane—4	0.16309	0.00359	316	**0.15347**	0.00237	314
rbf—drift-1	0.11462	0.00053	397	**0.11451**	0.00053	394
rbf—drift-2	0.2858	0.00155	482	**0.28429**	0.00154	476
rbf—drift-3	**0.13821**	0.00068	365	0.13853	0.00069	362
rbf—drift-4	0.40874	0.00141	288	**0.40525**	0.00154	316
recurrent—abrupt—222	0.35403	0.02056	4	**0.00233**	7e-05	4
recurrent—abrupt—322	0.37862	0.01248	4	**0.00344**	7e-05	4
recurrent—abrupt—332	0.3504	0.08913	8	**0.01317**	0.00033	8
recurrent—abrupt—333	0.36505	0.01499	27	**0.05836**	0.00124	27
recurrent—abrupt—334	0.39687	0.00217	64	**0.15157**	0.00543	64
recurrent—abrupt—335	0.39622	0.00319	121	**0.23064**	0.00776	121
recurrent—abrupt—422	0.33416	0.02931	4	**0.00933**	0.00025	4
recurrent—abrupt—444	0.40671	0.00636	235	**0.34795**	0.00688	235
recurrent—abrupt—522	0.3309	0.03999	4	**0.00972**	0.00031	4
recurrent—abrupt—555	0.46461	0.00584	1405	**0.46356**	0.00568	1405
Unique Wins	**2**			**22**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 2e-05**	**Confidence Interval: 0.7602 — 1**

5 Unspecified and Emergent Behaviors in the MOA implementation of Hoeffding Adaptive Tree

We also investigate the success of Hoeffding Adaptive Tree (HAT)—the state-of-the-art adaptive tree learner—for learning under concept drift. We show that in addition to its main proposed strategy of building alternate subtrees upon drift detection and replacing mainline subtrees with alternates with lower error, HAT's excellent performance is influenced, to various degrees, by factors such as:

- Alternate voting, explained in Sections 5.2, 5.3, and 5.4
- Partial weighting of examples at leaves, explained in Section 5.5
- Effects from the VFDT base—explained in Sections 5.1, 5.7, 5.11

We elaborate on these mechanisms, perform comparisons with and without unspecified strategies enabled, and show the effects each strategy has on prequential error. We also describe unspecified features that do not result in significant effects.

5.1 Resplitting on nominal attributes

This unspecified feature of MOA-VFDT is responsible for a significant performance gain for HAT. As explained in Section 4.3, nominal attributes that have been used once to split are "reused", in that a single child is created if a previously used attribute leads to greater Information Gain than any other attribute under drifting streams. This turns out to be a particularly effective performance booster, as it forces the creation of a fresh leaf with empty node statistics, implying that every iteration of self-splitting leads to telescoping amnesia in the node statistics. We need

the node statistics to compute Information Gain; when these are set to zero, we are effectively working with a blank slate, having deleted the historical node statistics that would otherwise retain information from older concepts. In addition, further splits will also have a predictor—a class distribution—derived solely from the freshly cleared node statistics of the parent that represent the post-drift concept. This enables learning a newer concept rapidly and is useful particularly in concept drift settings, more so for HAT than for VFDT, as is clear from our results (Table 8, p-value 0.00024). This attests to the complexity that simple design decisions may produce in conjunction, leading to significant effects on algorithm performance—even in such simple artifacts as decision trees.

Table 8 Performance of the resplitting on nominal attributes strategy

Streams	**HAT that does not "resplit" on nominal attributes**			**HAT that "resplits" on nominal attributes**		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.13645	0.00037	14	**0.12454**	0.0011	92
recurrent—led	0.26767	0.00085	12	**0.26761**	0.00084	11
recurrent—randomtree	0.10104	0.00269	250	**0.09866**	0.00282	281
recurrent—sea	0.12579	0.00014	5	**0.11207**	0.00011	74
recurrent—stagger	0.00215	3e-05	19	**0.00213**	5e-05	18
recurrent—waveform	0.17867	0.00031	33	**0.17756**	3e-04	41
hyperplane—1	**0.10801**	0.00017	84	0.10875	0.00016	137
hyperplane—2	0.11934	3e-04	23	**0.11823**	0.00028	47
hyperplane—3	**0.10588**	0.00014	98	0.10693	0.00015	199
hyperplane—4	0.11254	4e-04	10	**0.11008**	4e-04	29
rbf—drift-1	0.13508	0.00083	47	**0.1197**	5e-04	66
rbf—drift-2	0.19549	0.00155	14	**0.19199**	0.00143	13
rbf—drift-3	0.16636	0.00107	46	**0.15267**	0.00087	97
rbf—drift-4	0.33926	0.00334	3	**0.33877**	0.00337	2
recurrent—abrupt—222	0.00185	9e-05	3	**0.00085**	4e-05	4
recurrent—abrupt—322	0.0016	0.00013	3	**0.00087**	6e-05	5
recurrent—abrupt—332	0.00252	0.00037	6	**0.00169**	0.00019	6
recurrent—abrupt—333	0.00425	0.00019	23	**0.00404**	0.00018	23
recurrent—abrupt—334	**0.01212**	0.00052	58	0.01228	0.00054	58
recurrent—abrupt—335	0.01687	0.00147	111	**0.01675**	0.00148	111
recurrent—abrupt—422	0.00177	0.00011	3	**0.00095**	4e-05	4
recurrent—abrupt—444	0.06662	0.00297	171	**0.06643**	0.00298	171
recurrent—abrupt—522	0.00177	0.00012	4	**0.00113**	9e-05	4
recurrent—abrupt—555	***0.34892***	0.00896	329	***0.34892***	0.00896	329
Unique Wins	**3**			**20**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.00024**	**Confidence Interval: 0.69636 — 1**

5.2 Alternate Voting

HAT makes use of change detectors at each node to determine whether a concept drift has occurred. Upon detecting a concept drift, an alternate subtree is grown. After a set period, the alternate subtree is made eligible to replace the mainline subtree if it has lower prequential error. Alternates are only specified to be potential replacements, not to vote. After all, it is possible that an alternate is grown as a result of a false drift detection due to noise.

However, should alternates be allowed to vote, they can represent a lookahead ability; while the model currently holds on to an older concept due to underlying statistical considerations of the HoeffdingTree base, exploratory submodels that catch on to fresh concepts may represent the best future state of the tree in concordance with ongoing concept drift. Giving these alternates a say in model predictions would be dependent on the scenario: it is a tradeoff between identifying

subtrees that represent new concepts early and allowing in predictions from subtrees built as a result of noise.

Alternates voting was one of the unspecified features in the original implementation of HAT. We have distilled this feature in order to study it in isolation.

Table 9 compares a baseline, standardized HAT (without the HoeffdingTree unspecified effects described in Sections 4.2, 4.3, and 4.4 or the other unspecified HAT features described in this section), with a version of HAT where only a single alternate is grown and *allowed to vote*.

Table 9 Performance of the single alternate voting strategy: a single alternate may be grown and allowed to vote.
While the base algorithm makes no provision for alternates voting, allowing them to do so provides the system with a form of lookahead wherein alternate subtrees under development—concepts being developed as potential future replacements—begin to take part in providing predictions instead of waiting until a replacement has taken place to start doing so.

Streams	**HAT**			**HAT with a single voting alternate**		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.13645	0.00037	14	**0.13643**	0.00036	14
recurrent—led	0.26767	0.00085	12	**0.26669**	8e-04	12
recurrent—randomtree	**0.10104**	0.00269	250	0.10615	0.00264	250
recurrent—sea	0.12579	0.00014	5	**0.12538**	0.00014	5
recurrent—stagger	0.00215	3e-05	19	**0.00136**	0	19
recurrent—waveform	0.17867	0.00031	33	**0.17789**	0.00026	33
hyperplane—1	0.10801	0.00017	84	**0.10576**	0.00015	84
hyperplane—2	0.11934	3e-04	23	**0.11606**	0.00026	23
hyperplane—3	0.10588	0.00014	98	**0.10412**	0.00013	98
hyperplane—4	0.11254	4e-04	10	**0.10947**	0.00034	10
rbf—drift-1	0.13508	0.00083	47	**0.13109**	0.00078	47
rbf—drift-2	0.19549	0.00155	14	**0.18094**	0.00124	14
rbf—drift-3	0.16636	0.00107	46	**0.16503**	0.00101	46
rbf—drift-4	0.33926	0.00334	3	**0.32535**	0.00305	3
recurrent—abrupt—222	0.00185	9e-05	3	**7e-04**	5e-05	3
recurrent—abrupt—322	0.0016	0.00013	3	**0.00056**	4e-05	3
recurrent—abrupt—332	0.00252	0.00037	6	**0.00144**	0.00027	6
recurrent—abrupt—333	**0.00425**	0.00019	23	0.00475	0.00022	23
recurrent—abrupt—334	**0.01212**	0.00052	58	0.01389	6e-04	58
recurrent—abrupt—335	**0.01687**	0.00147	111	0.01978	0.00149	111
recurrent—abrupt—422	0.00177	0.00011	3	**0.00049**	3e-05	3
recurrent—abrupt—444	**0.06662**	0.00297	171	0.07347	0.00302	171
recurrent—abrupt—522	0.00177	0.00012	4	**0.00037**	0	4
recurrent—abrupt—555	**0.34892**	0.00896	329	0.35605	0.00868	329
Unique Wins	**6**			**18**		
A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.						
The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.				**Test Statistics**	**p-value: 0.01133**	**Confidence Interval: 0.56531 — 1**

It is clear from Table 9 that allowing alternates to vote is beneficial for prequential accuracy on a standard testbench of drifting streams (p-value 0.01133); we also notice a clear trend where increasing dimensionality of the data leads to loss of performance for the alternate voting strategy (as seen in the last ten rows).

The base strategy wins on streams with greater data dimensionality. As previously mentioned, the recurrent abrupt drift streams are notated with the number of classes, followed by the number of nominal attributes, and the number of values per attribute; a 5x4x3 stream has 5 classes, 4 nominal attributes, and 3 possible values per attribute. The experimental results in Table 9 show us that the base HAT strategy wins on 5x5x5, 4x4x4, 3x3x5, 3x3x4, and 3x3x3 streams, and the Random Tree Generator stream; in this test setting, the base strategy loses to the alternate-voting strategy on all those streams with under 21 degrees of freedom, winning on all of the others. Conversely, the strategy with alternate voting wins on all low-dimensional streams, losing on all the high-dimensional streams with 21 degrees of freedom or over.

Given that the high-dimensional recurrent drift streams are noise free, we hypothesize that the effect due to alternate voting—an aggressive lookahead strategy—results in higher bias (implying a larger bias space or set of hypothesis functions to choose from). In increasingly higher dimensional scenarios, the larger bias—and the ability to traverse it thanks to model adaptation—would lead to overfitting recent examples and thus a drop in prequential accuracy performance.

Thus on a typical concept drift testbench found in the literature that does not include a high dimensional recurrent abrupt drift generator, the alternate voting strategy leads to almost universal outperformance over standard HAT. It might be interesting to explore the extent to which existing results on the standard testbench generalize to higher dimensionality.

Our conclusion with respect to unspecified alternate voting is that it has a profound, fundamental impact on the performance of HAT, in a positive sense on a standard concept drift testbench.

5.3 Alternates of alternates

In addition to the losses on higher dimensional streams, enabling alternates to sprout their own alternates leads to losses in some gradual drift Hyperplane and RBF settings (Table 10). We offer this observation without explanation; it merits further study. Overall, our conclusion based on this testbench is that this unspecified strategy does not provide improvement in prequential accuracy, and that a single voting alternate is significantly better (p-value 0.02452).

Table 10 Performance of the multiple alternates voting strategy: multiple alternates may be grown and allowed to vote

Streams	**HAT with multiple voting alternates**			**HAT with single voting alternate**		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	***0.13643***	0.00036	14	***0.13643***	0.00036	14
recurrent—led	0.26671	8e-04	11	**0.26669**	8e-04	12
recurrent—randomtree	0.10663	0.00262	250	**0.10615**	0.00264	250
recurrent—sea	***0.12538***	0.00014	5	***0.12538***	0.00014	5
recurrent—stagger	***0.00136***	0	19	***0.00136***	0	19
recurrent—waveform	**0.17772**	0.00026	34	0.17789	0.00026	33
hyperplane—1	**0.10562**	0.00016	65	0.10576	0.00015	84
hyperplane—2	**0.11571**	0.00026	22	0.11606	0.00026	23
hyperplane—3	0.1044	0.00013	92	**0.10412**	0.00013	98
hyperplane—4	0.10963	0.00034	11	**0.10947**	0.00034	10
rbf—drift-1	0.13617	0.00083	39	**0.13109**	0.00078	47
rbf—drift-2	0.18238	0.00135	13	**0.18094**	0.00124	14
rbf—drift-3	0.17105	0.00129	37	**0.16503**	0.00101	46
rbf—drift-4	**0.32534**	0.00305	3	0.32535	0.00305	3
recurrent—abrupt—222	***7e-04***	5e-05	3	***7e-04***	5e-05	3
recurrent—abrupt—322	***0.00056***	4e-05	3	***0.00056***	4e-05	3
recurrent—abrupt—332	0.00157	0.00028	6	**0.00144**	0.00027	6
recurrent—abrupt—333	0.00793	0.00031	23	**0.00475**	0.00022	23
recurrent—abrupt—334	0.01693	7e-04	58	**0.01389**	6e-04	58
recurrent—abrupt—335	0.0212	0.00149	111	**0.01978**	0.00149	111
recurrent—abrupt—422	***0.00049***	3e-05	3	***0.00049***	3e-05	3
recurrent—abrupt—444	0.07469	0.00299	171	**0.07347**	0.00302	171
recurrent—abrupt—522	***0.00037***	0	4	***0.00037***	0	4
recurrent—abrupt—555	0.35638	0.00866	329	**0.35605**	0.00868	329
Unique Wins	**4**			**13**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.02452**	**Confidence Interval: 0.53945 — 1**

5.4 Alternates of alternates allowed, but only single leaf alternates do not vote

This is the unspecified voting strategy followed in the implementation used to report results in the original paper; alternates are allowed to vote, and alternates may grow alternates, which are also allowed to vote—but only if the alternates are not simple leaves but have more structure. Table 11 demonstrates that, interestingly, support for the underperformance of this strategy falls within a significance level of 0.05 when compared to the strategy where alternates are allowed to vote regardless of how simple their structure is (p-value 0.03196). The strategy especially underperforms across the board on streams with gradual drift. The implication is that the lookahead provided by single-leaf alternates is valuable in gradually drifting environments.

Table 11 Performance of the multiple alternates voting strategies: when single leaves are not allowed to vote, and when they are

Streams	**HAT with multiple voting alternates, excepting single leaves**			**HAT with multiple voting alternates, including single leaves**		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	**0.1364**	0.00037	14	0.13643	0.00036	14
recurrent—led	0.27417	0.00326	11	**0.26671**	8e-04	11
recurrent—randomtree	**0.10284**	0.00262	250	0.10663	0.00262	250
recurrent—sea	0.1256	0.00014	5	**0.12538**	0.00014	5
recurrent—stagger	0.002	3e-05	19	**0.00136**	0	19
recurrent—waveform	0.18552	0.00197	34	**0.17772**	0.00026	34
hyperplane—1	0.13141	0.00382	65	**0.10562**	0.00016	65
hyperplane—2	0.12012	0.00076	22	**0.11571**	0.00026	22
hyperplane—3	0.13913	0.00771	92	**0.1044**	0.00013	92
hyperplane—4	0.11189	0.00073	11	**0.10963**	0.00034	11
rbf—drift-1	0.14943	0.00167	39	**0.13617**	0.00083	39
rbf—drift-2	0.19643	0.00186	13	**0.18238**	0.00135	13
rbf—drift-3	0.20134	0.00698	37	**0.17105**	0.00129	37
rbf—drift-4	0.33891	0.00336	3	**0.32534**	0.00305	3
recurrent—abrupt—222	0.00159	7e-05	3	**7e-04**	5e-05	3
recurrent—abrupt—322	0.00147	9e-05	3	**0.00056**	4e-05	3
recurrent—abrupt—332	0.00246	0.00034	6	**0.00157**	0.00028	6
recurrent—abrupt—333	**0.00699**	0.00026	23	0.00793	0.00031	23
recurrent—abrupt—334	**0.01497**	0.00061	58	0.01693	7e-04	58
recurrent—abrupt—335	**0.01796**	0.00146	111	0.0212	0.00149	111
recurrent—abrupt—422	0.00149	7e-05	3	**0.00049**	3e-05	3
recurrent—abrupt—444	**0.06767**	0.00295	171	0.07469	0.00299	171
recurrent—abrupt—522	0.00148	7e-05	4	**0.00037**	0	4
recurrent—abrupt—555	**0.34955**	0.00892	329	0.35638	0.00866	329
Unique Wins	**7**			**17**		
A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.						
The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.				**Test Statistics**	**p-value: 0.03196**	**Confidence Interval: 0.52127 — 1**

5.5 Partial weighting

Weighting examples is a common strategy to force a learner to focus on some examples more so than others. For instance, online boosting (Oza 2005) weights misclassified examples higher than correctly classified ones, creating a virtual "heavier" history for examples that are being misclassified to aid learners in classifying them correctly if the reason for the misclassification happens to be sparsity of examples complicated by noisy observations.

Table 12 demonstrates that, with HAT, adding a Poisson(1) weighting for examples at leaves outperforms only letting alternates that are not single leaves vote (within a 0.05 significance level, p-value 0.00331); some instances are weighted more than others at leaves, prioritizing them. A Poisson(1) weighting assigns weights to examples stochastically—the weight is randomly drawn

from the Poisson(1) distribution, with a mean value of 1, and otherwise taking nonnegative integer values. This strategy is followed in the original HAT implementation with the likely intention of creating a semblance of ensemble-like diversity within subtrees. Given the seeming success of this unspecified strategy, it might be worthwhile exploring other Poisson values—Poisson(1) evaluates to 0 about a third of the time, so a third of all instances are not counted at leaves.

Table 12 Performance of the partial weighting strategy: leaves are weighted Poisson(1)

Streams	**HAT with multiple voting alternates, excepting single leaves**			**...with partial weighting also enabled**		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.1364	0.00037	14	**0.13354**	0.00049	23
recurrent—led	**0.27417**	0.00326	11	0.28105	0.00528	16
recurrent—randomtree	0.10284	0.00262	250	**0.10111**	0.00252	305
recurrent—sea	0.1256	0.00014	5	**0.12522**	0.00014	5
recurrent—stagger	**0.002**	3e-05	19	0.00205	5e-05	17
recurrent—waveform	0.18552	0.00197	34	**0.17895**	0.00077	39
hyperplane—1	**0.13141**	0.00382	65	0.15097	0.00376	72
hyperplane—2	**0.12012**	0.00076	22	0.14423	0.00435	27
hyperplane—3	0.13913	0.00771	92	**0.13817**	0.00256	124
hyperplane—4	**0.11189**	0.00073	11	0.11276	0.00048	12
rbf—drift-1	0.14943	0.00167	39	**0.1475**	0.00363	62
rbf—drift-2	0.19643	0.00186	13	**0.18406**	0.0016	22
rbf—drift-3	0.20134	0.00698	37	**0.19871**	0.00393	44
rbf—drift-4	0.33891	0.00336	3	**0.32724**	0.00291	4
recurrent—abrupt—222	0.00159	7e-05	3	**0.00154**	0.00014	3
recurrent—abrupt—322	0.00147	9e-05	3	**0.00135**	0.00011	4
recurrent—abrupt—332	0.00246	0.00034	6	**0.00185**	2e-04	6
recurrent—abrupt—333	0.00699	0.00026	23	**0.00698**	0.00024	23
recurrent—abrupt—334	0.01497	0.00061	58	**0.01312**	0.00058	58
recurrent—abrupt—335	0.01796	0.00146	111	**0.01528**	0.00099	112
recurrent—abrupt—422	0.00149	7e-05	3	**0.00146**	0.00011	3
recurrent—abrupt—444	0.06767	0.00295	171	**0.05249**	0.00232	182
recurrent—abrupt—522	0.00148	7e-05	4	**0.00143**	8e-05	3
recurrent—abrupt—555	0.34955	0.00892	329	**0.28932**	0.00648	541
Unique Wins	**5**			**19**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.00331**	**Confidence Interval: 0.61086 — 1**

5.6 Using weight accumulated at leaves to measure split evaluation intervals instead of number of instances seen

If examples at leaves are weighted, and split evaluation is done at *intervals of a certain amount of weight seen*, weighted examples will cause more frequent splitting. As Poisson(1) is used for weighting, on the whole, the effects should be negligible. As with VFDT, we find (Table 13) that there is no notable effect on the synthetic streams due to replacing a weight-based split evaluation timer (as in the original implementation) with an instance based one—prequential error is identical on all streams.

Table 13 Using getweightSeen instead of nodeTime: No measurable difference in prequential accuracy

Streams	HAT with nodeTime			HAT with getWeightSeen		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	***0.13645***	0.00037	14	***0.13645***	0.00037	14
recurrent—led	***0.26767***	0.00085	12	***0.26767***	0.00085	12
recurrent—randomtree	***0.10104***	0.00269	250	***0.10104***	0.00269	250
recurrent—sea	***0.12579***	0.00014	5	***0.12579***	0.00014	5
recurrent—stagger	***0.00215***	3e-05	19	***0.00215***	3e-05	19
recurrent—waveform	***0.17867***	0.00031	33	***0.17867***	0.00031	33
hyperplane—1	***0.10801***	0.00017	84	***0.10801***	0.00017	84
hyperplane—2	***0.11934***	3e-04	23	***0.11934***	3e-04	23
hyperplane—3	***0.10588***	0.00014	98	***0.10588***	0.00014	98
hyperplane—4	***0.11254***	4e-04	10	***0.11254***	4e-04	10
rbf—drift-1	***0.13508***	0.00083	47	***0.13508***	0.00083	47
rbf—drift-2	***0.19549***	0.00155	14	***0.19549***	0.00155	14
rbf—drift-3	***0.16636***	0.00107	46	***0.16636***	0.00107	46
rbf—drift-4	***0.33926***	0.00334	3	***0.33926***	0.00334	3
recurrent—abrupt—222	***0.00185***	9e-05	3	***0.00185***	9e-05	3
recurrent—abrupt—322	***0.0016***	0.00013	3	***0.0016***	0.00013	3
recurrent—abrupt—332	***0.00252***	0.00037	6	***0.00252***	0.00037	6
recurrent—abrupt—333	***0.00425***	0.00019	23	***0.00425***	0.00019	23
recurrent—abrupt—334	***0.01212***	0.00052	58	***0.01212***	0.00052	58
recurrent—abrupt—335	***0.01687***	0.00147	111	***0.01687***	0.00147	111
recurrent—abrupt—422	***0.00177***	0.00011	3	***0.00177***	0.00011	3
recurrent—abrupt—444	***0.06662***	0.00297	171	***0.06662***	0.00297	171
recurrent—abrupt—522	***0.00177***	0.00012	4	***0.00177***	0.00012	4
recurrent—abrupt—555	***0.34892***	0.00896	329	***0.34892***	0.00896	329

5.7 Approximating Information Gain

According to the original specification, Information Gain needs to be averaged across all timesteps, as explained in Section 4.3. Table 14 shows us that the unspecified HAT approximation strategy of using the latest computed infogain divided by the number of instances results in a net performance gain in prequential accuracy (p-value 0.04657, within a significance level of 0.05). Instantaneous Information Gain should be higher than a smoothed average if there is a trend favoring an attribute—which is particularly relevant in drifting settings and contributes to outperformance.

Table 14 Approximating Information Gain

Streams	HAT that averages infogain			HAT that approximates infogain		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	**0.13645**	0.00037	14	0.13871	0.00021	12
recurrent—led	0.26767	0.00085	12	**0.2675**	0.00062	9
recurrent—randomtree	0.10104	0.00269	250	**0.10079**	0.00285	252
recurrent—sea	0.12579	0.00014	5	**0.12578**	0.00014	5
recurrent—stagger	***0.00215***	3e-05	19	***0.00215***	3e-05	19
recurrent—waveform	0.17867	0.00031	33	**0.17862**	0.00031	32
hyperplane—1	**0.10801**	0.00017	84	0.10832	0.00016	80
hyperplane—2	0.11934	3e-04	23	**0.11929**	3e-04	19
hyperplane—3	**0.10588**	0.00014	98	0.1063	0.00014	102
hyperplane—4	0.11254	4e-04	10	**0.11148**	0.00043	16
rbf—drift-1	**0.13508**	0.00083	47	0.1372	0.00088	46
rbf—drift-2	0.19549	0.00155	14	**0.19518**	0.00149	16
rbf—drift-3	0.16636	0.00107	46	**0.16541**	0.00101	45
rbf—drift-4	**0.33926**	0.00334	3	0.33939	0.00337	2
recurrent—abrupt—222	0.00185	9e-05	3	**0.00183**	9e-05	3
recurrent—abrupt—322	**0.0016**	0.00013	3	0.00161	0.00012	3
recurrent—abrupt—332	**0.00252**	0.00037	6	0.0028	0.00045	6
recurrent—abrupt—333	0.00425	0.00019	23	**0.00421**	0.00017	23
recurrent—abrupt—334	0.01212	0.00052	58	**0.0118**	0.00056	59
recurrent—abrupt—335	0.01687	0.00147	111	**0.01646**	0.00143	111
recurrent—abrupt—422	0.00177	0.00011	3	**0.00176**	0.00013	3
recurrent—abrupt—444	0.06662	0.00297	171	**0.06559**	0.00312	172
recurrent—abrupt—522	0.00177	0.00012	4	**0.00173**	0.00014	4
recurrent—abrupt—555	0.34892	0.00896	329	**0.34081**	0.00849	342
Unique Wins	**7**			**16**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics | **p-value: 0.04657** | **Confidence Interval: 0.50356 — 1**

5.8 Split replacement behaviors—1

The original HAT exhibits some split replacement behaviors that appear to be detrimental to its performance under certain conditions. When a root level alternate leaf node l_{RA} is ready to split, it replaces the root node *Root*. Note that replacement shouldn't be occurring unless the alternate has better accuracy!

Table 15 studies the replacement of the root by a root level alternate when the alternate splits, with all of the VFDT unspecified features enabled. It is clear that this behavior leads to a significant drop in performance. The explanation could lie in the fact that all of the knowledge in the form of subtree structure accumulated under the root is prematurely lost.

Table 15 When an root alternate splits, it replaces the root

Streams	**HAT with VFDT unspecified features**			**...with added root substitution when root alternate splits**		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	**0.11303**	0.0024	98	0.11828	0.00193	99
recurrent—led	***0.2675***	0.00062	9	***0.2675***	0.00062	9
recurrent—randomtree	***0.09855***	0.00287	284	***0.09855***	0.00287	284
recurrent—sea	***0.11203***	0.00011	74	***0.11203***	0.00011	74
recurrent—stagger	0.00213	5e-05	18	**0.00177**	2e-05	19
recurrent—waveform	**0.17746**	3e-04	41	0.17768	3e-04	40
hyperplane—1	**0.10882**	0.00016	140	0.10884	0.00017	144
hyperplane—2	**0.11838**	0.00028	47	0.11855	0.00029	44
hyperplane—3	**0.10675**	0.00015	215	0.10696	0.00015	207
hyperplane—4	**0.11007**	4e-04	39	0.11033	0.00041	48
rbf—drift-1	***0.11975***	0.00053	68	***0.11975***	0.00053	68
rbf—drift-2	0.19195	0.00143	14	**0.19142**	0.00141	16
rbf—drift-3	***0.15236***	0.00085	104	***0.15236***	0.00085	104
rbf—drift-4	0.33901	0.00334	4	**0.3385**	0.00342	3
recurrent—abrupt—222	**0.00087**	5e-05	4	0.00093	4e-05	4
recurrent—abrupt—322	**0.00083**	5e-05	5	0.00097	4e-05	4
recurrent—abrupt—332	0.00161	0.00017	6	**0.00152**	0.00018	6
recurrent—abrupt—333	**0.00412**	0.00019	23	0.00416	0.00017	23
recurrent—abrupt—334	0.01182	0.00057	59	**0.0117**	0.00057	58
recurrent—abrupt—335	**0.0164**	0.00143	111	0.01642	0.00143	111
recurrent—abrupt—422	***0.00101***	6e-05	5	***0.00101***	6e-05	4
recurrent—abrupt—444	***0.06557***	0.00312	172	***0.06557***	0.00312	172
recurrent—abrupt—522	0.00104	6e-05	4	**0.00101**	5e-05	5
recurrent—abrupt—555	***0.34081***	0.00849	342	***0.34081***	0.00849	342
Unique Wins	**10**			**6**		
A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.						
The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.				**Test Statistics**	**p-value: 0.89494**	**Confidence Interval: 0.17777 — 1**

5.9 Split replacement behaviors—2

When a leaf alternate l_A of a mainline node $notRoot_1$ that is not the root splits, it replaces the corresponding mainline subtree of $notRoot_1$—again, it is not meant to do so until it has sprouted a subtree with higher accuracy. We study these behaviors with respect to a HAT with all the VFDT unspecified behaviors enabled.

Table 16 studies the replacement of the mainline subtrees by alternates when the alternates split. This also turns out to be detrimental—lookahead in these cases is too rapid.

Table 16 When a non-root alternate splits, it replaces its corresponding mainline node

Streams	**HAT with VFDT unspecified features**			**...with substitution of mainline node when alternate splits (not root level)**		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.11303	0.0024	98	**0.11167**	0.00257	88
recurrent—led	**0.2675**	0.00062	9	0.26807	0.00092	7
recurrent—randomtree	**0.09855**	0.00287	284	0.1128	0.0024	184
recurrent—sea	**0.11203**	0.00011	74	0.11727	0.00015	51
recurrent—stagger	0.00213	5e-05	18	**0.00211**	4e-05	18
recurrent—waveform	**0.17746**	3e-04	41	0.18341	0.00036	28
hyperplane—1	**0.10882**	0.00016	140	0.11003	0.00018	95
hyperplane—2	0.11838	0.00028	47	**0.11828**	0.00028	37
hyperplane—3	**0.10675**	0.00015	215	0.10986	2e-04	111
hyperplane—4	0.11007	4e-04	39	**0.11003**	4e-04	29
rbf—drift-1	**0.11975**	0.00053	68	0.12437	0.00058	41
rbf—drift-2	0.19195	0.00143	14	**0.19145**	0.00148	18
rbf—drift-3	**0.15236**	0.00085	104	0.15928	0.00089	55
rbf—drift-4	0.33901	0.00334	4	**0.33875**	0.00336	3
recurrent—abrupt—222	0.00087	5e-05	4	**0.00085**	4e-05	3
recurrent—abrupt—322	**0.00083**	5e-05	5	0.00084	5e-05	4
recurrent—abrupt—332	**0.00161**	0.00017	6	0.00164	0.00017	6
recurrent—abrupt—333	0.00412	0.00019	23	**0.00405**	0.00016	23
recurrent—abrupt—334	**0.01182**	0.00057	59	0.01201	0.00058	58
recurrent—abrupt—335	**0.0164**	0.00143	111	0.01832	0.00144	97
recurrent—abrupt—422	0.00101	6e-05	5	**0.00098**	5e-05	4
recurrent—abrupt—444	**0.06557**	0.00312	172	0.0675	0.00306	163
recurrent—abrupt—522	**0.00104**	6e-05	4	0.00106	5e-05	4
recurrent—abrupt—555	**0.34081**	0.00849	342	0.34213	0.00848	302
Unique Wins	**15**			**9**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.92421**	**Confidence Interval: 0.21157 — 1**

5.10 Split replacement behaviors—3

Following on from Sections 5.8 and 5.9, we should expect the combination of premature root replacement and premature non-root replacement should lead to an even larger drop in prequential accuracy. Table 17 studies the case with both replacement of the mainline subtrees by alternates when the alternates split, and of the root when a root alternate splits. Interestingly, combining these strategies leads to an improvement over each strategy alone as shown in Tables 15 and 16! While it does not perform significantly better than our rewritten baseline HAT, this can tip results in studies that focus mainly on naive rankings of algorithm performance.

Table 17 When an root alternate splits, it replaces the root; similarly, when any alternate splits, it replaces it's corresponding mainline node

Streams	**HAT with VFDT unspecified features and multiple alternates excepting single leaves voting**			**...with premature replacement by alternates at all levels added**		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.13098	0.00153	170	**0.12793**	0.00103	152
recurrent—led	0.28362	0.0058	18	**0.2678**	0.00041	17
recurrent—randomtree	**0.0994**	0.00251	354	0.12227	0.00206	206
recurrent—sea	**0.11224**	0.00011	100	0.11853	0.00016	51
recurrent—stagger	**0.00187**	4e-05	19	0.00188	2e-05	19
recurrent—waveform	0.18374	0.00193	46	**0.18298**	0.00034	38
hyperplane—1	0.11936	0.00058	171	**0.11644**	0.00021	105
hyperplane—2	0.13425	0.0028	59	**0.1241**	0.00033	37
hyperplane—3	0.13868	0.0039	249	**0.11753**	0.00024	94
hyperplane—4	**0.11149**	0.00055	39	0.11526	0.00042	24
rbf—drift-1	0.13354	0.00244	99	**0.12037**	0.00056	64
rbf—drift-2	0.18126	0.00141	22	**0.18017**	0.00118	16
rbf—drift-3	0.19556	0.00728	121	**0.15522**	0.00085	60
rbf—drift-4	**0.32677**	0.00288	3	0.32757	0.00278	3
recurrent—abrupt—222	**0.00065**	2e-05	5	0.00068	2e-05	3
recurrent—abrupt—322	0.00068	3e-05	5	**0.00062**	4e-05	4
recurrent—abrupt—332	0.00167	0.00013	7	**0.00122**	0.00012	6
recurrent—abrupt—333	0.00642	2e-04	23	**0.00307**	0.00011	23
recurrent—abrupt—334	0.01252	0.00047	57	**0.0093**	0.00044	57
recurrent—abrupt—335	**0.01473**	0.0011	113	0.01475	0.00105	103
recurrent—abrupt—422	**0.00069**	3e-05	5	0.00073	2e-05	4
recurrent—abrupt—444	**0.05099**	0.00233	180	0.05314	0.00238	172
recurrent—abrupt—522	0.00079	2e-05	4	**0.00078**	4e-05	4
recurrent—abrupt—555	**0.28127**	0.00627	558	0.28637	0.00589	443
Unique Wins	**10**			**14**		
A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.						
The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.				**Test Statistics**	**p-value: 0.27063**	**Confidence Interval: 0.39679 — 1**

5.11 Effects from HoeffdingTree

Table 18 shows us that adding just the unspecified features from HoeffdingTree is significantly beneficial for HAT, as it was found to be for VFDT (p-value 0.00002).

There are many more combinations and possibilities to study—given there are 9 options, a simple binomial sum gives us 19,863 possibilities to study. It is possible there are scenarios in which split replacement behaviors lead to an improvement under concept drift on the testbench.

Table 18 With and without unspecified VFDT features

Streams	**HAT**			**HAT with VFDT unspecified features**		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.13645	0.00037	14	**0.11303**	0.0024	98
recurrent—led	0.26767	0.00085	12	**0.2675**	0.00062	9
recurrent—randomtree	0.10104	0.00269	250	**0.09855**	0.00287	284
recurrent—sea	0.12579	0.00014	5	**0.11203**	0.00011	74
recurrent—stagger	0.00215	3e-05	19	**0.00213**	5e-05	18
recurrent—waveform	0.17867	0.00031	33	**0.17746**	3e-04	41
hyperplane—1	**0.10801**	0.00017	84	0.10882	0.00016	140
hyperplane—2	0.11934	3e-04	23	**0.11838**	0.00028	47
hyperplane—3	**0.10588**	0.00014	98	0.10675	0.00015	215
hyperplane—4	0.11254	4e-04	10	**0.11007**	4e-04	39
rbf—drift-1	0.13508	0.00083	47	**0.11975**	0.00053	68
rbf—drift-2	0.19549	0.00155	14	**0.19195**	0.00143	14
rbf—drift-3	0.16636	0.00107	46	**0.15236**	0.00085	104
rbf—drift-4	0.33926	0.00334	3	**0.33901**	0.00334	4
recurrent—abrupt—222	0.00185	9e-05	3	**0.00087**	5e-05	4
recurrent—abrupt—322	0.0016	0.00013	3	**0.00083**	5e-05	5
recurrent—abrupt—332	0.00252	0.00037	6	**0.00161**	0.00017	6
recurrent—abrupt—333	0.00425	0.00019	23	**0.00412**	0.00019	23
recurrent—abrupt—334	0.01212	0.00052	58	**0.01182**	0.00057	59
recurrent—abrupt—335	0.01687	0.00147	111	**0.0164**	0.00143	111
recurrent—abrupt—422	0.00177	0.00011	3	**0.00101**	6e-05	5
recurrent—abrupt—444	0.06662	0.00297	171	**0.06557**	0.00312	172
recurrent—abrupt—522	0.00177	0.00012	4	**0.00104**	6e-05	4
recurrent—abrupt—555	0.34892	0.00896	329	**0.34081**	0.00849	342
Unique Wins	**2**			**22**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 2e-05**	**Confidence Interval: 0.7602 — 1**

6 Conclusions

It is instructive to understand why a particular implementation of a proposed strategy works in terms of both unspecified features and intended mechanisms. We find that both VFDT and HAT have unspecified features that interact in complex ways to determine algorithm performance under concept drift.

In studying HAT, we came to understand the effects due to VFDT; in particular, the significance of inadvertent amnesia both by design of Hoeffding Tree (the theoretical artifact VFDT implements) and by design of VFDT-MOA (due to resplitting on used attributes). Proceeding along this line of inquiry, we also found that both major VFDT implementations (VFML and MOA) do not average infogain. Given Hoeffding Bound itself might not be suitable to use in the current context (Rutkowski et al. 2012), we find the approximations in computing the Hoeffding Bound are generally effective and harmless... with the added benefit of better performance on drifting streams.

We find the VFDT strategies of erasing node statistics through resplitting and approximating infogains individually somewhat effective, but significantly so when combined in terms of evoking a good response to concept drift from both VFDT, and HAT which derives from VFDT. We also find that the unspecified strategy of allowing alternates to vote in HAT significantly improves performance on streams with concept drift, with other strategies such as weighting leaves or optimistically early replacement of subtrees turning out to be beneficial or detrimental. We note that there are potentially thousands of possible interactions that need to be studied in order to extricate the true breadth of performance characteristics of simple strategies such as VFDT and HAT, and the choice of testbench adds further complexity to the conclusions one can draw.

A standardized, rationalized testbench that accounts for data stream characteristics does not exist in spite of decades of development in the field; we hope to have made a start in offering such a testbench.

In practice, we suggest that the node evisceration strategy proposed in Section 4.6 is integrated into standard Hoeffding Tree (and consequently into HAT) when they are expected to encounter concept drift, and that not averaging information gain 4.3 should continue being used in place of averaging information gain given the lack of evidence for adverse effect in stationary settings and due to the positive effects in drifting settings. However, it might be instructive to study specific cases in stationary settings in which an effect is observed due to not averaging information gain, and cases in drifting settings where not averaging infogain is detrimental. We also recommend that HAT is used with a single alternate until the effects of using multiple alternates are studied in greater depth, and that split replacement behaviors are normalized in general use (without the unspecified behaviors in Sections 5.1, 5.8, and 5.9), but that the mixture of unspecified strategies in Section 5.10 be studied in more detail.

It would be of immense utility to distill unspecified features from other basic machine learning methods so we may further our understanding of how learning strategies work.

References

Agrawal, Rakesh, Sakti Ghosh, Tomasz Imielinski, Balakrishna Iyer, and Arun Swami (Jan. 1992). "An Interval Classifier for Database Mining Applications." In: pp. 560–573.

Bifet, Albert and Ricard Gavalda (2007). "Learning from time-changing data with adaptive windowing". In: *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, pp. 443–448.

Bifet, Albert and Ricard Gavaldà (2009). "Adaptive learning from evolving data streams". In: *International Symposium on Intelligent Data Analysis*. Springer, pp. 249–260.

Bifet, Albert, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer (2010). "Moa: Massive online analysis". In: *Journal of Machine Learning Research* 11.May, pp. 1601–1604.

Bifet, Albert, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà (2009). "New ensemble methods for evolving data streams". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 139–148.

Breiman, L., J.H. Friedman, R.A. Olshen, and C.J. Stone (1984). *Classification and regression trees*. Chapman and Hall, New York.

Domingos, Pedro and Geoff Hulten (2000). "Mining high-speed data streams". In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 71–80.

Gama, João, Raquel Sebastião, and Pedro Pereira Rodrigues (Mar. 2013). "On Evaluating Stream Learning Algorithms". In: *Mach. Learn.* 90.3, pp. 317–346. ISSN: 0885-6125. DOI: 10.1007/s10994-012-5320-9. URL: http://dx.doi.org/10.1007/s10994-012-5320-9.

Gehrke, Johannes, Venkatesh Ganti, Raghu Ramakrishnan, and Wei-Yin Loh (1999). "BOAT—Optimistic Decision Tree Construction". In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. SIGMOD '99. Philadelphia, Pennsylvania, USA: ACM, pp. 169–180. ISBN: 1-58113-084-8. DOI: 10.1145/304182.304197. URL: http://doi.acm.org/10.1145/304182.304197.

Gehrke, Johannes, Raghu Ramakrishnan, and Venkatesh Ganti (2000). "RainForest—a framework for fast decision tree construction of large datasets". In: *Data Mining and Knowledge Discovery* 4.2-3, pp. 127–162.

Grossberg, Stephen (1988). "Nonlinear neural networks: Principles, mechanisms, and architectures". In: *Neural networks* 1.1, pp. 17–61.

Hoeffding, Wassily (1963). "Probability inequalities for sums of bounded random variables". In: *Journal of the American statistical association* 58.301, pp. 13–30.

Hoens, T Ryan, Robi Polikar, and Nitesh V Chawla (2012). "Learning from streaming data with concept drift and imbalance: an overview". In: *Progress in Artificial Intelligence* 1.1, pp. 89–101.

Hulten, Geoff, Laurie Spencer, and Pedro Domingos (2001). "Mining time-changing data streams". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 97–106.

Hunt, Earl Busby (1962). *Concept learning: An information processing problem.* John Wiley and Sons Inc.

Hunt, Earl Busby, Janet Marin, and Philip James Stone (1966). *Experiments in Induction*. Academic Press. URL: https://books.google.com.au/books?id=60NDAAAAIAAJ.

Oza, Nikunj C. (Oct. 2005). "Online Bagging and Boosting". In: *International Conference on Systems, Man, and Cybernetics, Special Session on Ensemble Methods for Extreme Environments*. Ed. by Mo Jamshidi. New Jersey: Institute for Electrical and Electronics Engineers, pp. 2340–2345.

Quinlan, John Ross (1979). "Discovering rules by induction from large collections of examples". In: *Expert systems in the micro electronics age*.

— (1983). "Learning efficient classification procedures and their application to chess end games". In: *Machine learning*. Springer, pp. 463–482.

— (1986). "Induction of Decision Trees". In: *MACH. LEARN* 1, pp. 81–106.

— (1992). *C4.5: programs for machine learning*. San Mateo, CA: Morgan Kaufmann. URL: http://cds.cern.ch/record/2031749.

Rutkowski, Leszek, Lena Pietruczuk, Piotr Duda, and Maciej Jaworski (2012). "Decision trees for mining data streams based on the McDiarmid's bound". In: *IEEE Transactions on Knowledge and Data Engineering* 25.6, pp. 1272–1279.

Schlimmer, Jeffrey and Douglas Fisher (1986). "A case study of incremental concept induction". In: *AAAI*. Vol. 86, pp. 496–501.

Schlimmer, Jeffrey and Richard Granger (1986). "Incremental learning from noisy data". In: *Machine Learning* 1.3, pp. 317–354. ISSN: 1573-0565. DOI: 10.1007/BF00116895. URL: http://dx.doi.org/10.1007/BF00116895.

Street, W Nick and YongSeog Kim (2001). "A streaming ensemble algorithm (SEA) for large-scale classification". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 377–382.

Utgoff, Paul E (1989). "Incremental induction of decision trees". In: *Machine learning* 4.2, pp. 161–186.

Webb, Geoffrey I, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean (2016). "Characterizing concept drift". In: *Data Mining and Knowledge Discovery* 30.4, pp. 964–994.

Chapter 4

An Improved Base Learner

Better Atoms For Machine Learning! Get yours now!

The process of identifying unspecified features had significant overlap with the process of aiming to design strategies that could improve these algorithms. Having understood the workings of Hoeffding Tree both in terms of its designed and unspecified features, we began looking for ways to increase plasticity without compromising statistical guarantees. We experimented with changing split mechanisms, the functional core of decision trees.

By tweaking the attribute selection mechanism and adding the possibility of revising poor split decisions, we created a decision tree algorithm that exhibited improved prequential accuracy with negligible time overhead, did not have an increased runtime complexity, and, unlike Hoeffding Tree, converged to the ideal batch tree.

We named our tree Hoeffding AnyTime Tree on account of its ability to revise split decisions and its relative eagerness in splitting compared to Hoeffding Tree. We presented our work as "Extremely Fast Decision Tree" (EFDT) at KDD 2018.

EFDT contains a subtree revision mechanism, one that is not designed for concept drift adaptation but for helping with the long-term stability needed to learn streams generated from stationary distributions. This is distinctly different from subtree revision mechanisms focused on concept-drift adaptation such as those found in HAT and CVFDT geared towards high plasticity.

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 *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3219819.3220005

1 INTRODUCTION

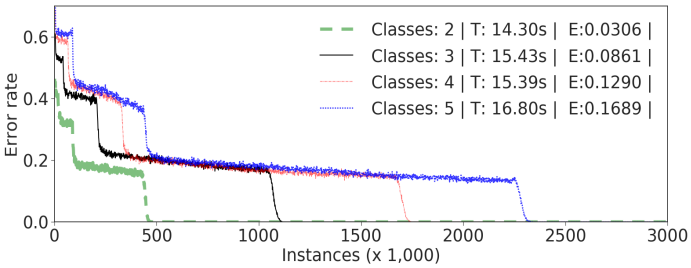
We present a novel stream learning algorithm, Hoeffding Anytime Tree (HATT)[1]. The de facto standard for learning decision trees from streaming data is Hoeffding Tree (HT) [11], which is used as a base for many state-of-the-art drift learners [3, 6, 8, 10, 16, 18, 24]. We improve upon HT by learning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

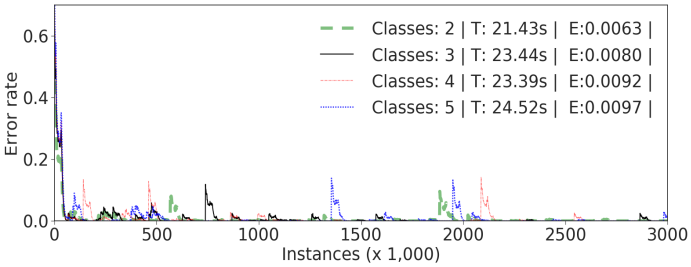
[1]In order to distinguish it from Hoeffding Adaptive Tree, or HAT [6]

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220005



(a) VFDT: the current de facto standard for incremental tree learning



(b) EFDT: our more statistically efficient variant

Figure 1.1: The evolution of prequential error over the duration of a data stream. For each learner we plot error for 4 different levels of complexity, resulting from varying the number of classes from 2 to 5. The legend includes time in CPU seconds (T) and the total error rate over the entire duration of the stream (E). This illustrates how EFDT learns much more rapidly than VFDT and is less affected by the complexity of the learning task, albeit incurring a modest computational overhead to do so. The data are generated by MOA RandomTreeGenerator, 5 classes, 5 nominal attributes, 5 values per attribute, 10 stream average.

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. This statistical efficiency in producing the batch tree equivalent is the reason we denote our implementation Extremely Fast Decision Tree; in the long run, where numerical features are involved, VFDT will grow interminably in depth causing increasing fragmentation and slowing while EFDT will readjust its splits maintaining a steady

tree size and thus computational load as long as the batch tree is finite. 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 2.1), 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.

Hoeffding Bound: If we have n independent random variables $r_1..r_n$, with range R and mean $\bar{r}$, the Hoeffding bound states that with probability $1-\delta$ the true mean is at least $\bar{r}-\epsilon$ where [11, 15]:

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

Hoeffding Tree is a tree that uses this probabilistic guarantee to test at each leaf whether the computed difference of information gains $\Delta\overline{G}$ between the attributes X_a and X_b with highest information gains respectively, $\Delta\overline{G}(X_a) - \Delta\overline{G}(X_b)$, is positive and non-zero. If, for the specified tolerance δ, we have $\Delta\overline{G} > \epsilon$, then we assert with confidence that X_a is the better split.

Note that we are seeking to determine the best split out-of-sample. The above controls the risk that X_a is inferior to X_b, but it does not control the risk that X_a is inferior to some other attribute X_c. It is increasingly likely that some other split will turn out to be superior as the total number of attributes increases. There is no recourse to alter the tree in such a scenario.

3 HOEFDDING 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.

Algorithm 2.1: Hoeffding Tree, Domingos & Hulten (2000)
–Reproduced verbatim from original–

Input: S, a sequence of examples,
$\mathbf{X}$, a set of discrete attributes,
G(.), a split evaluation function
δ, one minus the desired probability of choosing the correct attribute at any given node
Output: HT, a decision tree.
begin
  Let HT be a tree with a single leaf l_1 (the *root*).
  Let $\mathbf{X_1} = \mathbf{X} \cup X_\emptyset$.
  Let $\overline{G_1}(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class in S
  foreach *class* y_k **do**
    foreach *value* x_{ij} *of each attribute* $X_i \in \mathbf{X}$ **do**
      Let $n_{ijk}(l_1) = 0$
    end
  end
  foreach *example* $(\vec{x}, y)$ *in* S **do**
    Sort $(\vec{x}, y)$ into a leaf l using HT
    foreach x_{ij} *in* $\vec{x}$ *such that* $X_i \in \mathbf{X}_l$ **do**
      Increment $n_{ijk}(l)$
    end
    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 $\overline{G_l}(X_i)$ for each attribute $X_i \in \mathbf{X}_l - \{X_\emptyset\}$ using the counts $n_{ijk}(l)$
      Let X_a be the attribute with highest $\overline{G_l}$
      Let X_b be the attribute with second-highest $\overline{G_l}$
      Compute ϵ using Equation 1
      if $\overline{G_l}(X_a) - \overline{G_l}(X_b) > \epsilon$ *and* $X_a \neq X_\emptyset$ **then**
        Replace l by an internal node that splits on X_a
        foreach *branch of the split* **do**
          Add a new leaf l_m and let $\mathbf{X}_m = \mathbf{X} - \{X_\emptyset\}$
          Let $\overline{G}_m(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class at l_m
          foreach *class* y_k *and each value* X_{ij} *of each attribute* $X_i \in \mathbf{X}_m - \{X_\emptyset\}$]] **do**
            Let $n_{ijk}(l_m) = 0$
          end
        end
      end
    end
  end
  Return HT
end

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.

Algorithm 3.1: Hoeffding Anytime Tree

Input: S, a sequence of examples. At time t, the observed sequence is $S^t = ((\vec{x}_1, y_1), (\vec{x}_2, y_2), ...(\vec{x}_t, y_t))$
$\mathbf{X} = \{X_1, X_2 ... X_m\}$, a set of m attributes
δ, the acceptable probability of choosing the wrong split attribute at a given node
G(.), a split evaluation function

Result: $HATT^t$, 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 ∪ X_∅
    Let G_1(X_∅) 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 ∈ X do
            Set counter n_ijk(root) = 0
        end
    end
    foreach example (x⃗, y) in S do
        Sort (x⃗, y) into a leaf l using HATT
        foreach node in path (root...l) do
            foreach x_ij in x⃗ such that X_i ∈ X_node do
                Increment n_ijk(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 Ḡ_l(X_i) for each attribute X_l − {X_∅} using
          the counts n_ijk(l)
        Let X_a be the attribute with the highest Ḡ_l
        Let X_b = X_∅
        Compute ε using equation 1
        if Ḡ_l(X_a) − Ḡ_l(X_b) > ε and X_a ≠ X_∅ 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 Ḡ_m(X_∅) 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 ∈ X_m − {X_∅} do
                    Let n_ijk(l_m) = 0.
                end
            end
        end
    end
end
```

Function 3.3: ReEvaluateBestSplit(internalNode int)

```
begin
    Compute Ḡ_int(X_i) for each attribute X_int − {X_∅} using
      the counts n_ijk(int)
    Let X_a be the attribute with the highest Ḡ_int
    Let X_current be the current split attribute
    Compute ε using equation 1
    if Ḡ_l(X_a) − Ḡ_l(X_current) > ε then
        if X_a = X_∅ then
            Replace internal node int with a leaf (kills subtree)
        else if X_a ≠ X_current then
            Replace 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 Ḡ_m(X_∅) 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 ∈ X_m − {X_∅} do
                    Let n_ijk(l_m) = 0.
                end
            end
        end
    end
end
```

3.1 Convergence

Hoeffding Tree offers guarantees on the expected disagreement from a batch tree trained on an infinite dataset (which is denoted DT_* in [11], a convention we will follow). "Extensional disagreement" is defined as the probability that a pair of decision trees will

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 δ 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 ϵ 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.*

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 $\vec{x}$ take values corresponding to attributes X_1, X_2, ... X_m, if we have m attributes.

We are interested in attaining confidence $1-\delta$ that $\sum_{i=0}^{n} \Delta G/n - \mu_{\Delta G} \leq \epsilon$, where $\mu_{\Delta G}$ is the true mean of the summed historical differences in information gain between the top and second-best attributes (the matter of whether a sum must be used appears to be ambiguous in the original Hoeffding Tree proposed in [11]; because it is necessary in order to meaningfully use the Hoeffding bound, we assumed this was their intent. In implementation, there appears to be no consequence). We don't know $\mu_{\Delta G}$, but we would like it to be non-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 $\overline{\Delta G}$ differs from zero by at least ϵ. In other words, we are using a corollary of Hoeffding's Inequality to state with confidence that our random variable $\overline{\Delta G}$ diverges from 0.

In order for this to happen, we need $\overline{\Delta G}$ to be greater than ϵ. ϵ 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 ϵ.

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 $\overline{\Delta G}$, which can be different for HT and HATT. That is, it is possible that for $t \leq T$, $\overline{\Delta G}(HT) \neq \overline{\Delta G}(HATT)$. 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 $X_\emptyset$, which may happen to be the current split.

Clearly, at any timestep, $X^{i:t}(N_0)(HT) = X^{i:t}(N_0)(HATT)$. 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, $G(X^{i:t}(N_0)(HT)) = G(X^{i:t}(N_0)(HATT))$– 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 $\overline{\Delta G}^t$ as $\overline{G}^t_{top} - \overline{G}^t_{bot}$, we will have $\overline{G}^t_{top}(HT) = \overline{G}^t_{top}(HATT)$, but $\overline{G}^t_{bot}(HT)$ and $\overline{G}^t_{bot}(HATT)$ 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 $G(X^{currentSplit})$ where $G(X^{currentSplit}) \leq G(X^{2:T})$ (unless $X^{1:T} = X^{currentSplit}$, in which case $G^T_{bot}(HATT)$ is $G(X^{2:T})$), we have $\overline{G}^t_{bot}(HATT) \leq \overline{G}^t_{bot}(HT)$ for all t. (In the case that $X^{currentSplit} = X^{nullSplit}$, either $X^{currentSplit} = X^{1:T}$, which we have just accounted for, or $X^{currentSplit} = X^{2:T}$).

Because we have $\overline{G}^t_{bot}(HATT) \leq \overline{G}^t_{bot}(HT)$, we will have $\overline{\Delta G}^T(HATT) \geq \overline{\Delta G}^T(HT)$, and $\overline{\Delta G}^T(HT) > \epsilon$ implies $\overline{\Delta G}^T(HATT) > \epsilon$, 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_R^{HATT} at the root node of HATT converges in probability to the split attribute $X_R^{DT_*}$ 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_R^{HATT} that matches the split $X_R^{DT_*}$ at the root node of DT_* goes to 1.*

Proof. Let us denote the attributes available at the root X_i and the information gain of each attribute computed at time t as $G(X_i)^t$, based on the observed sequence of examples $S^t = ((\vec{x}_1, y_1), (\vec{x}_2, y_2)...(\vec{x}_t, y_t))$.

Now, we are working under the assumption that each X_i has a finite, constant information gain associated with it—DT_* would not converge, and thus any guarantees about HT's deviation from DT_* would not hold without making this assumption. Let us denote this gain $G(X_i)^\infty$.

This in turn implies that all pairwise differences in information gain: $\Delta G^\infty = G(X_a)^\infty - G(X_b)^\infty$ for any two attributes X_a and X_b must also be finite and constant over any given infinite dataset (from which we generate a stationary stream).

As $t \to \infty$, we expect the frequencies of our data $(\vec{x}, y)$ to approach their long-term frequencies given by P. Consequently, we expect our measured sequences of averaged pairwise differences in information gain $\overline{\Delta G}(X_{ij})^t$ to converge to their respective constant values on the infinite dataset $\Delta G(X_{ij})^\infty$, which implies we will effectively have the chosen split attribute for $HATT$ converging in probability to the chosen split attribute for DT_* as $t \to \infty$.

Why would this convergence only be in probability and not almost surely?

For any finite sequence of examples $S^t = ((\vec{x}_1, y_1), (\vec{x}_2, y_2) \ldots (\vec{x}_t, y_t))$ with frequencies of data that approach those given by P, we may observe with nonzero probability a followup sequence $((\vec{x}_{t+1}, y_{t+1}), (\vec{x}_{t+2}, y_{t+2}), \ldots (\vec{x}_{2t}, y_{2t}))$ that will result in a distribution that is unlike P over the observations. Obviously, we expect the probability of observing such an anomalous sequence to go to 0 as t grows large– if we didn't, we would not expect the observed frequencies of the instances to ever converge to their long-term frequencies.

Anytime we do observe such a sequence, we can expect to see anomalous values of $\overline{\Delta G}(X_{ij})^t$, which means that even if the top attribute has already been established as one that matches the attribute corresponding to $\Delta G(X_{ij})^\infty$, it may briefly be replaced by an attribute that is not the top attribute as per $G(X_i)^\infty$. We have already reasoned that the probability of observing such anomalous sequences must go to 0; so we expect that the probability of observing sequences with instance frequencies approaching those given by the measure P must go to 1. And for a sequence that is distributed as per P, we expect our information gain differences $\overline{\Delta G}(X_{ij})^t \rightarrow \Delta G(X_{ij})^\infty$.

Remember that we have assumed that the pairwise differences in information gain $\Delta G(X_{ij})^t$ are nonzero (by implication of no two attributes having identical information gain). Since ϵ is monotonically decreasing and no two attributes have been assumed to be identical, as t grows large, we will always pick the attribute with the largest information gain because its advantage over the next best attribute will exceed some fixed ϵ; and this picked top attribute will match, in probability, the one established by DT_*.

□

Lemma 3.3. *Hoeffding AnyTime Tree converges to the asymptotic batch tree in probability.*

Proof. From Lemma 3.2, we have that as $t \rightarrow \infty$, $X_R^{HATT} \xrightarrow{P} X_R^{DT_*}$, meaning that though it is possible to see at any individual timestep $X_R^{HATT} \neq X_R^{DT_*}$, we have have convergence in probability in the limit.

Consider immediate subtrees of the root node $HATT_i^1$ (denoting they are rooted at level 1). In all cases where the root split matches $X_R^{DT_*}$, the instances observed at the roots of $HATT_i^1$ will be drawn from the same data distribution that the respective DT_{*i}^1 draw their instances from. Do level 1 split attributes for HATT, $X_{i:L1}^{HATT}$ converge to $X_{i:L1}^{DT_*}$?

We can answer this by using the Law of Total Probability. Let us denote the event that for first level split i, $X_{i:L1}^{HATT} = X_{i:L1}^{DT_*}$ by $match_{i:L1}$. Then we have as $t \rightarrow \infty$:

$$\begin{aligned} &P(X_{i:L1}^{HATT} = X_{i:L1}^{DT_*}) \\ &= P(match_{i:L1}) \\ &= P(match_{i:L1}|match_{L0})P(match_{L0}) \\ &+P(match_{i:L1}|not\_match_{L0})P(not\_match_{L0}) \end{aligned}$$

We know that $P(match_{L0}) \rightarrow 1$ and $P(not\_match_{L0}) \rightarrow 0$ as $t \rightarrow \infty$ from Lemma 3.1. So we obtain $P(X_{i:L1}^{HATT} = X_{i:L1}^{DT_*})^\infty = P(match_{i:L1}|match_{L0})^\infty$.

Effectively, we end up only having to condition on the event $match_{L0}$. In other words, we may safely use a subset of the stream where only $match_{L0}$ has occurred to reason about whether $X_{i:L1}^{HATT} = X_{i:L1}^{DT_*}$ as $t \rightarrow \infty$.

Now, we need to show that $P(match_{i:L1}|match_{L0}) \rightarrow 1$ as $t \rightarrow \infty$ to prove convergence at level 1. This is straightforward. Since we are only considering instances that result in the event $match_{L0}$ occurring, the conditional distributions at level 1 of HATT match the ones at level 1 of DT_*. We may extend this argument to any number of levels; thus $HATT$ converges in probability to DT_* for any finite DT_*.

□

3.2 Time and Space Complexity

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

Time Complexity: There are two primary operations associated with learning for 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 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 HT and at any node for HATT, no more than d attribute evaluations will have to be considered. Each attribute evaluation at a node requires the computation of v 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 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 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 HT and $O(hdvc)$ for HATT, as one leaf and one path respectively have to be evaluated.

4 RELATED WORK

There is a sizable literature that adapts 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 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 HT are either significantly less

tight compared to 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.

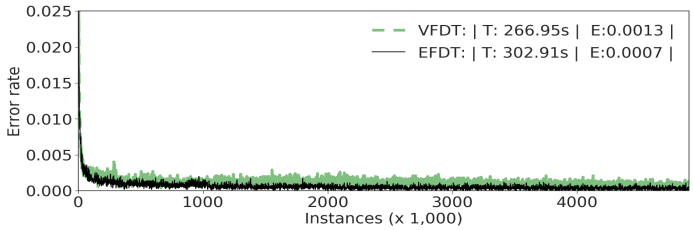
The idea common to both CVFDT and HATT is that of split re-evaluation. However, the circumstances, objectives, and methods are entirely different. CVFDT is explicitly designed for a drifting scenario; HATT for a stationary one. CVFDT's goal is to reduce prequential error for the current window in the expectation that this is the best way to respond to drift; HATT's goal is reduce prequential error overall for a stationary stream so that it asymptotically approaches that of a batch learner. CVFDT builds and substitutes alternate subtrees; HATT does not. CVFDT deliberately employs a range of forgetting mechanisms; 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 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.

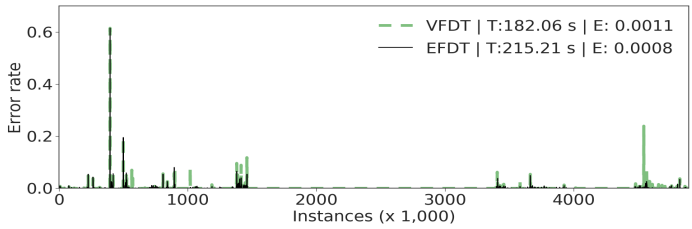
In order to avoid confusion, we will also mention the Hoeffding Adaptive Tree (HAT) [6]. This method builds a tree that grows alternate subtrees if a subtree is observed to have poorer prequential accuracy on more recent examples, and substitutes an alternate when it has better accuracy than the original 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



(a) 10 stream shuffled average.



(b) Unshuffled.

Figure 5.1: KDD intrusion detection dataset [21]

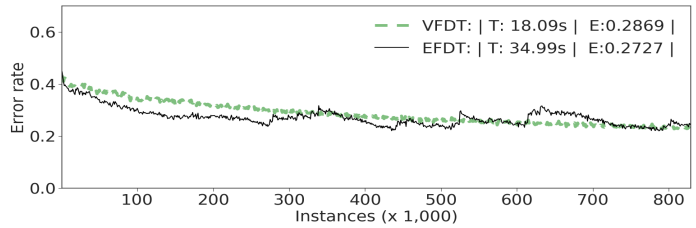
instances that had an obvious classification target variable, did not require text mining, and did not contain missing values (MOA has limited support for handling missing values). To augment this limited collection of large datasets, we also studied performance on the WISDM dataset [20]. In all, we 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.

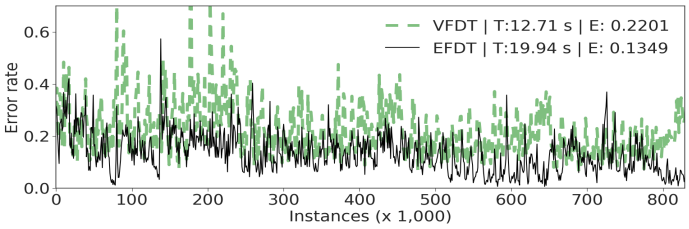
Our experiments are easily reproducible. Instructions for processing datasets, source code for VFDT and EFDT to be used with MOA, and Python scripts to run the experiments are all available at [https://github.com/chaitanya-m/kdd2018.git].

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

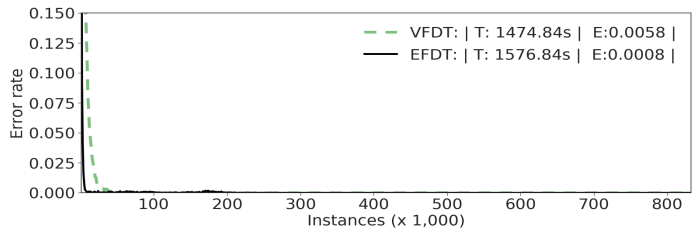


(a) 10 stream shuffled average.

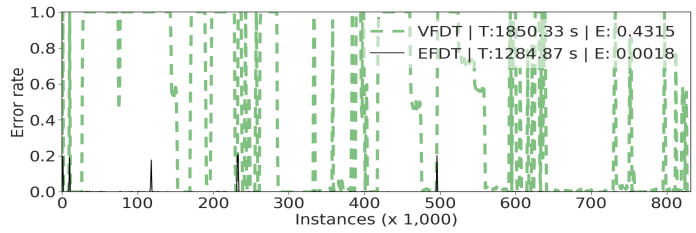


(b) Unshuffled.

Figure 5.2: Poker dataset [21]

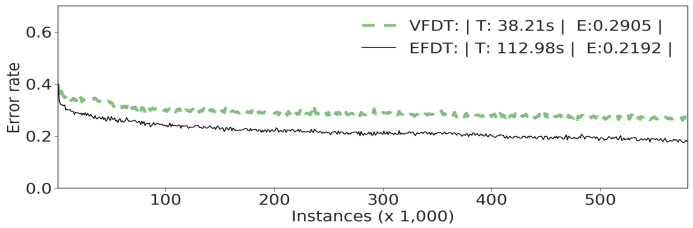


(a) 10 stream shuffled average.

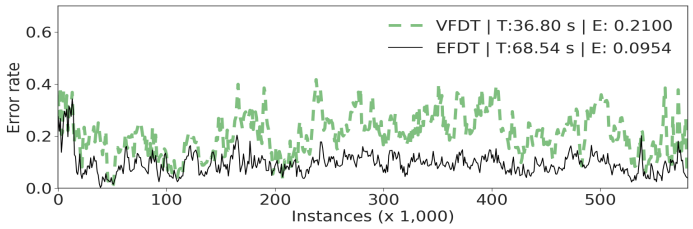


(b) Unshuffled.

Figure 5.3: Fonts dataset [21]

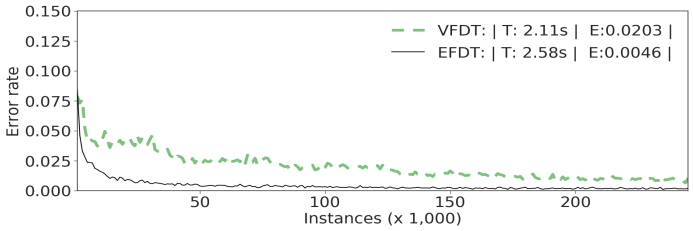


(a) 10 stream shuffled average.

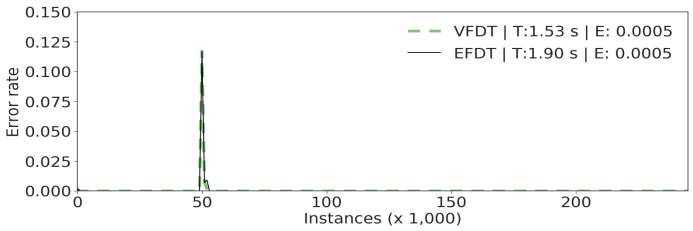


(b) Unshuffled.

Figure 5.4: Forest covertype dataset [9, 21]



(a) 10 stream shuffled average.



(b) Unshuffled.

Figure 5.5: Skin dataset [4, 21]

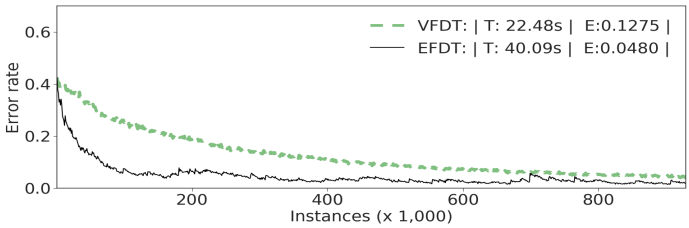
learn a more complex decision boundary, affecting performance for both learners.

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.

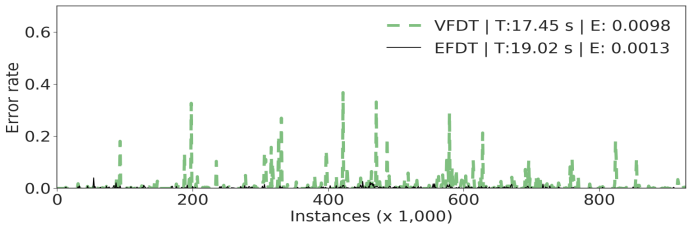
The results on the Poker and Forest-Covertype datasets (Figs. 5.2, 5.4) reflect both effects: EFDT performs significantly better on ordered data, and performance for both learners deteriorates with shuffled data in comparison with unshuffled data.

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.

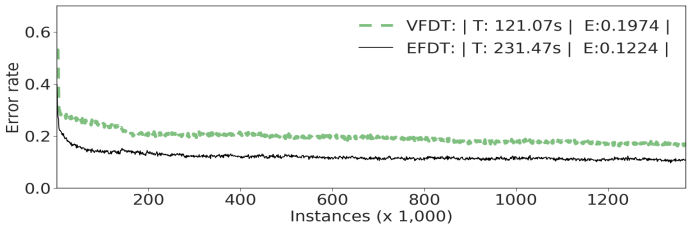


(a) 10 stream shuffled average.

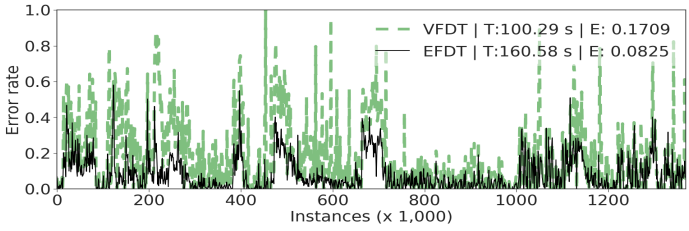


(b) Unshuffled.

Figure 5.6: Gas sensor dataset [17, 21]



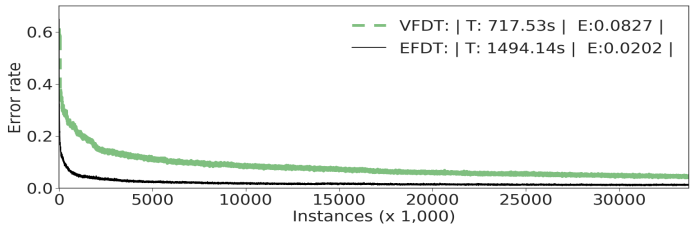
(a) 10 stream shuffled average.



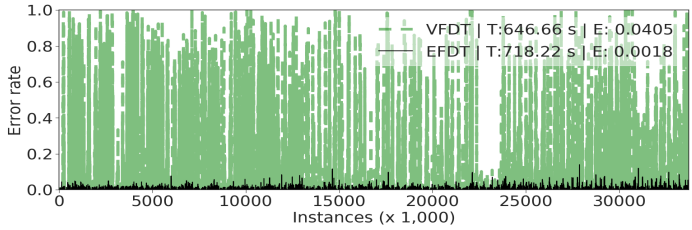
(b) Unshuffled.

Figure 5.7: WISDM dataset [20, 21]

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 (Figs. 5.10 to 5.12).

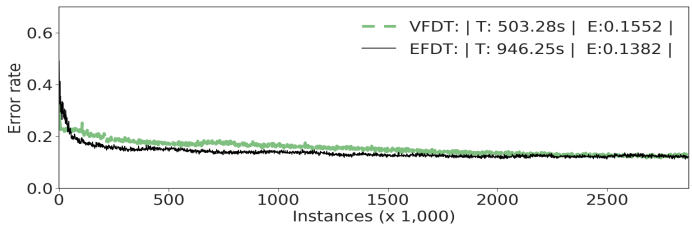


(a) 10 stream shuffled average.

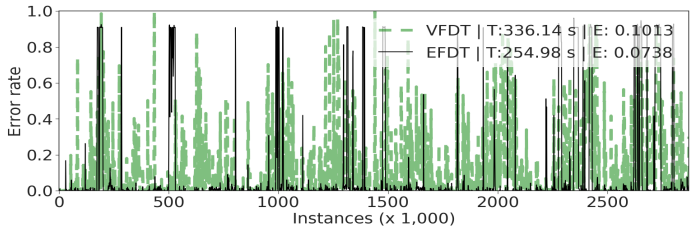


(b) Unshuffled.

Figure 5.8: Human Activity Recognition dataset: Phone, watch accelerometer, and gyrometer data combined. [21, 25]

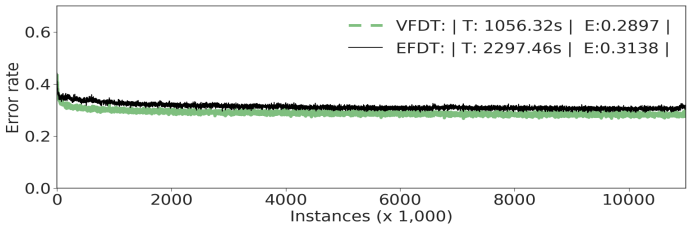


(a) 10 stream shuffled average.

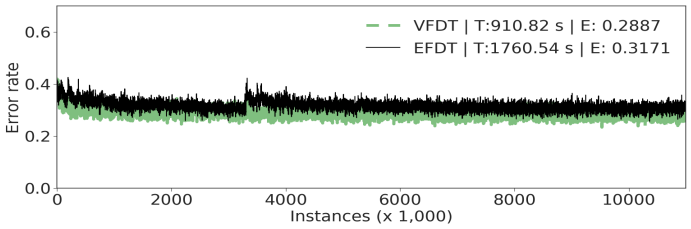


(b) Unshuffled.

Figure 5.9: PAMAP2 Activity Recognition dataset (UCI)– 9 subjects data combined [21, 22]

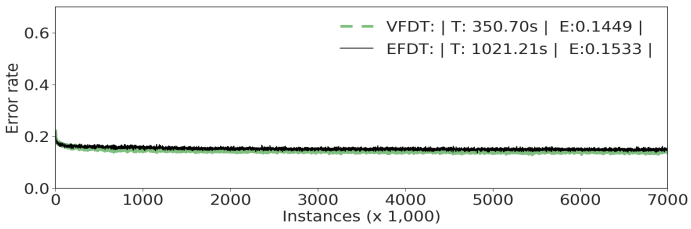


(a) 10 stream shuffled average.

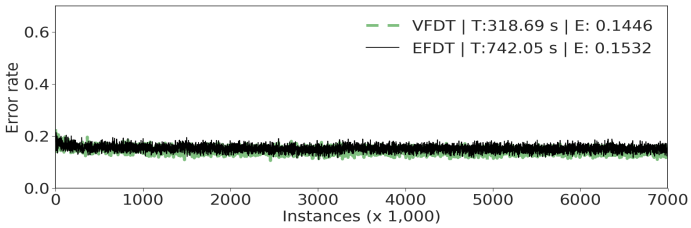


(b) Unshuffled.

Figure 5.10: Higgs dataset [1, 21]



(a) 10 stream shuffled average.



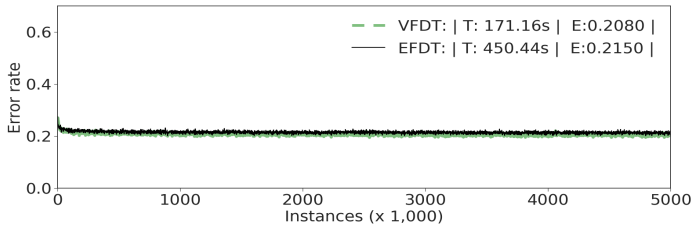
(b) Unshuffled.

Figure 5.11: Hepmass dataset [2, 21]

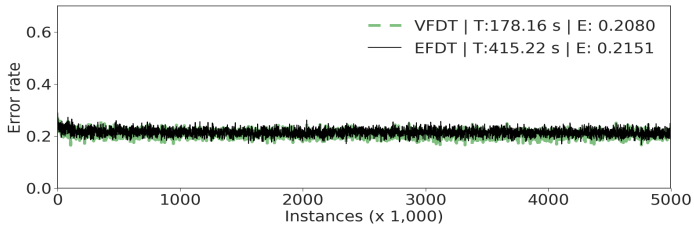
Fig. 5.13 shows us that with the MOA tree generator used in Fig. 1.1, even on a 100 million length stream, EFDT's prequential error is still an order of magnitude lower than that of VFDT.

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.

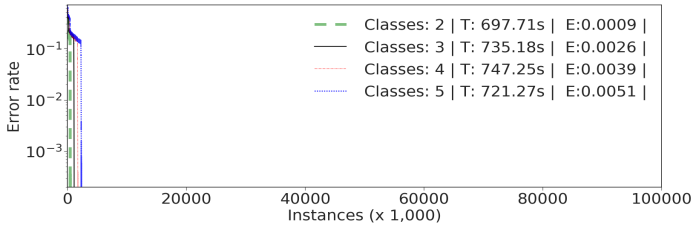


(a) 10 stream shuffled average.

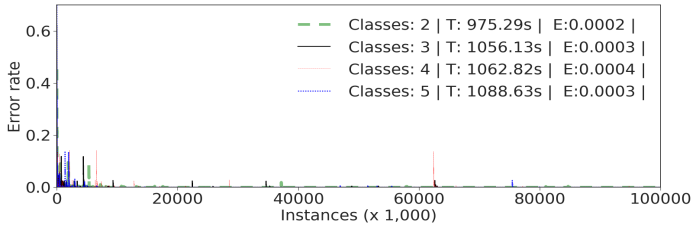


(b) Unshuffled.

Figure 5.12: SUSY dataset [1, 21]



(a) VFDT



(b) EFDT

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.

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.

HT cautiously works toward the asymptotic batch tree, ignoring, and thus not benefiting from potential improvements on the current state of the tree, until it is sufficiently confident that they will not need to be subsequently revised. If an incrementally learned tree is to be deployed to make predictions before fully learned, HATT's strategy of always utilizing the most useful splits identified to date has profound benefit.

ACKNOWLEDGEMENTS

This material is based upon work supported by the Air Force Office of Scientific Research, Asian Office of Aerospace Research and Development (AOARD) under award number FA2386-17-1-4033.

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.
[4] Rajen Bhatt and Abhinav Dhall. 2012. Skin Segmentation Dataset: UCI Machine Learning Repository. (2012).
[5] Albert Bifet and Ricard Gavalda. 2007. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, 443–448.
[6] Albert Bifet and Ricard Gavaldà. 2009. Adaptive learning from evolving data streams. In *International Symposium on Intelligent Data Analysis*. Springer, 249–260.
[7] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010. Moa: Massive online analysis. *Journal of Machine Learning Research* 11, May (2010), 1601–1604.
[8] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. 2009. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 139–148.
[9] Jock Blackard and Denis Dean. 1999. Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis in Predicting Forest Cover Types from Cartographic Variables. 24 (12 1999), 131–151.
[10] 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.
[11] Pedro Domingos and Geoff Hulten. 2000. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 71–80.
[12] 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
[13] 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
[14] Jonathan Gratch. 1996. Sequential inductive learning. In *Proceedings of the thirteenth national conference on Artificial intelligence-Volume 1*. AAAI Press, 779–786.
[15] Wassily Hoeffding. 1963. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association* 58, 301 (1963), 13–30.
[16] Stefan Hoeglinger and Russel Pears. 2007. Use of hoeffding trees in concept based data stream mining. In *Information and Automation for Sustainability, 2007. ICIAFS 2007. Third International Conference on*. IEEE, 57–62.
[17] Ramon Huerta, Mosqueiro, et al. 2016. Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. *Chemometrics and Intelligent Laboratory Systems* 157 (2016), 169–176.
[18] Geoff Hulten, Laurie Spencer, and Pedro Domingos. 2001. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 97–106.
[19] Ruoming Jin and Gagan Agrawal. 2003. Efficient decision tree construction on streaming data. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 571–576.
[20] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. 2010. Activity recognition using cell phone accelerometers. In *Proceedings of the Fourth International Workshop on Knowledge Discovery from Sensor Data*. 10–18.
[21] Moshe Lichman. 2013. UCI Machine Learning Repository. (2013). http://archive.ics.uci.edu/ml
[22] Attila Reiss and Didier Stricker. 2012. Introducing a new benchmarked dataset for activity monitoring. In *Wearable Computers (ISWC), 2012 16th International Symposium on*. IEEE, 108–109.
[23] 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), 1272–1279.
[24] Silas Santos et al. 2014. *Speeding Up Recovery from Concept Drifts*. Springer Berlin Heidelberg, Berlin, Heidelberg, 179–194.
[25] 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.
[26] Paul E Utgoff. 1989. Incremental induction of decision trees. *Machine learning* 4, 2 (1989), 161–186.

Chapter 5

Responding to Concept Drift: Extremely Fast Adaptive Tree

"Future Work"—We have no idea.

Chapter 4 details a learning mechanism, Hoeffding AnyTime Tree (HATT), designed as a replacement for Hoeffding Tree (HT) on stationary streams. HATT adds plasticity through eager splitting that results in greater statistical efficiency and consequent superior prequential accuracy performance on a UCI testbench. In this chapter, we study the effect of using the eager splitting strategy with an adaptive algorithm designed for learning with concept drift, Hoeffding Adaptive Tree [11]. In order to avoid confusion between HAT and HATT, in this chapter we will refer to HATT by the name of its implementation, Extremely Fast Decision Tree (EFDT).

By HAT we refer to the original implementation of Hoeffding Adaptive Tree with all the unspecified features—for this discussion, we will assume that the reader is familiar with Chapter 3. HAT, with all unspecified features, has outstanding prequential accuracy performance as a package that we could not beat by experimenting with combining some of the unspecified features that we identified in Chapter 3. It would appear the unspecified features—including the replacement of mainline subtrees anytime an alternate leaf is ready to split, instead of replacing the corresponding leaf—are incredibly powerful strategies in the face of concept drift.

HAT's change detection and subsequent update strategy is explicitly designed for addressing concept drift. As a result, we investigate whether replacing the simple update strategy of EFDT with HAT's revision strategy leads to better drift responsiveness. We call this strategy Extremely Fast Adaptive Tree (EFAT).

This simple change in strategy turns out to be profitable for EFAT with respect to both HAT and EFDT on concept-drifting streams as well as on real streams with minimal concept drift.

Overall, these results support the thesis that rationalized plasticity at multiple layers of a learning mechanism in different flavors (whether short-term plasticity aimed at long-term stability as with EFDT, or greedy plasticity aimed at rapid drift response as with HAT, or a combination) can be effective for learning in settings with concept drift as well in the stationary setting.

5.1 Experimental Results

The experimental design we use to compare EFDT, EFAT and HAT is the same as the one used in Chapter 3.

5.1.1 EFDT and HAT

EFDT is not designed to address concept drift, as is Hoeffding Adaptive Tree (HAT). EFDT is considerably more conservative than HAT in deciding when to modify tree structure.

As discussed in Chapters 2, 3 and 4, HAT theoretically modifies tree structure as soon as an alternate has reliably superior accuracy (based again on the Hoeffding Test). However, given the unspecified feature of voting alternates, structural modification effectively occurs as soon as an alternate is created. In either instance, HAT is far more plastic than is EFDT, which modifies tree structure only once it is confident that a new split is better than the existing one. HAT can replace a subtree trained on data from an older concept with an alternate subtree trained on fresh data from a new concept even if the replacement subtree splits on the same attribute as the mainline one at the top level. EFDT, which does not build alternate subtrees, replaces splits, deleting any underlying tree structure; it will replace a split only if it infers that on an underlying generating distribution assumed to be stationary, an alternative split attribute is better than the current one. EFDT is thus less sophisticated in its response to concept drift than is HAT.

On real data streams—mostly UCI testbenches with inherent concept drift, and incidental concept drift introduced due to concatenation of data files—HAT appears to have a slight advantage over EFDT, though it is not within a 0.05 significance level (Table 5.1, p-value 0.17964). When these UCI streams are shuffled to remove concept drift, as shown in Table 5.2 EFDT appears to have the advantage over HAT, though not within a 0.05 significance level. The event we are interested in is that of EFDT winning 13 or more times by chance—and this has a probability of 0.08353, which does not fall within a 0.05 level of significance, but still suggests that the greater stability of EFDT is more suitable in situations with minimal concept drift (note that the p-value 0.996822 in the table corresponds to the probability that HAT wins at least 6 times if wins and losses were equiprobable).

Table 5.1: Real Streams: EFDT and HAT

Streams		EFDT	HAT
aws—price-discretized		**0.14143**	0.14572
chess		0.31152	**0.08866**
covertype		**0.15431**	0.1808
covpokelec		**0.19391**	0.26455
fonts		**0.003**	0.0053
hhar		**0.00357**	0.00519
kdd		0.00094	**0.00092**
localization		0.09754	**0.059**
miniboone		0.00013	**0.00011**
nbaiot		0.00076	**0.00035**
nswelec		0.19283	**0.16772**
pamap2		0.06136	**0.03138**
poker		**0.218**	0.3315
pucrio		0.00147	**0.00125**
sensor—home-activity		0.00135	**0.00068**
sensor—CO-discretized		0.06655	**0.03226**
skin		0.00055	**3e-04**
tnelec		0.00496	**0.00376**
wisdm		**0.14576**	0.16293
A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.	**Unique Wins**	**7**	**12**
The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.	**Test Statistics**	**p-value: 0.17964**	**Confidence Interval: 0.41806 — 1**

Table 5.2: Real Streams, Shuffled: EFDT and HAT

Streams		EFDT	HAT
aws—price-discretized		**0.14139**	0.15319
chess		**0.60706**	0.67336
covertype		**0.27648**	0.28159
covpokelec		0.37454	**0.3386**
fonts		0.00068	**0.00062**
hhar		**0.05113**	0.06596
kdd		**0.00094**	0.00173
localization		**0.33857**	0.35673
miniboone		0.11875	**0.11652**
nbaiot		**0.02908**	0.33445
nswelec		**0.24057**	0.24114
pamap2		**0.15672**	0.218
poker		0.30909	**0.28636**
pucrio		**0.08329**	0.12773
sensor—home-activity		0.0955	**0.08867**
sensor—CO-discretized		0.2432	**0.22496**
skin		**0.01359**	0.01593
tnelec		**0.00586**	0.73151
wisdm		**0.13637**	0.19029
A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.	**Unique Wins**	**13**	**6**
The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.	**Test Statistics**	**p-value: 0.96822**	**Confidence Interval: 0.14747 — 1**

However, on a purely concept-drifting testbench, HAT has far superior prequential accuracy

performance (Table 5.3, p-value 0.00024), which is to be expected given its drift-specific plasticity.

Table 5.3: Synthetic Streams: EFDT and HAT

Streams		EFDT	HAT
recurrent—led		0.34679	**0.26767**
recurrent—randomtree		0.21817	**0.09415**
recurrent—sea		0.14958	**0.11169**
recurrent—stagger		0.19043	**0.00158**
recurrent—waveform		0.18957	**0.17667**
hyperplane—1		0.11677	**0.11503**
hyperplane—2		0.14056	**0.12429**
hyperplane—3		**0.11208**	0.11407
hyperplane—4		0.13384	**0.11399**
rbf—drift-1		**0.11255**	0.11806
rbf—drift-2		0.2623	**0.18264**
rbf—drift-3		**0.14308**	0.15343
rbf—drift-4		0.40597	**0.32701**
recurrent—abrupt—222		0.35381	**0.00052**
recurrent—abrupt—322		0.37596	**0.00055**
recurrent—abrupt—332		0.33376	**0.00132**
recurrent—abrupt—333		0.36583	**0.00295**
recurrent—abrupt—334		0.39001	**0.00904**
recurrent—abrupt—335		0.3945	**0.01264**
recurrent—abrupt—422		0.33569	**6e-04**
recurrent—abrupt—444		0.38967	**0.04815**
recurrent—abrupt—522		0.33374	**6e-04**
recurrent—abrupt—555		0.40866	**0.27943**
A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.	**Unique Wins**	**3**	**20**
The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.	**Test Statistics**	**p-value: 0.00024**	**Confidence Interval: 0.69636 — 1**

5.1.2 HAT and EFAT

Interestingly, EFAT—a tree with the eager splitting mechanism from EFDT and the drift adaptation mechanism from HAT—outperforms HAT on real streams whether shuffled (Table 5.4, p-value 0.02452) or unshuffled (Table 5.5, p-value 0.00636), as well as on synthetic streams with concept drift (Table 5.6, p-value 0.01734). That EFAT performs better than HAT on shuffled streams in spite of greater plasticity is interesting—this might be because useful structure is learned faster, thus deploying a higher quality classifier sooner.

Table 5.4: Real Streams: HAT and EFAT

Streams		HAT	EFAT
aws—price-discretized		0.14572	**0.13956**
chess		0.08866	**0.04397**
covertype		0.1808	**0.11259**
covpokelec		0.26455	**0.19816**
fonts		0.0053	**0.00147**
hhar		0.00519	**0.00132**
kdd		**0.00092**	0.00102
localization		0.059	**0.0469**
miniboone		***0.00011***	***0.00011***
nbaiot		0.00035	**0.00019**
nswelec		0.16772	**0.1418**
pamap2		**0.03138**	0.05903
poker		0.3315	**0.25468**
pucrio		**0.00125**	0.00127
sensor—home-activity		**0.00068**	0.00074
sensor—CO-discretized		0.03226	**0.02605**
skin		***3e-04***	***3e-04***
tnelec		0.00376	**0.00339**
wisdm		0.16293	**0.09262**
A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.	**Unique Wins**	**4**	**13**
The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.	**Test Statistics**	**p-value: 0.02452**	**Confidence Interval: 0.53945 — 1**

Table 5.5: Real Streams, Shuffled: HAT and EFAT

Streams		HAT	EFAT
aws—price-discretized		0.15319	**0.14962**
chess		0.67336	**0.60834**
covertype		0.28159	**0.27161**
covpokelec		0.3386	**0.30311**
fonts		***0.00062***	***0.00062***
hhar		**0.06596**	0.08036
kdd		**0.00173**	0.00186
localization		0.35673	**0.35505**
miniboone		0.11652	**0.10954**
nbaiot		0.33445	**0.00203**
nswelec		0.24114	**0.23722**
pamap2		0.218	**0.17819**
poker		**0.28636**	0.30697
pucrio		0.12773	**0.05015**
sensor—home-activity		0.08867	**0.03589**
sensor—CO-discretized		0.22496	**0.17362**
skin		0.01593	**0.00956**
tnelec		***0.73151***	***0.73151***
wisdm		0.19029	**0.12277**
A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.	**Unique Wins**	**3**	**14**
The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.	**Test Statistics**	**p-value: 0.00636**	**Confidence Interval: 0.60436 — 1**

Table 5.6: Synthetic Streams: HAT and EFAT

Streams		HAT	EFAT
recurrent—led		**0.26767**	0.26775
recurrent—randomtree		0.09415	**0.0855**
recurrent—sea		**0.11169**	0.11183
recurrent—stagger		0.00158	**0.00122**
recurrent—waveform		0.17667	**0.17475**
hyperplane—1		**0.11503**	0.12303
hyperplane—2		**0.12429**	0.13156
hyperplane—3		**0.11407**	0.12494
hyperplane—4		**0.11399**	0.12018
rbf—drift-1		0.11806	**0.11026**
rbf—drift-2		0.18264	**0.1586**
rbf—drift-3		0.15343	**0.14871**
rbf—drift-4		0.32701	**0.31819**
recurrent—abrupt—222		0.00052	**0.00049**
recurrent—abrupt—322		0.00055	**0.00053**
recurrent—abrupt—332		0.00132	**0.00094**
recurrent—abrupt—333		0.00295	**0.00137**
recurrent—abrupt—334		0.00904	**0.00273**
recurrent—abrupt—335		0.01264	**0.0053**
recurrent—abrupt—422		6e-04	**0.00051**
recurrent—abrupt—444		0.04815	**0.01342**
recurrent—abrupt—522		6e-04	**0.00051**
recurrent—abrupt—555		0.27943	**0.09666**
A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.	**Unique Wins**	**6**	**17**
The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.	**Test Statistics**	**p-value: 0.01734**	**Confidence Interval: 0.54902 — 1**

5.2 Future Work

While we make a start studying the interaction of eager splitting and change detection, detailed analysis of the combination of specified and unspecified strategies for both the case of individual and ensembled learners is left to future work. HAT as implemented has a tendency for exponential growth due to alternates being allowed their own alternates; but subtree replacement on the basis of unrelated alternate splitting reverses this tendency and adds plasticity. On the other hand, "re-splitting" may lead to degenerate paths and increased tree size—while simultaneously increasing drift adaptation. Overall, with the unspecified features, tree growth is difficult to predict... yet unspecified features interact in an intriguing, overall highly performant manner with respect to prequential accuracy, at least in the case of individual trees. Experiments with ensembles of HAT consumed far too much memory on our standard testbench, leading to experimental failure, especially so with eager splitting (we terminated our experimentation at a maximum of 128GB and a Java stack of 24GB); HAT/EFAT growth will have to be methodically addressed in future work.

HAT performs well on prequential accuracy in spite of the unspecified feature of discarding mainline trees and subtrees when the alternate is ready to split, long before a comparison of relative accuracy takes place.

It is plausible that for most tested scenarios, the concept in the mainline tree is already less effective than an alternate leaf by when the alternate is ready to split, and that it is thus usually suitable to discard the mainline tree when an alternate leaf is ready to split. Alternate leaves, unlike

child nodes, do not receive a class distribution. Perhaps that an alternate is ready to split signifies that it is more fruitful to discard the previous split at a given level and lose all of the structure underneath than aim to retain it—as long as the structure is quickly recovered through rapid tree growth.

An obvious next step involves creating testbenches that respond poorly to replacement of the main tree—we did not seem to quite achieve this in spite of our large testbench with recurrent drift and difficult-to-learn concepts. Clearly, designing a concept drift testbench is a problem that requires significant work. It might be the case that most problems that we are likely to encounter in the real world are addressed sufficiently well simply through bounded plasticity, though that is unlikely. We are building on top of a machine learning ecosystem designed to generalize well on stationary distributions. Our ecosystem of testbenches, evaluation methods, theoretical constructs and learning strategies is evolving, but slowly and incrementally. Given such an ecosystem, it is perhaps unsurprising in hindsight that increasing doses of plasticity in even uncontrolled and seemingly catastrophic ways helps learning under concept drift.

Ensembles that actively maintain both stable and plastic components might be a direction to continue to pursue. Naturally, the question of how many degrees of indirection are required arises; how do we decide what "stable" components store? Would we be "storing" concepts that we may be able to return to, in a "repository", as in [108]? If so, how many concepts do we store? If not, are we assuming that there are longer term features in the stream that are beneficial to model within a stable sub-component of the ensemble?

At this early stage of development, I believe it is premature to start optimizing for storage and retrieval of past knowledge in repositories, though it is clearly a promising direction to explore. We have yet to optimize prediction from the knowledge stored within "active" learning components—I believe there is ample scope for improvement of adaptive mechanisms in one-pass settings.

Chapter 6

Ensembles of EFDT

Is the sum of the parts...

VFDT remains the popular choice of base learner for random forests, and for boosting and bagging with online decision trees. Is EFDT a better choice?

In this chapter, we study the effects of short-term plasticity and long-term stability—as obtained from EFDT—in base learners for tree ensembles.

Noname manuscript No.
(will be inserted by the editor)

An Eager Splitting Strategy for Online Decision Trees

Chaitanya Manapragada · Heitor M Gomes · Mahsa Salehi · Albert Bifet · Geoffrey I Webb

Received: date / Accepted: date

Abstract **Keywords** Concept Drift · Hoeffding Tree · Explainability

We study the effectiveness of replacing the split strategy for the state-of-the-art online tree learner, Hoeffding Tree, with a rigorous but more eager splitting strategy. Our method, Hoeffding AnyTime Tree (HATT), uses the Hoeffding Test to determine whether the current best candidate split is superior to the current split, with the possibility of revision, while Hoeffding Tree aims to determine whether the top candidate is better than the second best and fixes it for all posterity. Our method converges to the ideal batch tree while Hoeffding Tree does not. Decision tree ensembles are widely used in practice, and in this work, we study the efficacy of HATT as a base learner for online bagging and online boosting ensembles. On UCI and synthetic streams, the success of Hoeffding AnyTime Tree in terms of prequential accuracy over Hoeffding Tree is established. HATT as a base learner component outperforms HT within a 0.05 significance level for the majority of tested ensembles on what we believe is the largest and most comprehensive set of testbenches in the online learning literature. Our results indicate that HATT is a superior alternative to Hoeffding Tree in a large number of ensemble settings.

1 Introduction

The traditional batch learning setting for machine learning is designed for finite datasets drawn from stationary distributions. Methods developed for learning from such datasets do not readily lend themselves to modern data processing applications dealing with streams of data where instances arrive continuously, generated by processes that may themselves be ever-changing. It is necessary to design new algorithms for learning from such settings, and a good place to start is from algorithms designed for batch settings.

Decision Trees are ubiquitous in batch learning settings, both as individual learners and in multiple ensembled forms such as Random Forests. Attempts at producing online versions of decision trees largely dominate work in online learning. *Hoeffding Tree* (HT) [17] has been the mainstay of online decision tree learning methods since its introduction in 2000 by Domingos &

Chaitanya Manapragada, Geoff Webb, Mahsa Salehi
Monash University, Australia
E-mail: FirstName.LastName@monash.edu

Albert Bifet, Heitor Murilo Gomes
University of Waikato, New Zealand
E-mail: FirstName.LastName@waikato.ac.nz

Hulten. To the best of our knowledge, it is the first attempt at incremental tree learning that provides guarantees of bounded divergence from a theoretical batch tree that has all examples in an infinite stream available to it at once. These guarantees are useful in that they enable us to compare theoretical performance with respect to longstanding, reliable decision tree methods. However, such guarantees often assume each data instance corresponds to a random variable from a stationary process that generates independent and identically distributed ($i.i.d$) random variables. This is usually not a valid assumption in streaming scenarios in which processes can change over time, and working around this assumption would require placing restrictions on the nature of change of the generating process. Such restrictions may be meaningful in the context of particular, well-studied processes where the nature of change is fully known.

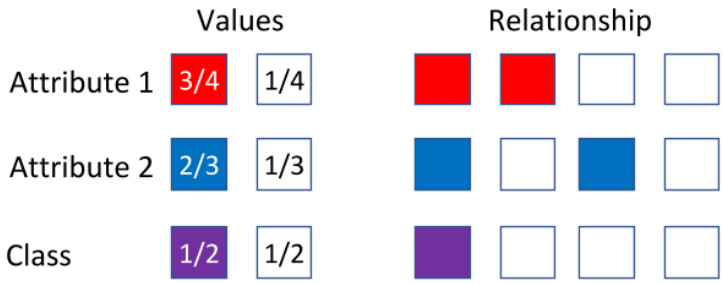
In the batch learning scenario, all target values are available at the outset, barring missing values. In the online setting, there is expected to be an infinite stream of instances and thus storage is considered impossible in the limit. Incremental tree learners are thus typically one-pass learners, in that they process each training instance exactly once. And because they are one-pass learners, incremental learners naturally allow for continuous evaluation; continuous evaluation of learning enables us, among other things, to detect or otherwise respond to concept drift in streams and adapt the learner accordingly.

Hoeffding Tree has dominated the development of incremental learning, winning a KDD Test of Time award [49] in 2015. It uses a statistical test—the Hoeffding Test— that is known to possibly lead to loose bounds, that is, the bound on the difference between the observed mean and the true mean is larger than would have been obtained with a tighter bound on the standard deviation. R, the range (or support) of the random variable may be considered too large an upper bound for the standard deviation in cases where variance is small [55]. The population of random variables[1] consists of the difference of cumulatively averaged information gain measurements of the top two split attributes after each learning step; a loose bound implies that a far larger number of such measurements is needed than actually required to establish the winning attribute [25]. Because the bound depends on the size of the population of random variables, which in turn depends directly on the number of observations of training instances, greater statistical efficiency (see Figure 1) may be achieved by either using an alternative test or by changing the application of the test. Our strategy, Hoeffding AnyTime Tree (HATT) [34], is an example of the latter.

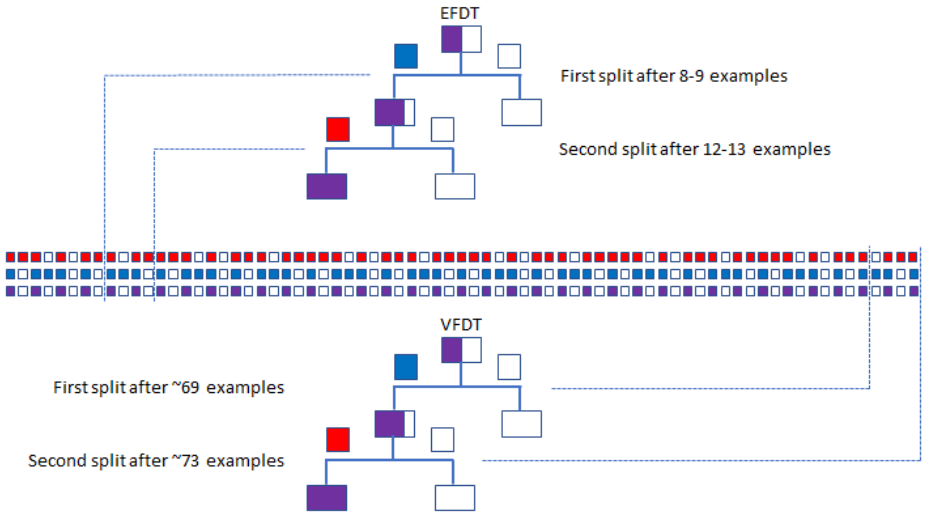
Our focus in this work is on online classification tasks, though our method can also be used for regression. In 2018, we presented preliminary results based on a simple and fundamental change [34]: HATT uses the current best available split attribute until a better one is found, as opposed to Hoeffding Tree which aims to find a split attribute that will never have to be replaced. HATT converges to the ideal batch tree while Hoeffding Tree does not. Our implementation of HATT, Extremely Fast Decision Tree (EFDT), was demonstrated to offer significant improvement in prequential accuracy[2] on UCI data streams over the Hoeffding Tree implementation Very Fast Decision Tree (VFDT) from the MOA toolkit [5] in a comparison setting that included only a minor change to VFDT in order to obtain EFDT. The current state-of-art tree-based online systems are ensembles that use VFDT as a base learner. In this work we provide a detailed assessment of the relative performance of VFDT and EFDT as base learners for online ensembling techniques.

[1] There is a common misconception that an individual random variable "changes, taking on a number of values during a process"; in fact, a process is a sequence of events, each of which corresponds to an individual random variable that has taken a particular value (which is fixed and never to change).

[2] In the prequential setting, training instances arrive in a sequence, and the true target value pertaining to each training instance is made available after the predictor has offered a prediction for a sequence of n instances. The loss function applied is necessarily incremental in nature. Choosing $n = 1$ — that is, updating the predictor after every instance—is an obvious transformation of a periodic evaluation process into an instantaneous one. While not typical of real world application scenarios, prequential accuracy serves as a useful approaximation thereto.



(a) Data Types — Data with Attribute 1 = Red and Attribute 2 = Blue are classified as Purple. All other data are classified as White. Attributes are independent.
Distribution of Data — Attribute 1 is red 3/4 of the time, Attribute 2 is blue 2/3 of the time, so each class occurs 1/2 the time.



(b) Streaming Example with Tree Construction.

Fig. 1: On a randomized stream generated from the data schema and distribution shown above in (a), EFDT first splits after 8-9 examples, then splits again after 12-13 examples to build the correct tree. VFDT takes around 69 examples for the first split, and makes the second split at around 73 examples. EFDT greatly increases statistical efficiency without compromising the use of a rigorous statistical test to determine split attributes, and revises splits in order to converge to the ideal batch tree.

Hoeffding Tree is the base learner for online versions of widely successful ensembling methods such as random forests, bagging and boosting. A host of derivative online ensemble methods have been proposed in the literature; the methods vary in the diversity of base learners they use, the the adaptability of the ensemble as a whole and of the component base learners. It is imperative that a proposed method works well in ensemble settings as well as standalone, as likely usage will often involve ensemble learners.

The ensemble methods we test generally far outperform plain EFDT and VFDT in terms of prequential accuracy, which is to be expected; bagging approaches specifically utilize diversity in order to aim to "stabilize" the predictive model, while boosting approaches modify the model in a principled manner to focus on learning misclassified examples. Generally speaking, the success of ensembling strategies on this testbench and in practice in the real world demonstrates that tree learners may suffer from high variance and a small considered hypothesis space (bias) when used as individual classifiers. Further, it should be noted that a formal definition of overfitting or underfitting is difficult to obtain in an online setting that may include concept drift unless the nature and magnitude of drift is bounded and fixed in a highly restrictive manner.

It is the prerogative of the user to pick an ensemble method that suits the application of interest; a corollary of the No Free Lunch Theorems is that no individual system will obtain superior performance on all possible problems [57, 58, 59]. With this proviso, in this work, we show that on the most common testbenches, it is highly beneficial to use HATT as a base classifier for ensemble methods in the place of Hoeffding Tree in terms of prequential accuracy.

The main contributions of this paper are:

1. A comprehensive experimental analysis comparing EFDT and VFDT as base learners for a large set of online ensembles across a diverse set of real and synthetic streams, building on our preliminary work in [34]
2. Identification of scenarios in which EFDT has a definitive advantage with high confidence
3. A working hypothesis for the observed outperformance of EFDT (Section 3.4)
4. Discussion of how change detection mechanisms may interact adversely with ensemble components

This paper is organized as follows: Section 2 presents a general overview of the context of stream learning within which this work is placed. Section 3 presents HATT. This is largely similar to our previous presentation in [34], but with the addition of a new subsection on HATT in ensembles (Section 3.4), the primary focus of this paper. Section 4 describes our experimental setup. Section 5 is a discussion of experimental results in terms of the types of ensembles and settings in which HATT is advantageous as a base learner. Section 6 summarizes our findings.

2 Background

2.1 Decision Trees for Batch Learning

Of the many approaches to inductive learning, Decision Trees are a particularly utile paradigm that store knowledge in an easily interpretable manner. Algorithms that build decision tree models recursively divide the sample space with hyperplane decision boundaries. Each division represents a conditioning of the sample space on a particular set of data attribute values or ranges. The knowledge obtained from a Decision Tree is represented in an elementary form; in the classification case, each path down the tree results in a conditional probability distribution $P(C|X_1 = v_1, X_2 = v_2, ...)$, that is, the probability distribution of the class values given the observations $X_1 = v_1$, $X_2 = v_2, ...$ The regression case may use, for instance, a simple average of observed target values.

Decision Trees are a natural starting point for the study of extending inductive strategies to streaming scenarios; their simplicity allows us to compute model complexity [37], and they are highly interpretable. Concept Learning System (CLS) [29] was one of the first decision tree algorithms, published in 1966. The next decision tree system of note was Iterative Dichotomizer (ID3), published in 1979 [39, 40]. ID3 introduced the idea of using information gain as a split heuristic, though it was limited to handling binary classification and assumed all instances were correctly labelled.

"Classification and Regression Trees" (CART), proposed in 1984, included pruning to adjust for overfitting, multiclass classification, and a solution for regression [11]. CART used the Gini coefficient as a heuristic, not Information Gain. Consequently, a major improvement over ID3 called C4.5 that also addressed pruning, multiclass classification and regression was released [41]. The ideas embedded in C4.5 and CART form the basis of most modern decision tree learning systems today designed for both batch and streaming scenarios.

2.2 Incremental Decision Trees

Work on incremental decision trees began appearing just as batch decision trees started to mature. ID4 [46] was an incremental extension to ID3 that stored instances used for a split at each level of the tree. ID4 was conceived for a binary classification problem; the test of choice is the χ^2 test to determine whether the attribute with maximal separation power is independent of the target variable. When confidence of dependence is reached with the χ^2 test, the maximal attribute is split upon. The limitation of ID4 was that storage required was in the order of the number of instances.

Storage is a primary issue to address in the construction of an incremental supervised learning, because streams may be assumed to be indefinite. Another key problem is determining when and how the algorithm should modify the model—unlike with batch learning, one does not have all training instances available in one go and must periodically decide whether one has enough data to modify the decision tree model with some degree of confidence.

Strategies that process each instance and discard it immediately afterwards—*one-pass* strategies—would hypothetically address the problem of storage by only requiring storage of the order of the size of the tree, not the number of instances.

HoeffdingTree (Algorithm 2.1) was one of several attempts [46, 53] to provide a one-pass solution. Its success may be attributed to the fact that it was the first one-pass learner to also offer a robust solution to the problem of how the algorithm should modify the tree model. This robustness lies in the guarantees provided by the HoeffdingTree algorithm on the deviation of the inducted tree from the ideal batch tree—the hypothetical tree that would be learned if all infinite examples from a stationary distribution were made available at once. Hoeffding Tree uses a statistical test—the Hoeffding Test [17, 26]—to determine the most appropriate time to split. Its success may be attributed to the fact that it provided both a one-pass solution and deviation guarantees in the same package.

The ideas that underlie HoeffdingTree were individually and independently developed in related contexts. Work on scalability of batch learners had helped set the foundation for one-pass learning in sequential prediction scenarios. Bootstrapped Optimistic Algorithm for Tree construction (BOAT) [19] in particular lays the groundwork of ideas for a tree refined in stages, though it deals with a batch setting. BOAT represents a typical attempt at learning from a large database that does not use a predictive sequential setting, by sampling fixed size chunks that are used to bootstrap multiple trees. A "coarse" tree is then extracted, based on the overlapping parts of the bootstrapped trees in terms of split decisions; this tree is further refined to produce

Algorithm 2.1: Hoeffding Tree, Domingos & Hulten (2000) –Reproduced verbatim from original–

Input: S, a sequence of examples,
$\mathbf{X}$, a set of discrete attributes,
G(.), a split evaluation function
δ, one minus the desired probability of choosing the correct attribute at any given node
Output: HT, a decision tree.
begin
  Let HT be a tree with a single leaf l_1 (the *root*).
  Let $\mathbf{X_1} = \mathbf{X} \cup X_\emptyset$.
  Let $\overline{G_1}(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class in S
  foreach *class* y_k **do**
    foreach *value* x_{ij} *of each attribute* $X_i \in \mathbf{X}$ **do**
      Let $n_{ijk}(l_1) = 0$
    end
  end
  foreach *example* $(\mathbf{x}, y)$ *in* S **do**
    Sort $(\mathbf{x}, y)$ into a leaf l using HT
    foreach x_{ij} *in* $\mathbf{x}$ *such that* $X_i \in \mathbf{X}_l$ **do**
      Increment $n_{ijk}(l)$
    end
    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 $\overline{G_l}(X_i)$ for each attribute $X_i \in \mathbf{X}_l - \{X_\emptyset\}$ using the counts $n_{ijk}(l)$
      Let X_a be the attribute with highest $\overline{G_l}$
      Let X_b be the attribute with second-highest $\overline{G_l}$
      Compute ϵ using Equation **??**
      if $\overline{G_l}(X_a) - \overline{G_l}(X_b) > \epsilon$ *and* $X_a \neq X_\emptyset$ **then**
        Replace l by an internal node that splits on X_a **foreach** *branch of the split* **do**
          Add a new leaf l_m and let $\mathbf{X}_m = \mathbf{X} - \{X_\emptyset\}$ Let $\overline{G}_m(X_\emptyset)$ be the $\overline{G}$ obtained by predicting the most frequent class at l_m **foreach** *class* y_k *and each value* X_{ij} *of each attribute* $X_i \in \mathbf{X}_m - \{X_\emptyset\}]\}$ **do**
            Let $n_{ijk}(l_m) = 0$
          end
        end
      end
    end
  end
  Return HT
end

a final tree by passing the whole dataset over it. The system is "incremental" in the sense that it can process additional datasets; and it is responsive to drift in that the system detects when a new dataset requires a change in split criterion at a node through a global assessment of split criterion, and causes a rebuild of the subtree rooted at that node. While key ideas that shape later trees are developed in this work, the sizes of the initial bootstrap samples are arbitrarily chosen, and concurrently the notion of anytime prediction is not entertained—there is no automated way of determining how many examples suffice to build a first reliable tree. Further, the focus is on minimizing utilization of main memory; it is assumed that the database D is available for a corrective step in the algorithm. On the other hand, Hoeffding Tree is truly one-pass, in that it is assumed that an example is seen only once, then discarded. Meanwhile, the RainForest framework [20] introduces the idea of storing attribute-value-class counts at nodes, which we see in Hoeffding Tree as *node statistics*. Node statistics are indispensable in HoeffdingTree; they solve the one-pass problem and are critical in the application of the Hoeffding Test for split evaluation.

Hoeffding Tree may be considered to be aiming to achieve the same evaluation objective as CART and C4.5—maximal class purity—with a heuristic designed to be relevant in the streaming scenario where the data generation distribution is assumed to be static.

2.3 Ensembles

Ensemble methods were proposed as a means of utilizing multiple predictors to represent and combine various parts of the instance space, for various heuristic reasons—using a set of base predictors may reduce the risk of building an overfitted individual predictor; reducing the complexity of each individual base predictor may also serve to reduce risk from overfitting [16]. Further, using multiple base predictors may allow us to achieve a rather different bias-variance profile compared to using a single predictor.

Bootstrap aggregation- or "bagging" ensembles were proposed for stability, that is, to limit variance [12]. By "perturbing" the learning set for each component of an ensemble of predictors, Breiman creates predictors that effectively represent a larger bias (the set of hypothesis functions being considered) through combination. Assuming then a stationary distribution over the instance space, the ensemble is likely to demonstrate lower dataset dependent variance than a single tree. The objective for bagging differs from boosting in its addressal of controlling variance, and the accompanying heuristic follows.

Boosting was first proposed as an approach to answer affirmatively the question of whether a combination of weak predictors—predictors that find hypotheses that perform only slightly better than random guessing—can learn strongly, that is, are able to find hypotheses "that are correct with high probability on all but an arbitrarily small fraction of instances" [45]. In the classification setting, boosting works by using some components of the ensemble to focus on weighting misclassified examples higher so they are preferentially learned with respect to their actual frequency of occurrence, and thus potentially classified better on a test set if they were not merely noisy instances. In practice, a major advantage is that ensemble components need to be less complex (thus less prone to individually overfit). The objective is to reduce the possibility of overfitting, and the heuristic is the idea that using individual models of lower complexity will mitigate the danger of overfitting while remaining effective as constituent ensemble components would have learned subconcepts that would otherwise have been learned by a single complex model.

Online versions of both boosting (OzaBoost) and bagging (OzaBag) were proposed in [38]. Hoeffding Tree versions of the ensembles were made available in MOA [5].

Because the online versions assume infinite streams, strategies to sample the input space in a manner equivalent to sampling from a finite sample space had to be devised. Online bagging as performed by OzaBag achieves the goal of providing each learner with a different subset of the sample space by weighting each example with a value drawn from a Poisson(1) distribution. Similarly, online boosting as performed by OzaBoost provides the first base predictor an example weighted as $Pois(\lambda = 1)$; the second base predictor receives weighted examples with λ adjusted so that misclassified examples comprise half the total weight, thus heavily stressing the learning of misclassified examples. This telescoping sequence of boosting misclassified examples is continued through the predictors in the ensemble.

While ensembles do tend to be more expensive in terms of memory and time than individual predictors, their ability to provide diversity particularly motivates their usage with evolving streams. As discussed, predictor diversity is central to bagging approaches; it plays a role in boosting approaches; in evolving scenarios, some base predictors may learn the latest versions of a changing concept, some may preserve portions of a concept that may recur, and some

predictors could be reset if they have not been of utility for a prolonged period. Note that there also exists the possibility that an ensemble approach may degenerate its base predictors into a set of redundant models [22].

3 Hoeffding AnyTime Tree (HATT)

Algorithm 3.1: Hoeffding Anytime Tree

Input: S, a sequence of examples. At time t, the observed sequence is $S^t = ((\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ...(\mathbf{x}_t, y_t))$
$\mathbf{X} = \{X_1, X_2...X_m\}$, a set of m attributes
δ, the acceptable probability of choosing the wrong split attribute at a given node
G(.), a split evaluation function
Result: $HATT^t$, 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 ∪ X_∅
    Let G_1(X_∅) 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 ∈ X do
            Set counter n_ijk(root) = 0
        end
    end
    foreach example (x, y) in S do
        Sort (x, y) into a leaf l using HATT
        foreach node in path (root...l) do
            foreach x_ij in x such that X_i ∈ X_node do
                Increment n_ijk(node)
                if node = l then
                    AttemptToSplit(l)
                else
                    ReEvaluateBestSplit(node)
                end
            end
        end
    end
end
```

Hoeffding AnyTime Tree (Algorithm 3.1) [34] makes a simple change to the attribute selection mechanism of Hoeffding Tree and achieves better performance on a large testbench. Given the streaming setting, the objective underwent a subtle change; convergence to the ideal batch tree was the aim, as divergence from the ideal batch tree grows as the number of leaves in HoeffdingTree, which may plausibly be expected to see exponential growth, invalidating it in the general case. Assuming the convergence objective can be met, a desired augmentation to the objective is statistical efficiency on the way to convergence. A strategy for convergence in a streaming scenario would necessarily re-evaluate the model; HATT improvises a split evaluation heuristic more conducive to statistical efficiency and eventual convergence than to maximizing confidence in historical splits as does HoeffdingTree.

As previously mentioned, in the classification setting, decision trees are grown by repetitively dividing the instance space and assigning a class (and possibly a probability distribution over classes) to each subspace. Where all training instances are available at the outset, as in batch learning, decision trees evaluate all available features (attributes) in order to decide which attribute should first be utilized to divide the space. According to the split heuristics we have

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 Ḡ_l(X_i) for each attribute X_l − {X_∅} using the counts n_ijk(l)
        Let X_a be the attribute with the highest Ḡ_l
        Let X_b = X_∅
        Compute ε using equation ??
        if Ḡ_l(X_a) − Ḡ_l(X_b) > ε and X_a ≠ X_∅ 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 Ḡ_m(X_∅) 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 ∈ X_m − {X_∅} do
                    Let n_ijk(l_m) = 0.
                end
            end
        end
    end
end
```

Function 3.3: ReEvaluateBestSplit(internalNode int)

```
begin
    Compute Ḡ_int(X_i) for each attribute X_int − {X_∅} using the counts n_ijk(int)
    Let X_a be the attribute with the highest Ḡ_int
    Let X_current be the current split attribute
    Compute ε using equation ??
    if Ḡ_l(X_a) − Ḡ_l(X_current) > ε then
        if X_a = X_∅ then
            Replace internal node int with a leaf (kills subtree)
        else if X_a ≠ X_current then
            Replace 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 Ḡ_m(X_∅) 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 ∈ X_m − {X_∅} do
                    Let n_ijk(l_m) = 0.
                end
            end
        end
    end
end
```

discussed, the best attribute is one that maximizes increase in class purity based on some heuristic measuring class purity such as Information Gain or the Gini Coefficient. For example, if we have available the nominal features "can-swim" and "is-a-mammal", the latter feature would perfectly classify sharks and dolphins, splitting the instance space into two parts assuming no noise in the data, while the former attribute provides no information, as both sharks and dolphins swim. Now suppose we wish to further classify the creatures as certain shark or dolphin species; we may use other attributes to recursively split the instance space.

Where all training instances are not available at the outset, as in the case of online learning, it becomes necessary to determine when and how split the instance space. Hoeffding Tree addresses the problem by aiming to attain statistical confidence (using the Hoeffding Test) in the top

attribute outperforming the second best attribute at each split decision. However, attaining this confidence is time-consuming. HATT utilizes information available before attaining this level of confidence by using the Hoeffding Test to determine whether a candidate attribute is better than the current split, rather than the second best attribute. If there is currently no split and a candidate split attribute tests to be significantly better, it is split upon; this decision may be revised as necessary. The net result is that in the stationary (*i.i.d*) setting that Hoeffding Tree assumes, HATT converges to the ideal tree that would have been obtained in a hypothetical batch setting that processes an infinity of training examples at once.

3.1 Convergence

Hoeffding Tree probabilistically bounds deviation from the ideal batch tree. It guarantees that the expected "extensional" or "intensional" disagreement from a batch tree are each independently bound by $\frac{\delta}{p}$, where δ is a tolerance level and p is the leaf probability– the probability that an example will fall into a leaf at a given level. "Extensional disagreement" is defined as the probability that a pair of decision trees will produce different predictions for a given example, and "intensional disagreement" is the probability that the path of an example will differ on the two trees [17]. It is assumed that these guarantees hold in the limit when trained on an infinite dataset denoted DT_*. p is assumed to be constant across all levels for simplicity.

Note that the guarantees will weaken significantly as the number of leaves increase, tending to probability bound 1 as tree size increases. Increasing the complexity and size of data streams such that a larger tree is required increases the chance of a greater deviation. This limits the utility of guarantees for Hoeffding Tree to lower dimensional scenarios, though the trees themselves may be highly useful.

HATT on the other hand, converges in probability to the batch decision tree under the following assumptions:

- No two attributes will have identical information gain. This is a simplifying assumption to ensure that we can always split given enough examples, because ϵ is monotonically decreasing.
- The data are independently and identically distributed (*i.i.d*)

Given these assumptions, we list three useful properties of HATT that follow as lemmas; the proofs are presented in the "Extremely Fast Decision Tree"work that introduces HATT [34].

Lemma 1 *For any input stream S, HATT learned from S will have the same split attribute at the root as HT learned from S at the time HT splits the root node.*

Lemma 2 *If the input stream S is i.i.d, the split attribute X_R^{HATT} at the root node of HATT converges in probability to the split attribute $X_R^{DT_*}$ 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_R^{HATT} that matches the split $X_R^{DT_*}$ at the root node of DT_* goes to 1.*

Lemma 3 *If the input stream S is i.i.d, Hoeffding AnyTime Tree converges to the asymptotic batch tree in probability.*

3.2 Time and Space Complexity

Space Complexity: As detailed in [34], on nominal data with d attributes, v values per attribute, and c classes, HATT requires $O(dvc)$ memory to store node statistics at each node, as does HT [17]. The worst case space complexity is $O(v^{d-1}dvc)$, because there may be a maximum of

$(1 - v^d)/(1 - v)$ nodes due to geometric tree growth. The space complexity for HT is given as $O(ldvc)$ in [17], where l is the current number of leaves; for HATT, the space complexity is $O(ndvc)$, where n is the current total number of nodes, including internal nodes. Because l is $O(n)$, space complexity is equivalent for HATT and HT.

In the ensemble setting, space complexity is simply multiplied by the maximum ensemble size m to obtain $O(mndvc)$. Ensembles which use change detectors add to the space complexity by maintaining a window (this is independent of whether the base learner is HT or HATT). The change detector used in such ensembles in our experiments is ADWIN, the default MOA option, which maintains a self-adjusting window of prediction errors. ADWIN is parameterized with a parameter M that determines maximum window size as $W = M \times \sum_{i=0}^{M} 2^i$. Each ADWIN instance uses $O(Mlog(W/M))$ memory [4], and one instance is used per ensemble component, so the space complexity added by ensembles that use ADWIN is $O(mMlog(W/M))$.

Time Complexity: Again, as detailed in [34], the worst-case cost of both split evaluation and updating node statistics while processing an example are $O(dvc)$ for HT and $O(hdvc)$ for HATT, where h is the maximum height of the tree. One leaf (for HT) or one path (for HATT) have to be evaluated for splits and have their node statistics updated.

In the ensemble setting, time complexity is also simply multiplied by maximum ensemble size m, giving us $O(mdvc)$ for HT and $O(mhdvc)$ for HATT for both split evaluation and updating node statistics. Ensembles that use ADWIN add an additional amortized cost of $O(1)$ and worst case cost of $O(W)$ for processing each example per ADWIN instance, which evaluates to $O(m)$ amortized and $O(mW)$ worst case, as one ADWIN instance is used per ensemble component.

3.3 HATT in the context of concept drift

Hoeffding Tree is surprisingly responsive to concept drift given that it is designed for learning from stationary distributions and has no capacity to revise internal nodes once they have been added to the model. A major reason for this responsiveness is that when a split is created, the new leaves that are created start with no memory of the examples seen by the learner previously. As a result, if a split occurs after drift, the new leaves start with a fresh slate and learns the new distribution [35].

HATT has the same property of new leaves starting with a fresh slate. However, it has the additional properties of—

1. forming new branches more readily (when it has evidence that a specific branch is better than none, rather than better than any potential alternative) and
2. replacing internal nodes with new splits when the new split becomes better than the existing one.

The first of these properties enhances the speed with which HATT adjusts to drift, as it will develop new leaves reflecting a new distribution more rapidly. However, the second has both positive and negative aspects in the context of drift. Replacing an internal node can potentially remove a large section of the model that reflects the old distribution and allow it to be replaced by a new subtree that will grow to reflect the new distribution. However, the closer to the root a node is, the more evidence of older distributions will be retained in its node statistics. Hence, the closer to the root a node is, the longer it is likely to take for the evidence to grow sufficient to replace it. As a result, by the time an internal node is replaced following a drift, the subtrees below it are likely to have already had time to adjust to the new distribution, and hence replacing them may be detrimental in the near term until a new tree can be grown to replace them.

3.4 HATT in the Ensemble Setting

The choice of base predictor is important and Hoeffding Tree has been the mainstay of online ensemble learning. It has been argued that uncorrelated predictions are important for error reduction effect [12]; consequently that learner diversity is key to uncorrelated predictions [31], implying that unstable learners are most suited as base learners for ensembles, so that small changes to the stream can cause significant changes to the base models, creating diversity in the ensemble. However, Hoeffding Tree is a stable learner [23]; that is, being provided slightly different versions of a stream does not greatly alter the decision tree model produced, on account of the Hoeffding Test that is used to decide each split with statistical confidence. Hoeffding AnyTime Tree can be argued to be perfectly stable in the long run—no matter the sequence in which the input is provided, it will converge to the ideal predictor as $t \to \infty$—however, it is far less stable than Hoeffding Tree in the short term on account of its incidentally adaptive nature as a result of constantly readjusting split decisions. This short term reduction in stability in HATT is a plausible reason to expect more accurate ensembles when it is used as a base predictor in ensembles in place of Hoeffding Tree.

4 Experimental Setup

We work with the MOA framework [5], which provides implementations of common ensembling strategies for decision trees, including Hoeffding Tree. Each ensemble method comprises either a bagging or boosting component, may involve a change detector and may also weight instances. Change detectors are usually used to determine whether to kill off the worst performing trees in order to replace them with a new one if a significant increase in the error is observed.

We use a testbench that to our best knowledge is the largest and most comprehensive in the literature, though there is ample room for improvement. We test on UCI datasets that are mostly drawn from real data as well as on synthetic streams with concept drift. We carefully selected twenty real datasets: two electricity datasets and the airlines and AWS dataset that are widely used in the concept drift literature, and as for the rest, the largest datasets from the UCI repository that involved an obvious classification task and had no missing values in order to reduce the confounding factor of how the algorithms handle missing values. We omitted the physics simulation datasets Hepmass, SUSY and Higgs as these are synthetic datasets with low information [34]. The price variable in the AWS dataset was discretized into ten equal buckets and set as the classification target; similarly, the CO concentration variable in the sensor-CO dataset was discretized into 5 equal buckets and set as the classification target. The datasets we use and their key features are listed in Table 1.

UCI datasets are often ordered on some basis. As such, when processed sequentially in their native order, they provide a non-stationary data stream. We use them in this way. To assess performance on real world data in the absence of concept drift, we also use shuffled versions of each dataset. This ensures a stationary distribution. Our shuffled runs on real data use 10 randomised shuffles of each stream with fixed seeds so the experiments are reproducible.

In order to assess ensembling strategies with VFDT and EFDT as base learners under high levels of concept drift, we chose a synthetic testbench that demonstrates noticeable differences in performance across different parametrizations. These synthetic streams and their parametrizations are listed in Table 2.

Hyperplane [28] and Radial Basis Function (RBF) [7] generators draw examples from a naturally evolving concept and allow exploration of the interactions between varying rates of drift and dimensionalities.

Table 1: Properties of Real Datasets

	Dataset	Instances	Attributes (Numeric, Nominal)	Classes
1	airlines [9]	539383	8 (3, 5)	2
2	aws—price-discretized [54]	27410309	7 (4, 3)	10
3	chess [18]	28056	6(3, 3)	18
4	covtype [10, 18]	581012	54 (10, 44)	7
5	cpe [8, 7]	1455525	72 (22, 50)	10
6	fonts [33, 18]	745000	411 (410, 1)	153
7	hhar [50, 18]	43930257	9 (6, 3)	6
8	kdd [18]	4000000	42 (34, 8)	23
9	localization [30, 18]	164860	8 (4, 4)	11
10	miniboone [43, 18]	130065	50 (50, 0)	2
11	nbaiot [36, 18]	7062606	115 (115, 0)	11
12	nswelec [24, 18]	45312	9 (7, 2)	2
13	pamap2 [42, 18]	3850505	53 (53,0)	25
14	poker [18]	1025010	10 (5, 5)	10
15	pucrio [52, 18]	165632	18 (15, 3)	5
16	sensor—home-activity [27, 18]	919438	11 (11, 0)	3
17	sensor—CO-discretized [13, 14, 18]	4095000	19 (19, 0)	5
18	skin [3, 18]	245057	3 (3, 0)	2
19	tnelec [18]	45781	4 (2, 2)	20
20	wisdm [32, 18]	15630426	44 (43, 1)	6

Our Recurrent AbruptDrift generator tweaks the AbruptDrift generator from [56] to add the option of generating recurrent abrupt drifts. It models a full conditional probability table for the target distribution $p_{Y|X}$ that grows exponentially in the number of input variables. Learning problems generated by this generator become significantly harder with small increases in dimensionality, possibly because of the relative independence of the variables. The suffix notation in the shorthand "recurrent—abrupt—522" used in tables conveys that that particular stream has 5 classes, 2 nominal attributes, and 2 values per attribute.

We use the RecurrentConceptDriftStream generator [5] to generate drift between different parametrizations of synthetic streams commonly used in the concept drift literature that do not have in-built drift generation; these are the Agrawal [1], LED [11], RandomTree [5], SEA [51], STAGGER [47], and Waveform [11] generators.

All synthetic streams generate one million examples. Each synthetic stream generator is run with 10 differently initialized random seeds. Now, prequential accuracy is a curve that plots error at every timestep, so the results are reported thus: a measure of the mean error is obtained by averaging error across seeded runs for each epoch, and then averaging over the epoch-wise error averages. A measure of variance is obtained by computing the variance of the error across the seeded runs for each epoch, then averaging the epoch-wise variances. Note that the error reported for each epoch is itself the average across a thousand examples, which is the default for the evaluator in MOA. The number of leaves, and the CPU time, are simply the averages of the final values across the seeded runs for a given stream.

All ensembles were requisitioned with ten trees each; this was about the limit on a fairly large university cluster that made available 60 CPUs at a time for embarrassingly parallel workloads. Ensembles were otherwise parameterized with their MOA defaults. The boosting ensembles we use are: OzaBoost [38], OzaBoost with the ADWIN change detector (OzaBoostADWIN) [4], OnlineSmoothBoost [48, 15], Adaptable Diversity-based Online Boosting (ADOB) [44], and Boosting-Like Online Ensemble (BOLE) [2]. The bagging ensembles in our experiments are:

Table 2: Synthetic Datasets

	MOA Stream	Shorthand
1	-s (RecurrentConceptDriftStream -x 200000 -y 200000 -z 100 -s (generators.AgrawalGenerator -f 2 -i 2) -d (generators.AgrawalGenerator -f 3 -i 3))	recurrent—agrawal
2	-s (RecurrentConceptDriftStream -x 200000 -y 200000 -z 100 -s (generators.LEDGenerator -i 2) -d (generators.LEDGeneratorDrift -i 3 -d 7))	recurrent—led
3	-s (RecurrentConceptDriftStream -x 200000 -y 200000 -z 100 -s (generators.RandomTreeGenerator -r 1 -i 1) -d (generators.RandomTreeGenerator -r 2 -i 2))	recurrent—randomtree
4	-s (RecurrentConceptDriftStream -x 200000 -y 200000 -z 100 -s (generators.SEAGenerator -f 2 -i 2) -d (generators.SEAGenerator -f 3 -i 3))	recurrent—sea
5	-s (RecurrentConceptDriftStream -x 200000 -y 200000 -z 100 -s (generators.STAGGERGenerator -i 2 -f 2) -d (generators.STAGGERGenerator -i 3 -f 3))	recurrent—stagger
6	-s (RecurrentConceptDriftStream -x 200000 -y 200000 -z 100 -s (generators.WaveformGenerator -i 2 -n) -d (generators.WaveformGeneratorDrift -i 3 -d 40 -n))	recurrent—waveform
7	-s (generators.HyperplaneGenerator -k 10 -t 0.0001 -i 2)	hyperplane—1
8	-s (generators.HyperplaneGenerator -k 10 -t 0.001 -i 2)	hyperplane—2
9	-s (generators.HyperplaneGenerator -k 5 -t 0.0001 -i 2)	hyperplane—3
10	-s (generators.HyperplaneGenerator -k 5 -t 0.001 -i 2)	hyperplane—4
11	-s (generators.RandomRBFGeneratorDrift -s 0.0001 -k 10 -i 2 -r 2)	rbf—drift-1
12	-s (generators.RandomRBFGeneratorDrift -s 0.0001 -k 50 -i 2 -r 2)	rbf—drift-2
13	-s (generators.RandomRBFGeneratorDrift -s 0.001 -k 10 -i 2 -r 2)	rbf—drift-3
14	-s (generators.RandomRBFGeneratorDrift -s 0.001 -k 50 -i 2 -r 2)	rbf—drift-4
15	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 2 -n 2 -v 2 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—222
16	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 3 -n 2 -v 2 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—322
17	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 3 -n 3 -v 2 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—332
18	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 3 -n 3 -v 3 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—333
19	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 3 -n 3 -v 4 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—334
20	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 3 -n 3 -v 5 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—335
21	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 4 -n 2 -v 2 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—422
22	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 4 -n 4 -v 4 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—444
23	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 5 -n 2 -v 2 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—522
24	-s (generators.monash.AbruptDriftGenerator -c -o 1.0 -z 5 -n 5 -v 5 -r 2 -b 200000 -d Recurrent)	recurrent—abrupt—555

OzaBag [38], OzaBag with ADWIN (OzaBagADWIN), Leveraging Bagging (LevBag) [6], Leveraging Bagging without ADWIN (LevBagNoADWIN), and Adaptive Random Forest [21]. Note that OzaBag and OzaBoost are two state-of-the-art online bagging and boosting algorithms—their core mechanisms extending bagging and boosting to online scenarios underpin all the other MOA ensembles that perform online bagging or boosting. We also compare just the individual EFDT and VFDT learners.

All trees use NBAdaptive prediction at the leaves [5], that is, they either use Naive Bayes or Majority Class for prediction depending on whichever has been more accurate overall — the cumulative accuracy of each approach from the beginning of the learning process at each leaf is recorded in order to facilitate this switching behavior.

Note that variance is reported in Sections 5.1.2 and 5.2, because these report error averages over 10 seeded runs for shuffled UCI streams and synthetic streams respectively, but not in Section 5.1.1 where standard UCI streams are used unshuffled. Tables 7, 18, and 29 do not present leaf counts as the MOA result files do not contain this information.

Our implementations of EFDT and VFDT differ only in their split selection (and reselection) strategy; there are no implementation details that are otherwise different. All our experimental results and scripts are available for replicability at `github.com/chaitanya-m` in the *exp_analysis*, *results2* and *moa_experiment_scripts (ensembles branch)* repositories.

We use prequential accuracy as the primary evaluation measure. While the instantaneous feedback provided to the learner in prequential evaluation does not reflect a common setup in applied incremental learning, none of the learners in the study relies on or exploits it. All learners examined are general incremental learners, capable at any point in time of either updating the current model by learning from an example or applying that model to classify an example. The prequential evaluation strategy is simply a convenient incremental setting for evaluating such systems. The critical feature of prequential learning is the it ensures that a learner does learn from an example before classifying it. We use prequential evaluation because it is the defacto standard for evaluation in the field of research.

5 Experimental Results

In order to compare the relative prequential accuracy of EFDT as a base learner with the prequential accuracy of VFDT as a base learner, we tabulated their wins by ensemble and class of streams (UCI unshuffled, UCI shuffled 10-seed, Synthetic 10-seed). Overall, EFDT does not have a significant CPU time overhead given its prequential accuracy performance: we list CPU run-times in Appendix A, Tables 37, 38 and 39.

5.1 UCI Streams

As explained above, following the experimental method of [34], we first use UCI datasets in their original order to assess netural datasets with inherent drift. We then shuffle the same datasets, in order to assess performance when an input stream comes from a sstationary distribution.

5.1.1 Standard UCI streams

Tables 3—13 show the effect due to EFDT on 20 UCI data streams, which are largely drawn from real data. The prequential accuracy performance matches expectations; there is minimal complex drift in these scenarios, mostly arising from concatenating files together, as many streams were made available in files that were each associated with a single class. Under these conditions, class boundaries change periodically, an instance of a trivial concept drift.

Five of these tables show bagging ensembles, which favor EFDT over VFDT. Table 3 shows the performance of vanilla OzaBag, in which EFDT outperforms VFDT on 16 streams, underperforms on 2 streams, and draws level on the remaining 2 streams. The p-value noted is 0.00066, well within a standard significance level of 0.05. Table 4 shows the performance of OzaBagADWIN, a version of OzaBag with the ADWIN change detector, which maintains a variable-length, self-adjusting window of observed error; when a change is detected in one of the base classifiers, it is replaced with a new one. This makes little difference to the performance of EFDT over VFDT, with outperformance on 15 streams (p-value 0.00377), potentially implying that in scenarios without significant levels of concept drift, ADWIN does not have an affect on EFDT bagging ensembles. However, as we discuss in Section 5.2, ADWIN interacts adversely with EFDT-based bagging ensembles when significant concept drift is present.

Table 5 shows the prequential accuracy performance comparison with Leveraging Bagging without ADWIN (LevBagNoADWIN), and Table 6 shows the comparison for LeveragingBagging (LevBag). Leveraging Bagging [6] is a variant of OzaBag that makes the critical observation that the original OzaBag algorithm parameterizes the Poisson distribution used for determining the weight of a sample with value 1—which causes a third of examples to be ignored as the value drawn from $Pois(\lambda = 1)$ is 0 about 34% of the time. Leveraging Bagging uses instead

Poisson(6) which allows using many more examples, giving it a significant advantage in the streaming setting while retaining the bagging heuristic through differential weighting of examples. Leveraging Bagging (Table 6) performs better with EFDT than with VFDT—we obtain p-value of 0.04813, which is significant at the 0.05 significance level. The removal of the change detector causes a slight deterioration, with a p-value of 0.08353 that falls outside a significance level of 0.05 (Table 5), and one more stream on which the EFDT Based system underperforms compared to the VFDT based system. Based on this observation and the observation concerning OzaBag, we do not find the role of ADWIN to be significant in streams with minimal concept drift when bagging strategies are applied.

Table 7 shows the performance of Adaptive Random Forest [21], an adaptive, online variant of classic random forests. The EFDT based system registers a clear outperformance on prequential accuracy (p-value 0.00377).

As for boosting ensembles, neither OzaBoost (Table 8, p-value 0.11894) nor its variant with ADWIN change detection (Table 9, p-value 0.3238) benefit EFDT with significance. The general success of online bagging strategies over straightforward online boosting strategies is noted in [6], and while tangential to ensemble comparison, this discrepancy appears to work in favor of further setting apart EFDT based bagging ensembles in their outperformance.

However, boosting regimes with greater particularity in weighting misclassified examples favor EFDT with significance. We observe this with ADOB (Table 10, p-value 0.04813), a variant of online boosting that rearranges ensemble components in increasing order of accuracy—misclassified examples are given half the total weight when passed to the next ensemble component in online boosting, and thus ADOB's strategy optimizes for learning misclassified instances sooner. BOLE (Table 11), a variant of ADOB which allows poorly performing ensemble members to vote, and OnlineSmoothBoost(Table 12), which provides a rationalized continuous weighting scheme for examples in contrast with the stepped Poisson weighting provided by OzaBoost, both also benefit EFDT with high significance, with p-values of 0.04813 and 0.02069 respectively. Note that we use ADOB and BOLE for their core strategies without change detectors (their MOA implementations default to wrapping ensemble components with change detectors, while the respective papers require wrapping the entire ensemble—this ambiguity was not of interest to our experimentation, but the core strategies were).

Plain EFDT outperforms plain VFDT on 13 streams (Table 13), with VFDT registering lower prequential accuracy on the remaining 7.

In summary, as expected, the greater variance induced by EFDT's less conservative splitting mechanism proves productive when drift is not extreme. All types of ensemble learner achieve lower accuracy for more datasets than not when EFDT is used as a base learner in place of VFDT. The frequency of these wins is statistically significant at the 0.05 level for all ensemble techniques other than LeveragingBagging in the case that ADWIN is disabled (when the use of EFDT still achieves lower error for 13 datasets, but VFDT achieves lower error for 6, preventing a statistically significant win).

Table 3: OzaBag - UCI streams processed in original order

Streams	**VFDT Base**		**EFDT Base**	
	Error	Leaves	Error	Leaves
airlines	0.34988	8445	**0.34133**	17366
aws—price-discretized	0.1469	5052	**0.14027**	6117
chess	0.57514	4	**0.25848**	69
covertype	0.14551	162	**0.11684**	196
covpokelec	0.146	315	**0.13372**	313
fonts	0.01729	207	**0.00245**	211
hhar	0.02375	2430	**0.00115**	1173
kdd	0.00088	135	**0.00086**	178
localization	0.09551	164	**0.06576**	74
miniboone	***8e-05***	4	***8e-05***	4
nbaiot	0.00522	43	**0.00091**	27
nswelec	0.17367	38	**0.17226**	52
pamap2	0.04046	152	**0.01335**	104
poker	0.1613	129	**0.15943**	104
pucrio	0.02202	22	**0.00089**	7
sensor—home-activity	0.00208	240	**0.00049**	20
sensor—CO-discretized	**0.01362**	608	0.03894	450
skin	***0.00062***	4	***0.00062***	4
tnelec	**0.0025**	2	0.00528	8
wisdm	0.11476	2692	**0.0812**	2296
Unique Wins	**2**		**16**	

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics | **p-value: 0.00066** | **Confidence Interval: 0.68974 — 1**

Table 4: OzaBagADWIN - UCI streams processed in original order

Streams	**VFDT Base**		**EFDT Base**	
	Error	Leaves	Error	Leaves
airlines	**0.33446**	3488	0.33988	2637
aws—price-discretized	0.14711	4879	**0.14057**	5897
chess	0.01741	3	**0.01379**	2
covertype	0.15069	1	**0.10075**	4
covpokelec	0.2132	2	**0.18577**	2
fonts	0.00128	156	**8e-04**	156
hhar	0.0041	4	**0.00093**	4
kdd	**0.00047**	41	0.00055	36
localization	0.06409	5	**0.04699**	4
miniboone	***8e-05***	2	***8e-05***	2
nbaiot	0.00028	2	**0.00014**	2
nswelec	0.16307	3	**0.12798**	4
pamap2	0.00858	1	**0.00282**	1
poker	0.25234	2	**0.24115**	4
pucrio	0.00092	2	**0.00066**	5
sensor—home-activity	3e-04	2	**0.00029**	2
sensor—CO-discretized	0.02445	2	**0.02192**	4
skin	***0.00036***	2	***0.00036***	2
tnelec	**0.00278**	4	0.00343	11
wisdm	0.10881	1	**0.07696**	2
Unique Wins	**3**		**15**	

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics | **p-value: 0.00377** | **Confidence Interval: 0.62332 — 1**

Table 5: LevBagNoADWIN - UCI streams processed in original order

Streams	VFDT Base		EFDT Base	
	Error	Leaves	Error	Leaves
airlines	0.35124	13743	**0.34829**	27354
aws—price-discretized	0.13042	10612	**0.12233**	14160
chess	0.48134	66	**0.18238**	164
covertype	0.099	697	**0.07295**	379
covpokelec	**0.06218**	1560	0.08186	376
fonts	0.01641	208	**0.00156**	210
hhar	0.01012	8911	**0.00109**	2567
kdd	**0.00073**	156	0.00075	275
localization	0.05593	305	**0.05218**	84
miniboone	**4e-05**	4	5e-05	4
nbaiot	0.00089	59	**0.00038**	24
nswelec	0.15643	110	**0.14809**	108
pamap2	0.15539	515	**0.02625**	107
poker	**0.05333**	804	0.09355	292
pucrio	0.01239	47	**0.00115**	7
sensor—home-activity	0.0029	597	**0.00071**	26
sensor—CO-discretized	**0.01437**	1949	0.03077	446
skin	**0.00015**	4	0.00016	4
tnelec	***0.00324***	7	***0.00324***	7
wisdm	0.08572	6174	**0.06601**	3856
Unique Wins	**6**		**13**	

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value:**	**Confidence Interval:**
	0.08353	**0.47003 — 1**

Table 6: LevBag - UCI streams processed in original order

Streams	VFDT Base		EFDT Base	
	Error	Leaves	Error	Leaves
airlines	**0.34316**	1676	0.34607	1410
aws—price-discretized	0.13038	10576	**0.12272**	13174
chess	0.01576	2	**0.01355**	2
covertype	0.09768	2	**0.06784**	9
covpokelec	0.17382	6	**0.13015**	7
fonts	0.00114	155	**0.00067**	161
hhar	0.00088	10	**0.00063**	6
kdd	**0.00034**	69	0.00037	37
localization	0.03834	5	**0.03781**	5
miniboone	**4e-05**	4	5e-05	4
nbaiot	9e-05	2	**8e-05**	3
nswelec	0.11276	12	**0.10457**	19
pamap2	0.003	2	**0.00138**	2
poker	0.222	11	**0.1684**	26
pucrio	**0.00101**	3	0.00103	6
sensor—home-activity	3e-04	1	**0.00025**	2
sensor—CO-discretized	**0.01412**	2	0.0142	4
skin	***0.00014***	3	***0.00014***	3
tnelec	***0.00267***	7	***0.00267***	7
wisdm	0.10197	2	**0.06728**	3
Unique Wins	**5**		**13**	

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value:**	**Confidence Interval:**
	0.04813	**0.50217 — 1**

Table 7: Adaptive Random Forest - UCI streams processed in original order

Streams	**VFDT Base** Error	**EFDT Base** Error
airlines	0.34994	**0.33776**
aws—price-discretized	**0.18187**	0.19474
chess	0.01924	**0.0139**
covertype	0.11075	**0.07057**
covpokelec	0.18964	**0.15533**
fonts	0.006	**0.00599**
hhar	0.0015	**0.00075**
kdd	0.00044	**0.00043**
localization	**0.03333**	0.03576
miniboone	**4e-05**	5e-05
nbaiot	***4e-05***	***4e-05***
nswelec	0.13567	**0.11857**
pamap2	0.00151	**0.00113**
poker	0.24551	**0.20895**
pucrio	***0.00044***	***0.00044***
sensor—home-activity	5e-04	**0.00024**
sensor—CO-discretized	0.00834	**0.00765**
skin	0.00013	**0.00011**
tnelec	0.01257	**0.00748**
wisdm	0.10885	**0.07728**
Unique Wins	**3**	**15**

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.00377**	**Confidence Interval: 0.62332 — 1**

Table 8: OzaBoost - UCI streams processed in original order

Streams	**VFDT Base**		**EFDT Base**	
	Error	Leaves	Error	Leaves
airlines	**0.36114**	9095	0.36415	10139
aws—price-discretized	0.14541	5526	**0.13773**	7987
chess	0.84848	14	**0.24224**	66
covertype	0.08286	155	**0.05701**	224
covpokelec	0.09133	334	**0.09009**	334
fonts	0.00167	208	**0.00166**	220
hhar	0.00142	2493	**0.00103**	1104
kdd	**0.00176**	315	0.00183	437
localization	0.04061	98	**0.03807**	102
miniboone	***9e-05***	3	***9e-05***	3
nbaiot	0.0092	31	**0.00124**	9
nswelec	0.14113	27	**0.13004**	36
pamap2	0.28883	132	**0.00364**	55
poker	**0.10215**	142	0.10375	135
pucrio	**0.00532**	14	0.02177	9
sensor—home-activity	0.00145	161	**8e-04**	12
sensor—CO-discretized	**0.00385**	507	0.01015	220
skin	***0.20681***	4	***0.20681***	4
tnelec	**0.00148**	4	0.00152	7
wisdm	0.10207	2174	**0.06448**	2263
Unique Wins	**6**		**12**	

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.11894**	**Confidence Interval: 0.44595 — 1**

Table 9: OzaBoostADWIN - UCI streams processed in original order

Streams	VFDT Base		EFDT Base	
	Error	Leaves	Error	Leaves
airlines	**0.38107**	2679	0.38181	5095
aws—price-discretized	0.14548	5300	**0.13786**	7636
chess	0.18648	2	**0.10631**	4
covertype	0.08812	4	**0.0856**	4
covpokelec	**0.12989**	11	0.14332	6
fonts	0.42996	150	**0.00853**	128
hhar	0.14148	2	**0.12517**	2
kdd	**0.2722**	42	0.28153	28
localization	0.0501	4	**0.04536**	4
miniboone	**6e-05**	2	7e-05	2
nbaiot	0.43386	2	**0.37867**	2
nswelec	0.09989	11	**0.0983**	9
pamap2	0.32267	2	**0.24907**	2
poker	**0.14994**	61	0.18364	11
pucrio	**0.00124**	2	0.2646	2
sensor—home-activity	0.32472	2	**0.00088**	2
sensor—CO-discretized	**0.30825**	5	0.34715	3
skin	***0.99995***	2	***0.99995***	2
tnelec	**0.00143**	11	0.00165	19
wisdm	0.08342	5	**0.07332**	4
Unique Wins	**8**		**11**	

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics | **p-value: 0.3238** | **Confidence Interval: 0.36812 — 1**

Table 10: ADOB - UCI streams processed in original order

Streams	VFDT Base		EFDT Base	
	Error	Leaves	Error	Leaves
airlines	**0.36174**	9302	0.36964	9879
aws—price-discretized	0.14263	5432	**0.13371**	7627
chess	0.60293	14	**0.16659**	78
covertype	0.07985	154	**0.05784**	239
covpokelec	0.09125	325	**0.08324**	370
fonts	***0.00038***	211	***0.00038***	225
hhar	0.00152	2654	**8e-04**	1708
kdd	**0.00023**	321	0.00026	529
localization	0.04004	103	**0.03828**	95
miniboone	0.00017	3	**0.00016**	3
nbaiot	**0.00354**	36	0.00688	11
nswelec	0.13748	35	**0.13211**	36
pamap2	0.08868	138	**0.00342**	56
poker	**0.0915**	140	0.10831	101
pucrio	0.00289	13	**0.00117**	10
sensor—home-activity	0.00206	176	**0.00056**	16
sensor—CO-discretized	**0.00295**	437	0.00713	145
skin	***0.00019***	4	***0.00019***	4
tnelec	0.00172	2	**0.00159**	8
wisdm	0.09067	2641	**0.06247**	2174
Unique Wins	**5**		**13**	

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics | **p-value: 0.04813** | **Confidence Interval: 0.50217 — 1**

Table 11: BOLE - UCI streams processed in original order

Streams	**VFDT Base**		**EFDT Base**	
	Error	Leaves	Error	Leaves
airlines	**0.36173**	9302	0.36963	9879
aws—price-discretized	0.14263	5432	**0.13371**	7627
chess	0.45517	14	**0.16648**	78
covertype	0.07985	154	**0.05773**	239
covpokelec	0.09122	325	**0.08315**	370
fonts	***0.00039***	211	***0.00039***	225
hhar	0.00152	2654	**8e-04**	1708
kdd	**0.00023**	321	0.00026	529
localization	0.04005	103	**0.0383**	95
miniboone	0.00017	3	**0.00016**	3
nbaiot	**0.00355**	36	0.00688	11
nswelec	0.13737	35	**0.132**	36
pamap2	0.04702	138	**0.00342**	56
poker	**0.09138**	140	0.10831	101
pucrio	0.00288	13	**0.00116**	10
sensor—home-activity	0.00207	176	**0.00057**	16
sensor—CO-discretized	**0.00294**	437	0.00713	145
skin	***0.00017***	4	***0.00017***	4
tnelec	0.00172	2	**0.00159**	8
wisdm	0.09066	2641	**0.06247**	2174
Unique Wins	**5**		**13**	

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value:**	**Confidence Interval:**
	0.04813	**0.50217 — 1**

Table 12: OnlineSmoothBoost - UCI streams processed in original order

Streams	**VFDT Base**		**EFDT Base**	
	Error	Leaves	Error	Leaves
airlines	0.34321	8124	**0.33784**	12517
aws—price-discretized	0.14576	4924	**0.13951**	5981
chess	0.47655	10	**0.28121**	70
covertype	0.15224	183	**0.11009**	280
covpokelec	0.13829	358	**0.12604**	365
fonts	0.01291	207	**0.00228**	211
hhar	0.01685	2214	**0.00245**	2898
kdd	0.00086	151	**0.00076**	180
localization	0.09631	170	**0.06973**	88
miniboone	**0.00011**	4	0.00013	4
nbaiot	0.00358	36	**0.00097**	29
nswelec	0.17422	36	**0.16039**	58
pamap2	0.05008	182	**0.03045**	88
poker	**0.13754**	146	0.16386	84
pucrio	0.02314	21	**0.00119**	12
sensor—home-activity	0.00382	215	**9e-04**	30
sensor—CO-discretized	**0.01256**	502	0.03383	541
skin	**0.00051**	4	0.00055	4
tnelec	**0.00237**	2	0.00498	7
wisdm	0.12901	2340	**0.09535**	2665
Unique Wins	**5**		**15**	

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value:**	**Confidence Interval:**
	0.02069	**0.54442 — 1**

Table 13: Plain single learners (no ensemble) - UCI streams processed in original order

Streams	**VFDT Base**		**EFDT Base**	
	Error	Leaves	Error	Leaves
airlines	0.34955	8420	**0.34782**	14361
aws—price-discretized	0.14728	5041	**0.14143**	6144
chess	0.65807	5	**0.31152**	76
covertype	0.18905	217	**0.15431**	389
covpokelec	**0.18665**	368	0.19391	505
fonts	0.01781	207	**0.003**	211
hhar	0.03749	2590	**0.00357**	2947
kdd	**0.00093**	141	0.00094	181
localization	0.13299	177	**0.09754**	115
miniboone	**0.00011**	4	0.00013	4
nbaiot	0.00368	36	**0.00076**	29
nswelec	0.19861	40	**0.19283**	71
pamap2	0.0972	234	**0.06136**	88
poker	**0.20509**	142	0.218	150
pucrio	0.01457	26	**0.00147**	12
sensor—home-activity	0.00906	238	**0.00135**	39
sensor—CO-discretized	**0.02633**	605	0.06655	794
skin	**0.00052**	4	0.00055	4
tnelec	**0.00235**	2	0.00496	7
wisdm	0.18796	2752	**0.14576**	2598
Unique Wins	**7**		**13**	

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics

p-value: 0.13159

Confidence Interval: 0.44197 — 1

5.1.2 Shuffled UCI streams

We shuffled the UCI streams in order to test prequential accuracy performance when concept drift was removed so as to understand if the advantage is retained in the case of a static generating distribution. The results on shuffled data streams are based on averaged prequential performance over 10 shuffles of each data stream, with fixed random seeds so the experiments can be replicated easily. These results are shown in Tables 14 through 24.

The pattern of comparative prequential accuracy performance is roughly the same for three of the bagging approaches, but not for the Leveraged Bagging approaches. OzaBag (Table 14), OzaBagADWIN (Table 15) and Adaptive Random Forest (Table 18) all favor EFDT with p-values of 0.00377, 0.00377, and 0.00591 respectively. However, in the absence of residual concept drift, it appears that EFDT based Leveraging Bagging without ADWIN (Table 16, p-value 0.24034) does not attain significance, and Leveraging Bagging (Table 17, p-value 0.75966) also loses significance (VFDT outperforms slightly, 10 wins to 8 and 2 draws). Change detection with Leveraging Bagging appears to make the effect more severe, but changing the choice of Poisson value alone (to 6) as compared to OzaBag negates the advantage of EFDT when the stream is highly uniform. This erosion of advantage for EFDT based Leveraging Bagging when supplied highly uniform streams merits further investigation.

Among the boosting approaches, we note the failure of OzaBoost and OzaBoostADWIN to reach significance (Tables 19, 20, p-values 0.31453 and 0.24034 respectively), and the success of ADOB, BOLE and OnlineSmoothBoost (Tables 21, 22, and 23, p-values 0.04813, 0.04813, and 0.00377 respectively) well within a 0.05 significance level. Clearly, improving strategies for weighting misclassified examples is of particular interest for future work on boosting in general.

Plain EFDT outperforms plain VFDT on 13 streams (Table 24), with VFDT registering lower prequential accuracy on remaining 5 and drawing on two.

Table 14: OzaBag - Shuffled UCI streams

Streams	**VFDT Base**			**EFDT Base**		
	Error	Variance	Leaves	Error	Variance	Leaves
airlines	**0.35662**	0.00023	9331	0.36085	0.00023	17292
aws—price-discretized	0.14668	0.00013	5054	**0.14018**	0.00012	6133
chess	0.67124	0.00023	1	**0.59518**	3e-04	78
covertype	0.27185	0.00021	47	**0.25813**	2e-04	73
covpokelec	**0.29823**	0.00029	110	0.34712	0.00043	53
fonts	***0.00075***	0	243	***0.00075***	0	244
hhar	0.05203	6e-05	1821	**0.03088**	3e-05	3418
kdd	0.00098	0	144	**8e-04**	0	188
localization	0.35064	0.00023	52	**0.32112**	0.00068	122
miniboone	0.10228	0.00012	45	**0.10025**	0.00011	37
nbaiot	0.04758	0.0026	52	**0.00216**	3e-05	103
nswelec	0.23156	2e-04	26	**0.23054**	2e-04	33
pamap2	0.1234	0.00016	152	**0.10654**	0.00014	369
poker	**0.26286**	0.00024	60	0.27569	0.00043	56
pucrio	0.1063	0.00018	21	**0.04309**	8e-05	56
sensor—home-activity	0.06297	9e-05	242	**0.03715**	0.00011	328
sensor—CO-discretized	0.18668	0.00022	350	**0.17228**	0.00022	508
skin	0.01661	2e-05	25	**0.00973**	1e-05	34
tnelec	***0.00663***	1e-05	31	***0.00663***	1e-05	31
wisdm	0.17683	0.00018	231	**0.11814**	0.00011	510
Unique Wins	**3**			**15**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics | **p-value: 0.00377** | **Confidence Interval: 0.62332 — 1**

Table 15: OzaBagADWIN - Shuffled UCI streams

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
airlines	**0.3794**	0.00025	4186	0.38762	0.00026	4477
aws—price-discretized	0.14701	0.00013	4875	**0.14066**	0.00012	5887
chess	0.67127	0.00022	1	**0.59529**	0.00029	78
covertype	0.27194	0.00021	46	**0.25837**	2e-04	70
covpokelec	**0.3107**	0.00037	79	0.37881	0.00065	14
fonts	***0.00075***	0	243	***0.00075***	0	244
hhar	0.05104	6e-05	1704	**0.03142**	3e-05	3149
kdd	0.00096	0	142	**0.00079**	0	187
localization	0.35122	0.00023	51	**0.32013**	0.00061	123
miniboone	0.10225	0.00011	45	**0.10132**	0.00011	33
nbaiot	0.33052	0.0013	1	**0.0037**	3e-05	92
nswelec	0.23162	0.00019	25	**0.23045**	0.00019	33
pamap2	0.11963	0.00012	127	**0.10477**	0.00014	296
poker	**0.26353**	0.00024	59	0.28535	0.00047	46
pucrio	0.10893	0.00019	19	**0.0449**	9e-05	51
sensor—home-activity	0.0638	9e-05	240	**0.04238**	0.00019	285
sensor—CO-discretized	0.18565	0.00024	335	**0.18344**	0.00028	418
skin	0.01661	2e-05	25	**0.00973**	1e-05	34
tnelec	***0.00678***	0	31	***0.00678***	0	31
wisdm	0.17588	0.00018	223	**0.11891**	0.00012	475
Unique Wins	**3**			**15**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.00377**	**Confidence Interval: 0.62332 — 1**

Table 16: LevBagNoADWIN - Shuffled UCI streams

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
airlines	**0.35897**	0.00023	16509	0.36134	0.00024	26322
aws—price-discretized	0.13032	0.00011	10600	**0.12266**	0.00011	14197
chess	0.61357	0.00032	83	**0.60015**	0.00037	154
covertype	0.22738	0.00019	394	**0.22208**	0.00019	416
covpokelec	**0.19074**	0.00034	991	0.28548	0.00091	498
fonts	***0.00061***	0	248	***0.00061***	0	248
hhar	0.02496	3e-05	4391	**0.01873**	2e-05	5739
kdd	0.00064	0	207	**0.00054**	0	267
localization	0.33575	0.00047	116	**0.30653**	0.00072	283
miniboone	**0.08894**	8e-05	191	0.09681	0.00011	91
nbaiot	0.04423	0.00346	97	**0.03933**	0.00354	138
nswelec	**0.22097**	2e-04	107	0.22363	2e-04	97
pamap2	**0.0833**	1e-04	488	0.10028	0.00015	633
poker	**0.16567**	0.00042	614	0.21305	0.00218	336
pucrio	0.0453	0.00012	93	**0.04027**	0.00024	91
sensor—home-activity	**0.01491**	2e-05	621	0.01644	6e-05	482
sensor—CO-discretized	0.1251	0.00012	1861	**0.12438**	2e-04	1705
skin	0.00529	1e-05	66	**0.00394**	0	71
tnelec	***0.00356***	0	31	***0.00356***	0	31
wisdm	0.13369	0.00013	574	**0.11135**	0.00013	888
Unique Wins	**7**			**11**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.24034**	**Confidence Interval: 0.39216 — 1**

Table 17: LevBag - Shuffled UCI streams

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
airlines	**0.41108**	0.00032	1129	0.42212	0.00032	405
aws—price-discretized	0.1311	0.00011	9615	**0.12379**	0.00011	12693
chess	**0.62328**	0.00071	21	0.63585	0.00083	47
covertype	0.22735	0.00019	389	**0.22286**	2e-04	377
covpokelec	**0.18993**	0.00023	904	0.36941	0.00084	37
fonts	***6e-04***	0	248	***6e-04***	0	248
hhar	0.02357	3e-05	4119	**0.01815**	3e-05	5253
kdd	0.00063	0	207	**0.00054**	0	266
localization	**0.38614**	0.00069	28	0.44004	0.00041	23
miniboone	**0.08798**	8e-05	189	0.09766	0.00011	79
nbaiot	0.10372	0.00349	57	**0.00304**	3e-05	121
nswelec	**0.22097**	2e-04	107	0.22377	2e-04	93
pamap2	**0.33327**	0.00182	5	0.36266	0.00155	6
poker	**0.16436**	4e-04	618	0.25652	0.00141	132
pucrio	0.04534	0.00011	91	**0.04214**	0.00014	61
sensor—home-activity	**0.01491**	2e-05	621	0.02001	0.00011	401
sensor—CO-discretized	**0.12546**	0.00012	1782	0.13818	0.00021	1295
skin	0.00529	1e-05	66	**0.00399**	0	67
tnelec	***0.00356***	0	31	***0.00356***	0	31
wisdm	0.13278	0.00013	561	**0.11152**	0.00013	785
Unique Wins	**10**			**8**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.75966**	**Confidence Interval: 0.24396 — 1**

Table 18: Adaptive Random Forest - Shuffled UCI streams

Streams	VFDT Base		EFDT Base	
	Error	Variance	Error	Variance
airlines	0.40058	0.00033	**0.39267**	0.00028
aws—price-discretized	**0.18202**	0.00021	0.19533	0.00027
chess	0.70999	0.00062	**0.63728**	0.00046
covertype	0.2876	0.00024	**0.2796**	0.00025
covpokelec	0.48633	0.00075	**0.44633**	0.00054
fonts	0.80344	5e-04	**0.68368**	0.00081
hhar	0.82982	0.00054	**0.33478**	0.00069
kdd	**0.00191**	0	0.00284	0
localization	0.43867	0.00031	**0.43129**	0.00056
miniboone	0.10704	0.00014	**0.10339**	0.00011
nbaiot	0.28421	0.00232	**0.11822**	0.00154
nswelec	**0.22344**	0.00021	0.2238	2e-04
pamap2	**0.21005**	0.00036	0.27404	0.00324
poker	0.37042	0.00031	**0.35486**	0.00029
pucrio	0.16029	0.00033	**0.0893**	0.00017
sensor—home-activity	0.32884	0.00049	**0.28902**	0.00044
sensor—CO-discretized	0.39529	5e-04	**0.32805**	0.00044
skin	0.19835	0.00033	**0.04533**	0.00023
tnelec	0.35672	0.01894	**0.28284**	0.00168
wisdm	0.22926	0.00045	**0.1882**	0.00032
Unique Wins	**4**		**16**	

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.00591**	**Confidence Interval: 0.59897 — 1**

Table 19: OzaBoost - Shuffled UCI streams

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
airlines	**0.38601**	0.00024	9872	0.39146	0.00025	11529
aws—price-discretized	0.14527	0.00012	5513	**0.13761**	0.00012	7952
chess	***0.90007***	1e-04	9	***0.90007***	1e-04	81
covertype	0.26591	0.00022	50	**0.25115**	0.00021	72
covpokelec	**0.29062**	0.00053	112	0.34335	0.0013	76
fonts	***0.00107***	0	244	***0.00107***	0	245
hhar	0.02277	7e-05	2207	**0.00953**	1e-05	4025
kdd	**0.00034**	0	347	0.00035	0	414
localization	0.33213	5e-04	50	**0.31382**	0.00072	103
miniboone	**0.09859**	0.00021	49	0.10123	0.00012	41
nbaiot	0.02137	0.00022	99	**0.00879**	0.00013	81
nswelec	**0.23942**	0.00029	25	0.24197	0.00031	25
pamap2	**0.08673**	0.00015	143	0.08794	0.00014	294
poker	**0.23568**	0.00032	63	0.24227	0.00075	88
pucrio	0.07653	3e-04	21	**0.02393**	4e-05	52
sensor—home-activity	0.04266	7e-05	216	**0.02925**	6e-05	263
sensor—CO-discretized	0.15979	0.00071	398	**0.13586**	0.00056	538
skin	0.00485	1e-05	39	**0.0033**	1e-05	47
tnelec	***0.0065***	2e-05	31	***0.0065***	2e-05	31
wisdm	0.17925	0.00022	245	**0.1267**	0.00014	600
Unique Wins	**7**			**10**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.31453**	**Confidence Interval: 0.36401 — 1**

Table 20: OzaBoostADWIN - Shuffled UCI streams

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
airlines	**0.43251**	0.00028	6946	0.4347	0.00029	5690
aws—price-discretized	0.14084	0.00019	7638	**0.1388**	0.00013	8993
chess	**0.89967**	0.00011	30	0.89985	1e-04	80
covertype	0.26289	0.00035	183	**0.26169**	0.00034	83
covpokelec	**0.34818**	0.00481	1191	0.40584	0.00587	231
fonts	***0.00107***	0	244	***0.00107***	0	245
hhar	0.02243	0.00012	2693	**0.01147**	2e-05	3763
kdd	0.00072	5e-05	305	**0.00061**	0	358
localization	0.33738	0.0014	84	**0.31842**	0.00141	102
miniboone	**0.10724**	0.00014	87	0.12055	2e-04	18
nbaiot	0.02992	0.00104	103	**0.00998**	0.00033	77
nswelec	**0.26887**	0.00045	37	0.27118	0.00041	26
pamap2	**0.06545**	0.00018	443	0.09245	0.00027	270
poker	**0.27171**	0.00135	571	0.305	0.0026	164
pucrio	0.0417	5e-04	60	**0.02821**	5e-05	52
sensor—home-activity	0.03694	0.00039	231	**0.03507**	0.00016	217
sensor—CO-discretized	0.16919	0.00097	706	**0.1599**	0.00142	476
skin	0.00362	1e-05	39	**0.00329**	1e-05	37
tnelec	***0.00787***	0.00026	31	***0.00787***	0.00026	31
wisdm	0.14083	0.00052	440	**0.12696**	0.00014	559
Unique Wins	**7**			**11**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.24034**	**Confidence Interval: 0.39216 — 1**

Table 21: ADOB - Shuffled UCI streams

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
airlines	**0.38375**	0.00025	9965	0.38679	0.00025	11128
aws—price-discretized	0.14249	0.00012	5435	**0.13355**	0.00012	7628
chess	0.89206	0.00177	8	**0.75812**	0.02005	81
covertype	0.2607	0.00021	45	**0.24908**	2e-04	64
covpokelec	**0.27884**	0.00043	108	0.3403	0.00113	70
fonts	***0.00102***	1e-05	244	***0.00102***	1e-05	245
hhar	0.02323	8e-05	2176	**0.00933**	1e-05	4133
kdd	**0.00032**	0	240	0.00033	0	420
localization	0.33964	0.00042	51	**0.32868**	0.00093	99
miniboone	0.10164	0.00039	49	**0.10128**	0.00012	40
nbaiot	0.01748	0.00064	96	**0.0053**	7e-05	85
nswelec	**0.2455**	0.00033	24	0.2483	0.00035	27
pamap2	0.08814	0.00015	142	**0.08401**	0.00016	319
poker	**0.22355**	0.00044	57	0.2413	0.00089	93
pucrio	0.06842	0.00025	19	**0.02511**	5e-05	59
sensor—home-activity	0.04036	9e-05	218	**0.02613**	6e-05	264
sensor—CO-discretized	0.14869	0.00075	372	**0.12618**	0.00043	527
skin	0.00552	2e-05	36	**0.00384**	1e-05	42
tnelec	***0.00646***	2e-05	31	***0.00646***	2e-05	31
wisdm	0.17089	2e-04	240	**0.12382**	0.00013	602
Unique Wins	**5**			**13**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.04813**	**Confidence Interval: 0.50217 — 1**

Table 22: BOLE - Shuffled UCI streams

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
airlines	**0.38375**	0.00025	9965	0.3868	0.00025	11128
aws—price-discretized	0.14249	0.00012	5435	**0.13355**	0.00012	7628
chess	0.83787	0.00973	8	**0.60847**	0.00046	81
covertype	0.26065	0.00021	45	**0.24906**	2e-04	64
covpokelec	**0.27706**	0.00041	108	0.33397	0.00093	70
fonts	***0.00087***	0	244	***0.00087***	0	245
hhar	0.02299	4e-05	2176	**0.00933**	1e-05	4133
kdd	**0.00032**	0	240	0.00033	0	420
localization	0.33935	0.00041	51	**0.32839**	0.00092	99
miniboone	0.10164	0.00039	49	**0.10127**	0.00012	40
nbaiot	0.01617	2e-04	96	**0.0053**	7e-05	85
nswelec	**0.2455**	0.00033	24	0.2483	0.00035	27
pamap2	0.08814	0.00015	142	**0.08401**	0.00016	319
poker	**0.22348**	0.00044	57	0.24122	0.00088	93
pucrio	0.0684	0.00025	19	**0.02508**	5e-05	59
sensor—home-activity	0.04023	6e-05	218	**0.02611**	6e-05	264
sensor—CO-discretized	0.14675	0.00022	372	**0.12573**	0.00029	527
skin	0.00551	2e-05	36	**0.00382**	1e-05	42
tnelec	***0.00583***	0	31	***0.00583***	0	31
wisdm	0.17085	2e-04	240	**0.12378**	0.00013	602
Unique Wins	**5**			**13**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.04813**	**Confidence Interval: 0.50217 — 1**

Table 23: OnlineSmoothBoost - Shuffled UCI streams

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
airlines	**0.35598**	0.00023	8766	0.35792	0.00023	12326
aws—price-discretized	0.14553	0.00012	4922	**0.13925**	0.00012	5972
chess	0.6674	0.00023	1	**0.58559**	0.00035	101
covertype	0.27164	0.00021	40	**0.26133**	0.00021	61
covpokelec	**0.3003**	3e-04	94	0.33391	0.00073	70
fonts	***0.00068***	0	245	***0.00068***	0	245
hhar	0.05521	8e-05	1712	**0.0363**	4e-05	3342
kdd	0.00106	0	149	**0.00082**	0	187
localization	0.34715	0.00032	54	**0.32556**	9e-04	138
miniboone	0.10497	0.00011	39	**0.10297**	0.00015	29
nbaiot	0.03079	0.00032	52	**0.01775**	0.00174	103
nswelec	0.23471	0.00021	23	**0.23173**	0.00021	31
pamap2	0.12848	0.00017	154	**0.10889**	0.00019	379
poker	**0.26615**	0.00027	49	0.2724	0.00084	66
pucrio	0.12499	0.00039	17	**0.05627**	0.00045	57
sensor—home-activity	0.07077	0.00012	215	**0.03824**	0.00012	353
sensor—CO-discretized	0.18328	0.00023	333	**0.16407**	0.00028	516
skin	0.01677	3e-05	22	**0.01178**	2e-05	32
tnelec	***0.00588***	1e-05	31	***0.00588***	1e-05	31
wisdm	0.18723	0.00022	222	**0.13099**	0.00018	522
Unique Wins	**3**			**15**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.00377**	**Confidence Interval: 0.62332 — 1**

Table 24: Plain single learners (no ensemble) - Shuffled UCI streams

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
airlines	**0.35926**	0.00023	9233	0.36099	0.00023	14143
aws—price-discretized	0.14715	0.00013	5050	**0.14139**	0.00012	6134
chess	0.67174	0.00022	1	**0.60706**	0.00058	99
covertype	0.28618	0.00023	46	**0.27648**	0.00022	69
covpokelec	**0.32762**	0.00044	110	0.37454	0.00207	74
fonts	***0.00068***	0	245	***0.00068***	0	245
hhar	0.07311	8e-05	1824	**0.05113**	6e-05	3483
kdd	0.00114	0	165	**0.00094**	0	190
localization	0.35944	0.00025	46	**0.33857**	0.00128	152
miniboone	**0.1163**	0.00015	46	0.11875	0.00022	36
nbaiot	0.0381	0.00165	51	**0.02908**	0.00275	103
nswelec	0.24141	0.00027	27	**0.24057**	0.00023	38
pamap2	0.15764	0.00032	169	**0.15672**	3e-04	385
poker	**0.28636**	0.00032	59	0.30909	0.00142	49
pucrio	0.13639	5e-04	20	**0.08329**	0.00084	67
sensor—home-activity	0.11511	0.00022	241	**0.0955**	0.00187	333
sensor—CO-discretized	**0.23193**	4e-04	387	0.2432	0.00145	542
skin	0.01908	3e-05	26	**0.01359**	3e-05	37
tnelec	***0.00586***	1e-05	31	***0.00586***	1e-05	31
wisdm	0.1888	0.00021	231	**0.13637**	0.00019	532
Unique Wins	**5**			**13**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.04813**	**Confidence Interval: 0.50217 — 1**

5.2 Synthetic Streams

Table 25: OzaBag - Synthetic streams with concept drift

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.1968	0.00031	533	**0.18373**	0.00031	806
recurrent—led	**0.32315**	0.00027	54	0.32699	0.00026	303
recurrent—randomtree	0.21247	0.0022	1041	**0.20024**	0.00299	1243
recurrent—sea	0.14635	0.00014	150	**0.13948**	0.00013	168
recurrent—stagger	**0.17477**	0.00123	23	0.17949	0.00078	23
recurrent—waveform	0.16931	0.0015	149	**0.16033**	0.0012	147
hyperplane—1	0.10901	0.00019	277	**0.10626**	0.00014	196
hyperplane—2	0.15844	0.00119	326	**0.12861**	0.00056	83
hyperplane—3	**0.10194**	0.00012	275	0.10271	0.00011	226
hyperplane—4	0.15448	0.00349	304	**0.12442**	0.00131	96
rbf—drift-1	0.07945	0.00032	381	**0.07194**	0.00026	266
rbf—drift-2	0.18388	0.00088	434	**0.16055**	0.00094	705
rbf—drift-3	0.11286	0.00063	355	**0.10982**	6e-04	258
rbf—drift-4	0.36223	0.00134	272	**0.35289**	0.00127	320
recurrent—abrupt—222	0.3431	0.02363	4	**0.33746**	0.02201	4
recurrent—abrupt—322	0.37099	0.01543	4	**0.35744**	0.03263	4
recurrent—abrupt—332	0.3459	0.08753	8	**0.32838**	0.10325	8
recurrent—abrupt—333	0.36143	0.01448	27	**0.35037**	0.04007	27
recurrent—abrupt—334	0.39642	0.00214	64	**0.3698**	0.05898	62
recurrent—abrupt—335	0.39614	0.00314	121	**0.38665**	0.04395	123
recurrent—abrupt—422	**0.34057**	0.02736	4	0.34205	0.04047	4
recurrent—abrupt—444	0.40099	0.00751	228	**0.36757**	0.01922	226
recurrent—abrupt—522	**0.33857**	0.0361	4	0.34167	0.07319	4
recurrent—abrupt—555	0.40625	0.00469	1299	**0.35437**	0.00596	1705
Unique Wins	**5**			**19**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.00331**	**Confidence Interval: 0.61086 — 1**

Tables 25 through 35 compare EFDT and VFDT based ensembles on a large number of parameterized synthetic concept drift streams found in the literature.

With bagging ensembles, EFDT as a base learner demonstrates an advantage with OzaBag (Table 25, p-value 0.00331), Leveraging Bagging without ADWIN (Table 27, p-value 0.00077), and Adaptive Random Forest (Table 29, p-value < 0.00001). EFDT has no advantage as a base learner with OzaBagADWIN (Table 26, p-value 0.41941, arising from 13 wins for EFDT to 11 for VFDT) and Leveraging Bagging (Table 28, p-value 0.84627 arising from 10 wins for EFDT to 14 for VFDT).

These two cases may be explained as follows. When EFDT replaces a test, there is a risk that the model as a whole will derease in accuracy in the short term, as the substrees that have been removed, while suboptimal, may be better than the simple split with which they are replaced. Ensembles with change detection use ADWIN change detectors to determine if change is occurring, and if so, replace trees with poor performance. The change detectors are triggered when accuracy drops. When there is drift, EFDT is likely to replace nodes (and thus corresponding subtrees)—triggering change detectors due to loss of accuracy. Thus, EFDT's response to drift, in which it is already adjusting the tree to the new distribution, will trigger total removal of the tree.

Therefore in settings with concept drift (as all our synthetic streams are), where ensembles happen to feature both a bagging component and an ADWIN change detector, a tree created by EFDT is more likely to be removed when a change is detected without the tree growing large enough or persisting long enough to meaningfully contribute to prediction.

Table 26: OzaBagADWIN - Synthetic streams with concept drift

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	**0.11929**	0.00097	60	0.12652	0.00042	137
recurrent—led	**0.26175**	2e-04	4	0.26285	2e-04	28
recurrent—randomtree	0.09511	0.00212	207	**0.0846**	0.00232	273
recurrent—sea	0.11452	0.00023	49	**0.11262**	0.00018	69
recurrent—stagger	**0.00172**	0.00014	15	0.00299	0.00065	15
recurrent—waveform	0.15625	0.00021	37	**0.14762**	0.00015	55
hyperplane—1	**0.10114**	0.00012	115	0.10371	0.00012	119
hyperplane—2	**0.10998**	0.00019	38	0.11387	0.00019	52
hyperplane—3	**0.09972**	0.00011	170	0.10153	0.00011	163
hyperplane—4	**0.10515**	0.00029	26	0.10753	0.00031	40
rbf—drift-1	0.07738	0.00029	275	**0.07201**	0.00026	220
rbf—drift-2	0.13282	0.00082	15	**0.11136**	0.00059	41
rbf—drift-3	0.11143	0.00062	261	**0.10947**	6e-04	185
rbf—drift-4	0.30834	0.00193	2	**0.29876**	0.00209	2
recurrent—abrupt—222	0.12285	0.03384	3	**0.11331**	0.03473	3
recurrent—abrupt—322	0.16658	0.02614	4	**0.15517**	0.02375	4
recurrent—abrupt—332	**0.11804**	0.04584	7	0.12188	0.05251	7
recurrent—abrupt—333	0.14935	0.01806	24	**0.14827**	0.02198	24
recurrent—abrupt—334	**0.1531**	0.04044	61	0.17691	0.03843	59
recurrent—abrupt—335	0.18661	0.0267	112	**0.15904**	0.05559	111
recurrent—abrupt—422	**0.12408**	0.03756	4	0.1336	0.04196	4
recurrent—abrupt—444	0.17194	0.01932	182	**0.15267**	0.04052	185
recurrent—abrupt—522	**0.11097**	0.04375	4	0.11422	0.04112	4
recurrent—abrupt—555	0.29009	0.00912	418	**0.15781**	0.01077	777
Unique Wins	**11**			**13**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics | **p-value: 0.41941** | **Confidence Interval: 0.35756 — 1**

Table 27: LevBagNoADWIN - Synthetic streams with concept drift

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.2001	0.00046	1697	**0.1739**	0.00048	1625
recurrent—led	0.32774	0.00024	236	**0.31783**	0.00033	665
recurrent—randomtree	0.19747	0.00163	3020	**0.18482**	0.00199	2605
recurrent—sea	0.14106	0.00013	463	**0.13493**	0.00015	411
recurrent—stagger	**0.19141**	0.00029	23	0.1926	0.00016	23
recurrent—waveform	0.16861	0.00146	877	**0.16064**	0.00085	423
hyperplane—1	0.1141	0.00019	1235	**0.11174**	0.00014	522
hyperplane—2	0.16105	0.00108	1478	**0.1276**	0.00042	193
hyperplane—3	**0.10639**	0.00013	1223	0.10955	0.00012	618
hyperplane—4	0.15599	0.00347	1380	**0.12183**	0.00098	217
rbf—drift-1	0.06514	0.00017	1366	**0.06263**	0.00017	525
rbf—drift-2	0.15039	0.00048	1890	**0.11581**	0.00051	1408
rbf—drift-3	0.09594	0.00043	1339	**0.09147**	0.00038	681
rbf—drift-4	0.30655	0.001	1554	**0.29518**	0.00124	1588
recurrent—abrupt—222	0.32199	0.03015	4	**0.3104**	0.02955	4
recurrent—abrupt—322	0.37013	0.01279	4	**0.36394**	0.02985	4
recurrent—abrupt—332	0.34405	0.08873	8	**0.32268**	0.09904	8
recurrent—abrupt—333	0.35421	0.00811	27	**0.35155**	0.04098	27
recurrent—abrupt—334	0.39439	0.00079	64	**0.37892**	0.05352	63
recurrent—abrupt—335	0.39358	0.00095	124	**0.37639**	0.04541	123
recurrent—abrupt—422	**0.31978**	0.0289	4	0.32019	0.0455	4
recurrent—abrupt—444	0.39685	0.00128	239	**0.37445**	0.04024	239
recurrent—abrupt—522	**0.33077**	0.03754	4	0.33627	0.05462	4
recurrent—abrupt—555	0.3999	0.00472	1939	**0.34243**	0.01166	1915
Unique Wins	**4**			**20**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics | **p-value: 0.00077** | **Confidence Interval: 0.65819 — 1**

EFDT based ensembles demonstrate superior prequential accuracy performance within a significance level of 0.05 when used as a base learner with all boosting strategies: OzaBoost

Table 28: LevBag - Synthetic streams with concept drift

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.11408	0.00043	310	**0.10205**	0.00066	307
recurrent—led	**0.26217**	2e-04	6	0.28177	0.00034	3
recurrent—randomtree	0.06448	0.00103	400	**0.0642**	0.0013	441
recurrent—sea	0.10672	0.00011	111	**0.10639**	0.00011	96
recurrent—stagger	**0.00137**	3e-05	18	0.00141	3e-05	17
recurrent—waveform	**0.15119**	0.00016	223	0.15233	0.00016	155
hyperplane—1	**0.10714**	0.00013	563	0.11024	0.00013	388
hyperplane—2	**0.11543**	2e-04	163	0.11736	2e-04	144
hyperplane—3	**0.10488**	0.00012	822	0.10881	0.00012	510
hyperplane—4	**0.10807**	0.00031	86	0.10908	0.00032	88
rbf—drift-1	**0.0605**	0.00015	708	0.06153	0.00017	323
rbf—drift-2	0.08728	0.00029	84	**0.0825**	0.00027	95
rbf—drift-3	0.08917	0.00037	719	**0.08741**	0.00035	352
rbf—drift-4	0.18823	9e-04	8	**0.17382**	0.00075	16
recurrent—abrupt—222	**0.08568**	0.03789	3	0.09068	0.03983	3
recurrent—abrupt—322	**0.15231**	0.02417	4	0.15752	0.01998	4
recurrent—abrupt—332	**0.11588**	0.04298	7	0.11655	0.04618	7
recurrent—abrupt—333	**0.15151**	0.02658	24	0.16132	0.02506	25
recurrent—abrupt—334	0.21071	0.00723	61	**0.17869**	0.03142	61
recurrent—abrupt—335	0.20627	0.01275	115	**0.1693**	0.04913	114
recurrent—abrupt—422	**0.10922**	0.04752	4	0.11094	0.05145	4
recurrent—abrupt—444	0.17353	0.02142	219	**0.12842**	0.06028	198
recurrent—abrupt—522	**0.11904**	0.04243	4	0.13362	0.05295	4
recurrent—abrupt—555	0.13153	0.01712	977	**0.08684**	0.01581	968
Unique Wins	**14**			**10**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value:** **0.84627**	**Confidence Interval:** **0.24639 — 1**

Table 29: Adaptive Random Forest - Synthetic streams with concept drift

Streams	VFDT Base		EFDT Base	
	Error	Variance	Error	Variance
recurrent—agrawal	0.34164	0.00079	**0.23287**	0.00131
recurrent—led	0.28165	0.00065	**0.26301**	2e-04
recurrent—randomtree	0.24865	0.0037	**0.21096**	0.00363
recurrent—sea	0.15533	0.00024	**0.14944**	0.00023
recurrent—stagger	0.08174	0.00094	**0.00365**	0.00028
recurrent—waveform	**0.15365**	0.00017	0.15474	0.00016
hyperplane—1	0.13866	3e-04	**0.13727**	0.00015
hyperplane—2	0.13581	3e-04	**0.13454**	0.00026
hyperplane—3	0.14088	0.00021	**0.13967**	0.00015
hyperplane—4	0.14896	0.00222	**0.12455**	0.00042
rbf—drift-1	0.17788	0.00129	**0.16278**	0.00143
rbf—drift-2	0.16438	0.00119	**0.14905**	0.00121
rbf—drift-3	0.16781	0.0011	**0.15322**	0.00115
rbf—drift-4	0.20026	0.00079	**0.18832**	0.00076
recurrent—abrupt—222	0.11886	0.01174	**0.00548**	0.00033
recurrent—abrupt—322	0.07783	0.00916	**0.0182**	0.00182
recurrent—abrupt—332	0.16737	0.01644	**0.06472**	0.01184
recurrent—abrupt—333	0.21102	0.01086	**0.11341**	0.0059
recurrent—abrupt—334	0.22163	0.01839	**0.15657**	0.00494
recurrent—abrupt—335	0.28508	0.01309	**0.15258**	0.00231
recurrent—abrupt—422	0.09237	0.01104	**0.01034**	0.00065
recurrent—abrupt—444	0.43723	0.01824	**0.21796**	0.01087
recurrent—abrupt—522	0.06868	0.00863	**0.01917**	0.00181
recurrent—abrupt—555	0.67892	0.00197	**0.58899**	0.01103
Unique Wins	**1**		**23**	

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value:** **<** **0.00001**	**Confidence Interval:** **0.81711 — 1**

(Table 30, p-value 0.00014), OzaBoostADWIN (Table 31, p-value 0.01133), ADOB, BOLE and

Table 30: OzaBoost - Synthetic streams with concept drift

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.1812	0.00087	962	**0.15742**	0.00042	1247
recurrent—led	**0.33442**	0.00043	59	0.33533	0.00041	237
recurrent—randomtree	0.14317	0.00089	898	**0.13294**	0.00115	1269
recurrent—sea	0.13049	0.00012	298	**0.12719**	0.00013	329
recurrent—stagger	0.10382	0.00238	23	**0.08442**	0.00194	22
recurrent—waveform	0.17132	0.00086	167	**0.16516**	0.00044	119
hyperplane—1	**0.10869**	0.00014	309	0.11215	0.00016	298
hyperplane—2	0.13259	0.00037	321	**0.12523**	0.00028	291
hyperplane—3	**0.10511**	0.00012	308	0.11043	0.00015	302
hyperplane—4	0.12736	0.00093	319	**0.11864**	0.00058	297
rbf—drift-1	0.06847	0.00018	363	**0.06664**	0.00017	396
rbf—drift-2	0.14149	0.00039	391	**0.13559**	0.00036	552
rbf—drift-3	0.10352	0.00039	350	**0.10126**	0.00038	363
rbf—drift-4	0.31758	0.00094	339	**0.31195**	0.00088	363
recurrent—abrupt—222	0.24091	0.08901	4	**0.22166**	0.08186	4
recurrent—abrupt—322	0.21457	0.05245	4	**0.17571**	0.06421	4
recurrent—abrupt—332	0.2061	0.05159	8	**0.09681**	0.04476	8
recurrent—abrupt—333	0.21236	0.01179	27	**0.11988**	0.02705	26
recurrent—abrupt—334	0.21905	0.00198	63	**0.10993**	0.02107	61
recurrent—abrupt—335	0.21731	0.00135	121	**0.12118**	0.02446	117
recurrent—abrupt—422	0.2311	0.08934	4	**0.18369**	0.08844	4
recurrent—abrupt—444	0.2074	0.013	227	**0.05954**	0.00795	210
recurrent—abrupt—522	0.2181	0.06227	4	**0.18467**	0.06745	4
recurrent—abrupt—555	0.3756	0.02529	1259	**0.11939**	0.00508	1472
Unique Wins	**3**			**21**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.00014**	**Confidence Interval: 0.70773 — 1**

Table 31: OzaBoostADWIN - Synthetic streams with concept drift

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	**0.16874**	0.001	581	0.1748	0.00088	370
recurrent—led	**0.27943**	6e-04	90	0.28262	0.00156	58
recurrent—randomtree	0.13345	0.00305	886	**0.12445**	0.00323	500
recurrent—sea	0.16147	0.00044	567	**0.1449**	0.00038	177
recurrent—stagger	**0.00152**	0.00011	13	0.01347	0.00807	13
recurrent—waveform	0.18972	0.00032	109	**0.18629**	0.00031	68
hyperplane—1	0.16723	0.00032	560	**0.15898**	0.00028	167
hyperplane—2	0.188	0.00051	482	**0.16057**	0.00042	141
hyperplane—3	0.16682	3e-04	599	**0.15808**	0.00033	174
hyperplane—4	0.17259	0.00081	492	**0.14838**	0.00062	165
rbf—drift-1	0.07998	0.00027	480	**0.07398**	0.00022	269
rbf—drift-2	0.13404	0.00096	111	**0.11857**	0.00057	132
rbf—drift-3	0.11905	0.00052	476	**0.10793**	0.00039	276
rbf—drift-4	0.2341	0.00145	17	**0.22421**	0.00128	13
recurrent—abrupt—222	**0.07473**	0.0385	3	0.09672	0.04831	3
recurrent—abrupt—322	**0.01106**	0.00578	4	0.03094	0.02574	4
recurrent—abrupt—332	0.00034	4e-05	6	**0.00031**	4e-05	6
recurrent—abrupt—333	0.00053	1e-05	22	**0.00042**	0	20
recurrent—abrupt—334	0.00243	0.00014	49	**0.00093**	2e-05	48
recurrent—abrupt—335	0.00475	0.00113	91	**0.0023**	3e-04	89
recurrent—abrupt—422	0.08024	0.0799	4	**0.02032**	0.02003	4
recurrent—abrupt—444	0.04236	0.01526	153	**0.01399**	0.00533	142
recurrent—abrupt—522	**0.01728**	0.01396	4	0.0188	0.01558	4
recurrent—abrupt—555	0.41908	0.06705	580	**0.17661**	0.02764	565
Unique Wins	**6**			**18**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.01133**	**Confidence Interval: 0.56531 — 1**

OnlineSmoothBoost (Tables 32, 33, and 34, all three p-values 0.00077 on account of 20 wins and 4 losses).

Table 32: ADOB - Synthetic streams with concept drift

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.17084	0.00025	1010	**0.15346**	0.00019	1566
recurrent—led	0.31423	0.00025	60	**0.30981**	0.00041	243
recurrent—randomtree	0.13912	0.00104	865	**0.1323**	0.00104	1187
recurrent—sea	0.12015	0.00011	166	**0.11842**	0.00011	187
recurrent—stagger	0.09834	0.00252	23	**0.06311**	0.00206	22
recurrent—waveform	0.17092	0.00075	159	**0.16448**	0.00043	124
hyperplane—1	**0.11168**	0.00012	249	0.11396	0.00013	285
hyperplane—2	0.12744	0.00027	280	**0.12698**	0.00025	238
hyperplane—3	**0.11004**	0.00013	244	0.11241	0.00012	278
hyperplane—4	0.12168	0.00057	277	**0.11813**	0.00046	204
rbf—drift-1	0.06694	0.00017	370	**0.06479**	0.00016	422
rbf—drift-2	0.13939	0.00039	386	**0.13468**	0.00037	554
rbf—drift-3	0.10173	0.00036	350	**0.09913**	0.00034	351
rbf—drift-4	0.3171	0.00092	338	**0.31098**	0.00089	365
recurrent—abrupt—222	**0.12251**	0.03055	4	0.12888	0.03745	4
recurrent—abrupt—322	0.15083	0.05514	4	**0.12394**	0.04509	4
recurrent—abrupt—332	0.12366	0.02724	8	**0.05914**	0.02407	8
recurrent—abrupt—333	0.17282	0.01709	27	**0.10713**	0.01841	27
recurrent—abrupt—334	0.17084	0.00555	63	**0.08981**	0.01511	61
recurrent—abrupt—335	0.1672	0.00628	122	**0.09548**	0.01524	120
recurrent—abrupt—422	0.15191	0.04498	4	**0.14134**	0.04526	4
recurrent—abrupt—444	0.11911	0.00339	228	**0.04105**	0.00638	207
recurrent—abrupt—522	**0.1249**	0.03142	4	0.13476	0.04532	4
recurrent—abrupt—555	0.32364	0.0243	1240	**0.08112**	0.00314	1525
Unique Wins	**4**			**20**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.00077**	**Confidence Interval: 0.65819 — 1**

Table 33: BOLE - Synthetic streams with concept drift

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.17084	0.00025	1010	**0.15346**	0.00019	1566
recurrent—led	0.31417	0.00025	60	**0.30975**	0.00041	243
recurrent—randomtree	0.13912	0.00104	865	**0.1323**	0.00104	1187
recurrent—sea	0.12015	0.00011	166	**0.11841**	0.00011	187
recurrent—stagger	0.09834	0.00252	23	**0.06311**	0.00206	22
recurrent—waveform	0.17092	0.00075	159	**0.16447**	0.00043	124
hyperplane—1	**0.11168**	0.00012	249	0.11396	0.00013	285
hyperplane—2	0.12744	0.00027	280	**0.12698**	0.00025	238
hyperplane—3	**0.11004**	0.00013	244	0.11241	0.00012	278
hyperplane—4	0.12168	0.00057	277	**0.11813**	0.00046	204
rbf—drift-1	0.06694	0.00017	370	**0.06479**	0.00016	422
rbf—drift-2	0.13939	0.00039	386	**0.13468**	0.00037	554
rbf—drift-3	0.10173	0.00036	350	**0.09913**	0.00034	351
rbf—drift-4	0.3171	0.00092	338	**0.31099**	0.00089	365
recurrent—abrupt—222	**0.12248**	0.03055	4	0.12743	0.03735	4
recurrent—abrupt—322	0.1498	0.05508	4	**0.12291**	0.04503	4
recurrent—abrupt—332	0.12366	0.02724	8	**0.05914**	0.02407	8
recurrent—abrupt—333	0.17282	0.01709	27	**0.10713**	0.01841	27
recurrent—abrupt—334	0.17083	0.00555	63	**0.08965**	0.01502	61
recurrent—abrupt—335	0.16706	0.00622	122	**0.0949**	0.01492	120
recurrent—abrupt—422	0.15191	0.04498	4	**0.14134**	0.04526	4
recurrent—abrupt—444	0.11737	0.00302	228	**0.03993**	0.00592	207
recurrent—abrupt—522	**0.12386**	0.03136	4	0.13373	0.04526	4
recurrent—abrupt—555	0.29368	0.01059	1240	**0.07874**	0.00196	1525
Unique Wins	**4**			**20**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.00077**	**Confidence Interval: 0.65819 — 1**

Plain EFDT outperforms plain VFDT on 16 synthetic streams (Table 35, with VFDT outperforming on the remaining 8.

Table 34: OnlineSmoothBoost - Synthetic streams with concept drift

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.19145	0.00037	435	**0.17937**	0.00038	813
recurrent—led	**0.31266**	0.00027	55	0.32583	0.00036	302
recurrent—randomtree	0.20745	0.00268	1064	**0.19807**	0.0023	1355
recurrent—sea	0.1442	0.00013	146	**0.13578**	0.00013	163
recurrent—stagger	**0.17514**	0.00101	23	0.18746	0.00046	23
recurrent—waveform	0.1712	0.00148	127	**0.16125**	0.00119	139
hyperplane—1	0.10209	0.00017	244	**0.098**	0.00013	169
hyperplane—2	0.14834	0.00104	282	**0.12153**	0.00054	76
hyperplane—3	0.09573	0.00011	241	**0.09475**	1e-04	205
hyperplane—4	0.14618	0.00297	273	**0.1171**	0.00112	92
rbf—drift-1	0.084	0.00035	344	**0.07305**	0.00027	278
rbf—drift-2	0.18332	0.00084	424	**0.15977**	0.00076	745
rbf—drift-3	0.11502	0.00058	322	**0.10907**	0.00057	239
rbf—drift-4	0.33978	0.00122	288	**0.33032**	0.0012	345
recurrent—abrupt—222	0.33921	0.02677	4	**0.33905**	0.02662	4
recurrent—abrupt—322	**0.3587**	0.01792	4	0.36061	0.02554	4
recurrent—abrupt—332	0.32643	0.05302	8	**0.30822**	0.06033	8
recurrent—abrupt—333	0.35755	0.0149	27	**0.34818**	0.0266	27
recurrent—abrupt—334	0.38918	0.00202	64	**0.35711**	0.03683	63
recurrent—abrupt—335	0.38905	0.00225	121	**0.37848**	0.02378	123
recurrent—abrupt—422	0.32939	0.03697	4	**0.32422**	0.04199	4
recurrent—abrupt—444	0.38303	0.00529	231	**0.36083**	0.01028	233
recurrent—abrupt—522	**0.32769**	0.04879	4	0.32929	0.04613	4
recurrent—abrupt—555	0.39253	0.00499	1358	**0.34937**	0.0076	1867
Unique Wins	**4**			**20**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.00077**	**Confidence Interval: 0.65819 — 1**

Table 35: Plain single learners (no ensemble) - Synthetic streams with concept drift

Streams	VFDT Base			EFDT Base		
	Error	Variance	Leaves	Error	Variance	Leaves
recurrent—agrawal	0.20846	0.00043	482	**0.19639**	0.00062	829
recurrent—led	**0.33838**	0.00068	54	0.34679	0.00037	320
recurrent—randomtree	0.22404	0.00231	1155	**0.21817**	0.00219	1432
recurrent—sea	0.15251	0.00016	164	**0.14958**	0.00019	183
recurrent—stagger	**0.1882**	0.00047	23	0.19043	0.00047	23
recurrent—waveform	0.19355	0.00171	147	**0.18957**	0.00135	156
hyperplane—1	**0.11566**	0.00021	285	0.11677	2e-04	187
hyperplane—2	0.16785	0.00117	333	**0.14056**	0.00069	76
hyperplane—3	**0.1074**	0.00013	284	0.11208	0.00015	234
hyperplane—4	0.16309	0.00359	316	**0.13384**	0.00147	109
rbf—drift-1	0.11462	0.00053	397	**0.11255**	0.00063	310
rbf—drift-2	0.2858	0.00155	482	**0.2623**	0.00171	766
rbf—drift-3	**0.13821**	0.00068	365	0.14308	0.00087	278
rbf—drift-4	0.40874	0.00141	288	**0.40597**	0.00144	343
recurrent—abrupt—222	0.35403	0.02056	4	**0.35381**	0.02053	4
recurrent—abrupt—322	0.37862	0.01248	4	**0.37596**	0.03198	4
recurrent—abrupt—332	0.3504	0.08913	8	**0.33376**	0.10312	8
recurrent—abrupt—333	**0.36505**	0.01499	27	0.36583	0.04067	27
recurrent—abrupt—334	0.39687	0.00217	64	**0.39001**	0.04158	62
recurrent—abrupt—335	0.39622	0.00319	121	**0.3945**	0.02991	123
recurrent—abrupt—422	**0.33416**	0.02931	4	0.33569	0.04239	4
recurrent—abrupt—444	0.40671	0.00636	235	**0.38967**	0.02007	236
recurrent—abrupt—522	**0.3309**	0.03999	4	0.33374	0.04749	4
recurrent—abrupt—555	0.46461	0.00584	1405	**0.40866**	0.01097	1891
Unique Wins	**8**			**16**		

A **bold** value indicates higher accuracy, and ***bold italics*** indicate a tie.

The test is a one-tailed binomial test to determine the probability that the strategy in the rightmost column would achieve so many wins if wins and losses were equiprobable.

Test Statistics	**p-value: 0.07579**	**Confidence Interval: 0.47858 — 1**

6 Conclusions

EFDT has an advantage relative to VFDT in 28 out of 30 ensemble/stream combination settings when employed as the base learner for online ensembling techniques. For 21 of these settings, the

Table 36: Summary of results

Ensembles	UCI Streams			UCI Shuffled Streams			Synthetic Streams		
	VFDT wins	EFDT wins	p-value	VFDT wins	EFDT wins	p-value	VFDT wins	EFDT wins	p-value
OzaBag	2	16	0.00066	3	15	0.00377	5	19	0.00331
OzaBagAdwin	3	15	0.00377	3	15	0.00377	11	13	0.41941
LevBag without Adwin	6	13	0.08353	7	11	0.24034	4	20	0.00077
LevBag	5	13	0.04813	10	8	0.75966	14	10	0.84627
ARF	3	15	0.00377	4	16	0.00591	1	23	<0.00001
OzaBoost	6	12	0.11894	7	10	0.31453	3	21	0.00014
OzaBoostAdwin	8	11	0.3238	7	11	0.24034	6	18	0.01133
ADOB	5	13	0.04813	5	13	0.04813	4	20	0.00077
BOLE	5	13	0.04813	5	13	0.04813	4	20	0.00077
OnlineSmoothBoost	5	15	0.02069	3	15	0.00377	4	20	0.00077
Plain (no ensemble)	7	13	0.13159	5	13	0.04813	8	16	0.07579

EFDT largely outperforms VFDT as a base learner for ensembles, achieving significance at a standard 0.05 level with **7 ensembles** on UCI streams, **6 ensembles** on shuffled UCI streams, and **8 ensembles** on the synthetic testbench.

The test is a one-tailed binomial test to determine the probability that EFDT-based ensembles would achieve so many wins if wins and losses were equiprobable.

advantage is within a statistically significant level of 0.05. For the 2 (out of 30) ensemble/scenario combinations for which EFDT "loses" (LevBag with synthetic and shuffled UCI streams), the win for VFDT does not reach statistical significance.

EFDT shows promise as a base learner for both boosting and bagging ensembles in synthetic concept-drifting scenarios, achieving lower error than VFDT within a 0.05 significance level with 8 out of 10 ensembles. However, in concept-drifting scenarios, given EFDT's greater short term instability, interactions between EFDT and change detectors may cause some EFDT ensemble components to be prematurely terminated, leading to an erosion of advantage. This interaction needs further study. The influence of change detectors appears to be negligible when concept drift is not present.

On UCI data streams that are not shuffled, using EFDT as a base learner leads to significant wins for 7 ensembles at the 0.05 significance level, with the remaining results falling outside the level of significance. Results on shuffled versions of the streams are similar, with significant wins for EFDT on 6 ensembles. Bagging ensembles are overwhelmingly favored, with four out of five achieving significance for unshuffled streams; however, three boosting strategies, ADOB, BOLE, and OnlineSmoothBoost, with rationalized weighting mechanisms also lead to a win for the EFDT base learner within the 0.05 significance level. The indication seems to be that rationalized regimes for weighting, as noted in ADOB, BOLE and OnlineSmoothBoost, interact positively with HATT to deliver a performance gain over the boosting regimes that weight more naively—OzaBoost and OzaBoostAdwin.

The observations that EFDT does not work well with change detectors based on model error under concept drift, and that it can revise models without requiring a change detector both suggest that bespoke EFDT-based ensemble methods might be effective under concept drift. Existing online ensemble techniques have been developed for the rigid unrevisable models of VFDT. New online ensemble techniques that exploit the flexibility of EFDT provide a promising direction for future research.

Tangentially, while we have not explicitly compared the general performance of bagging and boosting strategies in this paper, as our focus is on the impact of using EFDT as a base learner in ensemble techniques, we offer the hypothesis that strategies that rationalize weighting in boosting meta-algorithms using base learners with limited instability might reduce the performance gap between online bagging and online boosting ensembles pointed out in [6].

Leveraging Bagging is significantly advantaged by an EFDT base learner over a VFDT base learner when mild concept drift is present, as in UCI streams, but the advantage erodes when one moves to a scenario with a more uniform, non-evolving stream, as noted in Section 5.1.2.

This erosion appears to result mostly from the change of Poisson λ value to 6 (from 1, thus ensuring 34% of instances are not discarded by drawing a 0), hinting at the thesis that when the generating distribution is uniform, online bagging with EFDT is advantaged through a batch-like bagging process that mimics leaving out a third of the input instances for each base learner—but when mild concept drift is present the extra instances are more helpful for learning. In a scenario with larger and more continual concept drift, as with our synthetic streams, interaction with change detectors appears to work adversely for EFDT with Leveraging Bagging.

We posit that the outperformance of ensembles with HATT as a base learner over those with HT as a base learner is due to the greater amount of short term instability in HATT, allowing for greater component diversity—which is associated with more effective error reduction—making HATT particularly suitable for use as a base learner in ensembled approaches.

To sum up, our results show that Hoeffding AnyTime Tree (implemented as Extremely Fast Decision Tree, EFDT) significantly outperforms Hoeffding Tree (implemented as Very Fast Decision Tree, VFDT) on prequential accuracy as a base learner for bagging and boosting ensembles on a large set of real and synthetic testbenches, and never underperforms with significance.

References

[1] Rakesh Agrawal, Sakti Ghosh, Tomasz Imielinski, Balakrishna Iyer, and Arun Swami (Jan. 1992). "An Interval Classifier for Database Mining Applications." In: pp. 560–573.

[2] R.S.M. de Barros, S.G.T. de Carvalho Santos, and P. M. G. Junior (July 2016). "A Boosting-like Online Learning Ensemble". In: *2016 International Joint Conference on Neural Networks (IJCNN)*, pp. 1871–1878. DOI: 10.1109/IJCNN.2016.7727427.

[3] Rajen Bhatt and Abhinav Dhall (2012). *Skin Segmentation Dataset: UCI Machine Learning Repository*. URL: https://archive.ics.uci.edu/ml/datasets/skin+segmentation.

[4] Albert Bifet and Ricard Gavalda (2007). "Learning from time-changing data with adaptive windowing". In: *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM, pp. 443–448.

[5] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer (2010). "Moa: Massive online analysis". In: *Journal of Machine Learning Research* 11.May, pp. 1601–1604.

[6] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer (2010). "Leveraging bagging for evolving data streams". In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer, pp. 135–150.

[7] Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà (2009). "New ensemble methods for evolving data streams". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 139–148.

[8] — (n.d.). *CovPokElec Dataset from "New Ensemble Methods for Evolving Data Streams", KDD '09*. URL: https://www.openml.org/d/149.

[9] Albert Bifet and Elena Ikonomovska (n.d.). *Airlines Dataset*. URL: https://www.openml.org/d/1169.

[10] Jock Blackard and Denis Dean (Dec. 1999). "Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis in Predicting Forest Cover Types from Cartographic Variables". In: 24, pp. 131–151.

[11] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone (1984). *Classification and regression trees*. Chapman and Hall, New York.

[12] Leo Breiman (1996). "Bagging predictors". In: *Machine learning* 24.2, pp. 123–140.

[13] Javier Burgués, Juan Manuel Jiménez-Soto, and Santiago Marco (Feb. 2018). "Estimation of the limit of detection in semiconductor gas sensors through linearized calibration models". In: *Analytica Chimica Acta* 1013. DOI: 10.1016/j.aca.2018.01.062.

[14] Javier Burgués and Santiago Marco (2018). "Multivariate estimation of the limit of detection by orthogonal partial least squares in temperature-modulated MOX sensors". In: *Analytica Chimica Acta* 1019, pp. 49–64. ISSN: 0003-2670. DOI: https://doi.org/10.1016/j.aca.2018.03.005. URL: http://www.sciencedirect.com/science/article/pii/S0003267018303702.

[15] Shang-Tse Chen, Hsuan-Tien Lin, and Chi-Jen Lu (2012). "An online boosting algorithm with theoretical justifications". In: *arXiv preprint arXiv:1206.6422*.

[16] Thomas G Dietterich (2000). "Ensemble methods in machine learning". In: *International workshop on multiple classifier systems*. Springer, pp. 1–15.

[17] Pedro Domingos and Geoff Hulten (2000). "Mining high-speed data streams". In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 71–80.

[18] Dheeru Dua and Casey Graff (2017). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

[19] Johannes Gehrke, Venkatesh Ganti, Raghu Ramakrishnan, and Wei-Yin Loh (1999). "BOAT—Optimistic Decision Tree Construction". In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. SIGMOD '99. Philadelphia, Pennsylvania, USA: ACM, pp. 169–180. ISBN: 1-58113-084-8. DOI: 10.1145/304182.304197. URL: http://doi.acm.org/10.1145/304182.304197.

[20] Johannes Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti (2000). "RainForest—a framework for fast decision tree construction of large datasets". In: *Data Mining and Knowledge Discovery* 4.2-3, pp. 127–162.

[21] Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabricio Enembreck, Bernhard Pfahringer, Geoff Holmes, and Talel Abdessalem (2017a). "Adaptive random forests for evolving data stream classification". In: *Machine Learning* 106.9-10, pp. 1469–1495.

[22] Heitor Murilo Gomes, Jean Paul Barddal, Fabricio Enembreck, and Albert Bifet (2017b). "A survey on ensemble learning for data stream classification". In: *ACM Computing Surveys (CSUR)* 50.2, pp. 1–36.

[23] Heitor Murilo Gomes, Jesse Read, and Albert Bifet (2019). "Streaming Random Patches for Evolving Data Stream Classification". In: *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*. Ed. by Jianyong Wang, Kyuseok Shim, and Xindong Wu. IEEE, pp. 240–249. DOI: 10.1109/ICDM.2019.00034. URL: https://doi.org/10.1109/ICDM.2019.00034.

[24] M Harries, J Gama, and A Bifet (n.d.). *NSW Electricity dataset*. URL: https://www.openml.org/d/151.

[25] Verena Heidrich-Meisner and Christian Igel (2009). "Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search". In: *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 401–408.

[26] Wassily Hoeffding (1963). "Probability inequalities for sums of bounded random variables". In: *Journal of the American statistical association* 58.301, pp. 13–30.

[27] Ramon Huerta, Thiago Mosqueiro, Jordi Fonollosa, and Nikolai Rulkov (2016). "Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring". In: *Chemometrics and Intelligent Laboratory Systems* 157, pp. 169–176.

[28] Geoff Hulten, Laurie Spencer, and Pedro Domingos (2001). "Mining time-changing data streams". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 97–106.

[29] Earl Busby Hunt, Janet Marin, and Philip James Stone (1966). *Experiments in Induction*. Academic Press. URL: https://books.google.com.au/books?id=60NDAAAAIAAJ.

[30] Bostjan Kaluza, Violeta Mirchevska, Erik Dovgan, Mitja Lustrek, and Matjaz Gams (Oct. 2010). "An Agent-Based Approach to Care in Independent Living". In: pp. 177–186. DOI: 10.1007/978-3-642-16917-5_18.

[31] Ludmila I Kuncheva (2003). "That elusive diversity in classifier ensembles". In: *Iberian conference on pattern recognition and image analysis*. Springer, pp. 1126–1138.

[32] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore (2010). "Activity recognition using cell phone accelerometers". In: *Proceedings of the Fourth International Workshop on Knowledge Discovery from Sensor Data*, pp. 10–18.

[33] Richard Lyman (2016). *Character Font Images Data Set: UCI Machine Learning Repository*. URL: https://archive.ics.uci.edu/ml/datasets/Character+Font+Images.

[34] Chaitanya Manapragada, Geoffrey I Webb, and Mahsa Salehi (2018). "Extremely fast decision tree". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, pp. 1953–1962.

[35] Chaitanya Manapragada, Geoffrey I Webb, Mahsa Salehi, and Albert Bifet (2020). *Emergent and Unspecified Behaviors in Streaming Decision Trees*. arXiv: 2010.08199 [cs.LG].

[36] Yair Meidan, Michael Bohadana, Yael Mathov, Yisroel Mirsky, Asaf Shabtai, Dominik Breitenbacher, and Yuval Elovici (July 2018). "N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders". In: *IEEE Pervasive Computing* 17, pp. 12–22. DOI: 10.1109/MPRV.2018.03367731.

[37] Rodrigo F de Mello, Chaitanya Manapragada, and Albert Bifet (2019). "Measuring the Shattering coefficient of Decision Tree models". In: *Expert Systems with Applications* 137, pp. 443–452.

[38] Nikunj C. Oza (Oct. 2005). "Online Bagging and Boosting". In: *International Conference on Systems, Man, and Cybernetics, Special Session on Ensemble Methods for Extreme Environments*. Ed. by Mo Jamshidi. New Jersey: Institute for Electrical and Electronics Engineers, pp. 2340–2345.

[39] John Ross Quinlan (1979). "Discovering rules by induction from large collections of examples". In: *Expert systems in the micro electronics age*.

[40] — (1983). "Learning efficient classification procedures and their application to chess end games". In: *Machine learning*. Springer, pp. 463–482.

[41] — (1992). *C4.5: programs for machine learning*. San Mateo, CA: Morgan Kaufmann. URL: http://cds.cern.ch/record/2031749.

[42] Attila Reiss and Didier Stricker (2012). "Introducing a new benchmarked dataset for activity monitoring". In: *Wearable Computers (ISWC), 2012 16th International Symposium on*. IEEE, pp. 108–109.

[43] Byron Roe, Haijun Yang, Ji Zhu, Yong Liu, Ion Stancu, and Gordon McGregor (Sept. 2004). "Boosted Decision Trees as an Alternative to Artificial Neural Networks for Particle Identification". In: *Nuclear Instruments and Methods in Physics Research A* 543. DOI: 10.1016/j.nima.2004.12.018.

[44] Silas Garrido Teixeira de Carvalho Santos, Junior Paulo Mauricio Gonçalves, Geyson Daniel dos Santos Silva, and Roberto Souto Maior" de Barros (2014). "Speeding Up Recovery from Concept Drifts". In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part III*. Ed. by Toon Calders, Floriana Esposito, Eyke Hüllermeier, and Rosa Meo. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 179–194. DOI: 10.1007/978-3-662-44845-8_12.

[45] Robert E Schapire (1990). "The strength of weak learnability". In: *Machine learning* 5.2, pp. 197–227.

[46] Jeffrey Schlimmer and Douglas Fisher (1986). "A case study of incremental concept induction". In: *AAAI*. Vol. 86, pp. 496–501.

[47] Jeffrey Schlimmer and Richard Granger (1986). "Incremental learning from noisy data". In: *Machine Learning* 1.3, pp. 317–354. ISSN: 1573-0565. DOI: 10.1007/BF00116895. URL: http://dx.doi.org/10.1007/BF00116895.

[48] Rocco A Servedio (2003). "Smooth boosting and learning with malicious noise". In: *Journal of Machine Learning Research* 4.Sep, pp. 633–648.

[49] SIGKDD (2015). *2015 KDD Test of Time Award Winners*. URL: https://www.kdd.org/awards/view/2015-kdd-test-of-time (visited on 12/10/2019).

[50] Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Prentow, Mikkel Kjaergaard, Anind Dey, Tobias Sonne, and Mads Jensen (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. Seoul, South Korea: ACM, pp. 127–140. ISBN: 978-1-4503-3631-4.

[51] W Nick Street and YongSeog Kim (2001). "A streaming ensemble algorithm (SEA) for large-scale classification". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, pp. 377–382.

[52] Wallace Ugulino, Débora Cardador, Katia Vega, Eduardo Velloso, Ruy Milidiu, and Hugo Fuks (Oct. 2012). "Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements". In: vol. 7589. ISBN: 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6_6.

[53] Paul E Utgoff (1989). "Incremental induction of decision trees". In: *Machine learning* 4.2, pp. 161–186.

[54] Benjamin Visser and Henry Gouk (n.d.). *AWS Dataset*. URL: https://www.openml.org/d/41424.

[55] Larry Wasserman (n.d.). *Lecture Notes 3 — Review: Bounded Random Variables - Hoeffding's bound*. URL: https://www.stat.cmu.edu/~larry/=stat705/Lecture3.pdf.

[56] Geoffrey I Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean (2016). "Characterizing concept drift". In: *Data Mining and Knowledge Discovery* 30.4, pp. 964–994.

[57] David H Wolpert and William G Macready (1997). "No Free Lunch Theorems for Optimization". In: *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION* 1.1, p. 67.

[58] — (2005). "Coevolutionary free lunches". In: *IEEE Transactions on Evolutionary Computation* 9.6, pp. 721–735.

[59] David H. Wolpert (1996). "The Lack of A Priori Distinctions Between Learning Algorithms". In: *Neural Computation* 8.7, pp. 1341–1390. DOI: 10.1162/neco.1996.8.7.1341. eprint: https://doi.org/10.1162/neco.1996.8.7.1341. URL: https://doi.org/10.1162/neco.1996.8.7.1341.

A CPU Time comparison

Table 37: Real data streams, unshuffled: CPU Times in Seconds

(a) See Table 1 for key to data streams

	Learners ↓ *Streams* →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	ARF EFDT	154	4135	2	213	669	1692	1297	479	11	20	1609	6	497	70	11	50	200	4	2	390
2	ARF VFDT	98	2694	2	143	426	1526	1004	430	8	18	1118	4	456	53	7	35	179	4	2	337
3	ADOB EFDT	48	2325	2	136	453	1398	997	273	11	9	853	3	490	69	5	28	229	2	3	292
4	ADOB VFDT	39	1719	2	96	300	1398	1592	314	6	10	760	3	448	48	7	41	270	2	3	266
5	BOLE EFDT	51	2325	2	136	439	1446	1047	222	10	11	845	3	520	69	6	28	329	2	3	282
6	BOLE VFDT	39	1579	3	93	258	1127	1245	266	6	10	773	2	438	50	7	36	279	2	2	302
7	LevBagNoAdwin EFDT	97	4323	3	235	826	5414	2626	690	22	20	2066	5	881	110	11	59	796	3	5	475
8	LevBagNoAdwin VFDT	43	2173	3	132	333	3914	2622	340	11	16	1581	4	752	47	10	52	401	3	4	246
9	LeveragingBag EFDT	211	5850	2	208	642	4094	1477	556	14	16	1473	6	548	63	12	47	330	5	5	435
10	LeveragingBag VFDT	149	2598	2	143	425	2130	1073	493	9	20	1432	4	480	51	9	37	252	4	5	404
11	OnlineSmoothBoost EFDT	73	3012	3	169	659	3787	2671	718	27	18	2368	4	841	97	12	58	644	3	5	471
12	OnlineSmoothBoost EFDT	46	2023	2	112	365	5038	1978	532	11	17	1596	3	921	56	11	48	340	3	4	360
13	OzaBag EFDT	54	2364	2	123	438	3060	1675	470	16	13	1502	3	606	48	8	38	547	3	3	307
14	OzaBag VFDT	31	1483	2	72	228	2304	1256	313	7	12	792	2	513	34	8	35	247	2	3	175
15	OzaBagAdwin EFDT	145	2874	1	148	403	2653	756	392	7	15	1032	4	438	46	9	35	224	4	2	296
16	OzaBagAdwin VFDT	113	2011	1	135	345	2506	1199	421	7	18	1079	2	417	43	8	24	245	3	3	356
17	OzaBoost EFDT	49	2762	3	128	445	1210	1025	282	10	11	724	2	461	66	6	27	341	2	3	301
18	OzaBoost VFDT	39	1820	2	70	302	1142	1552	348	7	10	953	2	484	40	7	39	260	2	3	268
19	OzaBoostAdwin EFDT	147	2908	1	207	702	1591	1005	305	9	13	916	4	354	77	5	25	190	4	1	422
20	OzaBoostAdwin VFDT	148	2273	1	154	543	1233	731	266	7	13	784	4	306	65	4	24	203	3	2	331
21	Plain EFDT	9	362	1	25	86	609	370	91	4	4	413	1	109	9	2	7	91	1	1	54
22	Plain VFDT	7	251	1	16	50	463	296	89	2	4	443	1	144	6	2	8	55	1	1	45

Table 38: Real data streams, shuffled: Average CPU Times in Seconds

(a) See Table 1 for key to data streams

	Learners ↓ *Streams* →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	ARF EFDT	128	3894	5	235	700	40692	4616	1109	13	82	13049	5	5153	84	40	124	1122	8	8	962
2	ARF VFDT	125	2832	5	127	273	33697	2489	824	14	46	20393	4	2974	89	32	89	792	6	6	606
3	ADOB EFDT	45	2672	3	178	750	3293	3011	397	11	45	3160	3	2493	102	24	107	941	4	5	398
4	ADOB VFDT	41	1655	3	96	319	2452	1975	383	9	26	2292	2	1282	70	22	69	539	4	5	325
5	BOLE EFDT	44	2826	4	169	763	3156	4778	442	11	41	3428	3	2326	73	25	104	804	4	6	418
6	BOLE VFDT	42	1756	3	98	363	2450	2028	382	10	26	2701	2	1239	73	22	69	504	3	6	326
7	LevBagNoAdwin EFDT	103	4767	5	379	2300	6559	5563	1310	17	91	9014	6	4399	152	40	208	1740	7	7	791
8	LevBagNoAdwin VFDT	55	2639	4	124	438	4742	2409	557	12	40	4182	4	1187	78	23	75	732	4	6	338
9	LeveragingBag EFDT	152	4551	5	345	995	7005	5789	1185	13	82	10610	5	4735	126	39	176	1487	8	7	778
10	LeveragingBag VFDT	145	2708	5	128	448	4658	2540	612	14	45	9802	4	4905	84	22	82	764	5	6	320
11	OnlineSmoothBoost EFDT	81	3508	5	224	1435	7114	5038	1119	14	58	10257	4	4425	133	36	171	1338	6	7	559
12	OnlineSmoothBoost EFDT	44	1985	4	91	378	5160	2230	773	11	26	5626	2	1852	65	29	77	709	4	7	386
13	OzaBag EFDT	70	2269	4	142	928	4557	3963	884	9	44	5811	2	2563	80	20	126	820	5	4	380
14	OzaBag VFDT	40	1412	3	72	277	2893	1833	391	8	23	3262	2	1176	50	17	52	450	2	4	244
15	OzaBagAdwin EFDT	135	2858	5	165	601	4698	3919	987	12	44	6411	3	3123	108	30	118	852	5	7	447
16	OzaBagAdwin VFDT	96	1853	5	101	340	3473	2455	634	12	23	17646	2	1526	83	20	67	617	5	7	311
17	OzaBoost EFDT	56	2805	3	172	1006	3318	2478	334	10	41	3287	3	2219	88	23	103	872	3	6	592
18	OzaBoost VFDT	41	1902	2	104	293	2170	2076	513	10	27	2437	2	961	70	20	58	560	3	6	329
19	OzaBoostAdwin EFDT	92	3615	3	173	625	3287	3001	501	9	44	3929	3	3307	89	23	95	817	5	5	506
20	OzaBoostAdwin VFDT	69	2350	3	96	314	2268	2384	382	10	31	2747	3	1371	73	21	57	562	4	6	329
21	Plain EFDT	10	354	1	19	74	997	459	105	2	8	881	1	320	11	4	16	100	1	1	66
22	Plain VFDT	6	252	1	16	54	797	336	106	2	6	704	1	194	6	3	11	65	1	1	50

Table 39: Synthetic data streams: Average CPU Times in Seconds

(a) See Table 2 for key to data streams

	Learners ↓ *Streams* →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	ARF EFDT	133	306	143	64	64	635	126	128	133	112	112	116	115	128	17	17	19	22	22	23	20	39	19	47
2	ARF VFDT	121	171	95	68	76	332	83	79	77	73	79	80	80	85	16	18	17	20	21	21	19	26	19	29
3	ADOB EFDT	123	233	100	70	48	364	73	82	72	71	88	110	88	109	6	7	9	11	11	12	7	19	7	44
4	ADOB VFDT	105	173	77	62	64	254	48	51	44	46	51	64	52	65	5	6	7	9	10	11	6	14	7	27
5	BOLE EFDT	109	264	107	61	55	346	78	78	71	71	88	109	88	107	6	7	9	10	12	13	7	19	8	45
6	BOLE VFDT	105	139	80	62	64	241	44	49	47	50	54	67	54	69	5	7	7	8	9	11	7	15	7	28
7	LevBagNoAdwin EFDT	144	453	264	75	59	647	152	139	152	109	194	195	194	196	8	7	12	14	16	13	8	20	8	64
8	LevBagNoAdwin VFDT	93	198	105	63	55	383	81	80	86	89	73	90	76	89	6	7	7	8	8	11	6	16	6	33
9	LeveragingBag EFDT	108	283	136	73	67	635	156	141	168	129	198	164	208	134	15	17	19	22	23	26	17	37	17	81
10	LeveragingBag VFDT	90	158	99	76	54	341	93	89	92	89	99	106	99	97	16	16	16	17	20	21	13	26	16	43
11	OnlineSmoothBoost EFDT	113	370	132	78	50	555	111	101	123	100	147	163	149	139	7	8	12	14	15	16	9	27	10	64
12	OnlineSmoothBoost EFDT	88	186	103	80	65	360	67	76	71	65	66	77	64	72	6	6	9	10	10	11	7	15	8	30
13	OzaBag EFDT	86	262	105	68	55	375	77	70	78	71	95	114	94	91	5	6	8	10	11	12	6	19	6	48
14	OzaBag VFDT	66	136	77	51	52	228	45	46	49	46	45	53	46	44	4	4	6	5	7	8	5	11	5	24
15	OzaBagAdwin EFDT	81	204	107	64	63	427	93	82	103	77	103	100	101	65	14	14	18	18	20	21	14	31	16	56
16	OzaBagAdwin VFDT	71	178	84	59	60	275	63	59	67	61	63	70	62	64	14	14	14	15	17	18	14	23	14	30
17	OzaBoost EFDT	105	267	106	69	55	326	76	83	76	77	85	109	91	109	6	7	9	10	11	12	6	19	7	44
18	OzaBoost VFDT	79	152	83	58	46	245	50	55	51	50	53	65	54	65	5	6	7	9	9	10	6	13	6	27
19	OzaBoostAdwin EFDT	106	203	133	70	61	400	82	83	82	84	90	105	93	109	11	13	16	17	17	19	13	25	14	42
20	OzaBoostAdwin VFDT	97	143	87	76	53	291	65	66	61	62	61	77	61	92	13	13	14	16	18	18	12	21	13	29
21	Plain EFDT	53	62	57	49	47	90	9	8	9	8	11	12	8	10	1	1	2	2	2	2	1	2	1	6
22	Plain VFDT	51	59	52	48	47	71	6	6	6	6	6	8	6	7	1	1	1	1	1	2	1	2	1	3

Chapter 7

Conclusions

This work studies the problem of learning from evolving data streams without making strong assumptions on the nature of data generating streams. We summarize our findings below.

Note that we assume a significance level of 0.05 for our tests, which are one-tailed binomial tests to determine the probability that the winning strategy would have achieved as many wins on as many streams if wins and losses were equiprobable.

7.1 Findings

- **Our online decision tree Hoeffding AnyTime Tree (HATT) is statistically more efficient than the state-of-the-art Hoeffding Tree (HT), converges to the ideal batch tree while Hoeffding Tree does not, and obtains improved prequential accuracy on a standard testbench.**

  Extremely Fast Decision Tree (EFDT, the concrete implementation of the theoretical construct HATT) outperforms Very Fast Decision Tree (VFDT, the implementation of the theoretical construct HT) on prequential accuracy on a testbench consisting of real and synthetic data streams, with a small overhead due to split revision that results in a minimal increase the order of time complexity.

  EFDT is both more statistically efficient at learning a stationary distribution and faster to adjust in the face of concept drift.

  On the stability-plasticity spectrum, EFDT is more plastic than VFDT, choosing intermediate splits on the basis of available data, and allowing splits to be revised after they have been made—in contrast to VFDT which fixes each split irreversibly and therefore waits to achieve greater certainty that a split will be indefinitely stable before splitting.

  EFDT chooses to split if the Hoeffding Test determines an attribute better than the current split attribute is available to split upon, while VFDT only splits if the Hoeffding Test determines that the top attribute is better than the second best one. Consequently, EFDT achieves greater statistical efficiency by growing tree structure earlier, and because it revises splits, it converges to the ideal batch tree.

- **Compared to Hoeffding Tree, HATT achieves superior prequential accuracy as a base learner across boosting and bagging ensembles on real streams with and without concept drift, and on a synthetic concept drift testbench. This supports the idea that greater plasticity in base learners, and thus greater ensemble diversity is advantageous for online learning.**

  Ensembles of decision trees are widely used in practice; decision trees are highly interpretable, and diversity of base models in ensembles is considered beneficial for learning as diverse base models would be able to capture different aspects of a concept. Thus it was important to study how EFDT performed as a base learner for ensembles; we conducted a large experimental study of the use of EFDT as a base learner with online boosting and bagging ensembles, including random forests, on UCI and synthetic data streams.

  We found that EFDT offers significantly improved prequential accuracy performance on most settings within a significance level of 0.05 on our testbench compared to VFDT. EFDT never underperforms with significance. This improved performance is possibly due to the greater plasticity of EFDT—plasticity of the base learner is hypothesized to improve ensemble diversity and hence boost ensemble performance.

- **Extremely Fast Decision Adaptive Tree (EFAT), which uses the eager splitting strategy of EFDT together with the adaptive subtree substitution mechanism from Hoeffding Adaptive Tree (HAT), outperforms HAT on prequential accuracy on real streams with and without concept drift, and on a synthetic concept drift testbench.**

  It is possible that building structure rapidly enables a higher quality classifier to be deployed sooner—reinforcing the idea that there might be an unsubstantiated bias in the development of machine learning for building highly stable classifiers.

- **Unspecified features and emergent behaviors in implementations of the state-of-the-art online tree learner Hoeffding Tree have significant previously unknown effects on its behavior and prequential accuracy performance**

  Unspecified details left to interpretation can significantly and consequentially influence algorithm behavior, and the true reasons for the experimental success of algorithms can deviate significantly from the theoretical justifications (as demonstrated in this work in Chapter 3). Therefore, even the assumption that an algorithm performs well in practice because of its demonstrated theoretical properties is subject to scrutiny. For instance, proving bounded divergence from an ideal batch tree is only useful in the scenario where streams are assumed to be stationary, and in practice, unspecified implementation details and associated emergent behaviors may unexpectedly account for a large portion of performance, as demonstrated in this work.

  – Inherent amnesia in Hoeffding Tree aids recovery from concept drift

- The unspecified feature of "re-splitting" on attributes that have already been used leads to a significant prequential accuracy improvement for MOA's VFDT (p-value < 0.00001 on a large concept drift testbench)
- The unspecified approximation of infogain (by choosing n as the number of data instances and not the number of $\overline{\Delta G_t}$ computations) appears to have no effect on VFDT prequential accuracy with concept-drifting streams
- Using the weight seen at leaf nodes instead of the actual number of node timesteps—that is, the number of examples that have actually been seen at the node—does not seem to have an effect on VFDT prequential accuracy under concept drift
- Combining all the unspecified strategies leads to a win (p-value 0.00002) implying they interact in an overall beneficial manner for VFDT prequential accuracy under concept drift
- We demonstrate "node evisceration", a strategy of deliberately clearing both node statistics and the class distribution within a leaf node when a previously used attribute emerges as the best one, and suggest that it be formally incorporated into VFDT as a policy for natural drift resilience in VFDT (on account of its improved prequential accuracy performance; p-value 0.00002) in lieu of the unspecified feature of "re-splitting"

- **Unspecified features and emergent behaviors in the state-of-the-art adaptive tree learner Hoeffding Adaptive Tree have significant previously unknown effects on its behavior and prequential accuracy performance under concept drift**

  - Because Hoeffding Adaptive Tree (HAT) is derived from Hoeffding Tree, the unspecified strategy of "re-splitting" on attributes that have already been used also benefits HAT. This strategy leads to a significant prequential accuracy improvement for MOA's HAT (p-value 0.00024, well within a significance level of 0.05 on a large concept drift testbench with significant diversity in streams)
  - While the base HAT algorithm makes no provision for alternates voting, the unspecified strategy in the implementation of allowing them to do so provides the system with a form of lookahead wherein alternate subtrees under development—concepts being developed as potential future replacements—begin to take part in providing predictions instead of waiting until a replacement has taken place to start doing so. Allowing a single alternate to vote results in significant improved prequential accuracy for HAT (p-value 0.01133)
  - When alternates of alternates are also allowed to vote, there is a deterioration compared to allowing only a single alternate to vote (p-value 0.02452)
  - The unspecified strategy in the HAT implementation of not allowing single leaves to vote but otherwise allowing multiple alternates to vote is outperformed on prequential accuracy by allowing multiple alternates, including single leaves, to vote (p-value 0.03196)
  - The unspecified strategy in the HAT implementation of weighting examples with weights drawn from a Poisson(1) distribution outperforms not weighting instances (p-value 0.00331)
  - Using the weight seen at leaf nodes instead of the actual number of node timesteps—that is, the number of examples that have actually been seen at the node—does not

improve HAT prequential accuracy under concept drift (performance is identical on our testbench)

- The unspecified approximation of infogain (by choosing n as the number of data instances and not the number of $\overline{\Delta G_t}$ computations) appears to favor HAT prequential accuracy on concept-drifting streams (p-value 0.04657)
- The unspecified feature of replacing the root when a root alternate splits leads to a decline in prequential accuracy for HAT compared to a baseline HAT
- The unspecified feature of a non-root alternate replacing its corresponding mainline node when it splits also leads to a decline in prequential accuracy for HAT compared to a baseline HAT
- The combination of the unspecified features relating to root and non-root replacement upon splitting reverses the decline in prequential accuracy performance!
- Combining the unspecified features from Hoeffding Tree leads to a significant improvement in prequential accuracy performance for HAT over a baseline HAT (p-value 0.00002)

7.2 Future Work

I have undertaken a very detailed investigation into the utility of many detailed learning mechanisms in the context of concept drift and proposed a new statistically efficient alternative to Hoeffding Trees. These productive lines of research open many avenues for further exploration.

- Ensemble methods tailored to work with EFDT might be worth exploring. The observations that EFDT doesn't work well with change detectors based on model error, but that it can revise models without requiring a change detector to replace them both suggest that bespoke EFDT-based ensemble methods might be effective under concept drift.

- The creation of a standardized testbench for online learning is a pressing matter for the development of the field. At present, small testbenches with limited diversity in data characteristics are used to validate hypotheses about strategies. While we massively expand the testbench size in our work, providing a discriminative testbench that was able to extract differences in algorithm behavior due to a host of unspecified as well as designed features, we believe this is only the start of a long line of work deliberating and proposing testbenches that enable a critical examination of differences in algorithm behavior and performance.

- While the development of the field has focused on decision tree models, it might be instructive to understand how other classes of learning models behave in online scenarios.

- The advantages and disadvantages of using adaptive models as opposed to simpler strategies of periodically rebuilding models without rationalised adaptation need to be understood. Adaptive models are far outnumbered by their counterparts designed for static streams, possibly because there is little understanding of the relative utility of adaptive models with respect to periodically rebuilding static models.

- There is a great need to develop evaluation methods for online learning that offer a useful and usable quantification of their utility. Measuring online learners by batch-learning standards of generalizability and divergence will not allow us to assess the true utility of online learners in the scenarios that they are designed for—scenarios with unpredictable change in the generating concept.
- The relative performance of ensembles needs to be studied, in particular, whether boosting ensembles may approach bagging ensembles in performance with carefully designed and rational weighting mechanisms for instances and ensemble components.
- Characterizations and measurements of concept drift need to be applied to real streams in order to generate a repository of examples of concept drift in real data streams.
- The assumption of identically and independently distributed data must be done away with to the extent possible if we are to take larger steps towards the goal of strong artificial intelligence.

Appendices

.1 Publication on Shattering Coefficient

Expert Systems With Applications 137 (2019) 443–452

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Measuring the Shattering coefficient of Decision Tree models

Rodrigo F. de Mello [a,*], Chaitanya Manapragada [b], Albert Bifet [c]

[a] Av. Trabalhador Saocarlense, 400, São Carlos, SP 13560-970, Brazil
[b] Monash University, Wellington Rd, Clayton VIC 3800, Australia
[c] LTCI, Télécom ParisTech, Office: C201-2, 46 rue Barrault, Paris, Cedex 13 75634, France

ARTICLE INFO

Article history:
Received 16 May 2019
Revised 1 July 2019
Accepted 6 July 2019
Available online 8 July 2019

Keywords:
Shattering coefficient
Decision Trees
Statistical Learning Theory
Learning guarantees

ABSTRACT

In spite of the relevance of Decision Trees (DTs), there is still a disconnection between their theoretical and practical results while selecting models to address specific learning tasks. A particular criterion is provided by the Shattering coefficient, a growth function formulated in the context of the Statistical Learning Theory (SLT), which measures the complexity of the algorithm bias as sample sizes increase. In attempt to establish the basis for a relative theoretical complexity analysis, this paper introduces a method to compute the Shattering coefficient of DT models using recurrence equations. Next, we assess the bias of models provided by DT algorithms while solving practical problems as well as their overall learning bounds in light of the SLT. As the main contribution, our results support other researchers to decide on the most adequate DT models to tackle specific supervised learning tasks.

1. Introduction

As important as the empirical validation is the process involved in justifying the choice of a model over another given some supervised learning task (Langley, 1988), in that sense, theoretical results establish incontrovertible truths about learning guarantees (de Mello & Ponti, 2018; Vapnik, 1995) and model complexities (Bousquet & Herrmann, 2003; Valiant, 1984). The most prominent example is the Statistical Learning Theory (SLT) which provides theoretical guarantees for supervised learning by using concentration inequalities (Devroye, Györfi, & Lugosi, 1996). Based on such a framework, one can formulate and analyze the learning convergence in the context of a static (fixed) joint probability distribution from which data are identically and independently sampled (de Mello & Ponti, 2018; Vapnik, 1995).

Theoretical proofs and guarantees have also been provided using a different set of tools, for instance, Schaffer (1994) provided the Machine Learning version of the No Free Lunch Theorem, Schapire (1990) confirmed boosting approaches the Bayes Error from weak learners that simply do better than random guessing, Oza and Russell (2001) showed that online bagging converges to its batch equivalent, Dempster, Laird, and Rubin (1977) proved the convergence of the Expectation Maximization algorithm, while Lee and Seung (2001) used an analogous argument to prove the convergence of algorithms using non-negative matrix factorization.

From the theoretical point of view, Decision Trees (DTs) have particularly motivated proofs due to the easy interpretation of their results in terms of the most relevant attributes and their corresponding intervals or values. This motivated attempts at determining conditions for pruning such learning models in order to gain predictive advantage (Kim & Koehler, 1994; Schaffer, 1991), at proving that binary DTs that optimally minimize the number of tests are NP-complete (Laurent & Rivest, 1976), and at showing that the divergence for online Hoeffding trees from batch DTs is bounded (Domingos & Hulten, 2000). Yíldíz (2015) prove the lower bounds of the Vapnik–Chervonenkis dimension for a univariate DT hypothesis class. Complementarily, a theoretical comparison of Gini and Information Gain for DTs concluded that over typically tested datasets, such two attribute-selection mechanisms present small divergences, backing up experimental evidences (Raileanu & Stoffel, 2004).

DTs are also versatile as confirmed by their great variety of applications in domains specially involving classification, but also regression, feature selection and even clustering (Breiman, Friedman, Olshen, & Stone, 1984; Pandya, Pandya, & Infotech, 2015). In addition, they also have the ability of handling different data types, such as nominal, numeric and textual (Rokach & Maimon, 2008), and are particularly effective in ensembles, as a large related literature confirms (Lucchese et al., 2017; Mashayekhi & Gras, 2017).

In spite of the relevance of DTs and the mentioned attempts at formalizing relevant aspects of them, there is still a disconnection

* Corresponding author.
E-mail addresses: mello@icmc.usp.br (R.F. de Mello), chaitanya.manapragada@monash.edu (C. Manapragada), albert.bifet@telecom-paristech.fr (A. Bifet).

https://doi.org/10.1016/j.eswa.2019.07.012
0957-4174/

between theoretical results, after the SLT, and practical decisions on how to select the best tree model to address some learning task. In real scenarios, we see the importance of deciding whether one given tree model is more or less complex than another, and from that building up some partial order or complexity rankings among DTs which, in conjunction with typical performance measurements (e.g. accuracy), support the model selection while addressing specific learning tasks.

One particularly useful indicator of relative complexity provided by the SLT is the Shattering coefficient (or growth function) (de Mello & Ponti, 2018; Vapnik & Chervonenkis, 1971) that measures how quickly the space of admissible functions (a.k.a. algorithm bias) grows as the sample size increases. Thus, a smaller coefficient corresponds to a less complex bias, so that less functions are considered by the algorithm when building up classifiers and regressors. While this is a central aspect of the SLT, in order to pick a classification or regression function with such a lower coefficient, we must first know all those growth functions for all algorithms under comparison. This has been left somewhat unexplored, even though the notion of a Shattering coefficient has existed for decades (Vapnik & Chervonenkis, 1971).

In an attempt to fill out this gap, we propose in this paper a method to numerically compute the Shattering coefficient of Decision Tree models to establish the basis for their relative theoretical complexity analysis (Vapnik, 1995). DTs are among the most commonly used supervised algorithms in Machine Learning and, to the best of our knowledge, this is the first time such a detailed theoretical analysis has been performed based on such growth function defining the space of admissible functions (a.k.a. bias) of DTs. The only remotely similar class of studies we have come across either aim to minimize message/description lengths for coding trees (Mehta, Rissanen, & Agrawal, 1995; Quinlan & Rivest, 1989; Wallace & Patrick, 1993), or provide a taxonomy of various types of algorithmic complexity (Buhrman & de Wolf, 2002). Both threads provide a very different view of "complexity" than in this manuscript, one that does not consider directly the size of the space of admissible functions.

In summary, our contribution estimates the Shattering coefficient from specific tree models rather than the overall complexity of a DT algorithm as a whole. The reader will observe throughout this paper that our computation is based on tree structures produced after applying a DT algorithm, most specially the fractions of training examples assigned to each tree node, including leaves.

In order to understand the practical impacts of the Shattering coefficient, we employed the following SLT bound after the Empirical Risk Minimization Principle (EMRP) (de Mello & Ponti, 2018):

$$P(\sup_{f\in\mathcal{F}} |R_{\text{emp}}(f) - R(f)| \geq \epsilon) \leq 2\mathcal{N}(\mathcal{F}, n)\exp(-2n\epsilon^2),$$

in which $\epsilon \in [0, 1]$ defines some acceptable divergence in between the empirical risk (or error) $R_{\text{emp}}(f) \in [0, 1]$ and its expected value $R(f) \in [0, 1]$, f is any classification or regression function contained in the algorithm bias $\mathcal{F}$, n is the sample size, and $\mathcal{N}(\mathcal{F}, n)$ is the Shattering coefficient (or growth function) defining the size of the algorithm bias as $n \to \infty$.

In this regard, by having a method for computing the Shattering Coefficient of DTs, one can take a set of tree models and define their relative complexity in terms of their spaces of admissible functions or, simply, the size of the bias of the algorithms responsible for producing those models. Of course the Shattering coefficient must be combined with model performance measurements, so that if models provide close enough accuracies but one of them has a significantly smaller coefficient, it should be selected as its bias is less complex but still efficient for representing the problem. On the other hand, one could have a DT model resultant from some simpler bias but providing insufficient representation to the target task, so that underfitting would happen.

We see this work as a starting point for a rigorous and thorough analysis of such growth functions for various commonly used classes of algorithms, so that the machine learning discourse would more frequently adopt the use of diligent theoretical arguments for performance comparisons in addition to the meticulous experimental arguments that have been considered in the current mainstay.

This paper is organized as follows: the necessary background on the Statistical Learning Theory is provided in Section 2. Our approach for measuring the complexity of DTs is introduced in Section 3. Experimental results are shown and discussed in Section 4. Finally, Section 6 presents our conclusions and future work.

2. Background

The Statistical Learning Theory (SLT) employs probabilistic convergence in conjunction with concentration inequalities to ensure supervised learning guarantees for general-purpose algorithms (von Luxburg & Schölkopf, 2011). SLT considers an input space $\mathcal{X}$ and an output space $\mathcal{Y}$ in which every $x_i \in \mathcal{X}$ corresponds to a data example (or feature vector) and $y_i \in \mathcal{Y}$ is an expected class label. In order to ensure learning, this theoretical framework assumes a joint probability distribution (JPD) $P(\mathcal{X} \times \mathcal{Y})$ mapping all possible combinations of input examples into their corresponding classes. Learning is then defined as the process of finding a classifier $f : \mathcal{X} \to \mathcal{Y}$, providing the minimum error or loss as possible so that it best represents $P(\mathcal{X} \times \mathcal{Y})$.

In order to complement, supervised learning requires the definition of a loss function to measure the risk of a classifier $f \in \mathcal{F}$, in which $\mathcal{F}$ corresponds to the algorithm bias, i.e., the set of admissible functions used by the supervised learning algorithm to represent every possible classifier. The two most common functions used in supervised learning are the $0-1$ loss and the squared-error functions (de Mello & Ponti, 2018). Algorithms typically employ the gradient descent method to adapt hyperparameters in order to minimize such loss functions (Bergstra, Bardenet, Bengio, & Kégl, 2011).

Five assumptions are taken by the SLT in order to ensure a supervised learning algorithm performs similarly on training data as on unseen examples (Vapnik, 1998; von Luxburg & Schölkopf, 2011): (i) examples must be independent from each other and sampled in an identical manner; (ii) no assumption is made about the joint probability distribution, otherwise one could simply estimate its parameters; (iii) labels can assume nondeterministic values due to noise and class overlapping; (iv) the joint probability distribution is fixed, i.e., it cannot change along time; and (v) this distribution is still unknown at the time of training, thus it must be estimated using training examples.

From those assumptions, Vapnik (1998) satisfied all necessary properties to employ the Law of Large Numbers (Devroye et al., 1996) to define the Empirical Risk Minimization Principle (ERMP):

$$P(|R(f) - R_{\text{emp}}(f)| > \epsilon) \to 0, \quad n \to \infty, \tag{1}$$

according to which the empirical risk of a classifier $R_{\text{emp}}(f)$ probabilistically converges to the risk $R(f)$ (also known as real risk or expected risk) as the sample size n tends to infinity, considering some $\epsilon \in \mathbb{R}_+$ (those two risks are in range [0,1]). The empirical risk is here seen as the training error, which is calculated based on the training sample, defined as follows $R_{\text{emp}}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(x_i, y_i, f(x_i))$, and the (expected) risk is given by $R(f) = \text{E}(l(\mathcal{X}, \mathcal{Y}, f(\mathcal{X})))$, in which E(.) is the expected value, and $\ell(x_i, y_i, f(x_i))$ is the loss function that measures the divergence between

the expected y_i versus the obtained label $f(x_i)$ for some input example x_i.

Inequality 1 was obtained after the definition of generalization in form $|R(f) - R_{\text{emp}}(f)|$, meaning that a classifier f is expected to result in a similar risk or error over unseen examples as it provides over training data, otherwise it overfits the training sample and, thus, it does not produce a general-purpose model for the target problem. Notice that some classifier presenting a good generalization capacity is not necessarily the one producing a low risk but the one in which the empirical risk (training error) is a good estimator for the expected risk.

Using this formulation, Vapnik obtained an upper limit for such probability in terms of Chernoff's bound (Devroye et al., 1996), ensuring learning for any supervised algorithm:

$$P(\sup_{f \in \mathcal{F}} |R(f) - R_{\text{emp}}(f)| > \epsilon) \leq 2\mathcal{N}(\mathcal{F}, n)e^{-n\epsilon^2/4}, \quad (2)$$

taking into account that the worst possible classifier $f \in \mathcal{F}$ contained in the algorithm bias converges to the best, inside the same function space, as the sample size $n \to \infty$, if and only if the algorithm bias $\mathcal{F}$ is characterized by a polynomial Shattering coefficient $\mathcal{N}(\mathcal{F}, n)$. According to this result, such a coefficient (or function) must have a polynomial trend, otherwise learning could never be ensured by the ERMP.

The Shattering coefficient of a supervised algorithm is a growth function that relates the sample size (number of examples in $\mathcal{X}$) to its maximum number of distinct classifications, according to the capacity of $\mathcal{F}$ in creating different space regions in $\mathcal{X}$. Since every distinct classification corresponds to a function, the Shattering coefficient effectively provides us a measurement of the hypotheses space (cardinality) $\mathcal{F}$ in terms of sample sizes. Depending on the rate at which this coefficient grows, one has learning guarantees or not. That coefficient also provides a capacity measure called the Vapnik–Chervonenkis dimension, which informs up until which sample size one has an exponential number of distinct classifications, afterwards it becomes polynomial, however such measure does not provide any additional detail on the growth of such a function. In brief, the smaller the growth is, the stronger are the learning guarantees. Of course that is not enough given without accuracy performances, the simple focus on the Shattering coefficient could lead to underfitting scenarios (de Mello & Ponti, 2018).

This is one of the most important formal steps (if not the most) for the area of ML, because it ensures the learning conditions for supervised algorithms. From that theoretical foundation, Vapnik derived the Generalization Bound (de Mello & Ponti, 2018; Vapnik, 1995), which can be related to other theoretical results such as the Large-Margin Bound, the Tikhonov complexity, the Rademacher complexity, and PAC Learning (von Luxburg & Schölkopf, 2011).

3. Theoretical framework

Let n training examples be organized in some k-dimensional space $\mathbf{X}^k$, a Decision Tree (DT) classifies such a sample by dividing each of those space axes using one or more hyperplanes. Every hyperplane of a DT builds up an orthogonal decision boundary along one of the axes of $\mathbf{X}^k$, so that the Shattering coefficient $\mathbf{N}(\mathcal{F}, n)$ of a DT must be computed in terms of the maximal number of distinct classifications provided by such orthogonal boundaries on $\mathbf{X}^k$.

As a simple example, consider the input space $\mathbf{X}^2 \subseteq \mathbb{R}^2$ having $n = 1, 2, 3$ training examples, as shown in Fig. 1. Observe that for a single hyperplane on a binary classification problem, there are two distinct classification possibilities for $n = 1$, which may be written as 2^1. For $n = 2$, we obtain $2^2 = 4$ and, for $n = 3$, there are six possibilities. This makes evident the Vapnik–Chervonenkis dimension is 2 when a single hyperplane is taken (de Mello & Ponti, 2018; Vapnik, 1998). However, such a capacity measure is not our main goal, but the growth function responsible for measuring the number of different classifications a.k.a. Shattering coefficient, so that we can compare the complexity of different learning models as sample sizes increase.

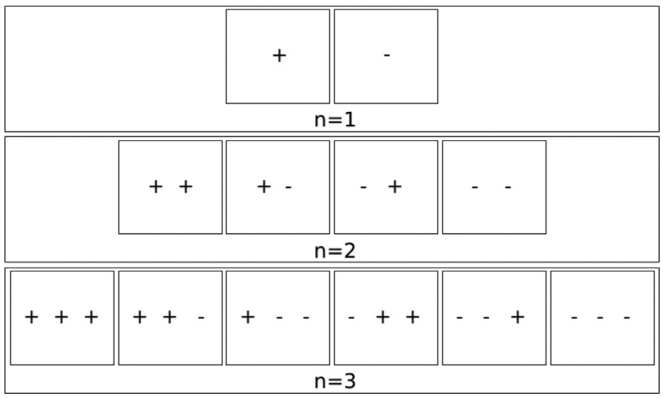


Fig. 1. Binary classification problem having a single hyperplane. Attempt to classify three collinear samples using DTs.

Algorithm 1 Function $F(s, n, h)$ to compute the Shattering coefficient $\mathcal{N}(\mathcal{F}, n)$ of a DT for h hyperplanes dividing along the same dimension of $\mathcal{X}^k$. Parameter s refers to a counter, which is set as zero when performing the first call. The source code is available at https://drive.google.com/file/d/1qFpXPhKpIouHuUENET3eKzWVq0L__VQ/view?usp=sharing. In order to run it, firstly the user must install the Java SDK, then open the command line and run "R CMD javareconf" in order to configure a wrapper in between Java and R. Next, the user must install the packages rJava and RWeka inside the R Statistical Software.

Require: $n \geq 0, s \geq 0, s < n$;
  $s = 0$;
  for $(i = s;\ i < n;\ i++)$ **do**
    $s++$;
    $s = s + F(s + 1, n, h - 1)$;
  end for
  Return s;

The pseudocode in Algorithm 1 represents the solution for h hyperplanes of a DT dividing along a single (same) dimension of $\mathbf{X}^k$, given a sample size equals to n and using parameter s to count the number of operations. Based on this procedure, we formulated the following recurrent equation:

$$F(s, n, h) = \sum_{i=s}^{n-1} F(s + 1, n, h - 1) + 1$$
$$= (n - s) \times F(s + 1, n, h - 1) + (n - s) \times 1$$

to provide the number of distinct classifications obtained for several hyperplanes dividing along a single space axis. We remind the reader this formulation results in the worst-case scenario as typically expected from the Shattering coefficient, as discussed in Vapnik (1998), von Luxburg and Schölkopf (2011) and de Mello and Ponti (2018). Notice that points could be collinear and parallel to the separating hyperplane so that the growth function would decrease, but that is not indeed the Shattering coefficient for general purpose scenarios (see Fig. 2). Thus, we found the following recurrent equation:

$$F(s, n, h) = (n - s) \times F(s + 1, n, h - 1) + (n - s) \times 1$$

to represent the Shattering coefficient for any number of hyperplanes and given any sample size, but still limited to a single space

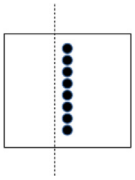


Fig. 2. Set of collinear points (black dots) parallel to the separating hyperplane (dashed line), resulting in a smaller number of distinct classifications.

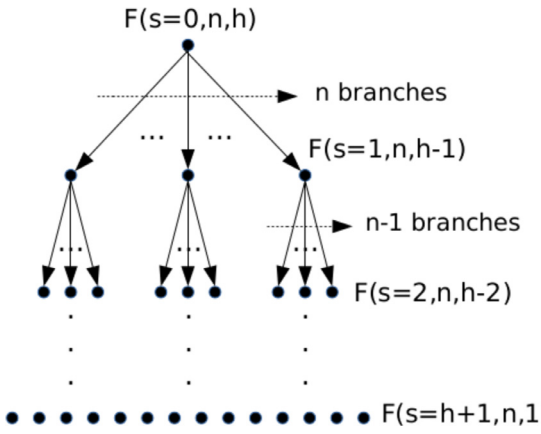


Fig. 3. Solving the recurrence equation with Tree Method (Cormen et al., 2009).

axis. To solve it, we employed discrete math in conjunction with the Tree Method (Cormen, Leiserson, Rivest, & Stein, 2009; Knuth, 1997) (see Fig. 3) to obtain the following:

$$F(s,n,h) = \sum_{i=0}^{h} \frac{n!}{(n-i)!} = e\left(\Gamma(n+1,1) - \frac{\Gamma(n+1)\Gamma(n-h,1)}{\Gamma(n-h)}\right), \quad (3)$$

resulting in the Shattering coefficient, having $\Gamma(.)$ as the gamma function and $\Gamma(.,.)$ as the incomplete gamma function. Observe the tree (based on the Tree Method (Cormen et al., 2009; Knuth, 1997) and not on DTs) from Fig. 3 grows until $h = 1$, meaning the assessment of the last hyperplane, assuming the sample size is greater than the total number of hyperplanes considered, what is a fair condition, otherwise overfitting would be a natural result as discussed in de Mello and Ponti (2018).

From Eq. (3), we found:

$$\begin{aligned} F(s,n,h) = \sum_{i=0}^{h} \frac{n!}{(n-i)!} &= 1 + n + n(n-1) \\ &+ \cdots + n(n-1) \times \ldots \times (n-(h-1)) \\ &\leq cn(n-1)(n-2) \times \ldots \times (n-(h-1)), \end{aligned}$$

so that the equation is upper bounded by the polynomial term at the greatest order multiplied by some constant $c > 0$, allowing us to define the upper bound complexity using the typical time complexity notation as $O(g(n)) = O(n^h)$, given $g(n) = cn^h$ (Cormen et al., 2009).

Based on this partial solution, we formulated the Shattering coefficient for any DT after assuming two axioms: i) any node of a DT builds up h hyperplanes to divide a given space axis into $h+1$ regions. For example, a single hyperplane would produce two regions, thus two hyperplanes would produce at most three regions along such an axis and so on; and ii) every space region defined by a DT node receives a fraction of the training examples. For instance, a DT node could somehow divide one space axis to produce half examples on each side of it.

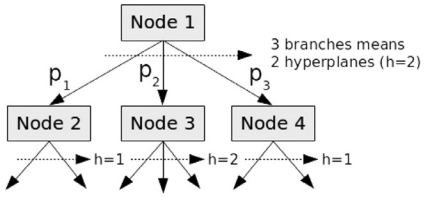


Fig. 4. Computing the Shattering coefficient on a DT example.

In this context, every DT node passes ahead a given fraction of its training examples, reducing the sample size for additional decision boundaries built up along other space axes. Therefore, our numerical computation of the Shattering coefficient is given by a product of functions according to the division built up by hyperplanes from the root node to the DT leaves (see Fig. 4), thus additional nodes and/or increments in the tree height directly impact on the DT model complexity. From this simple example, we compute the Shattering coefficient as follows:

$$\begin{aligned} \mathcal{N}_{DT_1}(\mathcal{F},n) &= F(s=0,n,h=2) \times [1 + F(s=0,p_1 n,h=1) \\ &+ F(s=0,p_2 n,h=2) + F(s=0,p_3 n,h=1)] \\ &= \sum_{i=0}^{2} \frac{n!}{(n-i)!} \times \left(1 + \sum_{i=0}^{1} \frac{(p_1 n)!}{(p_1 n-i)!} + \sum_{i=0}^{2} \frac{(p_2 n)!}{(p_2 n-i)!} \right. \\ &\left. + \sum_{i=0}^{1} \frac{(p_3 n)!}{(p_3 n-i)!}\right), \end{aligned}$$

in which p_1, p_2, p_3 are fractions or probability terms to characterize the number of training examples passed ahead to a next tree node level for further assessment. The number one multiplied by $F(s=0,n,h=2)$ makes sure there is at least such a division on the input data space. The precision of this formulation is directly dependent on the sample size used to induce the decision tree, so the more data is considered, the greater is such a precision as also formalized in the context of the SLT (de Mello & Ponti, 2018; Vapnik, 1998). In addition, observe h defines the number of hyperplanes at a given node, s is always set to be equal to zero (see Algorithm 1). From this formulation, we can compare the complexity of different Decision Trees while modeling the same training examples.

This multiplication-based formulation is motivated by a simple scenario. Consider an $\mathbb{R}^2$ data space whose points are spread out around the traditional orthogonal basis. In that situation, consider that a first and consequently more discriminative hyperplane is used to separate points along the x axis and it is placed on $x = 0$. Thus, we now have two regions of such a space, to mention $x \geq 0$ and $x < 0$ (for the sake of simplicity, we consider all points lying on the hyperplane as belonging to one of the regions). Now let a second and less discriminative hyperplane lie on $y = 0$, so that we have also two regions $y \geq 0$ and $y < 0$. Based on this simple example, we notice that the two classes of the first hyperplane or two-halve spaces can still be divided to form 4 possibilities, a natural result of the multiplication of the half spaces built up with two hyperplanes, this is $2 \times 2 = 4$. Of course that if we had another space dimension, this is, if points were on $\mathbb{R}^3$, and another hyperplane were lying on $z = 0$ whose dot product to any other hyperplane would be equals to zero, we would have $2^3 = 8$ possibilities as expected of this product of regions. In fact, we adopted the same estimation in this paper to formulate the complexity of DT models.

The reader may ask what happens if the same space dimension is divided in several regions using two or more hyperplanes, which would not end up in a direct multiplication of hyperplane regions, i.e., it would not provide a direct power of two, however that was already solved using the recurrence equation representing a set of

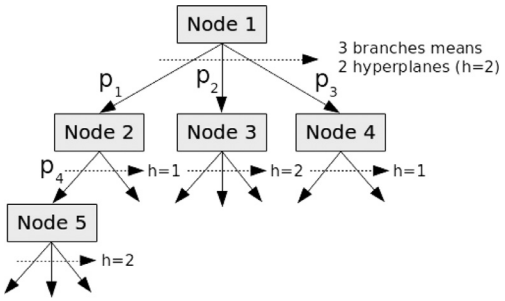


Fig. 5. A second Decision Tree to support an additional analysis.

hyperplanes orthogonally dividing along the same space axis (see Eq. (3)).

For instance, consider the comparison of the DTs from Figs. 4 and 5, knowing the latter has basically an additional node. The Shattering coefficient of this second tree (Fig. 5) is given by:

$$\begin{aligned}\mathcal{N}_{DT_2}(\mathcal{F},n) &= F(s=0,n,h=2)\times[1+F(s=0,p_1n,h=1)\\ &\times[1+F(s=0,p_4n,h=2)]+F(s=0,p_2n,h=2)\\ &+F(s=0,p_3n,h=1)]\\ &=\mathcal{N}_{DT_1}(\mathcal{F},n)+F(s=0,n,h=2)\\ &\times F(s=0,p_1n,h=1)\times F(s=0,p_4n,h=2),\end{aligned}$$

confirming an additional complexity in case probabilities p_1, p_2 and p_3 are maintained along both trees and $p_4 > 0$.

From this perspective, we can now compare DT models and take conclusions about their relative complexities even though our approach only gets more accurate as sample sizes used to induce trees also grow. To complement our study, the reader must remember that such complexity measurement must be combined with a risk quantification, as detailed in Vapnik (1998) and de Mello and Ponti (2018). Following the same principle, we must analyze the effects of the tree complexity in conjunction with its empirical risk, as provided by the Generalization Bound (a direct result from Inequality (2) as discussed in (von Luxburg & Schölkopf, 2011)):

$$R(f) \leq R_{\text{emp}}(f) + \sqrt{\frac{4}{n}(\log 2\mathcal{N}(\mathcal{F},n) - \log\delta)}, \tag{4}$$

taking into consideration the worst possible classifier $f \in \mathcal{F}$ (algorithm bias) as the sample size $n \to \infty$, some polynomial Shattering coefficient $\mathcal{N}(\mathcal{F},n)$, and finally some probability δ for which the following holds:

$$P(\sup_{f\in\mathcal{F}} |R(f) - R_{\text{emp}}(f)| > \epsilon) \leq \delta. \tag{5}$$

Inequality 4 is analyzed in depth by Vapnik (1998), von Luxburg and Schölkopf (2011) and de Mello and Ponti (2018), confirming some supervised model must be assessed in terms of its empirical risk $R_{\text{emp}}(f)$ in conjunction with $\sqrt{\frac{4}{n}(\log 2\mathcal{N}(\mathcal{F},n) - \log\delta)}$, so that we can ensure an upper bound for the expected risk $R(f)$, i.e., for unseen examples experienced in real-world scenarios.

In order to illustrate, consider the two DT models previously analyzed provided the accuracies and the empirical risks (this is the same as one minus the accuracy) listed in Table 1. The same table shows the Shattering coefficient for those trees after setting up $p_1 = p_2 = p_3 = \frac{1}{3}$ and $p_4 = \frac{1}{6}$ (this last value is only used on the second tree). Then, consider δ is equal to 0.1, meaning Inequality (5) will be studied to ensure a probability of 0.9 (in 90% of cases) that the empirical risk is a good estimate for the expected risk, given some divergence factor ϵ. So, let ϵ be equal to 0.05 which is the same as saying that any divergence greater than 5% between those risks will be considered as a fault.

Table 1
Using accuracies and Shattering coefficients to assess two Decision Tree models: n is the sample size (empirical risks are computed as one minus accuracy).

DT	Acc	$R_{\text{emp}}(f)$	$\mathcal{N}(\mathcal{F},n)$
4-node DT	0.95	0.05	$\frac{1}{9}n^4+\frac{2}{3}n^3+\frac{37}{9}n^2+\frac{2}{3}n+4$
5-node DT	0.955	0.045	$\frac{1}{108}n^5+\frac{5}{36}n^4+\frac{109}{108}n^3+\frac{185}{36}n^2+n+5$

If one simply analyzes accuracies from Table 1, the clear conclusion is that the 5-node DT is better (see Fig. 5). However, if we analyze the Generalization Bound (Inequality (4)) after the SLT as illustrated in Fig. 6, we conclude the 5-node DT is more complex and requires many more training examples to provide the same theoretical guarantee as the 4-node DT. This happens because the 5-node DT needs a greater sample size to amortize the complexity of its Shattering coefficient. Fig. 7 shows the growth of the Shattering coefficients for both DTs. The function growing faster is associated to the more complex model, this is the 5-node DT. From this simple introductory analysis, we conclude that: (i) by having

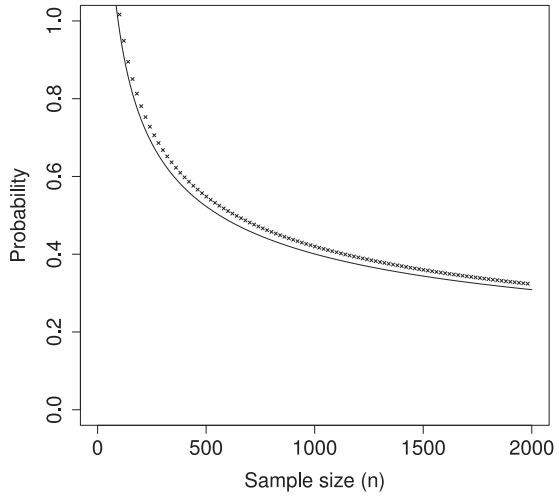


Fig. 6. Assessing the Generalization Bound (Inequality (4)) using information provided in Table 1. Solid line corresponds to the 4-node DT. Crossed points are related with the 5-node DT.

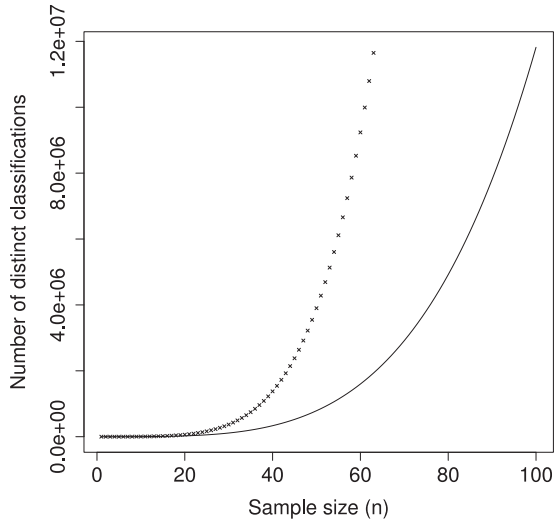


Fig. 7. Illustrating the Shattering coefficients as the sample size increases. Solid line corresponds to the 4-node DT. Crossed points are related with the 5-node DT.

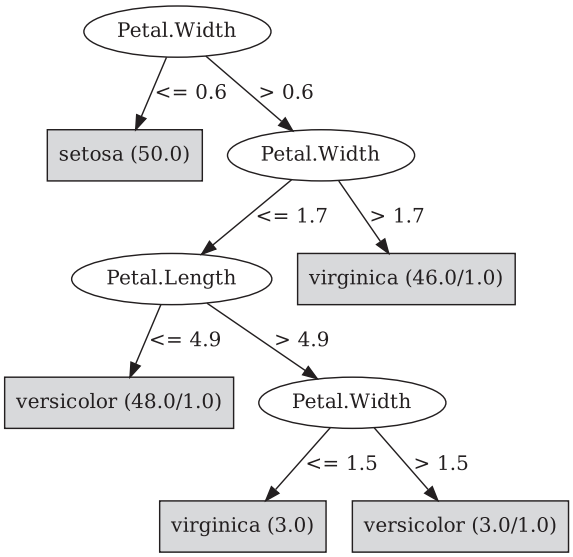


Fig. 8. J48 pruned version: DT built from the Iris dataset.

the Shattering coefficient, one can compare the sizes of algorithm biases, i.e. what are their relative complexities; (ii) one can take advantage of the Generalization Bound to understand which DT model converges faster, so that its learning guarantee would be tighter (greater); and (iii) assess if according to the current sample size, which DT model is the best to tackle a particular learning task.

Next section introduces some experimental results to make evident the relevance of computing the Shattering coefficients while assessing DT models.

4. Experimental results

In this section, we applied the J48 algorithm in its pruned and unpruned versions available with Weka (Hall et al., 2009) on the Iris and the Abalone datasets (Dheeru & Karra Taniskidou, 2017).[1] We start with the Iris dataset which was at first modeled using all examples, providing the trees illustrated in Figs. 8 (pruned) and 9 (unpruned), from which we computed the Shattering coefficient for the pruned tree:

$$\mathcal{N}(\mathcal{F}_{\text{pruned}}, n) = G\left(\frac{150}{150}n\right) \times \left[1 + G\left(\frac{100}{150}n\right) \times \left(1 + G\left(\frac{54}{100}n\right) \times \left(1 + G\left(\frac{6}{54}n\right)\right)\right)\right],$$

and for its unpruned version:

$$\mathcal{N}(\mathcal{F}_{\text{unpruned}}, n) = G\left(\frac{150}{150}n\right) \times \left[1 + G\left(\frac{100}{150}n\right) \times \left(G\left(\frac{54}{100}n\right) \times \left(G\left(\frac{48}{54}n\right) \times \left(1 + G\left(\frac{3}{48}n\right)\right) + G\left(\frac{6}{54}\right) \times \left(1 + G\left(\frac{3}{6}n\right)\right)\right) + G\left(\frac{46}{100}n\right) \times \left(1 + G\left(\frac{3}{46}n\right)\right)\right)\right],$$

having $G(n) = F(s = 0, n, h = 1)$, $F(s = 0, n, h = 1) = \sum_{i=0}^{h=1} \frac{n!}{(n-i)!} = n + 1$, and the probabilities associated with the sample sizes passed along the tree nodes. From that, we conclude the Shattering coefficients are:

$$\mathcal{N}(\mathcal{F}_{\text{pruned}}, n) = \frac{1}{25}n^4 + \frac{1207}{1350}n^3 + \frac{5461}{1350}n^2 + \frac{1618}{225}n + 4$$

$$\mathcal{N}(\mathcal{F}_{\text{unpruned}}, n) = \frac{1}{25}n^5 + \frac{12059}{10800}n^4 + \frac{3182789}{496800}n^3 + \frac{2490247}{165600}n^2 + \frac{153711}{9200}n + 7,$$

confirming the bias for the unpruned tree is greater and therefore more complex than for the pruned tree. This first result confirms other studies that have already concluded about the additional complexity of unpruned trees, but now having the numerical representation as the main contribution of this paper.

According to the Statistical Learning Theory (SLT), such a greater bias should impact the Generalization Bound (Inequality (4)) bringing greater variance to the expected risk $R(f)$. In such a way, we proceeded with an additional experiment[2] to measure the accuracies and their variances, as a way of analyzing the right-side term of Inequality (4).

Table 2
Iris: Average of accuracies and their variances after a leave-one-out procedure.

DT model	Accuracy	Variance
J48 pruned	0.953	0.0447
J48 unpruned	0.940	0.0567

Table 3
Abalone: Distribution of examples according to class labels ("# exmps" corresponds to the number of examples).

Class	# exmps	Class	# exmps	Class	# exmps
1	1	11	487	21	14
2	1	12	267	22	6
3	15	13	203	23	9
4	57	14	126	24	2
5	115	15	103	25	1
6	259	16	67	26	1
7	391	17	58	27	2
8	568	18	42	28	0
9	689	19	32	29	1
10	634	20	26	**Total**	4,177

As the results listed in Table 2 confirm, the unpruned decision tree provides a lower accuracy and a greater variance among all experiments performed using a leave-one-out procedure to hold out every example of the training set, thus totaling 150 experiments (the Iris dataset has 150 examples). This indicates the unpruned model suffers from an additional complexity resultant of a greater space of admissible functions (algorithm bias), which leads to a smaller accuracy, and therefore a greater probability of data overfitting, and a larger variance due to the increase in the cardinality of the algorithm bias (i.e. more unnecessary classification functions are available to build up such model).

After this first example, we performed the same sort of analysis considering the Abalone dataset, which is composed of 4,177 examples, 8 attributes, and 29 unbalanced class labels (see Table 3). The main objective of this experiment is the analysis of more complex DT models built upon an unbalanced scenario, what should impact accuracies and their relative variances as confirmed in Table 4. Those results were computed after performing 100 training stages considering 70% of data and the remaining 30% disjoint examples for testing. The leave-one-out strategy used before would

[1] The source codes are available at https://drive.google.com/file/d/1qFp-XPhKpIouHuUENET3eKzWVq0L__VQ/view?usp=sharing.

[2] Source code is available at https://drive.google.com/file/d/1qFp-XPhKpIouHuUENET3eKzWVq0L__VQ/view?usp=sharing.

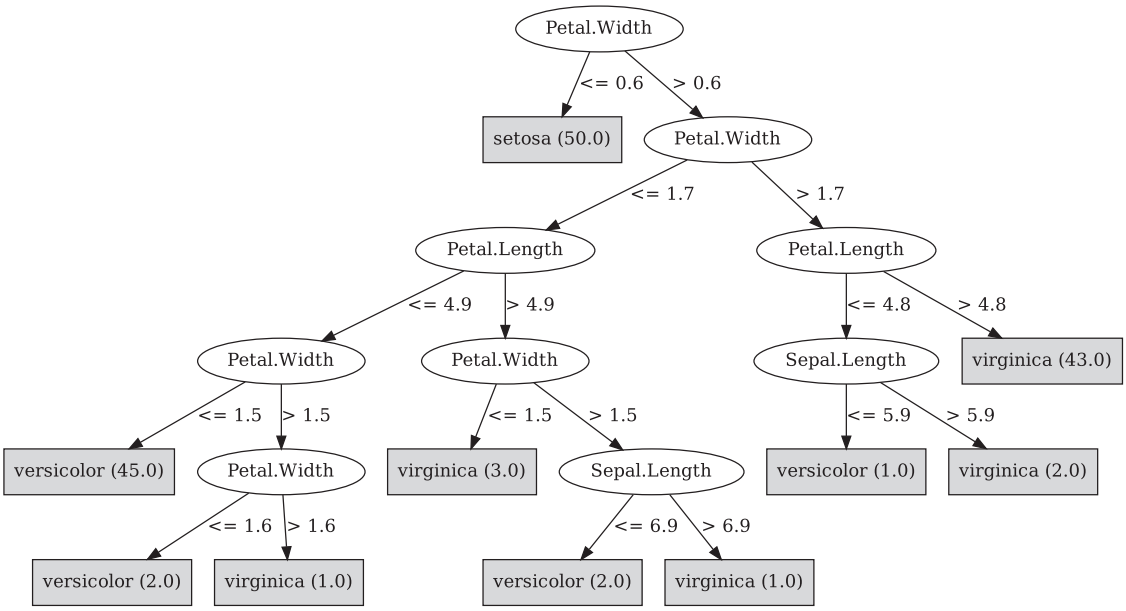


Fig. 9. J48 unpruned version: DT built from the Iris dataset.

Table 4
Abalone: Average accuracies and their variances after performing 100 iterations considering 70% of data for training and the remaining 30% of disjoint examples for testing (resampling).

DT model	Accuracy	Variance
J48 pruned	0.208	0.00013
J48 unpruned	0.194	0.00016

require too many computational resources to tackle this task, so we decided to build up several models using different data samples (resampling) and assess the overall accuracy and its relative variance. Similarly as before, the accuracy for the pruned model is superior and its variance is also smaller as expected.

Next we decided to analyze the Shattering coefficient for the pruned and unpruned DTs given all dataset examples, however that proved to be unfeasible once the resulting trees were huge and so difficult to be illustrated in this paper. For the sake of information, the DT obtained on the whole Abalone dataset had 1,179 leaves after pruning, while the most complex model, i.e. without pruning, reached a total of 2,250 leaves, already depicting the difference in terms of their relative complexities. Therefore, instead of analyzing those huge trees, we decided to consider only 1% of the available examples to illustrate the difference in the pruned and unpruned model complexities, as shown in Figs. 10 and 11. First of all, we mention the difference in terms of the number of leaves per tree. While the pruned tree had 13 leaves, the unpruned needed 25 leaves, an average number after building up one hundred models. After computing their Shattering coefficient, we obtained the following:

$$\begin{aligned}\mathcal{N}(\mathcal{F}_{\text{pruned}}, n) &= F(s=0, n, h=1) \times (1 + F(s=0, \frac{11}{41}n, h=1) \\ &\times (1 + F(s=0, \frac{9}{11}n, h=1) \times (1 + F(s=0, \frac{7}{9}n, h=1))) \\ &+ F(s=0, \frac{30}{41}n, h=1) \times (1 + F(s=0, \frac{18}{30}n, h=2)) \\ &\times (1 + F(s=0, \frac{6}{18}n, h=1) + F(s=0, \frac{9}{18}n, h=1)) \\ &+ F(s=0, \frac{12}{30}n, h=1) \times (1 + F(s=0, \frac{9}{12}n, h=1) \\ &\times (1 + F(s=0, \frac{4}{9}n, h=1)))))\end{aligned}$$

$$\begin{aligned}\mathcal{N}(\mathcal{F}_{\text{unpruned}}, n) &= F(s=0, n, h=1) \times (1 + F(s=0, \frac{11}{41}n, h=1) \\ &\times (1 + F(s=0, \frac{9}{11}n, h=1) \times (1 + F(s=0, \frac{7}{9}n, h=2))) \\ &+ F(s=0, \frac{30}{41}n, h=1) \times (1 + F(s=0, \frac{18}{30}n, h=2)) \\ &\times (1 + F(s=0, \frac{6}{18}n, h=1) \times (1 + F(s=0, \frac{4}{6}n, h=1) \\ &\times (F(s=0, \frac{2}{4}n, h=1)) + F(s=0, \frac{3}{18}n, h=1) \\ &\times (1 + F(s=0, \frac{2}{3}n, h=1))) + F(s=0, \frac{9}{18}n, h=1) \\ &\times (1 + F(s=0, \frac{2}{9}n, h=1) + F(s=0, \frac{7}{9}n, h=1))) \\ &+ F(s=0, \frac{12}{30}n, h=1) \times (1 + F(s=0, \frac{9}{12}n, h=1) \\ &\times (1 + F(s=0, \frac{4}{9}n, h=1) \times (1 + F(s=0, \frac{3}{4}n, h=1)) \\ &+ F(s=0, \frac{5}{9}n, h=1)) + F(s=0, \frac{3}{12}n, h=1) \\ &\times (1 + F(s=0, \frac{2}{3}n, h=1))))),\end{aligned}$$

and their Shattering coefficients:

$$\begin{aligned}\mathcal{N}(\mathcal{F}_{\text{pruned}}, n) &= \frac{9}{41}n^5 + \frac{397}{246}n^4 + \frac{1235761}{202950}n^3 + \frac{3473501}{202950}n^2 \\ &+ \frac{1031813}{40590}n + 13\mathcal{N}(\mathcal{F}_{\text{unpruned}}, n) = \frac{8}{205}n^7 + \frac{1634}{3075}n^6 \\ &+ \frac{6401}{2050}n^5 + \frac{83278277}{7306200}n^4 + \frac{431011481}{14612400}n^3 \\ &+ \frac{818845747}{14612400}n^2 + \frac{5058299}{81180}n + 27,\end{aligned}$$

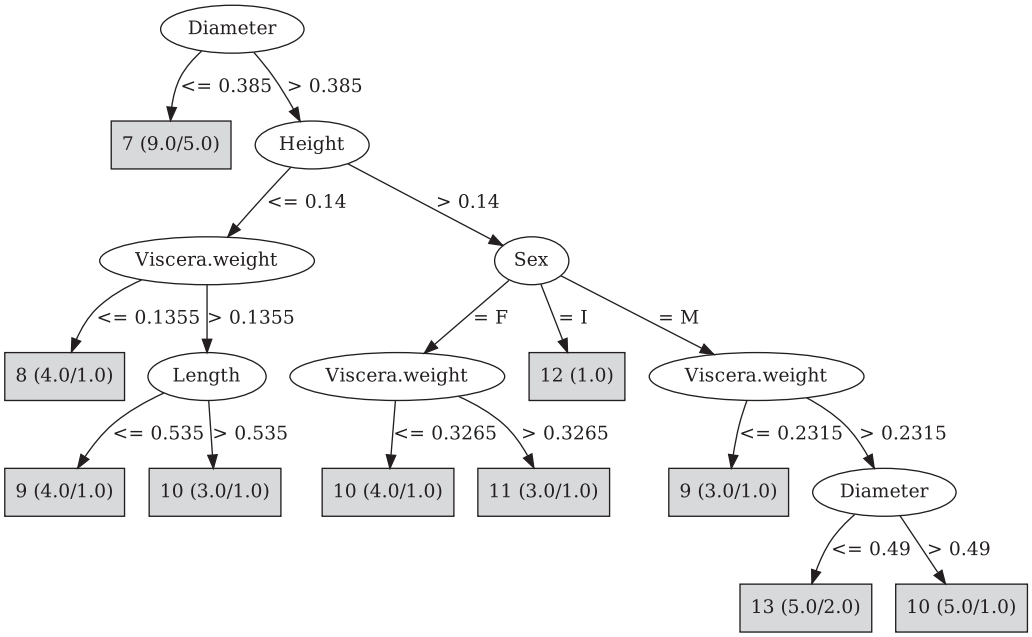


Fig. 10. J48 pruned version: DT built from the Abalone dataset (only for illustration purposes).

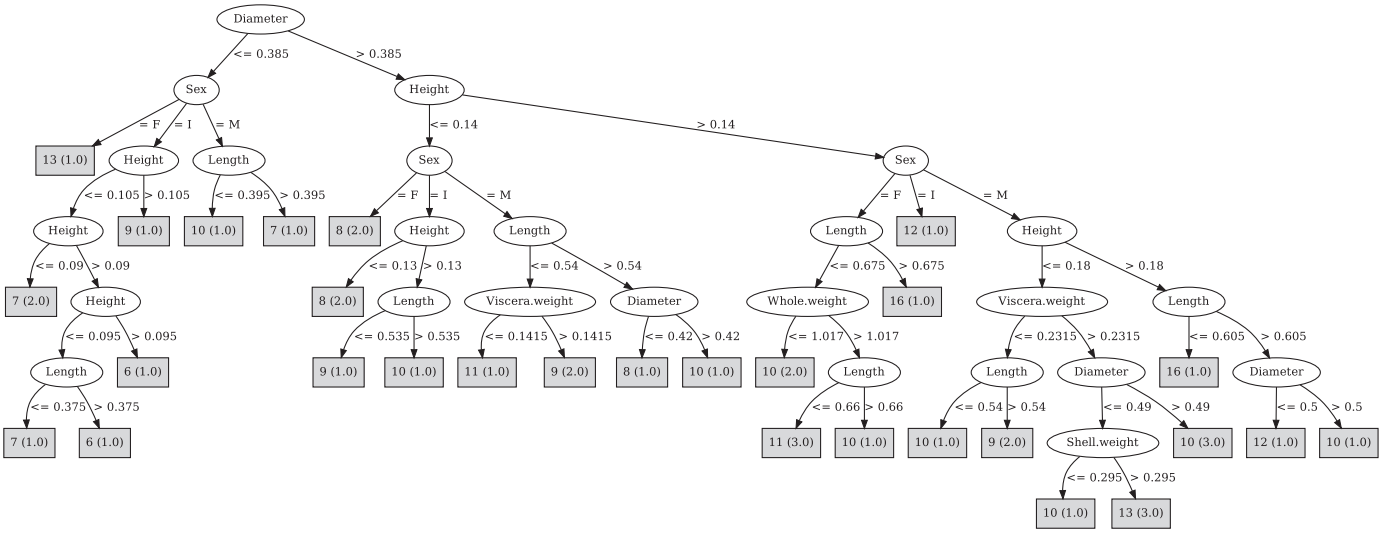


Fig. 11. J48 unpruned version: DT built from the Abalone dataset (only for illustration purposes).

thus their relative difference in terms of algorithm biases is $\frac{8}{205}n^7 + \frac{1634}{3075}n^6 + \frac{5951}{2050}n^5 + \frac{71487377}{7306200}n^4 + \frac{342036689}{14612400}n^3 + \frac{22750147}{584496}n^2 + \frac{272243}{7380}n + 14$, again confirming the additional complexity provided by the unpruned model.

This second experiment illustrates the Shattering coefficients and their relative complexities, allowing us to remind that every time a coefficient is greater than another its relative space of admissible functions (algorithm bias) is more relaxed and thus it contains more functions to model some dataset. By having more functions, the algorithm is capable of expressing more complex classification or regression functions, something necessary when the target problem is more complex (de Mello & Ponti, 2018). If the target learning task is less complex, by having less functions the algorithm is already enough to provide a fair accuracy. However, if accuracy is still low, we should at least try to increase such space of functions, making the bias more complex in attempt to better express the target problem.

5. Complementary discussions

It is still relevant to compare our approach to Breiman et al. (1984), who formulated Pruning in Section 8.5.1 of their book, defining the error-complexity measure $R_\alpha(T)$ of a given tree T as:

$$R_\alpha(T) = R(T) + \alpha|T|,$$

given the number of leaves of such a tree in form $|T|$ and a constant $\alpha > 0$. This result is used by the authors to analyze the convergence of $R_\alpha(T)$, or simply one minus the tree accuracy, as the number of leaf nodes increase, being directly connected to the overall complexity of the tree. Then, they use some criterion to prune the tree, by reducing nodes from leaves to the root up to some point, without excessively increasing $R_\alpha(T)$.

We simply mention that this result is different from ours, in the sense we consider the probabilistic convergence bound provided

by the Statistical Learning Theory (Inequality (2)), which is not associated with the model simplification itself but it formulates how a given classification function f converges to the best as possible function inside some algorithm bias $\mathcal{F}$ as the sample size tends to infinity $n \to 0$. For Breiman et al. (1984), there is no mention on how the sample size growth affects tree model complexities, on the other hand we can indeed say that pruning works as a model adaptation in attempt to find the best classifier inside some space of admissible functions.

It is also worth to mention the work by Ye (1998), who takes advantage of the concept of degrees of freedom to measure model complexities, in attempt to obtain an unbiased estimate of the error variance, and for comparison of different models. The proposed concept of generalized degrees of freedom is based on the sum of the sensitivities to data perturbations, based on expected values and variances. The same research line was still extended by Efron (2004), who also formulated data variances using a similar tool set. Both approaches were formulated from the point of view of Statistics, without taking into account the probabilistic convergence of classification functions according to some algorithm bias. In our point of view, our manuscript is complementary to all those discussed in this additional section once it gives a different perspective on the subject of DT which has classically been tackled from distinct points of view. Finally, our approach allows to connect DT to the general-purpose theoretical framework provided by SLT (de Mello & Ponti, 2018; Vapnik, 1995), which can be used to study any supervised learning algorithm and not only DTs.

In terms of scalability, we recognize that it is still necessary some further studies on how to estimate the Shattering coefficient when considering large datasets. Computations may indeed produce large numbers, however, if a given tree has only small changes, it can be considered stable, even after receiving many more training examples n, then the same Shattering result (or complexity) will be obtained. It is also relevant to mention that the Shattering coefficient is a function of the input size (size of the dataset) so that it cannot be represented by single numbers. That is, in fact, another advantage of our complexity measure, because it should become stable even after training the DT model with more examples. Therefore, the Shattering coefficient will represent the probabilistic convergence of the empirical risk to its expected value as the sample size increases (von Luxburg & Schölkopf, 2011).

An additional aspect on the comparison of DT models in terms of our Shattering formulation is that, in case of producing too complex polynomials, one can decide to compute only the polynomial of the difference between DT models, so that only the complexity divergence of them is represented. This would help researchers interested in relatively ranking models according to their corresponding divergences, without the need of fully measuring the individual coefficients of each tree.

6. Concluding remarks

This paper proposed an approach to numerically compute the Shattering coefficient of DT models, expressing their inherent complexities, i.e. the number of distinct classifications as the sample size increases. Our main contribution is the ability of computing such growth functions and compare the complexity differences of DT models built upon the same training set. In addition, we show how the Generalization Bound (Inequality (4)) can be used to analyze when a bias is enough or it should consider more functions (more complex) while tackling some target learning task.

Experimental results were performed on two commonly considered datasets from the literature. While using the Iris dataset, we simply had the intention to illustrate and compare complexities, the Abalone dataset confirmed the difficulty in manually computing the Shattering coefficients, a natural consequence of addressing a more complex problem. In attempt to improve the practical results of this paper, we will focus our future work to develop an automatic approach to compute the growth functions based on the DT models built up with Weka (Hall et al., 2009), which counts on wrappers for different and quite used languages such as R and Python.

Declaration of Competing Interest

Authors state that there is no conflict of interest to be mentioned.

Credit authorship contribution statement

Rodrigo F. de Mello: Conceptualization, Data curation, Formal analysis, Methodology, Project administration, Software, Validation, Writing - original draft. **Chaitanya Manapragada:** Data curation, Investigation, Project administration, Software, Supervision, Validation, Writing - original draft. **Albert Bifet:** Conceptualization, Investigation, Methodology, Funding acquisition, Supervision, Validation, Writing - review & editing.

Acknowledgments

We acknowledge sponsorships of FAPESP (São Paulo Research Foundation) and CNPq (National Counsel of Technological and Scientific Development), Brazil, under grants 2017/16548-6 and 302077/2017-0. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of FAPESP nor CNPq.

References

Bergstra, J. S., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems: 24* (pp. 2546–2554). Curran Associates, Inc..

Bousquet, O., & Herrmann, D. (2003). On the complexity of learning the kernel matrix. Max-Planck-GesellschaftCambridge, MA, USA. Advances in neural information processing systems 15, 399–406, The MIT Press.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth.

Buhrman, H., & de Wolf, R. (2002). Complexity measures and decision tree complexity: a survey. *Theoretical Computer Science, 288*, 21–43. doi:10.1016/S0304-3975(01)00144-X.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to algorithms* (3rd ed.). The MIT Press.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B, 39*, 1–38.

Devroye, L., Györfi, L., & Lugosi, G. (1996). *A probabilistic theory of pattern recognition*. Springer.

Dheeru, D., & Karra Taniskidou, E. (2017). UCI machine learning repository.

Domingos, P., & Hulten, G. (2000). Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 71–80). ACM.

Efron, B. (2004). The estimation of prediction error: covariance penalties and cross–validation. *Journal of the American Statistical Association*, 99–467.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The weka data mining software: an update. *SIGKDD Explorations Newsletter, 11*, 10–18. doi:10.1145/1656274.1656278.

Kim, H., & Koehler, G. J. (1994). An investigation on the conditions of pruning an induced decision tree. *European Journal of Operational Research, 77*, 82–95.

Knuth, D. E. (1997). The art of computer programming. *Fundamental algorithms*: 1. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc..

Langley, P. (1988). Machine learning as an experimental science. *Machine Learning, 3*, 5–8. doi:10.1023/A:1022623814640.

Laurent, H., & Rivest, R. L. (1976). Constructing optimal binary decision trees is np–complete. *Information Processing Letters, 5*, 15–17.

Lee, D. D., & Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In T. K. Leen, T. G. Dietterich, & V. Tresp (Eds.), *Advances in neural information processing systems: 13* (pp. 556–562). MIT Press.

Lucchese, C., Nardini, F. M., Orlando, S., Perego, R., Tonellotto, N., & Venturini, R. (2017). Quickscorer: efficient traversal of large ensembles of decision trees. In Y. Altun, K. Das, T. Mielikäinen, D. Malerba, J. Stefanowski, J. Read,

… S. Džeroski (Eds.), *Machine learning and knowledge discovery in databases* (pp. 383–387). Cham: Springer International Publishing.

Mashayekhi, M., & Gras, R. (2017). Rule extraction from decision trees ensembles: New algorithms based on heuristic search and sparse group lasso methods. *International Journal of Information Technology & Decision Making, 16*.

Mehta, M., Rissanen, J., & Agrawal, R. (1995). Mdl-based decision tree pruning. In *Proceedings of the first international conference on knowledge discovery and data mining, KDD'95* (pp. 216–221). AAAI Press.

de Mello, R. F., & Ponti, M. A. (2018). *Machine learning - a practical approach on the statistical learning theory*. Springer. doi:10.1007/978-3-319-94989-5.

Oza, N. C., & Russell, S. (2001). Online bagging and boosting. In *In artificial intelligence and statistics: 2001* (pp. 105–112). Morgan Kaufmann.

Pandya, R., Pandya, J., & Infotech, K. P. D. (2015). C5.0 algorithm to improved decision tree with feature selection and reduced error pruning. *International Journal of Computer Applications, 117*.

Quinlan, J. R., & Rivest, R. L. (1989). Inferring decision trees using the minimum description length principle. *Information and Computation, 80*, 227–248.

Raileanu, L. E., & Stoffel, K. (2004). Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence, 41*, 77–93. doi:10.1023/B:AMAI.0000018580.96245.c6.

Rokach, L., & Maimon, O. Z. (2008). *Data mining with decision trees: Theory and applications*: 69. World scientific.

Schaffer, C. (1991). When does overfitting decrease prediction accuracy in induced decision trees and rule sets?. In Y. Kodratoff (Ed.), *Machine learning — EWSL-91* (pp. 192–205). Berlin, Heidelberg: Springer Berlin Heidelberg.

Schaffer, C. (1994). A conservation law for generalization performance. In *Proceedings of the eleventh international conference on international conference on machine learning, ICML'94* (pp. 259–265). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning, 5*, 197–227. doi:10.1023/A:1022648800760.

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM, 27*, 1134–1142. doi:10.1145/1968.1972.

Vapnik, V. N. (1995). *The nature of statistical learning theory*. Berlin, Heidelberg: Springer-Verlag.

Vapnik, V. N. (1998). *Statistical learning theory* (1st ed.). Wiley.

Vapnik, V. N., & Chervonenkis, A. Y. (1971). On uniform convergence of the frequencies of events to their probabilities. *Teoriya Veroyatnostei i ee Primeneniya, 16*, 264–279.

von Luxburg, U., & Schölkopf, B. (2011). Statistical learning theory: Models, concepts, and results. In D. M. Gabbay, S. Hartmann, & J. Woods (Eds.), *Handbook of the history of logic, vol. 10: Inductive logic: vol. 10* (pp. 651–706). Amsterdam, Netherlands: Elsevier North Holland.

Wallace, C., & Patrick, J. (1993). Coding decision trees. *Machine Learning, 11*, 7–22. doi:10.1023/A:1022646101185.

Ye, J. (1998). On measuring and correcting the effects of data mining and model selection. *Journal of the American Statistical Association, 93*, 120–131.

Yıldız, O. T. (2015). Vc-dimension of univariate decision trees. *IEEE Transactions on Neural Networks and Learning Systems, 26*, 378–387. doi:10.1109/TNNLS.2014.2385837.

Bibliography

[1] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012. ISBN: 1600490069, 9781600490064.

[2] Rakesh Agrawal et al. "An Interval Classifier for Database Mining Applications." In: Jan. 1992, pp. 560–573.

[3] P. Almeida et al. "Handling Concept Drifts Using Dynamic Selection of Classifiers". In: *2016 IEEE 28th International Conference on Tools with Artificial Intelligence (ICTAI)*. Nov. 2016, pp. 989–995. DOI: 10.1109/ICTAI.2016.0153.

[4] Manuel Baena-Garcia et al. "Early drift detection method". In: (2006).

[5] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. "Searching for exotic particles in high-energy physics with deep learning". In: *Nature communications* 5 (2014), p. 4308.

[6] Pierre Baldi et al. "Parameterized neural networks for high-energy physics". In: *The European Physical Journal C* 76.5 (2016), p. 235.

[7] Jean Paul Barddal et al. "On Dynamic Feature Weighting for Feature Drifting Data Streams". In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part II*. Ed. by Paolo Frasconi et al. Cham: Springer International Publishing, 2016, pp. 129–144. ISBN: 978-3-319-46227-1. DOI: 10.1007/978-3-319-46227-1_9. URL: http://dx.doi.org/10.1007/978-3-319-46227-1_9.

[8] R.S.M. de Barros, S.G.T. de Carvalho Santos, and P. M. G. Junior. "A Boosting-like Online Learning Ensemble". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. July 2016, pp. 1871–1878. DOI: 10.1109/IJCNN.2016.7727427.

[9] Rajen Bhatt and Abhinav Dhall. *Skin Segmentation Dataset: UCI Machine Learning Repository*. 2012. URL: https://archive.ics.uci.edu/ml/datasets/skin+segmentation.

[10] Albert Bifet and Ricard Gavalda. "Learning from time-changing data with adaptive windowing". In: *Proceedings of the 2007 SIAM International Conference on Data Mining*. SIAM. 2007, pp. 443–448.

[11] Albert Bifet and Ricard Gavaldà. "Adaptive learning from evolving data streams". In: *International Symposium on Intelligent Data Analysis*. Springer. 2009, pp. 249–260.

[12] Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. "Leveraging bagging for evolving data streams". In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2010, pp. 135–150.

[13] Albert Bifet and Elena Ikonomovska. *Airlines Dataset*. URL: https://www.openml.org/d/1169.

[14] Albert Bifet et al. *CovPokElec Dataset from "New Ensemble Methods for Evolving Data Streams", KDD '09*. URL: https://www.openml.org/d/149.

[15] Albert Bifet et al. "Efficient Online Evaluation of Big Data Stream Classifiers". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '15. Sydney, NSW, Australia: ACM, 2015, pp. 59–68. ISBN: 978-1-4503-3664-2. DOI: 10.1145/2783258.2783372. URL: http://doi.acm.org/10.1145/2783258.2783372.

[16] Albert Bifet et al. "Moa: Massive online analysis". In: *Journal of Machine Learning Research* 11.May (2010), pp. 1601–1604.

[17] Albert Bifet et al. "New ensemble methods for evolving data streams". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, pp. 139–148.

[18] Albert Bifet et al. "Pitfalls in Benchmarking Data Stream Classification and How to Avoid Them". In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2013, Prague, Czech Republic, September 23-27, 2013, Proceedings, Part I*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 465–479. ISBN: 978-3-642-40988-2. DOI: 10.1007/978-3-642-40988-2_30. URL: http://dx.doi.org/10.1007/978-3-642-40988-2_30.

[19] Jock Blackard and Denis Dean. "Comparative Accuracies of Artificial Neural Networks and Discriminant Analysis in Predicting Forest Cover Types from Cartographic Variables". In: 24 (Dec. 1999), pp. 131–151.

[20] Zoran Bosnic et al. "Enhancing data stream predictions with reliability estimators and explanation". In: *Engineering Applications of Artificial Intelligence* 34 (Sept. 2014), pp. 178–192. DOI: 10.1016/j.engappai.2014.06.001.

[21] L. Breiman et al. *Classification and regression trees*. Chapman and Hall, New York, 1984.

[22] Leo Breiman. "Bagging predictors". In: *Machine learning* 24.2 (1996), pp. 123–140.

[23] Leo Breiman. "Statistical modeling: The two cultures (with comments and a rejoinder by the author)". In: *Statistical science* 16.3 (2001), pp. 199–231.

[24] Dariusz Brzezinski and Jerzy Stefanowski. "Classifiers for Concept-drifting Data Streams: Evaluating Things That Really Matter". In: *Real-World Challenges for Data Stream Mining* (2013), p. 10.

[25] Dariusz Brzezinski and Jerzy Stefanowski. "Prequential AUC for Classifier Evaluation and Drift Detection in Evolving Data Streams". In: *New Frontiers in Mining Complex Patterns: Third International Workshop, NFMCP 2014, Held in Conjunction with ECML-PKDD 2014, Nancy, France, September 19, 2014, Revised Selected Papers*. Ed. by Annalisa Appice et al. Cham: Springer International Publishing, 2015, pp. 87–101. ISBN: 978-3-319-17876-9. DOI: 10.1007/978-3-319-17876-9_6. URL: http://dx.doi.org/10.1007/978-3-319-17876-9_6.

[26] Dariusz Brzezinski and Jerzy Stefanowski. "Reacting to different types of concept drift: The accuracy updated ensemble algorithm". In: *IEEE Transactions on Neural Networks and Learning Systems* 25.1 (2014), pp. 81–94.

[27] Peter Bühlmann and Torsten Hothorn. "Boosting algorithms: Regularization, prediction and model fitting". In: *Statistical Science* (2007), pp. 477–505.

[28] Javier Burgués, Juan Manuel Jiménez-Soto, and Santiago Marco. "Estimation of the limit of detection in semiconductor gas sensors through linearized calibration models". In: *Analytica Chimica Acta* 1013 (Feb. 2018). DOI: 10.1016/j.aca.2018.01.062.

[29] Javier Burgués and Santiago Marco. "Multivariate estimation of the limit of detection by orthogonal partial least squares in temperature-modulated MOX sensors". In: *Analytica Chimica Acta* 1019 (2018), pp. 49–64. ISSN: 0003-2670. DOI: https://doi.org/10.1016/j.aca.2018.03.005. URL: http://www.sciencedirect.com/science/article/pii/S0003267018303702.

[30] G. A. Carpenter and S. Grossberg. "The ART of adaptive pattern recognition by a self-organizing neural network". In: *Computer* 21.3 (1988), pp. 77–88. DOI: 10.1109/2.33.

[31] Shang-Tse Chen, Hsuan-Tien Lin, and Chi-Jen Lu. "An online boosting algorithm with theoretical justifications". In: *arXiv preprint arXiv:1206.6422* (2012).

[32] A. P. Dawid. "Present Position and Potential Developments: Some Personal Views: Statistical Theory: The Prequential Approach". In: *Journal of the Royal Statistical Society. Series A (General)* 147.2 (1984), pp. 278–292. ISSN: 00359238. URL: http://www.jstor.org/stable/2981683.

[33] A. Philip Dawid and Ambuj Tewari. *On Learnability under General Stochastic Processes*. 2020. arXiv: 2005.07605 [stat.ML].

[34] Thomas G Dietterich. "Ensemble methods in machine learning". In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

[35] Pedro Domingos. "Occam's Two Razors: The Sharp and the Blunt". In: *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*. KDD'98. New York, NY: AAAI Press, 1998, pp. 37–43. URL: http://dl.acm.org/citation.cfm?id=3000292.3000299.

[36] Pedro Domingos and Geoff Hulten. "Mining high-speed data streams". In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2000, pp. 71–80.

[37] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[38] Yoav Freund and Robert E Schapire. "A decision-theoretic generalization of on-line learning and an application to boosting". In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[39] Yoav Freund and Robert E Schapire. "Experiments with a new boosting algorithm". In: Citeseer. 1996.

[40] Joao Gama, Ricardo Fernandes, and Ricardo Rocha. "Decision trees for mining data streams". In: *Intelligent Data Analysis* 10.1 (2006), pp. 23–45.

[41] João Gama and Pedro Medas. "Learning decision trees from dynamic data streams". In: *Journal of Universal Computer Science* 11 (Jan. 2005), pp. 1353–1366.

[42] João Gama, Ricardo Rocha, and Pedro Medas. "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. Washington, D.C.: ACM, 2003, pp. 523–528. ISBN: 1-58113-737-0. DOI: 10.1145/956750.956813. URL: http://doi.acm.org/10.1145/956750.956813.

[43] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. "Issues in evaluation of stream learning algorithms". In: *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2009, pp. 329–338.

[44] João Gama, Raquel Sebastião, and Pedro Pereira Rodrigues. "On Evaluating Stream Learning Algorithms". In: *Mach. Learn.* 90.3 (Mar. 2013), pp. 317–346. ISSN: 0885-6125. DOI: 10.1007/s10994-012-5320-9. URL: http://dx.doi.org/10.1007/s10994-012-5320-9.

[45] João Gama et al. "A survey on concept drift adaptation". In: *ACM Computing Surveys (CSUR)* 46.4 (2014), p. 44.

[46] João Gama et al. "Learning with drift detection". In: *Brazilian Symposium on Artificial Intelligence*. Springer. 2004, pp. 286–295.

[47] Johannes Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti. "RainForest—a framework for fast decision tree construction of large datasets". In: *Data Mining and Knowledge Discovery* 4.2-3 (2000), pp. 127–162.

[48] Johannes Gehrke et al. "BOAT—Optimistic Decision Tree Construction". In: *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*. SIGMOD '99. Philadelphia, Pennsylvania, USA: ACM, 1999, pp. 169–180. ISBN: 1-58113-084-8. DOI: 10.1145/304182.304197. URL: http://doi.acm.org/10.1145/304182.304197.

[49] *2.7–Sources of random data*. 2018. URL: https://www.gnu.org/software/coreutils/manual/html_node/Random-sources.html.

[50] Heitor M Gomes et al. "Adaptive random forests for evolving data stream classification". In: *Machine Learning* 106.9-10 (2017), pp. 1469–1495.

[51] Heitor Murilo Gomes, Jesse Read, and Albert Bifet. "Streaming Random Patches for Evolving Data Stream Classification". In: *2019 IEEE International Conference on Data Mining, ICDM 2019, Beijing, China, November 8-11, 2019*. Ed. by Jianyong Wang, Kyuseok Shim, and Xindong Wu. IEEE, 2019, pp. 240–249. DOI: 10.1109/ICDM.2019.00034. URL: https://doi.org/10.1109/ICDM.2019.00034.

[52] Heitor Murilo Gomes et al. "A survey on ensemble learning for data stream classification". In: *ACM Computing Surveys (CSUR)* 50.2 (2017), pp. 1–36.

[53] Paulo M Gonçalves et al. "A comparative study on concept drift detectors". In: *Expert Systems with Applications* 41.18 (2014), pp. 8144–8156.

[54] Jonathan Gratch. "Sequential inductive learning". In: *Proceedings of the thirteenth national conference on Artificial intelligence-Volume 1*. AAAI Press. 1996, pp. 779–786.

[55] Stephen Grossberg. "Nonlinear neural networks: Principles, mechanisms, and architectures". In: *Neural networks* 1.1 (1988), pp. 17–61.

[56] Peter D. Grünwald, In Jae Myung, and Mark A. Pitt. *Advances in Minimum Description Length: Theory and Applications (Neural Information Processing)*. The MIT Press, 2005. ISBN: 0262072629.

[57] M Harries, J Gama, and A Bifet. *NSW Electricity dataset*. URL: https://www.openml.org/d/151.

[58] Verena Heidrich-Meisner and Christian Igel. "Hoeffding and Bernstein races for selecting policies in evolutionary direct policy search". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. 2009, pp. 401–408.

[59] Wassily Hoeffding. "Probability inequalities for sums of bounded random variables". In: *Journal of the American statistical association* 58.301 (1963), pp. 13–30.

[60] Stefan Hoeglinger and Russel Pears. "Use of hoeffding trees in concept based data stream mining". In: *Information and Automation for Sustainability, 2007. ICIAFS 2007. Third International Conference on*. IEEE. 2007, pp. 57–62.

[61] T Ryan Hoens, Robi Polikar, and Nitesh V Chawla. "Learning from streaming data with concept drift and imbalance: an overview". In: *Progress in Artificial Intelligence* 1.1 (2012), pp. 89–101.

[62] Ramon Huerta et al. "Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring". In: *Chemometrics and Intelligent Laboratory Systems* 157 (2016), pp. 169–176.

[63] Geoff Hulten, Laurie Spencer, and Pedro Domingos. "Mining time-changing data streams". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 97–106.

[64] Earl Busby Hunt. *Concept learning: An information processing problem*. John Wiley and Sons Inc, 1962.

[65] Earl Busby Hunt, Janet Marin, and Philip James Stone. *Experiments in Induction*. Academic Press, 1966. URL: https://books.google.com.au/books?id=60NDAAAAIAAJ.

[66] Elena Ikonomovska et al. "Speeding-up hoeffding-based regression trees with options". In: *ICML*. 2011.

[67] Ruoming Jin and Gagan Agrawal. "Efficient decision tree construction on streaming data". In: *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2003, pp. 571–576.

[68] Bostjan Kaluza et al. "An Agent-Based Approach to Care in Independent Living". In: Oct. 2010, pp. 177–186. DOI: 10.1007/978-3-642-16917-5_18.

[69] Bartosz Krawczyk et al. "Ensemble learning for data stream analysis: A survey". In: *Information Fusion* 37 (Sept. 2017), pp. 132–156. DOI: 10.1016/j.inffus.2017.02.004.

[70] Georg Krempl et al. "Open Challenges for Data Stream Mining Research". In: *SIGKDD Explor. Newsl.* 16.1 (Sept. 2014), pp. 1–10. ISSN: 1931-0145. DOI: 10.1145/2674026.2674028. URL: http://doi.acm.org/10.1145/2674026.2674028.

[71] Ludmila I Kuncheva. "That elusive diversity in classifier ensembles". In: *Iberian conference on pattern recognition and image analysis*. Springer. 2003, pp. 1126–1138.

[72] Vitaly Kuznetsov and Mehryar Mohri. "Generalization bounds for non-stationary mixing processes". In: *Machine Learning* 106.1 (2017), pp. 93–117.

[73] Vitaly Kuznetsov and Mehryar Mohri. "Time series prediction and online learning". In: *COLT*. 2016.

[74] Jennifer R. Kwapisz, Gary M. Weiss, and Samuel A. Moore. "Activity recognition using cell phone accelerometers". In: *Proceedings of the Fourth International Workshop on Knowledge Discovery from Sensor Data*. 2010, pp. 10–18.

[75] Moshe Lichman. *UCI Machine Learning Repository*. 2013. URL: http://archive.ics.uci.edu/ml.

[76] Jing Liu, Xue Li, and Weicai Zhong. "Ambiguous decision trees for mining concept-drifting data streams". In: *Pattern Recognition Letters* 30.15 (2009), pp. 1347–1355.

[77] Richard Lyman. *Character Font Images Data Set: UCI Machine Learning Repository*. 2016. URL: https://archive.ics.uci.edu/ml/datasets/Character+Font+Images.

[78] Chaitanya Manapragada, Geoffrey I Webb, and Mahsa Salehi. "Extremely fast decision tree". In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM. 2018, pp. 1953–1962.

[79] Chaitanya Manapragada et al. *Emergent and Unspecified Behaviors in Streaming Decision Trees*. 2020. arXiv: 2010.08199 [cs.LG].

[80] Llew Mason et al. "Boosting algorithms as gradient descent". In: *Advances in neural information processing systems*. 2000, pp. 512–518.

[81] Llew Mason et al. "Boosting Algorithms as Gradient Descent in Function Space". In: (1999).

[82] Yair Meidan et al. "N-BaIoT—Network-Based Detection of IoT Botnet Attacks Using Deep Autoencoders". In: *IEEE Pervasive Computing* 17 (July 2018), pp. 12–22. DOI: 10.1109/MPRV.2018.03367731.

[83] Rodrigo F de Mello, Chaitanya Manapragada, and Albert Bifet. "Measuring the Shattering coefficient of Decision Tree models". In: *Expert Systems with Applications* 137 (2019), pp. 443–452.

[84] Martial Mermillod, Aurélia Bugaiska, and Patrick BONIN. "The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects". In: *Frontiers in Psychology* 4 (2013), p. 504. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2013.00504. URL: http://journal.frontiersin.org/article/10.3389/fpsyg.2013.00504.

[85] Nikunj C. Oza. "Online Bagging and Boosting". In: *International Conference on Systems, Man, and Cybernetics, Special Session on Ensemble Methods for Extreme Environments*. Ed. by Mo Jamshidi. New Jersey: Institute for Electrical and Electronics Engineers, Oct. 2005, pp. 2340–2345.

[86] Russel Pears, Sripirakas Sakthithasan, and Yun Sing Koh. "Detecting concept change in dynamic data streams". In: *Machine Learning* 97.3 (2014), pp. 259–293. ISSN: 1573-0565. DOI: 10.1007/s10994-013-5433-9. URL: http://dx.doi.org/10.1007/s10994-013-5433-9.

[87] Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby. "New Options for Hoeffding Trees". In: *AI 2007: Advances in Artificial Intelligence: 20th Australian Joint Conference on Artificial Intelligence, Gold Coast, Australia, December 2-6, 2007. Proceedings*. Ed. by Mehmet A. Orgun and John Thornton. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 90–99. ISBN: 978-3-540-76928-6. DOI: 10.1007/978-3-540-76928-6_11. URL: http://dx.doi.org/10.1007/978-3-540-76928-6_11.

[88] John Ross Quinlan. *C4.5: programs for machine learning*. San Mateo, CA: Morgan Kaufmann, 1992. URL: http://cds.cern.ch/record/2031749.

[89] John Ross Quinlan. "Discovering rules by induction from large collections of examples". In: *Expert systems in the micro electronics age* (1979).

[90] John Ross Quinlan. "Induction of Decision Trees". In: *MACH. LEARN* 1 (1986), pp. 81–106.

[91] John Ross Quinlan. "Learning efficient classification procedures and their application to chess end games". In: *Machine learning*. Springer, 1983, pp. 463–482.

[92] Jesse Read et al. "Streaming Multi-label Classification". In: *Proceedings of the Second Workshop on Applications of Pattern Analysis*. Ed. by Tom Diethe et al. Vol. 17. Proceedings of Machine Learning Research. CIEM, Castro Urdiales, Spain: PMLR, Oct. 2011, pp. 19–25. URL: http://proceedings.mlr.press/v17/read11a.html.

[93] Attila Reiss and Didier Stricker. "Introducing a new benchmarked dataset for activity monitoring". In: *Wearable Computers (ISWC), 2012 16th International Symposium on*. IEEE. 2012, pp. 108–109.

[94] Byron Roe et al. "Boosted Decision Trees as an Alternative to Artificial Neural Networks for Particle Identification". In: *Nuclear Instruments and Methods in Physics Research A* 543 (Sept. 2004). DOI: 10.1016/j.nima.2004.12.018.

[95] Leszek Rutkowski et al. "Decision trees for mining data streams based on the McDiarmid's bound". In: *IEEE Transactions on Knowledge and Data Engineering* 25.6 (2012), pp. 1272–1279.

[96] Sripirakas Sakthithasan, Russel Pears, and Yun Sing Koh. "One Pass Concept Change Detection for Data Streams". In: *Advances in Knowledge Discovery and Data Mining: 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, April 14-17, 2013, Proceedings, Part II*. Ed. by Jian Pei et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 461–472. ISBN: 978-3-642-37456-2. DOI: 10.1007/978-3-642-37456-2_39. URL: http://dx.doi.org/10.1007/978-3-642-37456-2_39.

[97] Sripirakas Sakthithasan, Russel Pears, and Yun Sing Koh. "One pass concept change detection for data streams". In: *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer. 2013, pp. 461–472.

[98] S.L. Salas and Einar Hille. *Calculus: One and Several Variable*. New York: John Wiley and Sons, 1978.

[99] Silas Garrido Teixeira de Carvalho Santos et al. "Speeding Up Recovery from Concept Drifts". In: *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part III*. Ed. by Toon Calders et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 179–194. DOI: 10.1007/978-3-662-44845-8_12.

[100] Cullen Schaffer. "A Conservation Law for Generalization Performance". In: *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*. ICML'94. New Brunswick, NJ, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 259–265. ISBN: 1-55860-335-2. URL: http://dl.acm.org/citation.cfm?id=3091574.3091606.

[101] Robert E Schapire. "The strength of weak learnability". In: *Machine learning* 5.2 (1990), pp. 197–227.

[102] Jeffrey Schlimmer and Douglas Fisher. "A case study of incremental concept induction". In: *AAAI*. Vol. 86. 1986, pp. 496–501.

[103] Jeffrey Schlimmer and Richard Granger. "Incremental learning from noisy data". In: *Machine Learning* 1.3 (1986), pp. 317–354. ISSN: 1573-0565. DOI: 10.1007/BF00116895. URL: http://dx.doi.org/10.1007/BF00116895.

[104] Rocco A Servedio. "Smooth boosting and learning with malicious noise". In: *Journal of Machine Learning Research* 4.Sep (2003), pp. 633–648.

[105] Ammar Shaker and Eyke Hüllermeier. "Recovery analysis for adaptive learning from non-stationary data streams: Experimental design and case study". In: *Neurocomputing* 150 (2015), pp. 250–264.

[106] Richard M. Shiffrin et al. "A Survey of Model Evaluation Approaches With a Tutorial on Hierarchical Bayesian Methods". In: *Cognitive Science* 32.8 (2008), pp. 1248–1284. ISSN: 1551-6709. DOI: 10.1080/03640210802414826. URL: http://dx.doi.org/10.1080/03640210802414826.

[107] SIGKDD. *2015 KDD Test of Time Award Winners*. 2015. URL: https://www.kdd.org/awards/view/2015-kdd-test-of-time (visited on 12/10/2019).

[108] Sakthithasan Sripirakas and Russel Pears. "Mining recurrent concepts in data streams using the discrete fourier transform". In: *International Conference on Data Warehousing and Knowledge Discovery*. Springer. 2014, pp. 439–451.

[109] Allan Stisen et al. "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. Seoul, South Korea: ACM, 2015, pp. 127–140. ISBN: 978-1-4503-3631-4.

[110] W Nick Street and YongSeog Kim. "A streaming ensemble algorithm (SEA) for large-scale classification". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 377–382.

[111] Matthew J Streeter. "Two broad classes of functions for which a no free lunch result does not hold". In: *Genetic and Evolutionary Computation Conference*. Springer. 2003, pp. 1418–1430.

[112] Wallace Ugulino et al. "Wearable Computing: Accelerometers' Data Classification of Body Postures and Movements". In: vol. 7589. Oct. 2012. ISBN: 978-3-642-34458-9. DOI: 10.1007/978-3-642-34459-6_6.

[113] Paul E Utgoff. "Incremental induction of decision trees". In: *Machine learning* 4.2 (1989), pp. 161–186.

[114] Jean-Philippe Uzan. "The fundamental constants and their variation: observational and theoretical status". In: *Reviews of modern physics* 75.2 (2003), p. 403.

[115] Vladimir Vapnik. "The Nature of Statistical Learning Theory". In: vol. 8. Jan. 2000, pp. 1–15. ISBN: 978-1-4419-3160-3. DOI: 10.1007/978-1-4757-3264-1_1.

[116] João Vinagre, Alípio Mário Jorge, and João Gama. "An overview on the exploitation of time in collaborative filtering". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 5.5 (2015), pp. 195–215.

[117] Benjamin Visser and Henry Gouk. *AWS Dataset*. URL: https://www.openml.org/d/41424.

[118] Larry Wasserman. *Lecture Notes 3 — Review: Bounded Random Variables - Hoeffding's bound*. URL: https://www.stat.cmu.edu/~larry/=stat705/Lecture3.pdf.

[119] Geoffrey I Webb. "Further experimental evidence against the utility of Occam's razor". In: *Journal of Artificial Intelligence Research* 4 (1996), pp. 397–417.

[120] Geoffrey I Webb et al. "Characterizing concept drift". In: *Data Mining and Knowledge Discovery* 30.4 (2016), pp. 964–994.

[121] Geoffrey I Webb et al. "Understanding Concept Drift". In: *arXiv preprint arXiv:1704.00362* (2017).

[122] Gerhard Widmer and Miroslav Kubat. "Learning in the presence of concept drift and hidden contexts". In: *Machine learning* 23.1 (1996), pp. 69–101.

[123] David H Wolpert and William G Macready. "Coevolutionary free lunches". In: *IEEE Transactions on Evolutionary Computation* 9.6 (2005), pp. 721–735.

[124] David H Wolpert and William G Macready. "No Free Lunch Theorems for Optimization". In: *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION* 1.1 (1997), p. 67.

[125] David H. Wolpert. "The Lack of A Priori Distinctions Between Learning Algorithms". In: *Neural Computation* 8.7 (1996), pp. 1341–1390. DOI: 10.1162/neco.1996.8.7.1341. eprint: https://doi.org/10.1162/neco.1996.8.7.1341. URL: https://doi.org/10.1162/neco.1996.8.7.1341.

[126] Indrė Žliobaitė et al. "Evaluation methods and decision theory for classification of streaming data with temporal dependence". In: *Machine Learning* 98.3 (2015), pp. 455–482. ISSN: 1573-0565. DOI: 10.1007/s10994-014-5441-4. URL: http://dx.doi.org/10.1007/s10994-014-5441-4.