Extremely Fast Hoeffding Adaptive Tree

Chaitanya Manapragada
Monash University
University of Melbourne
Australia
chaitanya.manapragada@unimelb.edu.au

Mahsa Salehi
Department of Data Science and AI
Monash University
Clayton, Victoria, Australia
mahsa.salehi@monash.edu

Geoffrey I Webb
Department of Data Science and AI
Monash University
Clayton, Victoria, Australia
geoff.webb@monash.edu

2022 IEEE International Conference on Data Mining (ICDM) | 978-1-6654-5099-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDM54844.2022.00042

***Abstract*—Many real-world data streams are non-stationary. Subject to concept drift, the distributions change over time. To retain accuracy in the face of such drift, online decision tree learners must discard parts of the tree that are no longer accurate and replace them by new subtrees that reflect the new distribution. The longstanding state-of-the-art online decision tree learner for non-stationary streams is Hoeffding Adaptive Tree (HAT), which adds a drift detection and response mechanism to the classic Very Fast Decision Tree (VFDT) online decision tree learner. However, for stationary distributions, VFDT has been superseded by Extremely Fast Decision Tree (EFDT), which uses a statistically more efficient learning mechanism than VFDT. This learning mechanism needs to be coupled with a compensatory revision mechanism that can compensate for circumstances where the learning mechanism is too eager.**

The current work develops a strategy to combine the best of both these state-of-the-art approaches, exploiting both the statistically efficient learning mechanism from EFDT and the highly effective drift detection and response mechanism of HAT. To do so requires decoupling of the EFDT splitting and revision mechanisms, as the latter incorrectly triggers the HAT drift detection mechanism. The resulting learner, Extremely Fast Hoeffding Adaptive Tree, responds to drift more rapidly and effectively than either HAT or EFDT, and attains a statistically significant advantage in accuracy even on stationary streams.

***Index Terms*—online learning, concept drift, data mining, decision trees**

Replicability Statement

All of our code is publicly available through Github.

https://github.com/chaitanya-m/stream-ml (*thesis* branch)
https://github.com/chaitanya-m/moa-experiment-scripts (*ensemble* branch)
https://github.com/chaitanya-m/exp_analysis

I. Introduction

Data streams are often subject to concept drift [1]. Such non-stationary streams pose a challenge for online learning systems that must adapt their models as the distributions generating the data evolve [2]. Hoeffding Adaptive Tree (HAT) [3] is the current state of the art algorithm for online decision tree learning under drift. It is based on the Hoeffding Tree, an online decision tree algorithm designed for learning from stationary distributions. The Hoeffding Tree [4] employs a conservative learning mechanism. It only adds a new branch to a tree when the risk is negligible that any alternative split could be better.

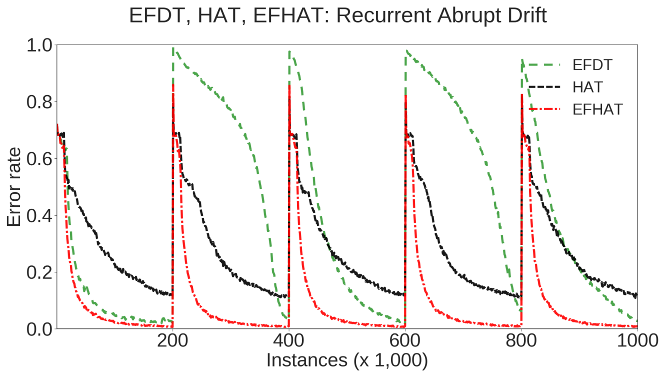

Fig. I.1. Extremely Fast Hoeffding Adaptive Tree (EFHAT) achieves superior recovery from concept drift induced error compared to the state of the art. It does so by coupling Hoeffding Adaptive Tree (HAT)'s ability to detect and immediately respond to drift with Extremely Fast Decision Tree (EFDT)'s superior learning rate for each new distribution. This plot shows prequential error over time for EFHAT relative to the algorithms from which it draws inspiration — EFDT, which has a fast learning rate and slow drift recovery; and HAT, which has fast drift recovery and a slow learning rate.

Extremely Fast Decision Tree (EFDT) [5] introduced a less conservative split mechanism for learning from stationary data streams. Its eager splitting strategy adds a new branch to a tree when the risk is low that doing so is worse than doing nothing. However, this introduces risk that while the new split is useful, it will not be the best split. In consequence, the splitting strategy must be coupled with a revision strategy that allows a split to be replaced by a better split if one becomes apparent. These two mechanisms in concert provide a statistically more efficient learning mechanism that the Hoeffding Tree, and is guaranteed to converge to the tree that would be learned by a batch learner in offline learning.

This paper refines HAT by incorporating the eager splitting strategy of EFDT. This results in Extremely Fast Hoeffding Adaptive Tree (EFHAT), a substantially more statistically efficient online learning algorithm for non-stationary distributions, as illustrated in Figure I.1.

In an online learning toolkit, where no concept drift is present, EFDT would be ideal for learning from distributions expected to be stationary. Where drift is present, EFDT is not as suitable as HAT is. HAT can generally be replaced by EFHAT as the choice of learner in concept drifting scenarios, on account of its ability to both quickly build structure to mitigate the risk of not learning a concept, and its ability to effectively penalise outdated concepts.

The core insights are that HAT with eager splitting requires that the EFDT revision strategy be abandoned when streams are concept drifting because

- the revision strategy interacts poorly with HAT's drift detection, triggering false alarms that result in loss of sub models; and
- in the context of drift it is more effective to learn good models quickly than perfect models slowly.
- EFDT's revision strategy only penalises structure that is unlikely to match the ideal batch tree in the long term (in favor of structure that is likely to be so), while HAT's revision strategy penalises structure that is not current with the evolving concept, in favor of structure that is likely to predict better given the current concept

The statistical efficiency introduced by EFDT is retained, making it the new "base learner" instead of Hoeffding Tree, but the subtree revision strategy native to EFDT is removed due to its harmful interaction with the drift detector. That this decoupling of the EFDT splitting and revision strategies is inobvious is demonstrated by two previous lines of related research having not uncovered it [6], [7].

We show that this new online learning strategy provides a fundamental improvement to HAT. As HAT, in spite of having been published in 2009, remains the state of the art decision tree for concept-drifting streams, this is a clear advance in the state of the art.

Candidate design elements for algorithms in this space are very many; other strategies considered prior to arriving at our particular design included various forms of differentially weighting leaves, introducing decay factors, utilising change detectors, and various pruning strategies. A process of systematic experimentation revealed that in order to improve upon HAT, a fundamental change in tree growth policy was necessary—simple embellishments did not work. While this work does combine two existing works, borrowing the idea of statistically efficient splitting from a recently published work, EFDT, and charting how to join the two completely different tree revision mechanisms from EFDT and HAT through a process of hypothesis formulation and experimentation were both inobvious and nontrivial steps.

The proposed algorithm Extremely Fast Hoeffding Adaptive Tree (EFHAT) carefully melds the best of the existing strategies HAT and EFDT. The primary innovations are to

- recognize the elements of each strategy that are beneficial to learning from non stationary data streams and those that hinder;
- have the vision to recognize that the beneficial elements of each might be productively combined; and
- architect an efficient and effective algorithm to realize that vision.

II. Background

A. Base Learners - Hoeffding Tree and EFDT

The primary challenge in online learning with decision trees is to determine *when* to split at a given node.

In the batch learning case, all examples that have filtered to a node are present at that node. Obviously, in the online learning scenario, it is impossible to decide a split based on collecting an infinite number of examples. Workarounds based around collecting some arbitrarily predetermined number N of examples before deciding to split are cumbersome on account of having to manually tune the parameter for each individual stream.

Hoeffding Tree [4] solves these challenges, in the first instance by storing statistics about the data at each node instead of the data themselves, and then by deciding when to split based on a statistically sound evaluation of a split heuristic; specifically, it employs the Hoeffding Test [8] to delay splitting until it is possible to be sufficiently confident that the top split candidate is better than any alternative.

Delaying splitting until there is confidence that the best split candidate is likely to be fixed in perpetuity slows down the acquisition of tree structure. The acquisition of relevant structure is critical for increasing prediction accuracy. Otherwise the model will underfit [9].

Extremely Fast Decision Tree (EFDT) [5] introduces a less conservative basis for splitting. A leaf will be split as soon as the current best candidate is found to be better than not splitting (in contrast to being better than any alternative split). As this strategy entails a substantial risk of selecting suboptimal splits, EFDT augments it with a revision strategy that allows a split to be replaced if another subsequently proves superior. This increase in short-term plasticity leads to better predictions with a smaller amount of training data, that is, statistical efficiency for prediction is increased.

Most extant online decision tree learners use the Hoeffding Tree split mechanism as a base, thus making Hoeffding Tree the "base learner" for the family of online decision trees. It is of obvious interest to explore the effects of using instead EFDT as a base learner, especially in the concept-drifting context. This promising direction of research has been explored by a two recent studies [6], [7], but both assumed that it is necessary to retain EFDT's revision strategy, failing to recognise that for non-stationary distributions it can be harmful.

B. Hoeffding Test

Constructed from the Hoeffding Inequality [10], the Hoeffding Test is widely used in machine learning theory to establish probably-approximately-correct (PAC) bounds [8], [11].

The Hoeffding Bound as used in Hoeffding Tree [4] is construed as: for n independent random variables $r_1..r_n$ with range R and mean $\bar{r}$ the probability P of the true mean being at least $\bar{r} - \epsilon$ is $1 - \delta$, where:

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

The test requires setting the confidence level parameter δ. Then, the true mean deviates by at least ϵ with a probability $P = 1 - \delta$; ϵ is easily computed using Equation 1. Should an acceptable threshold for ϵ be crossed, one can declare that the true mean is different from the population mean.

C. Hoeffding Tree

Hoeffding Trees represent a fundamental advance in one-pass incremental decision tree induction over preceding attempts [12], [13] by providing deviation guarantees from an ideal batch tree, and by using a particularly effective and statistically sound split determination mechanism based on the Hoeffding Inequality.

As previously mentioned, the central challenge in learning decision trees from potentially infinite streams is the question of how to grow the tree. A decision tree represents a sequence of decision boundaries drawn to divide the input space. Each node may be considered to represent a subspace of the input; a split at a node represents a decision boundary.

In the classification setting, a decision boundary drawn within a subspace would ideally result in two subspaces with greater class purity than the parent subspace. That is, splitting at a node should result in more class homogeneity within the instances that filter to the child nodes. Splits must obviously be based on some condition, such as the instance values taken by a particular attribute.

In order to choose a split attribute with higher separation power than others, we need an indication of the resultant class purity following conditioning on various split attributes. For a dataset that is fixed in size, this is straightforward to determine. All one would have to do is use some measure of class purity to determine which attribute has the best class separation power and should be split on. Information gain [14] and Gini [15] are commonly used split heuristics.

But it is infeasible for an infinite stream to use a measure that relies on a heuristic computed from a corresponding infinitely large data set, as we would not like to wait an eternity to decide our first decision boundary. We must compromise and use the information we have available. Can we, in a statistically sound manner, extrapolate into the future the relative advantage of our choice of split attribute?

This is the problem Hoeffding Tree solves with the use of the Hoeffding Test. Assuming a stationary data-generating distribution, the question of whether the best split candidate will remain so in perpetuity with probability $P = 1 - \delta$ is determined using the Hoeffding Test.

Given we are interested in establishing the best split attribute, Hoeffding Tree uses the formulation from the subsection "Background: Hoeffding Test" with random variables ΔG_t, which represents the advantage in split heuristic value the top candidate split attribute X_a has over the next best attribute X_b at each timestep t.

If there is no difference in the separation power of the two top attributes, the true mean of the population of infogain difference random variables would be zero. Should the measured average of all ΔG_t be greater than a threshold ϵ, Hoeffding Tree concludes that the population mean of infogain differences is nonzero, and thus that the attribute X_a has, with high probability, better separation power than X_b, and thus should be split on.

This is quite a strong constraint; attempting to find the likely best attribute over all time will necessitate acquiring a large number of examples to eventuate any split for which there is more than one useful attribute on which to split.

Note that the authors of the Hoeffding Tree [4] theoretical construct named their implementation Very Fast Decision Tree (VFDT), a naming convention that has been consistent across implementations.

D. EFDT: Extremely Fast Decision Tree

EFDT [5] improves on the statistical efficiency of Hoeffding Tree by relaxing the split criterion. Rather than splitting when the best split relative to all other splits is found, EFDT splits a leaf when the best candidate is better than no split. EFDT split decisions are Hoeffding tested; giving EFDT a strong element of stability, while allowing enough plasticity to allow quick growth of model structure, which aids classification accuracy.

As this eager split strategy fails to give assurance that the best possible split is selected, EFDT also employs a split revision strategy whereby splits are continually monitored and replaced if an alternative split passes a Hoeffding test for greater precision than the current split. This revision strategy is designed for use in stationary distributions, and as the experiments will show, EFDT is less effective than HAT in the presence of concept drift.

E. CVFDT: Concept-adapting Very Fast Decision Tree

CVFDT [16] uses Hoeffding Tree as a base, along with a moving window of instances—it deletes statistics due to older instances from Hoeffding Tree. CVFDT represented, in 2001, the state-of-art in learning from concept drifting streams. If a new winning attribute is found at a given node, CVFDT grows an alternate subtree. The existing subtree is retained and still used for classification until the new one demonstrates higher accuracy, at which stage the old subtree is replaced by the new one. This allows the model to adapt to drifting concepts.

CVFDT also uses the Hoeffding Test as the underlying mechanism to establish superiority of a split over any alternative, as does HAT (designed for concept-drifting streams).

F. HAT: Hoeffding Adaptive Tree

As mentioned in the Introduction, Hoeffding Adaptive Tree (HAT) [3] is a highly effective online decision tree learning algorithm with state-of-the-art capacity to adjust to drift. It utilizes the same conservative Hoeffding Test based splitting strategy as the Hoeffding Tree. Splitting in HAT is thus not as statistically efficient as splitting in EFDT. Similarly to CVFDT, HAT relies on replacing outdated subtrees with alternates, but alternates are grown when a change detector detects drift, instead of when a new winning attribute is found (the CVFDT mechanism). HAT's mechanism, being directly dependent on changing error rather than on changing split attributes, is more suited to adapting more rapidly to concept drift.

HAT may also use moving windows of instances, as does CVFDT. However, HAT also has the option of using the ADWIN change detector and error estimator instead of a moving window of instances.

ADWIN maintains a self-adjusting window of bits (which may represent a sequence of prediction errors), and evaluates possible changepoints. When it detects a change in the distribution of its input of error values, it can signal change and adjust the size of the error window [17], [18].

HAT achieves lower prequential error than CVFDT [18]. The flavor of HAT that uses ADWIN as a change detector and error estimator, being parameter-free, is more robust than versions of HAT that use moving windows of instances, which would require a user to carefully tune an appropriate window-size parameter for each scenario (and continually update it).

As far as we are aware, HAT-ADWIN (which hereafter we mean when we refer to HAT) has not been outperformed in a large experimental setting of concept-drifting streams.

HAT is less conservative than CVFDT in growing and replacing subtrees when drift is occurring. With HAT, alternate subtrees may be rooted with the same split as the subtree potentially being replaced; CVFDT, on the other hand, requires an alternate to be rooted with a different split attribute that has achieved the status of top split attribute. While HAT can grow an alternate on the basis of a change detection, CVFDT must wait for an attribute to win the top spot. HAT is thus far more plastic and responsive to concept drift than is CVFDT, as demonstrated in [3].

III. Extremely Fast Hoeffding Adaptive Tree

HAT is a highly plastic, responsive learner. However, it uses the conservative split mechanism inherited from Hoeffding Tree. We hypothesized that coupling the eager split mechanism of EFDT with HAT's drift detection and structure revision mechanisms would lead to a learner that has substantially more statistically efficient responsiveness to concept drift than either EFDT with its original revision mechanism, designed for stationary distributions, or HAT with its conservative structure acquisition strategy.

As we show in the "Experiments" section, this particular combination of strategies forms a concept-adapting learner that outperforms HAT on a testbench comprising 19 real data streams both with and without concept drift. As we eliminate EFDT's revision mechanism, EFHAT's time and space complexity hinge on the HAT mechanism alone, and EFHAT has the same time and space complexity as HAT.

EFHAT is listed in Algorithm III.1, Function III.2 and Function III.3. In order to conserve space, and to highlight the changes it entails, it is listed in the form of an update to the EFDT algorithm [5], with the additions required by EFHAT preceded by $\longrightarrow$ arrows and deletions indicated with ~~strikethroughs~~.

ADWIN0 [3] is an efficient change estimator and detector that is example free. The main parameters to be set are the number of buckets M and the tolerance δ, which is a confidence level for change not having occurred. Buckets contain 1-0 error and grow exponentially in size as history accumulates; when for some cutpoint the set of older windows vary in mean from the set of newer windows, and the confidence that this change has occurred surpasses the set threshold, the older buckets previous to the cutpoint are dropped. ADWIN is an efficient approximation of ADWIN0 also presented in the same work [3]. In practice, there is little need to tune ADWIN's parameters, as intended by the authors; the parameters M and δ are used with their MOA default values.

Algorithm III.1: Extremely Fast Hoeffding Adaptive Tree (EFHAT)

```
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
       $\delta$, the acceptable probability of choosing the
       wrong split attribute at a given node
       G(.), a split evaluation function
Result: $EFHAT^t$, the model at time $t$ constructed
        from having observed sequence $S^t$.
begin
    Let EFHAT be a tree with a single leaf, the $root$
    $\longrightarrow$ Initialise ADWIN estimators $A_{ijk}$ at root
    Let $\mathbf{X_1} = \mathbf{X} \cup X_\emptyset$
    Let $G_1(X_\emptyset)$ be the $G$ obtained by predicting the
      most frequent class in S
    foreach class $y_k$ do
        foreach value $x_{ij}$ of each attribute $X_i \in \mathbf{X}$ do
            Set counter $n_{ijk}(root) = 0$
        end
    end
    foreach example $(\vec{x}, y)$ in S do
        Find the leaf $l$ reached by $(\vec{x}, y)$ using
          $EFHAT$
        foreach node in path $(root...l)$ do
            foreach $x_{ij}$ in $\vec{x}$ such that $X_i \in \mathbf{X}_{node}$ do
                Increment $n_{ijk}(node)$
                $\longrightarrow$ Update estimators $A_{ijk}$
                if $node = l$ then
                    $AttemptToSplit(l)$
                else
                    ~~$ReEvaluateBestSplit(node)$~~ $\longrightarrow$
                    $HATGrow((\vec{x}, y), node,$
                    $EFHAT)$
                end
            end
        end
    end
end
```

IV. Experiments

We evaluate our proposed adaptive online decision tree algorithm EFHAT against the algorithm for stationary distributions

Function III.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 $\overline{G_l}(X_i)$ for each attribute $\mathbf{X}_l - \{\mathbf{X}_\emptyset\}$
          using the counts $n_{ijk}(l)$
        Let $X_a$ be the attribute with the highest $\overline{G_l}$
        Let $X_b = X_\emptyset$
        Compute $\epsilon$ 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
              $X_a$
            for each branch of the split do
                Add a new leaf $l_m$ and let
                  $\mathbf{X}_m = \mathbf{X} - X_a$
                Let $\overline{G_m}(X_\emptyset)$ be the $G$ obtained by
                  predicting the most frequent class at
                  $l_m$
                for each class $y_k$ and each value $x_{ij}$ of
                  each attribute $X_i \in X_m - \{X_\emptyset\}$ do
                    Let $n_{ijk}(l_m) = 0$
                    $\longrightarrow$ Initialize estimators $A_{ijk}$
                end
            end
        end
    end
end
```

Function III.3: HATGrow($\vec{x}$, y , $node$, T_{alt})

```
begin
    if this node has an alternate tree $T_{alt}$ then
        $HATGrow$ (($\vec{x}, y$), $root(T_{alt})$, $T_{alt}$)
    end
    if one of the change detectors has detected change
      then
        Create an alternate subtree if there is none
    end
    if existing alternate more accurate then
        Replace current node with alternate tree
    end
end
```

from which it is derived, EFDT, and against leading adaptive online decision tree learner HAT.

We carefully isolate the key determinants required to test our specific hypothesis; that coupling the eager split mechanism of EFDT with HAT's drift detection and structure revision mechanisms would lead to a learner that has substantially more statistically efficient responsiveness to concept drift than either EFDT with its original revision mechanism, designed for stationary distributions, or HAT with its conservative structure acquisition strategy. We also assess relative performance in the absence of drift. Accordingly, we present 2 tables of experimental results (Tables III, IV), one each with and without concept drift. The tables directly compare the averaged prequential error of EFHAT with each of the strategies on which it builds: EFDT and HAT.

A. Experimental Framework

We chose the largest UCI datasets[1] [20] that had a clear prediction objective, and that do not have missing values, in order to mitigate the confounding factor of how missing values are handled by the algorithms. We did not include datasets smaller than Chess, which has only $\sim 28,000$ instances. We also added CovPokElec [3], [23] and a discretized version (10 buckets) of the AWS pricing dataset [19], which with ~ 27 million examples is the largest dataset we use. These datasets are described in Table I.

While our conclusions may be drawn from the real data alone, experimental results on synthetic concept-drifting streams are provided for completeness. The synthetic streams we use comprise a commonly used concept drift testbench [1], [37]. The recurrent abrupt drift streams comprise data with widely varying dimensional complexity. For instance, the 555 flavor comprises data with 5 nominal attributes, 5 values per attribute, and 5 classes. The different parametrizations of the Hyperplane generator vary the rate at which the hyperplane rotates (i.e. rate at which the concept changes) and the number of attributes with drift. Similarly, The random RBF generator varies the velocity of the centroids (rate of concept change) and the number of attributes with drift.

The synthetic streams along with their MOA parametrizations are described in Table II. This synthetic testbench with its current set of parametrizations has demonstrated excellent discriminative power for comparing strategies that address concept drift [7]. Therefore, while the results in this paper skew heavily towards EFHAT, it should be noted that this is not because of high correlation between the streams.

UCI datasets tend to be ordered, either naturally or as a consequence of the process of collating published data. This implies that there is natural and incidental concept drift in these data. We run our experiments on the datasets in the order they are provided, with this inherent concept drift. In order to establish baseline algorithm behaviour on streams without concept drift, we also remove this inherent drift by randomly shuffling the order of the instances in the same datasets. This ensures that each instance encountered is a random draw from the distribution instantiated by the dataset as a whole. These shuffled results help establish an effective stationary stream baseline.

Results on shuffled streams represent averages over ten runs. In each run the streams are shuffled using a differently seeded random number generator. The seeds are pre-determined for replicability.

[1]We use a testbench of real datasets largely drawn from the UCI Machine Learning repository with natural or compilation-induced concept drift. Many individual UCI datasets are offered as multiple files that correspond to different experimental conditions or different classes. **Concatenating these files results in concept drift** at the concatenation boundaries.

TABLE I
PROPERTIES OF DATASETS

	Dataset	**Instances**	**Attributes (Numeric, Nominal)**	**Classes**
1	aws—price-discretized [19]	27410309	7 (4, 3)	10
2	chess [20]	28056	6(3, 3)	18
3	covtype [20], [21]	581012	54 (10, 44)	7
4	cpe [22], [23]	1455525	72 (22, 50)	10
5	fonts [20], [24]	745000	411 (410, 1)	153
6	hhar [20], [25]	43930257	9 (6, 3)	6
7	kdd [20]	4000000	42 (34, 8)	23
8	localization [20], [26]	164860	8 (4, 4)	11
9	miniboone [20], [27]	130065	50 (50, 0)	2
10	nbaiot [20], [28]	7062606	115 (115, 0)	11
11	nswelec [20], [29]	45312	9 (7, 2)	2
12	pamap2 [20], [30]	3850505	53 (53,0)	25
13	poker [20]	1025010	10 (5, 5)	10
14	pucrio [20], [31]	165632	18 (15, 3)	5
15	sensor—home-activity [20], [32]	919438	11 (11, 0)	3
16	sensor—CO-discretized [20], [33], [34]	4095000	19 (19, 0)	5
17	skin [20], [35]	245057	3 (3, 0)	2
18	tnelec [20]	45781	4 (2, 2)	20
19	wisdm [20], [36]	15630426	44 (43, 1)	6

TABLE II
SYNTHETIC DATA STREAMS

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

We evaluate the approaches in terms of averaged prequential error. Prequential accuracy/error, or predictive sequential accuracy/error, are widely used evaluation metrics for the stream learning scenario [2], [38]. In the prequential setting, the learner is first presented an example without the target value, and the target is made available as soon as the learner has made a prediction and has had its accuracy statistic updated.

We recognize that this experimental setup is a simplified version of real world online learning, where there may be considerable and varying delay before the true labels are revealed, some true labels may never be revealed and updates may be provided in batches. However, the prequential setting captures the essential property of online prediction and update, with true labels only revealed after classification. The prequential setting is standard in the research literature, as it allows us to make controlled comparisons of algorithms without loss of generality. We use it to promote comparability with related research.

For each stream we report prequential error averaged across all time steps. Prequential error is a time series, and while we average it to obtain a summary statistic for comparison, we also present some examples of prequential error series to illustrate the differing behaviors over time that lead to the final aggregate statistics that we report.

Our implementation is very straightforward: we change the MOA [37] Hoeffding Tree implementation so that it uses the eager splitting mechanism (without the split revision mechanism due to EFDT). We then inherit Hoeffding Adaptive Tree with the eager Hoeffding Tree as the base to give us EFHAT.

We report p-values with Holm-Bonferroni adjusted confidence levels. Holm-Bonferroni corrections account for multiple comparisons, mitigating the possibility that positive results are false positives. We report 95% confidence intervals over the proportion of wins out of all runs for which there is not a draw. Hence, the confidence interval $0.6738 - 1$ in Table III indicates 95% confidence that at least 64% of datasets will record wins or draws.

We present our results with a precision of 5 significant figures in order not to claim greater precision than we are confident about. Our experimental results had 6 significant figures; using 6 significant figures would not materially alter our findings — the p-values in Table III would be 0.0022 and 0.0245 respectively instead of 0.0012 and 0.0037, and would still be within the significance level of 0.05. The p-values in Table IV would not change.

All experimentation is performed in the MOA online learning system [37]. All of our code is publicly available.

B. EFDT and EFHAT

Tables III and IV compare the performance of EFDT with HAT and EFHAT on real data streams in their original order, and shuffled versions of real data streams that remove concept drift. Table V compares all three learners on commonly studied concept drifting streams.

EFHAT achieves lower error than EFDT on 15 UCI streams with concept drift and has higher error on 2 (Table III). EFHAT's advantage is statistically significant, with a p-value of 0.0012. This is significant at a 0.05 confidence level (which is also the Holm-Bonferroni confidence level).

On the stationary streams (Table IV) obtained by shuffling the UCI datasets, EFHAT's advantage is eroded, with 12 wins to 7 loses and a p-value of 0.1796, which does not reach significance. This is not unexpected, as EFDT is designed for a stable environment with stationary concepts. However, that EFHAT performs at a similar level in spite of the instability added by the HAT subtree substitution mechanism is an interesting result.

On concept-drifting synthetic streams (Table V), EFHAT has an advantage over EFDT on 20 streams, losing with a small margin on two Hyperplane streams and one drifting RBF stream. EFHAT's faster subtree replacements appear to slightly disadvantage it when drift is very slow. When a subtree is replaced to account for a changing subconcept, useful accumulated knowledge in the discarded subtree that can be used for prediction is discarded, and the subtree is

TABLE III
AVERAGED PREQUENTIAL ERROR WITH DRIFT

Streams	EFDT	HAT	EFHAT
aws—price-discretized	0.1414	0.1457	**0.1396**
chess	0.3115	0.0887	**0.0440**
covertype	0.1543	0.1808	**0.1126**
covpokelec	**0.1939**	0.2646	0.1982
fonts	0.0030	0.0053	**0.0015**
hhar	0.0036	0.0052	**0.0013**
kdd	0.0009	**0.0009**	0.0010
localization	0.0975	0.0590	**0.0469**
miniboone	***0.0001***	***0.0001***	***0.0001***
nbaiot	0.0008	0.0004	**0.0002**
nswelec	0.1928	0.1677	**0.1418**
pamap2	0.0614	**0.0314**	0.0590
poker	**0.2180**	0.3315	0.2547
pucrio	0.0015	***0.0013***	***0.0013***
sensor—home-activity	0.0014	***0.0007***	***0.0007***
sensor—CO-discretized	0.0666	0.0323	**0.0261**
skin	0.0006	***0.0003***	***0.0003***
tnelec	0.0050	0.0038	**0.0034**
wisdm	0.1458	0.1629	**0.0926**

A **bold** error value indicates higher accuracy, and ***bold italics*** indicate a tie.
Unique Wins EFDT=2 vs EFHAT=15.
One-tailed binomial test statistics: p-value: **0.0012**;
Confidence Interval: 0.6738 — 1

Unique Wins HAT=2 vs EFHAT=13
One-tailed binomial test statistics: **p-value: 0.0037**;
Confidence Interval: 0.6366 — 1

TABLE IV
AVERAGED PREQUENTIAL ERROR WITHOUT DRIFT

Streams	EFDT	HAT	EFHAT
aws—price-discretized	**0.1414**	0.1532	0.1496
chess	**0.6071**	0.6734	0.6083
covertype	0.2765	0.2816	**0.2716**
covpokelec	0.3745	0.3386	**0.3031**
fonts	0.0007	***0.0006***	***0.0006***
hhar	**0.0511**	0.0660	0.0804
kdd	**0.0009**	0.0017	0.0019
localization	**0.3386**	0.3567	0.3551
miniboone	0.1188	0.1165	**0.1095**
nbaiot	0.0291	0.3345	**0.0020**
nswelec	0.2406	0.2411	**0.2372**
pamap2	**0.1567**	0.2180	0.1782
poker	0.3091	**0.2864**	0.3070
pucrio	0.0833	0.1277	**0.0502**
sensor—home-activity	0.0955	0.0887	**0.0359**
sensor—CO-discretized	0.2432	0.2250	**0.1736**
skin	0.0136	0.0159	**0.0096**
tnelec	**0.0059**	0.7315	0.7315
wisdm	0.1364	0.1903	**0.1228**

A **bold** error value indicates higher accuracy, and ***bold italics*** indicate a tie.
Unique Wins EFDT=7 vs EFHAT=12.
One-tailed binomial test statistics: **p-value: 0.1796**;
Confidence Interval: 0.4181 — 1

Unique Wins HAT=3 vs EFHAT=14
One-tailed binomial test statistics: **p-value: 0.0064**;
Confidence Interval: 0.6044 — 1

TABLE V
AVERAGED PREQUENTIAL ERROR ON CONCEPT DRIFTING SYNTHETIC STREAMS

Streams	EFDT	HAT	EFHAT
recurrent—led	0.3468	**0.2677**	0.2678
recurrent—randomtree	0.2182	0.0942	**0.0855**
recurrent—sea	0.1496	**0.1117**	0.1118
recurrent—stagger	0.1904	0.0016	**0.0012**
recurrent—waveform	0.1896	0.1767	**0.1748**
hyperplane—1	0.1168	**0.1150**	0.1230
hyperplane—2	0.1406	**0.1243**	0.1316
hyperplane—3	**0.1121**	0.1141	0.1249
hyperplane—4	0.1338	**0.1140**	0.1202
rbf—drift-1	0.1126	0.1181	**0.1103**
rbf—drift-2	0.2623	0.1826	**0.1586**
rbf—drift-3	**0.1431**	0.1534	0.1487
rbf—drift-4	0.4060	0.3270	**0.3182**
recurrent—abrupt—222	0.3538	0.0005	**0.0005**
recurrent—abrupt—322	0.3760	0.0006	**0.0005**
recurrent—abrupt—332	0.3338	0.0013	**0.0009**
recurrent—abrupt—333	0.3658	0.0030	**0.0014**
recurrent—abrupt—334	0.3900	0.0090	**0.0027**
recurrent—abrupt—335	0.3945	0.0126	**0.0053**
recurrent—abrupt—422	0.3357	6e-04	**0.0005**
recurrent—abrupt—444	0.3897	0.0482	**0.0134**
recurrent—abrupt—522	0.3337	6e-04	**0.0005**
recurrent—abrupt—555	0.4087	0.2794	**0.0967**

A **bold** error value indicates higher accuracy, and ***bold italics*** indicate a tie.
Unique Wins EFDT=3 vs EFHAT=20.
One-tailed binomial test statistics: **p-value: 0.0002** ;
Confidence Interval: 0.6964 — 1

Unique Wins HAT=6 vs EFHAT=17.
One-tailed binomial test statistics: **p-value: 0.0173** ;
Confidence Interval: 0.5490 — 1

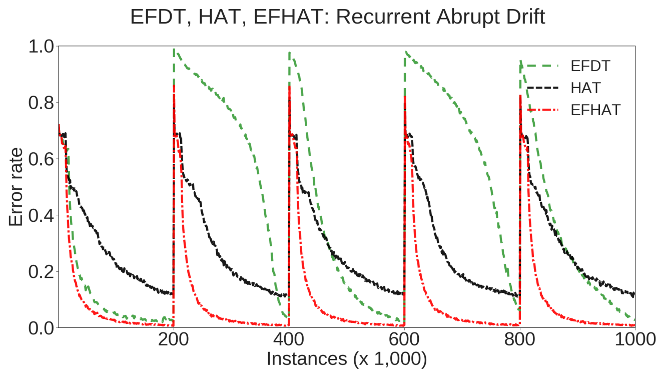

Fig. IV.1. Prequential Error with Abrupt Drift.

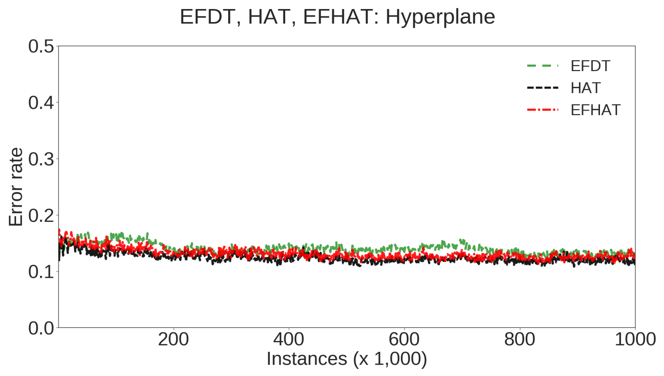

Fig. IV.2. Prequential Error with Gradual Drift - Hyperplane.

rebuilt; delaying this process will lead to better predictions on the previous concept. The p-value, 0.0002, is significant at a 0.05 significance level.

C. *HAT and EFHAT*

EFHAT achieves lower error than HAT on 13 drifting streams and higher on 2 (p-value 0.0037, Table III). It achieves lower error on 14 stationary streams and higher on 3 (p-value 0.0064, Table IV). Both results are significant at a 0.05 confidence level, and also at Holm-Bonferroni adjusted confidence levels of 0.05 and 0.025 respectively.

Many of the differences with drift are substantial, with HAT having more than three times the error of EFHAT on fonts and hhar and more than twice the error on chess.

As for concept-drifting synthetic streams (Table V), EFHAT has an advantage over HAT on 17 streams, losing with a small margin on all Hyperplane streams. Thus, EFHAT's faster subtree replacements resulting in structural loss appear to be a disadvantage with slow drift when compared to HAT as well as EFDT. The p-value, 0.0173, is significant at a 0.05 significance level.

The significant outperformance of EFHAT over HAT on streams without concept drift as well as streams with concept drift indicates that growing tree structure efficiently is critical both in responding to concept drift as well as for online prediction in the stationary setting.

This effect is particularly pronounced in scenarios with recurrent abrupt drift, where growing tree structure quickly is critical for rapid response. We illustrate this point with a prequential accuracy learning curve drawn on a synthetic stream designed to exacerbate responses to recurrent abrupt drift (Figure IV.1). The stream is generated using the method of [1] with parameters `AbruptDriftGenerator -c -o 1.0 -z 5 -n 5 -v 5 -r 2 -b 200000 -d Recurrent`. It comprises 5 nominal attributes taking on 5 values per attribute, with 5 possible classes for each input. An initial concept is generated with a full, random probability table, and a second concept with a specified Hellinger distance of 1 is used as the alternate concept for drift. Identical abrupt drifts are generated every 200,000 instances by switching concepts. The curves represent 10 stream shuffled averages.

Averaged prequential error for EFDT, HAT and EFHAT are **0.4087**, **0.2794** and **0.0967** respectively. Note how EFHAT recovery from abrupt drift is much quicker than that of HAT, which in turn prevails over EFDT. Both HAT and EFHAT display more immediate recovery, demonstrating the effect of the error-based drift detection and recovery strategy. Note, however, that the statistical efficiency of EFDT's eager split strategy prevails in the long run over HAT's initial advantage due to drift detection, reinforcing the value of both strategies, as combined in EFHAT. The upshot of the combination of these strategies is that EFHAT detects and responds to the drift more immediately than EFDT and then learns the new distribution more rapidly than HAT, effectively exploiting the relative strengths of each algorithm.

Similar results are obtained with a wide range of parameterizations of this abrupt drift stream generator.

However, with gradual drift generated by a moving hyperplane [16], there is little difference between EFHAT, HAT or EFDT. We illustrate this with a prequential accuracy learning curve drawn on a synthetic stream with gradual drift (Figure IV.2). Averaged prequential error is **0.1406** for EFDT, **0.1243** for HAT and **0.1316** for EFHAT. This result suggests that while the error-based, drift-responsive subtree substitution mechanism in HAT and EFHAT is particularly effective in the case of abrupt drift, EFDT, in-spite of being designed for stationary streams, is plastic enough to adapt effectively for gradual drift.

The hyperplane stream is generated using the MOA Hyperplane Generator [37] with parameters `HyperplaneGenerator -k 10 -t 0.001 -i 2`. Similar results are obtained with a wide range of parameterizations.

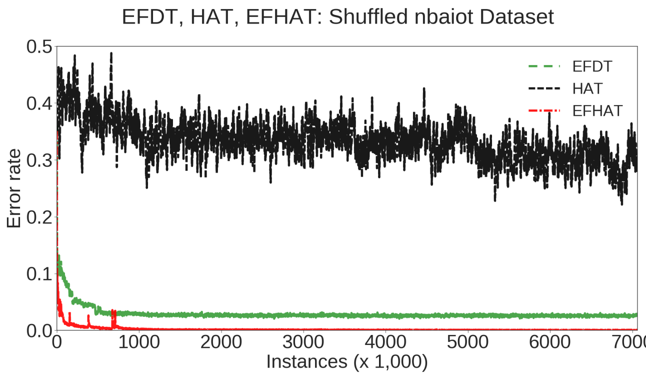

Fig. IV.3. Prequential Error, stationary real stream - NBAIOT.

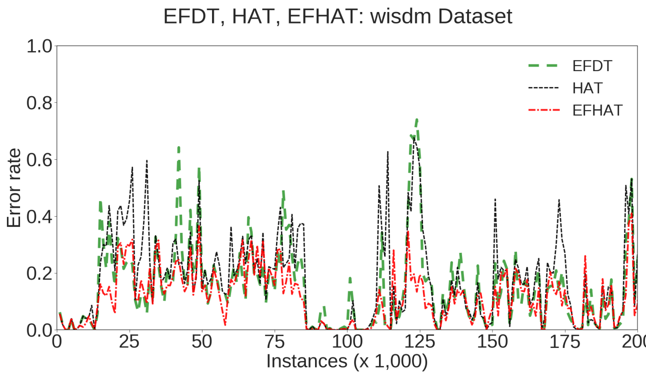

Fig. IV.4. Prequential Error, non-stationary real stream example - WISDM.

Figure IV.3 shows prequential error curves for all three learners on the NBAIOT UCI dataset [28] when it is shuffled to remove drift. This illustrates how the eager splitting of EFDT and EFHAT results in more statistically efficient learning than the conservative strategy of HAT (which it inherits from VFDT). The prequential error over this stationary stream is **EFHAT**: 0.0020; **EFDT**: 0.0291; and **HAT**: 0.3345.

Figure IV.4 shows prequential error curves for all three learners on the unshuffled WISDM human activity classification UCI dataset [36]. This non-stationary data stream clearly illustrates the real world benefit of combining the adaptive response to drift of HAT with the statistical efficiency when learning of EFDT. We see large spikes in error for both EFDT and HAT. For EFDT these result from the revision strategy, which was designed for stationary distributions, and routinely destroys much of the learned tree when the distribution changes. HAT suffers from the same problem, but at different times because the tree is revised when drift is detected. EFHAT does not suffer as much as HAT from these events as it learns the new distribution more rapidly than HAT. Overall, EFHAT clearly outperforms both other learners. We present the first 200,000 instances of around 1.5 million for readability of the chart. The respective prequential errors of the three learners over the entire stream are **EFHAT**: 0.09262; **EFDT**: 0.14576; and **HAT**: 0.16293.

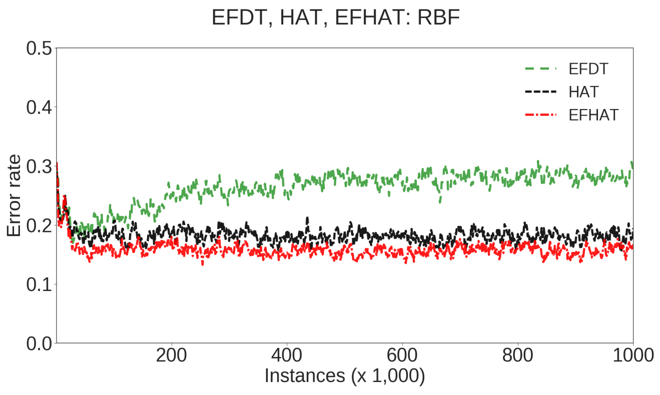

Fig. IV.5. Prequential Error with Gradual Drift - RBF.

With the RBF generator, which also generates gradual drift, averaged prequential error for EFDT, HAT and EFHAT is **0.2623**, **0.18264** and **0.1586** respectively (Figure IV.5). While this re-establishes the general order of performance for our learners, it is still not quite as remarkable as the difference observed with recurrent abrupt drift. The RBF stream was generated using the MOA RBF Generator [23], [37].

V. Conclusions

We have investigated the efficacy of the eager splitting strategy of EFDT in the context of concept drift. This strategy of eager splitting for online decision trees is shown to be highly effective not only for learning from stationary streams (as in the case of EFDT) but also for learning from concept-drifting streams, when combined with an appropriate adaptation strategy (as embodied by EFHAT), achieving lower error than HAT with statistically significant frequency, often by substantial margins and supporting our hypothesis that combining the statistical efficieny of EFDT's splitting mechanism with the drift detection and response mechanism of the leading decision tree algorithm for learning from non-stationary streams, HAT, would result in more accurate learning in a non-stationary environment.

EFHAT is highly plastic and responsive to concept drift. Due to the greater statistical efficiency of the eager splitting strategy, EFHAT can learn new structure much more rapidly than HAT, enabling alternate subtrees to grow much more rapidly after HAT's change detector triggers their growth.

The resulting algorithm is also being capable of being sufficiently stable in order to attain comparable prequential accuracy to EFDT on stationary data streams.

A promising direction for future research is to explore the combination of the effects of eager splitting and active concept drift adaptation in the context of ensemble learning, as ensembling is a highly effective and frequently deployed strategy for decision tree learners [39]–[41].

References

[1] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, "Characterizing concept drift," *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 964–994, 2016.

[2] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Computing Surveys (CSUR)*, vol. 46, no. 4, p. 44, 2014.

[3] A. Bifet and R. Gavaldà, "Adaptive learning from evolving data streams," in *International Symp. Intelligent Data Analysis*, pp. 249–260, Springer, 2009.

[4] P. Domingos and G. Hulten, "Mining high-speed data streams," in *Proc. sixth ACM SIGKDD Int. Conf. knowledge discovery and data mining*, pp. 71–80, ACM, 2000.

[5] C. Manapragada, G. I. Webb, and M. Salehi, "Extremely fast decision tree," in *Proc. 24th ACM SIGKDD Int. Conf. Knowledge Discovery & Data Mining*, pp. 1953–1962, ACM, 2018.

[6] S. G. T. de Carvalho Santos and R. S. M. de Barros, "Online adaboost-based methods for multiclass problems," *Artificial Intelligence Review*, vol. 53, no. 2, pp. 1293–1322, 2020.

[7] C. Manapragada, H. M. Gomes, M. Salehi, A. Bifet, and G. I. Webb, "An eager splitting strategy for online decision trees in ensembles," *Data Mining and Knowledge Discovery*, pp. 1–54, 2022.

[8] V. Vapnik, *The Nature of Statistical Learning Theory*, vol. 8, pp. 1–15. 01 2000.

[9] J. Gama and P. Medas, "Learning decision trees from dynamic data streams," *Journal of Universal Computer Science*, vol. 11, pp. 1353–1366, 01 2005.

[10] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American statistical association*, vol. 58, no. 301, pp. 13–30, 1963.

[11] M. J. Kearns and U. V. Vazirani, *An introduction to computational learning theory*. MIT press, 1994.

[12] J. Schlimmer and D. Fisher, "A case study of incremental concept induction," in *AAAI*, vol. 86, pp. 496–501, 1986.

[13] P. E. Utgoff, "Incremental induction of decision trees," *Machine learning*, vol. 4, no. 2, pp. 161–186, 1989.

[14] J. R. Quinlan, "Induction of decision trees," *MACH. LEARN*, vol. 1, pp. 81–106, 1986.

[15] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and regression trees*. Chapman and Hall, New York, 1984.

[16] G. Hulten, L. Spencer, and P. Domingos, "Mining time-changing data streams," in *Proc. seventh ACM SIGKDD Int. Conf. knowledge discovery and data mining*, pp. 97–106, ACM, 2001.

[17] M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," *SIAM journal on computing*, vol. 31, no. 6, pp. 1794–1813, 2002.

[18] A. Bifet and R. Gavalda, "Learning from time-changing data with adaptive windowing," in *Proc. 2007 SIAM Int. Conf. Data Mining*, pp. 443–448, SIAM, 2007.

[19] B. Visser and H. Gouk, "Aws dataset," 2018.

[20] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[21] J. Blackard and D. Dean, "Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables," vol. 24, pp. 131–151, 12 1999.

[22] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "Covpokelec dataset from "new ensemble methods for evolving data streams", kdd '09," 2009.

[23] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà, "New ensemble methods for evolving data streams," in *Proc. 15th ACM SIGKDD Int. Conf. knowledge discovery and data mining*, pp. 139–148, ACM, 2009.

[24] R. Lyman, "Character font images data set: Uci machine learning repository," 2016.

[25] A. Stisen, H. Blunck, S. Bhattacharya, T. Prentow, M. Kjaergaard, A. Dey, T. Sonne, and M. Jensen, "Smart devices are different: Assessing and mitigating mobile sensing heterogeneities for activity recognition," in *Proc. 13th ACM Conf. Embedded Networked Sensor Systems*, SenSys '15, (New York, NY, USA), pp. 127–140, ACM, 2015.

[26] B. Kaluza, V. Mirchevska, E. Dovgan, M. Lustrek, and M. Gams, "An agent-based approach to care in independent living," pp. 177–186, 10 2010.

[27] B. Roe, H. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor, "Boosted decision trees as an alternative to artificial neural networks for particle identification," *Nuclear Instruments and Methods in Physics Research A*, vol. 543, 09 2004.

[28] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-baiot—network-based detection of iot botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, pp. 12–22, 07 2018.

[29] M. Harries, J. Gama, and A. Bifet, "Nsw electricity dataset," 2009.

[30] A. Reiss and D. Stricker, "Introducing a new benchmarked dataset for activity monitoring," in *Wearable Computers (ISWC), 2012 16th International Symposium on*, pp. 108–109, IEEE, 2012.

[31] W. Ugulino, D. Cardador, K. Vega, E. Velloso, R. Milidiu, and H. Fuks, "Wearable computing: Accelerometers' data classification of body postures and movements," vol. 7589, 10 2012.

[32] R. Huerta, T. Mosqueiro, J. Fonollosa, and N. Rulkov, "Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring," *Chemometrics and Intelligent Laboratory Systems*, vol. 157, pp. 169–176, 2016.

[33] J. Burgués, J. M. Jiménez-Soto, and S. Marco, "Estimation of the limit of detection in semiconductor gas sensors through linearized calibration models," *Analytica Chimica Acta*, vol. 1013, 02 2018.

[34] J. Burgués and S. Marco, "Multivariate estimation of the limit of detection by orthogonal partial least squares in temperature-modulated mox sensors," *Analytica Chimica Acta*, vol. 1019, pp. 49 – 64, 2018.

[35] R. Bhatt and A. Dhall, "Skin segmentation dataset: Uci machine learning repository," 2012.

[36] J. R. Kwapisz, G. M. Weiss, and S. A. Moore, "Activity recognition using cell phone accelerometers," in *Proc. Fourth International Workshop on Knowledge Discovery from Sensor Data*, pp. 10–18, 2010.

[37] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, "Moa: Massive online analysis," *Journal of Machine Learning Research*, vol. 11, no. May, pp. 1601–1604, 2010.

[38] J. Gama, R. Sebastião, and P. P. Rodrigues, "On evaluating stream learning algorithms," *Mach. Learn.*, vol. 90, pp. 317–346, Mar. 2013.

[39] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfahringer, G. Holmes, and T. Abdessalem, "Adaptive random forests for evolving data stream classification," *Machine Learning*, vol. 106, no. 9-10, pp. 1469–1495, 2017.

[40] A. Bifet, G. Holmes, and B. Pfahringer, "Leveraging bagging for evolving data streams," in *Joint European Conf. machine learning and knowledge discovery in databases*, pp. 135–150, Springer, 2010.

[41] X. Dong, Z. Yu, W. Cao, Y. Shi, and Q. Ma, "A survey on ensemble learning," *Frontiers of Computer Science*, vol. 14, no. 2, pp. 241–258, 2020.