Machine Learning (2024) 113:3325–3352
https://doi.org/10.1007/s10994-024-06517-y

Gradient boosted trees for evolving data streams

Nuwan Gunasekara[1] · **Bernhard Pfahringer**[1] · **Heitor Gomes**[2] · **Albert Bifet**[1,3]

Received: 24 October 2023 / Revised: 22 December 2023 / Accepted: 19 January 2024 /
Published online: 22 March 2024

Abstract
Gradient Boosting is a widely-used machine learning technique that has proven highly effective in batch learning. However, its effectiveness in stream learning contexts lags behind bagging-based ensemble methods, which currently dominate the field. One reason for this discrepancy is the challenge of adapting the booster to new concept following a concept drift. Resetting the entire booster can lead to significant performance degradation as it struggles to learn the new concept. Resetting only some parts of the booster can be more effective, but identifying which parts to reset is difficult, given that each boosting step builds on the previous prediction. To overcome these difficulties, we propose Streaming Gradient Boosted Trees (SGBT), which is trained using weighted squared loss elicited in XGBOOST. SGBT exploits trees with a replacement strategy to detect and recover from drifts, thus enabling the ensemble to adapt without sacrificing the predictive performance. Our empirical evaluation of SGBT on a range of streaming datasets with challenging drift scenarios demonstrates that it outperforms current state-of-the-art methods for evolving data streams.

Keywords Gradient boosting · Stream learning · Gradient boosted trees · Concept drift

Editor: João Gama.

Bernhard Pfahringer, Heitor Gomes and Albert Bifet have contributed equally to this work.

✉ Nuwan Gunasekara
ng98@students.waikato.ac.nz

Bernhard Pfahringer
bernhard@waikato.ac.nz

Heitor Gomes
heitor.gomes@vuw.ac.nz

Albert Bifet
abifet@waikato.ac.nz

[1] AI Institute, University of Waikato, Hamilton, New Zealand

[2] Victoria University of Wellington, Wellington, New Zealand

[3] LTCI, Télécom Paris, IP Paris, Palaiseau, France

1 Introduction

Boosting methods have become increasingly successful in machine learning over the past decade. While early weighed boosting algorithms such as AdaBoost (Freund & Schapire, 1997) showed promise, they were later surpassed by gradient boosting methods (Friedman, 2001, 2002; Friedman et al., 2000). Gradient Boosting leverages the previous base learner's gradient information (i.e., the slope of the loss function) to boost the performance of the next learner in an ensemble. The eXtreme Gradient Boosting (XGBoost) (Chen & Guestrin, 2016) takes this approach to another level, achieving high efficiency and superior performance on various time-critical real-world problems. However, in many real-world scenarios, traditional batch learning with the Independent and Identically Distributed (iid) assumption cannot keep pace with the evolving nature of the underlying data stream (Gomes et al., 2017; Bifet et al., 2018).

On the other hand, Stream Learning (SL) accounts for the possibility of change in the underlying data distribution (concept drift) (Bifet et al., 2018). A model should respond efficiently in real-time when learning from an evolving data stream (Bifet et al., 2018). While methods such as Adaptive eXtreme Gradient Boosting (Axgb) (Montiel et al., 2020) and Adaptive Iterations (AdIter) (Wang et al., 2022) were proposed by the research community to enable gradient boosting for evolving data streams, they failed to outperform state-of-the-art ensemble learners like Adaptive Random Forest (Arf) (Gomes et al., 2017) and Streaming Random Patches (Srp) (Gomes et al., 2019).

The proposed work utilizes streaming regression trees with inbuilt drift detectors in a gradient-boosted setting. The paper makes the following contributions:

1. To our knowledge, Sgbt is the first instance where the weighted squared loss elicited in Friedman et al. (2000); Chen and Guestrin (2016) with *hessian* as the *weight* and *gradient* over *hessian* as the *target* considering the previous boosting step's loss, is used to develop a streaming gradient-boosted method for evolving data streams. This allows Sgbt to leverage any streaming regression tree as its base learner.
2. Sgbt utilizes trees with an internal Tree Replacement (TR) mechanism instead of externally monitoring each item in the boosting ensemble for drifts and adjusting each item like Axgb (Montiel et al., 2020) or resetting some parts as in AdIter (Wang et al., 2022). This Tree Replacement mechanism in Sgbt allows the trees in the booster to adapt dynamically to concept drifts. Unlike binary-class gradient-boosted streaming implementations: Axgb and AdIter, Sgbt can solve multi class problems using a committee of trees at each boosting step or a committee of Sgbts.
3. We present an extensive empirical evaluation of Sgbt against current state-of-the-art streaming bagging with random subspaces (Srp), random forest (Arf), boosting (OSB), and gradient boosting (AdIter) methods on 14 datasets with different drift types.

Overall, Sgbt outperforms existing techniques for evolving data streams. The paper is structured as follows. The next section reviews the current state-of-the-art stream learning methods and recent gradient boosting work for evolving data streams. The subsequent section explains our proposed Sgbt method. The experiments section describes the experimental setup where Sgbt was evaluated against state-of-the-art stream learning methods. The final section provides conclusions and directions for future research.

2 Related work

Boosting and bagging are two popular ensemble learning techniques used in machine learning. Bagging randomly samples instances with replacement to train each member of the ensemble. Boosting, on the other hand, attempts to boost the performance of the next base learner in the ensemble, considering the loss of the previous one. It combines the prediction of weak learners addictively to produce a strong learner (Friedman et al., 2000; Friedman, 2001). AdaBoost (Freund & Schapire, 1997) highly weights the misclassified instances by the current base learner to improve the next base learner. Gradient boosting uses the current base learner's gradient information of the loss to improve the next base learner (Friedman et al., 2000). XGBoost (Chen & Guestrin, 2016) uses this gradient information to derive a particular regression tree that predicts a raw score at the leaf for a given instance. It contains an efficient split-finding mechanism, cache-aware data processing, and parallel processing to produce a highly scalable and efficient algorithm for batch learning (Chen & Guestrin, 2016).

Compared to batch learning, Stream Learning model learns from an evolving data stream (non-iid data), processing one instance at a time. Here, the model should predict at any given moment using limited processing and memory (Bifet et al., 2018; Gomes et al., 2017). Also, it should adjust to distribution changes (concept drifts) in the underlying data stream (Bifet et al., 2018; Bifet & Gavalda, 2007; Gomes et al., 2017).

Data stream boosting is challenging due to the evolving nature of the data stream. Here, the model needs to adjust to the new input distribution of the stream after a concept drift (Bifet et al., 2018; Montiel et al., 2020). Online Bagging (OBg) and Online Boosting (OB) (Oza & Russell, 2001) were inspired by the observation that a binomial distribution $Binomial(p, n)$ can be approximated by a Poisson distribution $Poisson(\lambda)$ with $\lambda = np$ as $n \to \infty$. Here, n is the number of instances, and p is the probability of success in the binomial distribution. Since the probability of selecting a given example is $1/n$ in batch bagging, the uniform sampling with replacement of the bagging algorithm is approximated by $Poisson(1)$ in OBg. On the other hand, in OB, λ is computed by tracking the total weights of correctly classified and misclassified examples for each base learner. An online version of SmoothBoost (Servedio, 2003) was proposed in Chen et al. (2012). This Online Smooth Boost (OSB) uses smooth distributions that do not assign too much weight to a single example. When the number of weak learners and examples are sufficiently large, OSB is guaranteed to achieve an arbitrarily small error rate (Chen et al., 2012; Gomes et al., 2019). Recently, two notable approaches were proposed by the stream learning community to leverage gradient boosting for data streams: Axgb (Montiel et al., 2020) and AdIter (Wang et al., 2022). Axgb employs mini-batch trained XGBoost as its base learners and adjusts the ensemble in response to concept drifts, which it detects using ADWIN (Bifet & Gavalda, 2007). AdIter attempts to identify the weak learners in the ensemble and prune them when confronted with concept drift. It then employs multiple training iterations via majority vote among the ensemble to support different drift types. Both Axgb and AdIter only support binary classification. In contrast, our proposed streaming gradient boosting method (Sgbt) supports both binary and multi class problems.

Arf (Gomes et al., 2017) and Srp (Gomes et al., 2019) are popular ensemble learning methods for streaming data. They allow one to use efficient stream learning base learners like Hoeffding Tree (HT) in a random forest or random subspaces setup in conjunction with efficient drift detectors like ADWIN. Arf is a streaming random forest adaptation that combines re-sampling strategies, drift detection, and drift recovery strategies (Gomes

et al., 2017). SRP combines random subspaces and re-sampling (i.e., random patches) to leverage diversity among base incremental learners (Gomes et al., 2019). It uses the same drift detection and recovery strategy as ARF, but tends to outperform ARF (Gomes et al., 2019) in some benchmarks while not being limited to decision trees.

OSB performed better compared to OB (Chen et al., 2012). Empirical evaluation (Gomes et al., 2019) shows that even with 100 base learners, ARF and SRP outperform OSB by a large margin. In the same evaluation, SRP outperformed ARF. AXGB failed to outperform ARF in the Montiel et al. (2020) empirical evaluation. In Wang et al. (2022) experiments, ADITER also failed to surpass ARF on synthetic evolving datasets with 10,000 instances. However, in the same evaluation, ADITER surpassed ARF on real-world data. In that evaluation, all the other datasets had less than 100,000 instances apart from airlines. The above empirical evaluations suggest that the latest gradient boosting methods for evolving data streams are yet to surpass current state-of-the-art ensemble methods like SRP and ARF. However, our proposed SGBT was able to outperform SRP and ARF in a variety of evolving datasets.

3 Streaming gradient boosted trees (SGBT)

For a dataset with n instances, let x_i be the features for the i-th instance and y_i be its relevant target value. In gradient boosting, a model ϕ can be represented as S additive functions:

$$\hat{y}_i = \phi(x_i) = \sum_{s=1}^{S} f_s(x_i), f_s \in \mathcal{F}$$

to predict $\hat{y}_i$Friedman (2002); Chen and Guestrin (2016). Here, $\mathcal{F}$ is the space of regression trees. In XGBOOST (Chen & Guestrin, 2016), each f_s corresponds to an independent tree structure with leaf scores ω. Each regression tree contains a continuous score ω_i at the leaf for the i-th instance. The authors proposed to sum up the corresponding scores at the leaves of each tree for prediction. The learning objective is to minimize the regularized objective:

$$\mathcal{L}(\phi) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{s=1}^{S} \Omega(f_s) \tag{1}$$

where Ω penalizes the complexity of the tree f:

$$\Omega(f) = \gamma T + \frac{1}{2}\beta\|\omega\|$$

Here, γ penalizes adding a new leaf and β forces leaf predictions to be small. T is the number of leaves in the tree. l is a differentiable convex loss function that measures the difference between the prediction $\hat{y}_i$ and the target y_i. Furthermore, the loss at the s-th step is the loss incurred by the previous $(s-1)$ step and the loss incurred by tree f_s plus the regularization term:

$$\mathcal{L}^{(s)} = \sum_{i=1}^{n} l(y_i, \hat{y}^{(s-1)} + f_s(x_i)) + \Omega(f_s)$$

This loss could be approximated using second-order Taylor approximation (Chen & Guestrin, 2016) to:

$$\mathcal{L}^{(s)} \simeq \sum_{i=1}^{n} \left[l(y_i, \hat{y}^{(s-1)}) + g_i f_s(x_i) + \frac{1}{2} h_i f_s^2(x_i) \right] + \Omega(f_s)$$

Here, $g_i = \partial_{\hat{y}^{(s-1)}} l(y_i, \hat{y}^{(s-1)})$ and $h_i = \partial^2_{\hat{y}^{(s-1)}} l(y_i, \hat{y}^{(s-1)})$ are the first and second order (hessian) gradient statistics of the loss considering $s-1$-th prediction. Though the authors (Chen & Guestrin, 2016) use a simplified version of the above loss function by removing constants to derive raw score values at the leaves, the below version was elicited to explain it as a weighted squared loss with weight h_i and target g_i/h_i:

$$\sum_{i=1}^{n} \frac{1}{2} h_i (f_t(x_i) - g_i/h_i)^2 + \Omega(f_s) + constant. \tag{2}$$

This weighted squared loss with *hessian* as the *weight* and *gradient* over *hessian* as the *target*, considering the previous boosting step's loss, was first introduced in Friedman et al. (2000).

Algorithm 1 Training Sgbt

Input: y_i: label as a one-hot vector for x_i, lr: learning rate, S: # boosting steps, $\hat{y}^{(0)}$: initial prediction, l: loss function, m: % of features for training, C: # classes.

if *start of training* **then**
  Initialize M using m. ▷ M={sets of randomly picked m% of features for each boosting step}
end if
for $s := 1$ to S **do**
  Generate instance $x_{i,s}$ using feature set m_s ($\in M$) from instance x_i.
  $g_i \leftarrow \partial_{\hat{y}^{(s-1)}} l(y_i, \hat{y}^{(s-1)})$ ▷ gradient for committee
  $h_i \leftarrow \partial^2_{\hat{y}^{(s-1)}} l(y_i, \hat{y}^{(s-1)})$ ▷ hessian for committee
  for $c := 1$ to $C-1$ **do**
    TRAIN $f_{s,c}(x_{i,s}, g_{i,c}/h_{i,c})$ ▷ train $f_{s,c}$ using instance $x_{i,s}$ and label $g_{i,c}/h_{i,c}$
  end for
  $\hat{y}_i \leftarrow \hat{y}_i + lr * f_s(x_i)$ ▷ scale $f_s(x_{i,s})$ and add to $\hat{y}_i$
end for

Equation 2 provides the flexibility to utilize various streaming regression trees instead of the one employed in XGBoost. Moreover, depending on the implementation, the streaming regression tree's regularization term can diverge from that employed in XGBoost. In this work, the Tree Replacement strategy explained later in this paper, acts as a regularization mechanism.

3.1 Streaming regression trees with internal tree replacement strategy for gradient boosting

In data stream learning, n could be infinite, and learning happens online, where a model ϕ_{i-1} learned at the $i-1^{\text{th}}$ instance is used to predict the i^{th} instance. Also, from any i^{th} instance, the underlying distribution of x could change (concept drift). The model ϕ_i should adjust it's regression trees to adapt to this new distribution at i. Instead of externally monitoring and resetting each f_s tree like in AdIter (Wang et al., 2022), in Sgbt, the trees internally monitor their standardized absolute error and train an alternate tree if it goes above

a warning level. The tree f_s switches to its alternate tree once the error reaches a danger zone. f_s tree employs a drift detector to monitor its standardized absolute error to trigger these warning and danger signals. The rest of the paper identifies this strategy of replacing the active tree with an alternate tree on the drift detection signal as Tree Replacement (TR). In the experiments, we used two regression trees for data streams: FIMT-DD (Ikonomovska et al., 2011) and SGT (Gouk et al., 2019) with in-built drift detectors: Page-Hinckley Test (PHT) (Mouss et al., 2004) and DDM (Gama et al., 2004). The implementation of SGT with DDM is generic, and one could replace SGT with any other regression tree for data streams. Here, TR also serves as a dynamic regularization mechanism by replacing trees as data evolves during learning. ARF and SRP use a similar Tree Replacement strategy under random-forest and bagging settings for SL classification. But to our knowledge, this is the first instance, TR is used in gradient-boosted trees for SL classification. Here, the booster is allowed to dynamically adjust to underlying input distribution changes as some active trees are replaced by their alternate trees on drift detection.

The loss function in Eq. 2 requires the regression trees to support fractional weights, as h_i could be a fractional value for some loss functions. Streaming regression trees (SGT and FIMT-DD) considered in this work only support integer weights. Supporting fractional weights for them is not trivial. For example, SGT and FIMT-DD require the incremental calculation of variance and co-variance for fractional weights. Though recent work by Pébay et al. (2016) and Schubert and Gertz (2018) suggests this is possible, this itself is a separate research topic. Also, later in the text, it is clarified that the hessians for the popular categorical cross-entropy loss with softmax used in the experiments are consistently below 1. Hence, even though SGBT calculates these weights (hessians), it does not pass them to the underlying trees for these practical reasons. Alternatively, it passes a weight of 1 to the trees.

Instead of using all the features to train at each boosting step, SGBT uses a subset of features based on a predefined feature percentage. This approach of using a subset of features to train each ensemble member is also used in SRP (Gomes et al., 2019) to increase the diversity among the base learners. Algorithm 1 explains the training procedure of SGBT.

3.2 Multi class support

Two approaches are used to support multi class problems: SGBT and SGBT^{MC}.

- *SGBT* uses a committee of regression trees in a given boosting step s. Here, a single tree is trained for each class. The committee is composed of a softmax function, so the probability that an instance, x_i, belongs to class c is given by: $\hat{y}_{i,c} = \frac{exp(f_{s,c}(x_i))}{\sum_{c=1}^{C} exp(f_{s,c}(x_i))}$. Here $f_{s,c}$ is the regression tree trained to predict a real-valued score for class c at the s-th boosting step, and C is the number of classes. In practice, hard-wiring $f_{s,C}(x_i) = 0$ allows SGBT to reduce the number of trees being trained.[1] The categorical cross-entropy loss (l^{CE}) is used to train the model: $l^{CE}(y, \hat{y}) = -\sum_{c=1}^{C} y_c log(\hat{y}_c)$. Here, y is the ground truth encoded as a one-hot vector. For l^{CE}, gradient (g) is $y_c - \hat{y}_c$, and hessian (h) is $\hat{y}_c(1 - \hat{y}_c)$. The regression tree committee (composing $C - 1$ items) at the s-th boosting

[1] This practice is used in Gouk et al. (2019) as well.

step represents the base learner for the s-th boosting step. This approach is also used in SGT to support multi class classification.

- *SGBT*$^{\mathbf{MC}}$ uses the same loss function (l^{CE}) as in SGBT. But, it uses a wrapper classifier to invoke a binary SGBT classifier for each class. The task of the binary SGBT classifier is to distinguish a given class from all the other classes. All C classifier votes for the positive outcome are collected and normalized at prediction. The class associated with the classifier that predicted the positive outcome most confidently is considered the final class for the instance. This approach is very popular in batch learning and is commonly known as *one-vs-rest* or *one-vs-all* in literature (Witten et al., 2016). SGBT^{MC} reverts to SGBT for binary class problems to avoid any computing overhead.

Unlike AXGB and ADITER, the above two approaches allow SGBT to support gradient boosting for evolving data streams on multi class problems.

Algorithm 2 TRAINING SGBT$^{SK}_{MI}$

Input: y_i: label as a one-hot vector for x_i, lr: learning rate, S: # boosting steps, $\hat{y}^{(0)}$: initial prediction, l: loss function, m: % of features for training, C: # classes, k: randomly skip $1/k$ instances ($k \geq 1, \in \mathbb{N}$).

if *start of training* **then**
  Initialize M using m. ▷ M={sets of randomly picked m% of features for each boosting step}
end if
$train\_int \leftarrow RandInt(0, k)$ ▷ $0 \leq train\_int < k, \in \mathbb{N}$
if $k = 1$ OR $train\_int \neq 0$ **then** ▷ skip $1/k^{\text{th}}$ for $k > 1$
  for $s := 1$ to S **do**
    Generate instance $x_{i,s}$ using feature set m_s ($\in M$) from instance x_i.
    $g_i \leftarrow \partial_{\hat{y}^{(s-1)}} l(y_i, \hat{y}^{(s-1)})$ ▷ gradient for committee
    $h_i \leftarrow \partial^2_{\hat{y}^{(s-1)}} l(y_i, \hat{y}^{(s-1)})$ ▷ hessian for committee
    for $c := 1$ to $C - 1$ **do**
      $T \leftarrow ceiling(h_{i,c} * 10)$ ▷ $h_{i,c} < 1$ for l^{CE}
      for $t := 1$ to T **do** ▷ train $f_{s,c}$ T times
        TRAIN $f_{s,c}(x_{i,s}, g_{i,c}/h_{i,c})$ ▷ train $f_{s,c}$ using $x_{i,s}$ and label $g_{i,c}/h_{i,c}$
      end for
    end for
    $\hat{y}_i \leftarrow \hat{y}_i + lr * f_s(x_i)$ ▷ scale $f_s(x_{i,s})$ and add to $\hat{y}_i$
  end for
end if

3.3 Predicting and computing improvements

Two variants of SGBT are proposed below to improve the computing performance and utilise already calculated hessian weights.

- *SGBT*$^{\mathbf{SK}}$: In most streaming regression trees, the computation and memory complexities are affected by the number of instances they process. Some computation and memory savings could be achieved via skip training on random instances. SGBT^{SK} randomly skips $1/k$-th of instances ($k \geq 1, \in \mathbb{N}$). k is set to 1 by default, causing it to process all instances as in SGBT. Work by Gunasekara et al. (2022), Pavlovski et al. (2017) also

exploited skip training for Stream Learning. Line 5 in algorithm 2 highlights this skip training.

- $SGBT_{\mathbf{MI}}$: Even though current base learners only support integer weights, utilizing already calculated fractional hessian weights is helpful. For l^{CE}, hessian for class c at i-th instance is always less than 1 ($h_{i,c} < 1$). Even if one passes $h_{i,c}$ to a ceiling[2] function, it will always return 1. For all instances, multiplying $h_{i,c}$ by 10 and passing that to a ceiling function results in a positive integer weight that is greater than 1 for some instances. For all the other instances, the weight is set to 1. If $ceiling(h_{i,c} * 10) = T$, SGBT can train $f_{s,c}$ base learner T times using instance $x_{i,s}$ with label $g_{i,c}/h_{i,c}$. Here, multiplier 10 ensures that $T \leq 10$ for all instances, providing a reasonable upper limit to the computational cost of this approach. This technique of training a base learner multiple times based on a calculated integer weight for an instance is quite common in stream learning (Oza & Russell, 2001; Gomes et al., 2019). Line 12 in algorithm 2 highlights this multiple training iteration approach. Furthermore, this multiple-training iteration approach allows SGBT to use streaming regression trees that do not support weights.

Algorithm 2 explains the above two variants of SGBT in detail. In the experiments, we evaluate the effectiveness of these SGBT variants.

As SGBT allows different streaming regression trees for its base learners, its final time and memory complexities are influenced by the base learner's time and memory complexities. SGBT's time complexity can be derived as $\mathcal{O}(CSf)$, and its memory complexity as $\mathcal{O}(CSf)$, assuming $\mathcal{O}(f)$ for the base learner's time and memory complexities. Here, S is the number of boosting steps, and C is the number of classes. SGBT^{MC} has the same time and memory complexities as SGBT. The time complexity of SGBT^{MC} could be further improved by parallel training each SGBT. Our implementation of SGBT^{MC} leverages this parallel processing. This allows SGBT^{MC}'s time complexity to be $\mathcal{O}(Sf)$. This is similar to current state-of-the-art streaming bagging and random-forest based methods: SRP and ARF. For SGBT^{MC}_{SK}, this time complexity is further reduced to $\mathcal{O}((1 - 1/k)Sf)$ by skipping $1/k$-th of instances at training. Table 1 contains all the notations introduced in this section.

4 Experiments

We begin our experiments by comparing SGBT against current state-of-the-art streaming bagging with random subspaces (SRP), random forest (ARF), boosting (OSB), and gradient boosting (ADITER) methods on 14 datasets. We also conducted a parameter exploration to illustrate the effects of different SGBT components.

Finally, we show an in-depth analysis concerning the computational requirements of SGBT.

Datasets: AGR_a, AGR_g, LED_a, LED_g, RBF_f, RBF_m, electricity, airlines and covtype are from Gomes et al. (2019). RandomTree, LED, RBF5, RBF_B_m, RBF_B_f were generated using MOA synthetic generators. The synthetic datasets with drifts simulate different types of concept drifts, i.e., abrupt (AGR_a, LED_a), gradual (AGR_g, LED_g), fast incremental changes (RBF_B_f, RBF_f), and moderate incremental changes (RBF_B_m, RBF_m).

[2] Similar to *Java lang.Math.ceil*(v) that returns an integer value greater than or equal to the passed-in value v.

Table 1 Notations

Notation	Description
ϕ	Ensemble model
S	# Boosting steps
s	Boosting step
$\mathcal{F}$	The space of regression trees
f_s	Streaming regression tree at boosting step s
x_i	Input features at i-th instance
n	# Instances
ω	Leaf scores for f_s
Ω	Regularization term
γ	Penalizes adding a new leaf
β	Penalizes large leaf scores
T	Number of leaves in the tree
$L(\phi)$	Loss of the ensemble
$L^{(s)}$	Loss at s-th boosting step
l	Differentiable convex loss function
$\hat{y}$	Prediction by a given regression tree
y	Target value
g	Gradient of the loss considering the previous boosting step's prediction
h	Hessian of the loss considering the previous boosting step's prediction
C	Number of classes
c	Class index
m	Percentage of features used for training f_s
M	Randomly picked m % of features for each boosting step
l^{CE}	Categorical cross-entropy loss
SGBT	Vanilla SGBT explained in algorithm 1
SGBT^{MC}	SGBT that uses separate SGBT for each class
SGBT^{SK}	SGBT that randomly skip 1/k-th of instances at training (see line 5 of algorithm 2)
k	User-defined skip training parameter ($k \geq 1, \in \mathbb{N}$)
SGBT_{MI}	SGBT that does multiple training considering hessian (see line 12 of algorithm 2)
T	# Times to train a given regression tree using an instance for SGBT_{MI}
$\mathcal{O}(f)$	Time and memory complexity of regression tree f

AGR$_\mathbf{a}$ is a *binary class synthetic* dataset with 1 M instances, where *abrupt* concept drifts occur after every 250000 instances, with 50 instances drift width.

AGR$_\mathbf{g}$ also contains *binary class synthetic* data. Here, *gradual* concept drifts occur after every 250000 instances, with 50000 instances drift width. The dataset has 1 M instances.

LED$_\mathbf{a}$ is a *multi class synthetic* dataset with 1 M instances, where *abrupt* concept drifts occur after every 250000 instances, with 50 instances drift width.

LED$_\mathbf{g}$ is also a *multi class synthetic* dataset. The dataset has *gradual* concept drifts occurring after every 250000 instances, with 50000 instances drift width. The dataset has 1 M instances.

RBF$_\mathbf{f}$ contains *multi class synthetic* data. Here, *fast incremental* concept drifts occur with 0.001 centroid's speed of change. There are 1 M instances in this dataset.

Table 2 Dataset properties: has (**D**)rifts, (**R**)eal, (**S**)ynthetic

Name	Data	Drift	#	#	#	Class dist	
	Type	Type	Instances	Features	Classes	Max(%)	Min(%)
Binary class							
AGR_a	DS	Abrupt	1 M	9	2	52.83	47.17
AGR_g	DS	Gradual	1 M	9	2	52.83	47.17
RBF_B_f	DS	Fast	1 M	10	2	51.75	48.25
RBF_B_m	DS	Moderate	1 M	10	2	51.75	48.25
RandomTree	S	n/a	100K	10	2	57.84	42.16
Electricity	R	Unknown	≈45K	8	2	57.55	42.45
Airlines	R	Unknown	≈539K	7	2	55.46	44.54
Multi class							
LED_a	DS	Abrupt	1 M	24	10	10.08	9.94
LED_g	DS	Gradual	1 M	24	10	10.08	9.94
RBF_f	DS	Fast	1 M	10	5	30.01	9.27
RBF_m	DS	Moderate	1 M	10	5	30.01	9.27
RBF5	S	n/a	100K	10	5	32.17	8.10
LED	S	n/a	100K	24	10	10.00	9.96
Covtype	R	Unknown	≈581K	54	7	48.76	0.47

RBF_m also has *multi class synthetic* data. The dataset contains 1 M instances. Here, *moderate incremental* concept drifts occur with 0.0001 centroid's speed of change.

RBF_B_f is a *binary class synthetic* dataset with 1 M instances that includes *fast incremental* concept drifts with the centroid's speed of change set to 0.001.

RBF_B_m has 1 M instances. It is a *binary class synthetic* dataset with *moderate incremental* concept drifts occurring with 0.0001 centroid's speed of change.

RandomTree is a *binary class synthetic* dataset *without any drifts*. It was generated using MOA *RandomTreeGenerator*. It has 100K instances.

LED contains *multi class synthetic* data *without any drifts*. The dataset was generated using MOA *LEDGenerator*. It also has 100K instances.

RBF5 is a dataset with 100K instances. It contains *multi class synthetic* data *without drifts*. Data was generated using MOA *RandomRBFGenerator*.

Electricity contains the Australian New South Wales Electricity Market data when the prices are not fixed. These prices are affected by the supply and demand of the market itself and are set every five minutes. It is a *binary class real-world* dataset. The class label identifies the price changes (up or down) relative to a moving average of the last 24 h. The dataset exhibits *temporal dependencies*. It contains 45310 instances.

Airlines is a *binary class real-world* dataset. The task is to predict whether a given flight will be delayed, given information on the scheduled departure. The dataset has 539382 instances.

Covertype dataset represents forest cover type for 30 x 30-meter cells obtained from the US Forest Service Region 2 Resource Information System (RIS) data. Each class corresponds to a different cover type. The dataset contains a *multi class* problem with seven *imbalanced* class labels. It includes 581010 instances.

Datasets RBF_B_m, RBF_m, RBF_B_f and RBF_f were generated using MOA *RandomRBFGeneratorDrift*. While AGR_a and AGR_g were generated using MOA

Table 3 Accuracy: SGBT^{MC} against other baselines (values are rounded to 2 decimals). Relevant Shaffer Post-hoc test results are shown in figure 1

	SGBT^{MC}	SRP	ARF	OSB	ADITER
Binary class					
AGR_a	**94.45 ± 0.01**	92.81 ± 0.19	87.87 ± 0.08	90.39 ± 0.01	90.73 ± 0.18
AGR_g	**91.91 ± 0.01**	89.68 ± 0.19	82.45 ± 0.11	87.87 ± 0.03	87.66 ± 0.34
RBF_B_m	92.10 ± 0.66	90.76 ± 0.67	**92.10 ± 0.63**	89.27 ± 0.84	76.85 ± 1.30
RBF_B_f	84.33 ± 1.22	82.15 ± 1.46	**85.61 ± 1.31**	78.14 ± 1.25	72.16 ± 1.17
RandomTree	86.19 ± 8.21	87.58 ± 2.78	90.15 ± 3.38	**92.09 ± 2.59**	68.55 ± 10.79
Electricity	88.50 ± 0.06	89.68 ± 0.14	**90.62 ± 0.05**	89.51 ± 0.00	78.77 ± 0.08
Airlines	**68.79 ± 0.03**	68.54 ± 0.05	66.68 ± 0.03	64.56 ± 0.00	62.72 ± 0.07
Avg	**86.61**	85.89	85.07	84.55	76.78
Rank	**2.14**	2.43	2.57	3.29	4.57
Multi class					
LED_a	**74.04 ± 0.01**	**74.04 ± 0.01**	73.95 ± 0.01	72.48 ± 0.00	–
LED_g	**73.32 ± 0.01**	73.25 ± 0.01	73.12 ± 0.01	72.11 ± 0.01	–
RBF_m	**88.00 ± 0.76**	86.60 ± 0.84	87.82 ± 0.75	76.81 ± 0.99	–
RBF_f	76.98 ± 1.34	76.91 ± 1.21	**77.69 ± 1.44**	50.71 ± 1.06	–
LED	73.82 ± 0.14	**73.87 ± 0.12**	73.75 ± 0.15	73.86 ± 0.18	–
RBF5	90.13 ± 0.84	90.56 ± 0.96	**90.60 ± 0.99**	85.67 ± 1.18	–
Covtype	94.29 ± 0.03	**95.34 ± 0.01**	94.72 ± 0.02	92.69 ± 0.00	–
Avg	81.51	81.51	**81.66**	74.90	–
Rank	**2.00**	**2.00**	2.29	3.71	–
Avg (both)	**84.06**	83.70	83.37	79.73	–
Rank (both)	**2.07**	2.21	2.43	3.50	–

The best (highest) average accuracy and best (lowest) rank are given in bold

ConceptDriftStream with *AgrawalGenerator*. LED_a and LED_g datasets were generated using MOA *ConceptDriftStream* with *LEDGeneratorDrift*. Table 2 summarizes the characteristics of the datasets.

SGBT was compared against the current state-of-the-art stream learning baseline SRP, streaming random forest method ARF, the latest gradient-boosted method for data streams ADITER, and the stream-boosting method OSB. AXGB was not considered in the evaluation as it failed to outperform ARF (Montiel et al., 2020). SRP used the best parameter configurations explained in Gomes et al. (2019). As 100 base learners produced the best results for SRP in Gomes et al. (2019), all the baselines used 100 base learners. ARF and OSB used the same parameters in Gomes et al. (2019) evaluation. ARF and OSB used the same base learner (HT) in SRP with the same hyperparameters as in SRP.

We collected votes for each class on each instance from ADITER's Python implementation and ran it through a MOA dummy classifier to yield the same evaluation as the other methods. SGBT was implemented as an MOA classifier, and it used 100 boosting steps (S) to match other baselines 100 base learners. The SGBT^{MC} variant was compared against the above baselines. Here, the *one-vs-rest* wrapper classifier was also implemented in MOA. SGBT used a learning rate of 0.0125 and 75% of the features at each boosting step. As SGBT requires streaming regression trees as its base learners, the

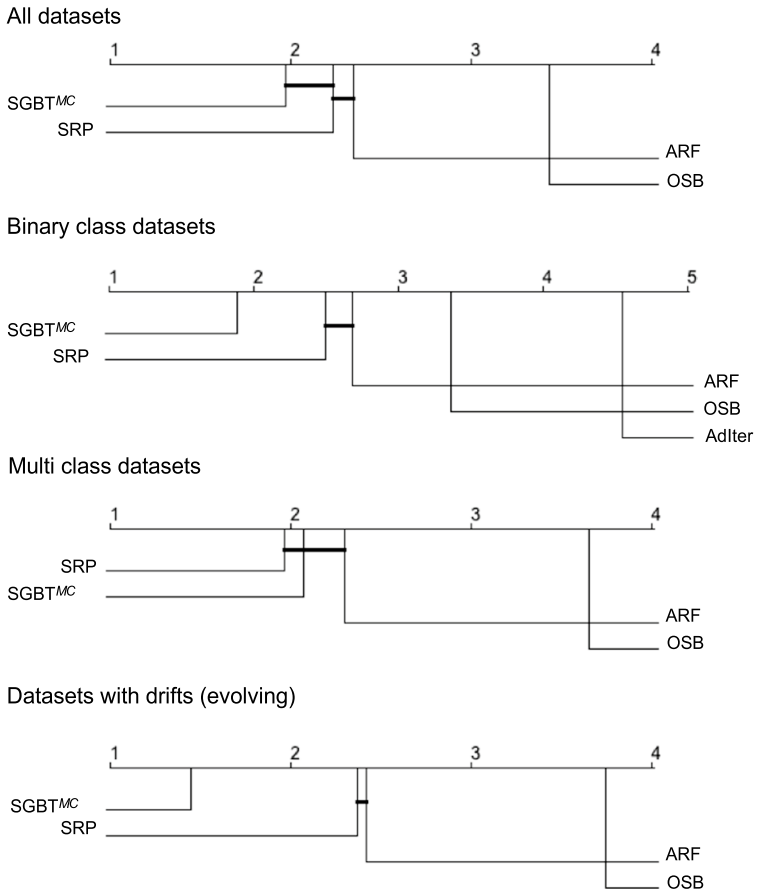

Fig. 1 Shaffer Post-hoc test with p-value 0.05 for **all, binary class, multi class, and evolving** (AGR_a, AGR_g, LED_a, LED_g, RBF_B_m, RBF_B_f, RBF_m, RBF_f) datasets (*accuracy*): $SGBT^{MC}$ against other baselines (10 iterations with different random seeds). A **lower rank is better**. Table 3 contains the individual accuracy values for each algorithm on each dataset

streaming classifier tree HT can not be used as a base learner. Therefore, the streaming regression tree FIMT-DD (Ikonomovska et al., 2011) was chosen as its base learner. FIMT-DD used a *variance reduction* split criterion, a *grace period* of 25, a *split confidence* interval of 0.05, a *constant learning rate at the leaves*, and the *regression tree option*.

Each algorithm was executed multiple times with different random seeds, and the average accuracy was considered in the evaluation process.[3]. Appendix A contains detailed information about the experimental setup.

Table 3 compares $SGBT^{MC}$'s accuracy against the baselines mentioned above. As one can see, $SGBT^{MC}$ outperforms all the baselines on binary class problems considering average

[3] Table 3, Figs. 1 and 8 used ten iterations. All the other experiments used three iterations. Code and data are available at https://github.com/nuwangunasekara/SGBT

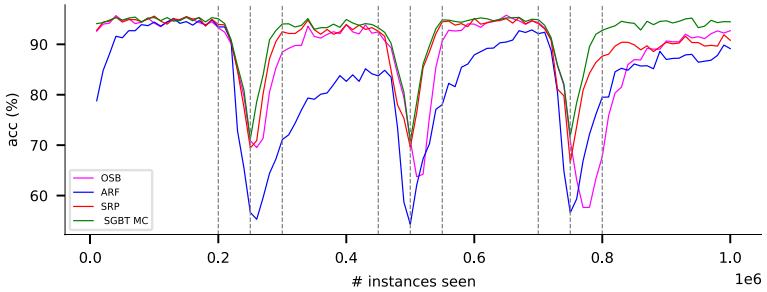

Fig. 2 Accuracy over time: SGBT^{MC} against SRP, ARF, and OSB on **AGR$_g$**. **X axis** is the number of instances seen so far. Vertical dotted lines mark a concept drift's start, center, and end. SGBT^{MC}[$S = 100$, $m = 75$, $lr = 1.25e{-}2$]

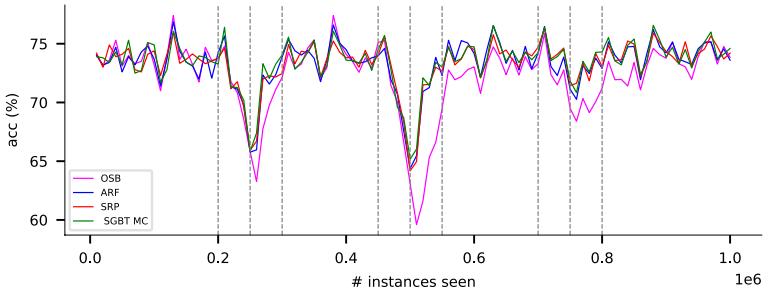

Fig. 3 Accuracy over time: SGBT^{MC} against SRP, ARF, and OSB on **LED$_g$**. **X axis** is the number of instances seen so far. Vertical dotted lines mark a concept drift's start, center, and end. SGBT^{MC}[$S = 100$, $m = 75$, $lr = 1.25e{-}2$]

accuracy and rank. It also performs equally well on multi class problems. It is also evident that SGBT^{MC} outperforms other methods on datasets with drifts: AGR$_a$, AGR$_g$, LED$_a$, LED$_g$, and RBF$_m$. This suggests that SGBT^{MC} is a good candidate not only for evolving data, but also for binary class problems.

It also performed well on the airlines dataset. On the other hand, SRP yielded good results on LED$_a$, LED, and covtype datasets, while ARF performed well on RBF_B$_m$, RBF_B$_f$, electricity, RBF$_f$, and RBF5. OSB performed well on the RandomTree dataset. The streaming gradient boosting method ADITER was the least performant among all methods. As it is a binary classifier, ADITER was only evaluated on binary class problems[4]. Furthermore, KappaM results in Appendix C (Table 9), which evaluate learner's performance on imbalanced data (Bifet et al., 2018), also align with accuracy rankings in Table 3.

Figure 1 shows the Shaffer Post-hoc test results with a p-value of 0.05 for: all, binary class, multi class, and evolving (AGR$_a$, AGR$_g$, LED$_a$, LED$_g$, RBF_B$_m$, RBF_B$_f$, RBF$_m$, RBF$_f$) datasets considering accuracy. It further highlights the fact that SGBT^{MC} outperforms other methods on binary and evolving datasets with statistical significance. For multi class problems it is on par with current state-of-the-art SRP. To our knowledge, this is the first

[4] Considering ADITER's weak performance on binary class problems and it's Python implementation, it was not evaluated on multi class problems using MOA *one-vs-rest* wrapper classifier.

Table 4 Time (seconds): SGBT^{MC} against SRP (values are rounded to 0 decimals, except ranks)

	SGBT$^{MC}_{ST}$	SGBT^{MC}	SRP
Binary class			
AGR$_a$	1423 ± 34	**1187 ± 32**	3208 ± 143
AGR$_g$	1401 ± 100	**1160 ± 37**	3838 ± 200
RBF_B$_m$	2278 ± 134	**1756 ± 190**	3697 ± 293
RBF_B$_f$	2027 ± 27	**1475 ± 130**	4728 ± 102
RandomTree	156 ± 7	**128 ± 13**	334 ± 90
Electricity	**44 ± 3**	48 ± 5	138 ± 10
Airlines	606 ± 27	**507 ± 41**	2892 ± 225
Avg	1134	**894**	2691
Rank	1.86	**1.14**	3.00
Multi class			
LED$_a$	17489 ± 2191	**1667 ± 7**	2920 ± 311
LED$_g$	16802 ± 1715	**1669 ± 16**	2901 ± 315
RBF$_m$	14511 ± 1058	**2767 ± 102**	3228 ± 114
RBF$_f$	13580 ± 1550	**2399 ± 46**	3624 ± 540
LED	1503 ± 104	**163 ± 2**	295 ± 20
RandomRBF5	1473 ± 119	268 ± 5	**163 ± 18**
Covtype	20067 ± 1869	**1789 ± 22**	3801 ± 26
Avg	12203	**1532**	2419
Rank	3.00	**1.14**	1.86
Avg (both)	6669	**1213**	2555
Rank (both)	2.43	**1.14**	2.43

The best (lowest) average time and best (lowest) rank are given in bold

time a streaming gradient boosted method is able to surpass current state-of-the-art bagging and random-forest based methods in wide range of evolving data and perform well on all types of data[5]. These Fig. 1 post-hoc test results for accuracy also align with KappaM post-hoc test results in Fig. 8 (Appendix C) with the same p-value.

We investigate each algorithm's performance on evolving data further in Figs. 2 and 3 by comparing accuracy over time for SGBT^{MC}, SRP, ARF, and OSB on AGR$_g$ and LED$_g$. Based on Figs. 2 and 3, it is evident that SGBT^{MC} had the lowest decrease in performance around drift points.

Table 4 compares the evaluation time in seconds reported by MOA among single-threaded SGBT^{MC} (SGBT$^{MC}_{ST}$), multi-threaded SGBT^{MC}, and SRP. SRP was chosen due to the fact that it had the best predictive performance among competitors considering Table 3 and Fig. 1. For binary class problems, both SGBT^{MC} variants perform faster than SRP. Maybe FIMT-DD in SGBT^{MC} is a faster base learner than HT in SRP. Compared to SGBT$^{MC}_{ST}$, SRP performs well on multi class problems. However, SGBT^{MC} performed the fastest on multi class problems leveraging parallel processing at training and prediction. Considering the

[5] AXGB failed to outperform ARF in the Montiel et al. (2020) empirical evaluation. In Wang et al. (2022) experiments, ADITER also failed to surpass ARF on synthetic evolving data.

Table 5 Accuracy: different variants of SGBT (values are rounded to 2 decimals)

	SGBT	SGBT^{MC}	SGBT_{MI}	SGBT^{MC}_{MI}
Binary class				
AGR_a	**94.45 ± 0.01**	**94.45 ± 0.01**	94.30 ± 0.01	94.30 ± 0.01
AGR_g	**91.92 ± 0.01**	**91.92 ± 0.01**	91.75 ± 0.01	91.75 ± 0.01
RBF_B_m	91.91 ± 0.19	91.91 ± 0.19	**92.58 ± 0.13**	**92.58 ± 0.13**
RBF_B_f	84.54 ± 0.67	84.54 ± 0.67	**87.36 ± 0.07**	**87.36 ± 0.07**
RandomTree	**85.72 ± 9.40**	**85.72 ± 9.40**	84.05 ± 8.36	84.05 ± 8.36
Electricity	88.54 ± 0.03	88.54 ± 0.03	**90.64 ± 0.06**	**90.64 ± 0.06**
Airlines	**68.77 ± 0.03**	**68.77 ± 0.03**	67.85 ± 0.03	67.85 ± 0.03
Avg	86.55	86.55	**86.93**	**86.93**
Rank	**2.36**	**2.36**	2.64	2.64
Multi class				
LED_a	73.96 ± 0.01	**74.05 ± 0.01**	73.71 ± 0.01	73.99 ± 0.01
LED_g	73.22 ± 0.00	**73.32 ± 0.01**	72.91 ± 0.01	73.18 ± 0.01
RBF_m	87.13 ± 0.71	87.96 ± 0.63	88.18 ± 0.91	**88.92 ± 0.57**
RBF_f	75.40 ± 1.84	77.03 ± 1.39	79.28 ± 1.62	**81.14 ± 1.19**
LED	73.81 ± 0.17	73.81 ± 0.19	73.56 ± 0.19	**73.82 ± 0.18**
RBF5	88.80 ± 0.91	89.76 ± 0.72	90.05 ± 0.81	**90.68 ± 0.58**
Covtype	94.31 ± 0.01	94.29 ± 0.02	**95.18 ± 0.02**	94.73 ± 0.02
Avg	80.95	81.46	81.84	**82.35**
Rank	3.29	2.43	2.71	**1.57**
Avg (both)	83.75	84.01	84.39	**84.64**
Rank (both)	2.82	2.39	2.68	**2.11**

Gomes et al. (2019) empirical evaluation on run time for 100 base learners, we would like to acknowledge that ARF and OSB can perform faster than SRP in practice. However, they have an inferior predictive performance compared to SRP in Table 3 evaluation and in the empirical evaluation of Gomes et al. (2019).

4.1 Multiple steps and multi class support

Another study was conducted to understand the performance of different SGBT variants: SGBT, SGBT^{MC}, SGBT_{MI}, and SGBT^{MC}_{MI}. SGBT^{MC} supports multi class problems using binary SGBTs, and SGBT_{MI} employs multiple iterations by *hessian* weights. Both SGBT^{MC} and SGBT_{MI} are orthogonal, so they can be fused to yield SGBT^{MC}_{MI}. All SGBT variants used the same hyperparameter configurations as in the previous experiments. Table 5 shows the results of the study. Since SGBT^{MC} reverts to SGBT and SGBT^{MC}_{MI} reverts to SGBT_{MI} on binary class problems, if one ignores SGBT^{MC}_{MI} and SGBT^{MC} for binary class problems, SGBT performs well on most of the binary class datasets compared to SGBT_{MI}. However, SGBT_{MI} has a higher average accuracy for that category. This suggests it performs exceptionally well on certain datasets such as RBF_B_m, RBF_B_f and electricity. This results on RBF_B_f, which has fast-evolving drifts, is interesting, as it suggests that multiple training iterations by hessian in SGBT_{MI} improve SGBT's performance on fast-evolving

Table 6 Test then train accuracy of SGBT^{MC} for different learning rates (*lr*), boosting steps (*S*), feature percentages (*m*), and Tree Replacement (TR) mechanisms: TR via drift detector and no TR. Notes: (i) Results are ranked and highlighted separately for *lr*, *S*, *m* and TR. (ii) TR: FIMT-DD does TR via PHT, and SGT uses a wrapper classifier to do TR via DDM. (iii) Values are rounded to 2 decimals, 4 decimals were considered to select the winner

	Learning rate (*lr*)			Boosting steps (*S*)					% of features (*m*)				TR	
Learner	FIMT-DD												SGT	
TR	via Page-Hinckley Test (PHT)												DDM	no TR
lr	6.25e−3	1.25e−2	2.50e−2	1.25e−2					1.25e−2				1.25e−2	
S	100			20	40	60	80	100	100				100	
m	75			75					45	60	75	100	75	
Binary class														
AGR_a	94.42	**94.45**	94.44	94.08	94.42	94.42	**94.45**	**94.45**	92.76	93.73	**94.45**	94.40	**93.86**	91.40
AGR_g	91.90	**91.92**	91.91	91.51	91.87	91.88	**91.93**	91.92	90.29	91.20	91.92	**91.97**	**91.60**	86.39
RBF_B_m	91.69	91.91	**92.04**	90.60	91.31	91.59	91.74	**91.91**	90.96	91.48	**91.91**	91.66	**78.85**	59.77
RBF_B_f	83.42	84.54	**84.82**	80.45	82.32	83.14	83.71	**84.54**	82.75	83.54	**84.54**	83.98	**57.41**	57.25
RandomTree	85.14	85.72	**86.36**	84.35	84.49	85.27	85.10	**85.72**	81.89	83.99	**85.72**	81.38	86.07	**87.08**
electricity	87.90	88.54	**89.27**	87.01	87.68	87.96	88.36	**88.54**	88.42	**88.57**	88.54	88.06	**76.83**	73.41
airlines	68.70	68.77	**68.83**	68.01	68.51	68.63	68.71	**68.77**	**68.85**	68.83	68.77	68.47	**64.70**	62.94
avg	86.17	86.55	**86.81**	85.14	85.80	86.13	86.29	**86.55**	85.13	85.91	**86.55**	85.70	**78.48**	74.03
rank	3.00	1.71	**1.29**	5.00	4.00	2.86	1.86	**1.29**	3.29	2.43	**1.57**	2.71	**1.14**	1.86

Table 6 (continued)

	Learning rate (*lr*)			Boosting steps (*S*)					% of features (*m*)				TR	
Learner	FIMT-DD												SGT	
TR	via Page-Hinckley Test (PHT)												DDM	no TR
lr	6.25e−3	1.25e−2	2.50e−2	1.25e−2					1.25e−2				1.25e−2	
S	100			20	40	60	80	100	100				100	
m	75			75					45	60	75	100	75	
Multi class														
LED_a	74.04	**74.05**	74.02	74.00	73.96	74.02	74.04	**74.05**	73.87	73.99	**74.05**	73.98	**73.68**	71.73
LED_g	73.32	73.32	**73.33**	73.23	73.26	73.29	73.29	**73.32**	72.91	73.26	**73.32**	73.27	**72.85**	71.42
RBF_m	87.70	87.96	**88.24**	86.42	87.25	87.59	87.81	**87.96**	86.88	87.53	**87.96**	87.81	**66.86**	36.64
RBF_f	76.12	77.03	**78.05**	72.39	74.75	75.78	76.54	**77.03**	75.64	76.47	**77.03**	76.81	**28.57**	27.79
LED	**73.84**	73.81	73.79	73.81	73.82	73.83	**73.84**	73.81	73.80	**73.89**	73.81	73.56	**72.67**	72.67
RBF5	89.58	89.76	**89.97**	89.16	89.49	89.59	89.67	**89.76**	89.24	**89.84**	89.76	85.92	82.01	**83.20**
covtype	93.86	94.31	**94.78**	93.17	93.61	93.92	94.07	**94.31**	94.29	**94.39**	94.31	93.85	**83.08**	69.78
avg	81.21	81.46	**81.74**	80.31	80.88	81.15	81.32	**81.46**	80.95	81.34	**81.46**	80.74	**68.53**	61.89
rank	2.57	1.86	**1.57**	4.86	4.00	2.86	1.86	**1.43**	3.57	2.00	**1.43**	3.00	**1.14**	1.86
avg (both)	83.69	83.97	**84.27**	82.73	83.34	83.64	83.80	**83.97**	83.04	83.62	**83.97**	83.22	**73.50**	67.96
rank (both)	2.79	1.79	**1.43**	4.93	4.00	2.86	1.86	**1.36**	3.43	2.21	**1.50**	2.86	**1.14**	1.86

Standard deviations are available in Appendix B (Table 8)

The best (highest) average accuracy and best (lowest) rank for each category are given in bold

data. For multi class problems, SGBT^{MC}_{MI} is the clear winner. When one compares SGBT^{MC} with SGBT, it is clear that multi class support using binary SGBTs performs better than SGBT with multi class support. On the other hand, multi class results on SGBT and SGBT_{MI} suggest that multiple iterations by hessian improve SGBT's accuracy on multi class problems. This explains why SGBT^{MC}_{MI} performs best on multi class problems, as it includes multi class support using binary SGBTs and multiple iterations by hessian approaches. Overall performance by SGBT^{MC}_{MI} exceeds the performance of SGBT^{MC}, which is compared against other baselines in Table 3. But SGBT^{MC} was used in Table 3 evaluation considering its computation efficiency compared to SGBT^{MC}_{MI}. On the other hand, SGBT^{MC}_{MI} is a good candidate for evolving data stream applications that prioritize predictive performance over computation efficiency.

4.2 Parameter exploration

A parameter exploration was conducted to understand the impact of learning rate (lr), boosting steps (S), weight (h_i) transfer methods, percentage of features (m), and the independent TR mechanism at each tree via drift detection on SGBT^{MC}'s predictive performance. The results for all these analyses are shown in Table 6 (ranked separately).

Three learning rates: 6.25e−3, 1.25e−2, and 2.50e−2, were used in the study to understand the effect of learning rate (lr) on SGBT^{MC}'s performance. All the other configurations: FIMT-DD base learner, 75% of features (m), and 100 boosting steps (S) were kept unchanged. As per Table 6, considering SGBT^{MC} [$S = 100$, $m = 75$, lr ={6.25e−3, 1.25e−2, 2.50e−2}, FIMT-DD] configurations, in general, larger learning rates (lr) seem to favour both binary and multi class problems.

In a separate study to understand the effect of boosting steps on SGBT^{MC}'s performance, five boosting steps (20, 40, 60, 80, 100) were considered. In this study, base learner (FIMT-DD), feature percentage (m=75%), and learning rate (lr=1.25e−2) were kept unchanged. According to Table 6, when considering SGBT^{MC} [$S = 20, 40, 60, 80, 100$, $m = 75$, lr = 1.25e−2, FIMT-DD] configurations, 100 boosting steps seem to yield good results than the smaller boosting steps for both binary and multi class problems. This aligns with OSB results in Gomes et al. (2019), where more boosting iterations performed better than fewer boosting iterations.

In another study investigating the influence of different feature percentages (m) on SGBT^{MC}'s performance, all SGBT^{MC} configurations remained constant, including the base learner (FIMT-DD), learning rate (lr=1.25e−2), and boosting steps (S=100), except for the feature percentage (m).

According to Table 6, among SGBT^{MC} [$S = 100$, $m = 45, 60, 75, 100$, lr =1.25e−2] configurations, 75% of features yield good accuracy on most datasets. Not having 100% of the features helps to increase the diversity of the ensemble, which avoids overfitting to data. These results match (Gomes et al., 2017, 2019) findings where ARF and SRP perform best with 60% of the features.

A separate study examines the effect of independent TR mechanisms by each base learner on SGBT^{MC}'s performance. For this study, SGT was selected as the base learner since FIMT-DD has a built-in TR mechanism. Hence, a generic regressor with an inbuilt TR mechanism based on DDM's *warning* and *out-of-control* signals was introduced into MOA. This allows us to enable or disable the underlying TR strategy using a generic regressor with SGT and DDM or just using SGT. The DDM settings were: *minimum number*

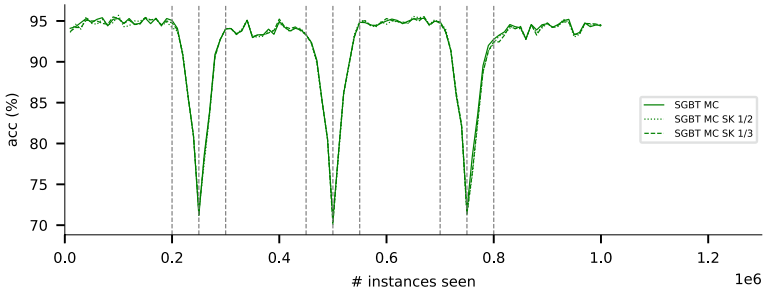

Fig. 4 Accuracy over time: Different SGBT^{MC} versions on $\mathbf{AGR_g}$. **X axis** is the number of instances seen so far. Vertical dotted lines mark a concept drift's start, center, and end. $\text{SGBT}^{MC}_{SK\_1/k}[S = 100, m = 75, lr = 1.25\text{e}{-2}]$

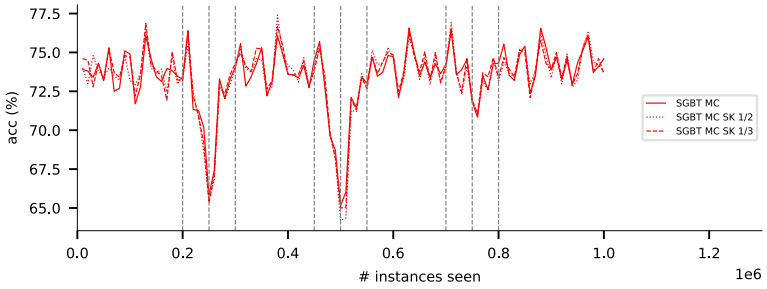

Fig. 5 Accuracy over time: Different SGBT^{MC} versions on $\mathbf{LED_g}$. **X axis** is the number of instances seen so far. Vertical dotted lines mark a concept drift's start, center, and end. $\text{SGBT}^{MC}_{SK\_1/k}[S = 100, m = 75, lr = 1.25\text{e}{-2}]$

of instances before permitting a change detection = 250, *warning level* = 2.0, and *out-of-control level* = 2.5. SGT used the same default configurations used in Gouk et al. (2019). From Table 6 results, one can see that having an internal TR mechanism often improves performance. Also, all the SGBT^{MC} configurations with SGT perform poorly on RBF_f. Maybe SGT's default *warmStart* (number of instances used to estimate bin boundaries for numeric values) 1000 is too large for RBF_f with fast-moving drifts.

4.3 Skip training on instances

Another study was conducted using $\text{SGBT}^{MC}_{SK\_1/k}[S = 100, m = 75, lr = 1.25\text{e}{-2}]$ with different k values to understand the effect of random skip training. Here, k was set to 1, 2, and 3 so that $\text{SGBT}^{MC}_{SK\_1/k}$ would not skip, skipping 1/2 and 1/3 of instances. As per Table 7, apart from RBF_B_f and RBF_f $\text{SGBT}^{MC}_{SK\_1/3}$, produced good results even with 1/3-rd of instances skipped. Here, slight poor accuracy in those two datasets may be because both RBF_B_f and RBF_f have fast-moving drifts.

To further illustrate the influence of random skipping a bit, another study was conducted using $\text{SGBT}^{MC}_{SK\_1/k}[S = 100, m = 75, lr = 1.25\text{e}{-2}]$ with different k values: 1, 2, 3 on AGR_g and LED_g datasets. The idea here is to understand the effect of skip training instances on $\text{SGBT}^{MC}_{SK\_1/k}$'s performance for binary and multi class problems. Both AGR_g

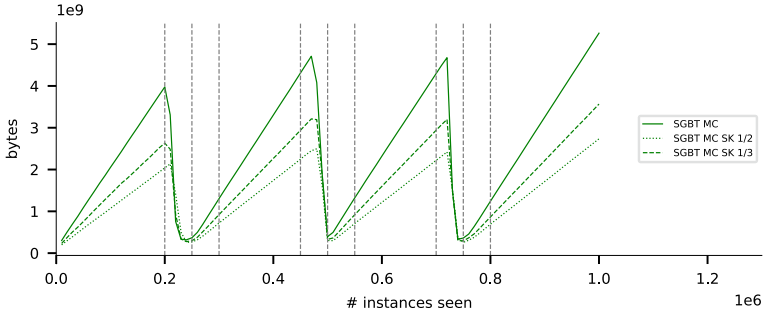

Fig. 6 Model size over time: Different SGBT^{MC} versions on **AGR$_g$**. **X axis** is the number of instances seen so far. Vertical dotted lines mark a concept drift's start, center, and end. SGBT$^{MC}_{SK_1/k}$[$S = 100$, $m = 75$, $lr =$ 1.25e−2]

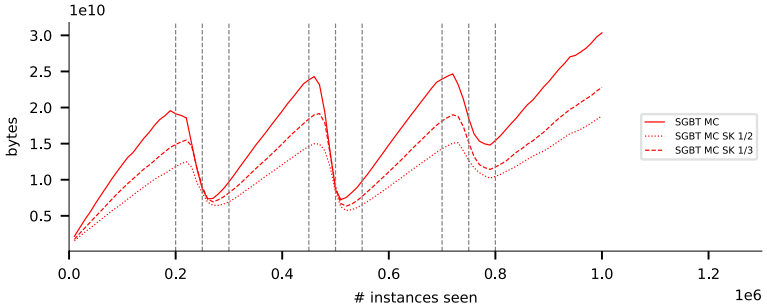

Fig. 7 Model size over time: Different SGBT^{MC} versions on **LED$_g$**. **X axis** is the number of instances seen so far. Vertical dotted lines mark a concept drift's start, center, and end. SGBT$^{MC}_{SK_1/k}$[$S = 100$, $m = 75$, $lr =$ 1.25e−2]

and LED$_g$ had drifts happening at the same time intervals. However, AGR$_g$ is a binary problem, and LED$_g$ is a multi class problem with 10 classes. Accuracy and model size statistics were collected every 10000 instances. When one considers the classification accuracy in Figs. 4 and 5, skipping instances for training does not significantly hinder the accuracy on both AGR$_g$ and LED$_g$. On the other hand, skipping instances results in significant memory savings on both datasets in Figs. 6 and 7. These savings are much more prevalent in LED$_g$ as SGBT$^{MC}_{SK_1/k}$ needs 10 SGBTs compared to 1 for AGR$_g$.

5 Conclusion

This work uses the generic weighted squared loss elicited in Friedman et al. (2000); Chen and Guestrin (2016) with *hessian* as the weight and *gradient* over *hessian* as the target, considering the loss of the previous boosting step with streaming regression trees with internal TR strategy to propose SGBT. In the experiments, SGBT variant SGBT^{MC}

Table 7 Accuracy and evaluation time(s) of $\text{SGBT}^{MC}_{SK\_1/k}[S = 100, m = 75, lr = 1.25\text{e}{-2}]$

	Accuracy (%)			Time (s)		
	$\text{SGBT}^{MC}_{SK\_1/k}$		SGBT^{MC}	$\text{SGBT}^{MC}_{SK\_1/k}$		SGBT^{MC}
k	2 (skip 1/2)	3 (skip 1/3)	1 (no skip)	2 (skip 1/2)	3 (skip 1/3)	1 (no skip)
Binary class						
AGR_a	94.32	94.36	**94.45**	**719.16**	818.16	1386.55
AGR_g	91.81	91.83	**91.92**	**663.56**	810.28	1359.81
RBF_B_m	90.55	91.20	**91.91**	**983.85**	1218.42	2126.22
RBF_B_f	78.11	80.89	**84.54**	**891.39**	1061.86	1749
RandomTree	84.73	85.62	**85.72**	**74.62**	93.09	148.69
Electricity	85.88	87.10	**88.54**	**31.43**	37.43	54.75
Airlines	67.99	68.31	**68.77**	**309.32**	378.3	584.59
Avg	84.77	85.62	**86.55**	**524.76**	631.07	1058.52
Rank	3.00	2.00	**1.00**	**1**	2	3
Multi class						
LED_a	73.91	73.97	**74.05**	**1002.87**	1197.72	1747.52
LED_g	73.18	73.22	**73.32**	**977.22**	1217.96	1718.98
RBF_m	86.17	87.01	**87.96**	**1441.83**	1875.82	2773.24
RBF_f	68.75	72.62	**77.03**	**1395.00**	1683.68	2439.11
LED	73.80	73.76	**73.81**	**91.99**	121.11	162.38
RBF5	88.51	89.06	**89.76**	**153.97**	201.3	279.21
Covtype	92.23	93.15	**94.28**	**1005.95**	1318.55	1944.42
Avg	79.51	80.40	**81.46**	**866.97**	1088.02	1580.69
Rank	2.86	2.14	**1.00**	**1**	2	3
Avg (both)	82.14	83.01	**83.97**	**695.87**	859.55	1319.61
Rank (both)	2.93	2.07	**1.00**	**1**	2	3

Standard deviations are available in Table 10

The best (highest) average accuracy, best (lowest) average time, and best (lowest) rank are given in bold

with FIMT-DD as the base learner produced superior results compared to the state-of-the-art streaming methods on large evolving data with multiple drifts and drift types.

SGBT calculated *hessian* weights result in fractions for most of the loss functions. To our knowledge, none of the streaming regression trees support non-integer weights. To circumvent this limitation, SGBT employs a weight of 1 or transformed weight (which yields a positive integer) to train the base learner. As future work, one could explore the work by Pébay et al. (2016) and Schubert and Gertz (2018) for incremental calculation of variance and covariance to support fractional weights for SGT and FIMT-DD base learners. Another future work is to skip training selectively on certain instances considering the loss above a certain threshold, like in Gunasekara et al. (2022), instead of random skipping.

Table 8 Test then train accuracy and Standard Deviations of SGBT^{MC} for different learning rates (*lr*), boosting steps (*S*), feature percentages (*m*), and Tree Replacement (TR) mechanisms: TR via drift detector and no TR

	Learning rate (*lr*)			Boosting steps (*S*)					% of features (*m*)				TR	
Learner	FIMT-DD												SGT	
TR	Via Page-Hinckley Test (PHT)												DDM	no TR
lr	6.25e−3	1.25e−2	2.50e−2	1.25e−2					1.25e−2				1.25e−2	
S	100			20	40	60	80	100	100				100	
m	75			75					45	60	75	100	75	
Binary class														
AGR_a	94.42 ± 0.00	**94.45 ± 0.01**	94.44 ± 0.01	94.08 ± 0.01	94.42 ± 0.01	94.42 ± 0.01	**94.45 ± 0.01**	94.45 ± 0.01	92.76 ± 0.01	93.73 ± 0.01	**94.45 ± 0.01**	94.40 ± 0.00	**93.86 ± 0.00**	91.40 ± 0.02
AGR_g	91.90 ± 0.01	**91.92 ± 0.01**	91.91 ± 0.01	91.51 ± 0.04	91.87 ± 0.02	91.88 ± 0.01	**91.93 ± 0.01**	91.92 ± 0.01	90.29 ± 0.02	91.20 ± 0.00	91.92 ± 0.01	**91.97 ± 0.01**	**91.60 ± 0.00**	86.39 ± 0.11
RBF_B_m	91.69 ± 0.09	91.91 ± 0.19	**92.04 ± 0.15**	90.60 ± 0.22	91.31 ± 0.15	91.59 ± 0.14	91.74 ± 0.13	**91.91 ± 0.19**	90.96 ± 0.00	91.48 ± 0.04	**91.91 ± 0.19**	91.66 ± 0.20	**78.85 ± 0.92**	59.77 ± 1.20
RBF_B_f	83.42 ± 0.39	84.54 ± 0.67	**84.82 ± 0.05**	80.45 ± 0.14	82.32 ± 0.10	83.14 ± 0.16	83.71 ± 0.13	**84.54 ± 0.67**	82.75 ± 0.41	83.54 ± 0.25	**84.54 ± 0.67**	83.98 ± 0.23	**57.41 ± 1.67**	57.25 ± 1.50
RandomTree	85.14 ± 9.96	85.72 ± 9.40	**86.36 ± 9.10**	84.35 ± 10.61	84.49 ± 11.00	85.27 ± 9.67	85.10 ± 9.86	**85.72 ± 9.40**	81.89 ± 5.95	83.99 ± 6.63	**85.72 ± 9.40**	81.38 ± 14.50	86.07 ± 2.15	**87.08 ± 2.08**
Electricity	87.90 ± 0.07	88.54 ± 0.03	**89.27 ± 0.04**	87.01 ± 0.17	87.68 ± 0.05	87.96 ± 0.07	88.36 ± 0.03	**88.54 ± 0.03**	88.42 ± 0.09	**88.57 ± 0.06**	88.54 ± 0.03	88.06 ± 0.00	**76.83 ± 0.08**	73.41 ± 0.22
Airlines	68.70 ± 0.00	68.77 ± 0.03	**68.83 ± 0.02**	68.01 ± 0.06	68.51 ± 0.01	68.63 ± 0.02	68.71 ± 0.03	**68.77 ± 0.03**	**68.85 ± 0.03**	68.83 ± 0.04	68.77 ± 0.03	68.47 ± 0.00	**64.70 ± 0.05**	62.94 ± 0.26
Avg	86.17	86.55	**86.81**	85.14	85.80	86.13	86.29	**86.55**	85.13	85.91	**86.55**	85.70	**78.48**	74.03
Rank	3.00	1.71	**1.29**	5.00	4.00	2.86	1.86	**1.29**	3.29	2.43	**1.57**	2.71	**1.14**	1.86

Table 8 (continued)

	Learning rate (*lr*)			Boosting steps (*S*)					% of features (*m*)				TR	
Learner	FIMT-DD												SGT	
TR	Via Page-Hinckley Test (PHT)												DDM	no TR
lr	6.25e−3	1.25e−2	2.50e−2	1.25e−2					1.25e−2				1.25e−2	
S	100			20	40	60	80	100	100				100	
m	75			75					45	60	75	100	75	
Multi class														
LED_a	74.04 ± 0.01	**74.05 ± 0.01**	74.02 ± 0.01	74.00 ± 0.00	73.96 ± 0.00	74.02 ± 0.01	74.04 ± 0.02	**74.05 ± 0.01**	73.87 ± 0.00	73.99 ± 0.01	**74.05 ± 0.01**	73.98 ± 0.01	**73.68 ± 0.00**	71.73 ± 0.00
LED_g	73.32 ± 0.02	73.32 ± 0.01	**73.33 ± 0.01**	73.23 ± 0.02	73.26 ± 0.01	73.29 ± 0.01	73.29 ± 0.01	**73.32 ± 0.01**	72.91 ± 0.01	73.26 ± 0.00	**73.32 ± 0.01**	73.27 ± 0.01	**72.85 ± 0.01**	71.42 ± 0.01
RBF_m	87.70 ± 0.64	87.96 ± 0.63	**88.24 ± 0.64**	86.42 ± 0.72	87.25 ± 0.70	87.59 ± 0.65	87.81 ± 0.65	**87.96 ± 0.63**	86.88 ± 0.65	87.53 ± 0.64	**87.96 ± 0.63**	87.81 ± 0.66	**66.86 ± 1.27**	36.64 ± 0.65
RBF_f	76.12 ± 1.45	77.03 ± 1.39	**78.05 ± 1.38**	72.39 ± 1.46	74.75 ± 1.49	75.78 ± 1.44	76.54 ± 1.44	**77.03 ± 1.39**	75.64 ± 1.52	76.47 ± 1.48	**77.03 ± 1.39**	76.81 ± 1.38	**28.57 ± 1.50**	27.79 ± 2.44
LED	**73.84 ± 0.22**	73.81 ± 0.19	73.79 ± 0.22	73.81 ± 0.21	73.82 ± 0.22	73.83 ± 0.16	**73.84 ± 0.17**	73.81 ± 0.19	73.80 ± 0.20	**73.89 ± 0.25**	73.81 ± 0.19	73.56 ± 0.16	**72.67 ± 0.16**	72.67 ± 0.33
RBF5	89.58 ± 0.70	89.76 ± 0.72	**89.97 ± 0.78**	89.16 ± 0.60	89.49 ± 0.65	89.59 ± 0.66	89.67 ± 0.74	**89.76 ± 0.72**	89.24 ± 0.34	**89.84 ± 0.55**	89.76 ± 0.72	85.92 ± 1.23	82.01 ± 0.51	**83.20 ± 0.39**
Covtype	93.86 ± 0.02	94.31 ± 0.02	**94.78 ± 0.01**	93.17 ± 0.01	93.61 ± 0.02	93.92 ± 0.01	94.07 ± 0.06	**94.31 ± 0.02**	94.29 ± 0.07	**94.39 ± 0.04**	94.31 ± 0.02	93.85 ± 0.00	**83.08 ± 0.02**	69.78 ± 4.71
Avg	81.21	81.46	**81.74**	80.31	80.88	81.15	81.32	**81.46**	80.95	81.34	**81.46**	80.74	**68.53**	61.89
Rank	2.57	1.86	**1.57**	4.86	4.00	2.86	1.86	**1.43**	3.57	2.00	**1.43**	3.00	**1.14**	1.86
Avg_{both}	83.69	83.97	**84.27**	82.73	83.34	83.64	83.80	**83.97**	83.04	83.62	**83.97**	83.22	**73.50**	67.96
$Rank_{both}$	2.79	1.79	**1.43**	4.93	4.00	2.86	1.86	**1.36**	3.43	2.21	**1.50**	2.86	**1.14**	1.86

(i) Results are ranked and highlighted separately for *lr*, *S*, *m* and TR. (ii) TR: FIMT-DD does TR via PHT, and SGT uses a wrapper classifier to do TR via DDM. (values rounded to 2 decimals, 4 decimals were considered to select the winner)

The best (highest) average accuracy and best (lowest) rank for each category are given in bold

Appendix A: Experimental setup

Experiments relating to Table 3, Figs. 1 and 8 used ten iterations with random seeds: 5, 9, 17, 13, 19, 23, 29, 31, 37 and 121. All the other experiments used three iterations with random seeds: 9, 17, and 121.

Experiments were run on an i) Ubuntu 18.04 LST system with AMD EPYC 7702 64-Core Processor at 4.00GHz, and with 1000GB RAM and on ii)Ubuntu 20.04.3 system with an Intel(R) Core(TM) i7-6700K CPU at 4.00GHz, and with 64GB RAM. All CPU Time experiments were done on the system i. The OpenJDK version was 11.0.11, and the JVM configurations were: -Xmx96g, -Xms50m, and -Xss1g.

Appendix B: Parameter exploration results

Table 8 contains average accuracy and standard deviation for parameter exploration experiments in Sect. 4.2.

Appendix C: KappaM results

Table 9 and Fig. 8 contain KappM results for learners: SGBT^{MC}, SRP, ARF, OSB and ADITER on all datasets discussed in Sect. 4. KappaM measures learner's performance against a majority class classifier (Bifet et al., 2018). It is used to evaluate learner's performance on an imbalanced dataset (Bifet et al., 2018). Here learner rankings in Table 9 and Fig. 8 align with rankings in Table 3 and Fig. 1.

Table 9 KappaM (percentage): SGBT^{MC} against other baselines (values rounded to 2 decimals). Relevant Shaffer Post-hoc test results are shown in Fig. 8

	SGBT^{MC}	SRP	ARF	OSB	ADITER
Binary class					
AGR_a	**88.23 ± 0.02**	84.76 ± 0.40	74.28 ± 0.16	79.62 ± 0.02	80.35 ± 0.37
AGR_g	**82.85 ± 0.02**	78.11 ± 0.39	62.80 ± 0.23	74.29 ± 0.06	73.83 ± 0.72
RBF_B_m	81.28 ± 2.96	78.41 ± 3.52	**81.55 ± 2.98**	74.63 ± 3.21	46.33 ± 3.03
RBF_B_f	62.76 ± 6.67	58.13 ± 8.04	**66.22 ± 6.91**	48.39 ± 5.26	35.39 ± 4.54
RandomTree	67.72 ± 17.18	69.50 ± 7.06	74.98 ± 10.45	**80.31 ± 7.28**	25.36 ± 19.02
Electricity	72.91 ± 0.14	75.69 ± 0.33	**77.91 ± 0.12**	75.30 ± 0.00	49.99 ± 0.19
Airlines	**29.93 ± 0.07**	29.37 ± 0.11	25.20 ± 0.06	20.43 ± 0.00	16.31 ± 0.17
Avg	**69.38**	67.71	66.13	64.71	46.79
Rank	**2.14**	2.43	2.57	3.29	4.57
Multi class					
LED_a	**71.11 ± 0.01**	**71.11 ± 0.01**	71.01 ± 0.01	69.37 ± 0.01	–
LED_g	**70.31 ± 0.01**	70.23 ± 0.01	70.08 ± 0.01	68.96 ± 0.01	–
RBF_m	**83.12 ± 1.56**	81.14 ± 1.79	82.85 ± 1.58	67.40 ± 2.13	–
RBF_f	67.61 ± 2.94	67.51 ± 2.85	**68.59 ± 3.04**	30.73 ± 3.08	–
LED	70.82 ± 0.15	**70.87 ± 0.13**	70.73 ± 0.16	70.86 ± 0.20	–
RBF5	86.10 ± 1.55	86.70 ± 1.74	**86.76 ± 1.77**	79.84 ± 2.03	–
covtype	88.85 ± 0.05	**90.91 ± 0.03**	89.69 ± 0.04	85.74 ± 0.00	–
Avg	76.84	76.92	**77.10**	67.56	–
Rank	**2.00**	**2.00**	2.29	3.71	–
Avg (both)	**73.11**	72.32	71.62	66.13	–
Rank (both)	**2.07**	2.21	2.43	3.50	–

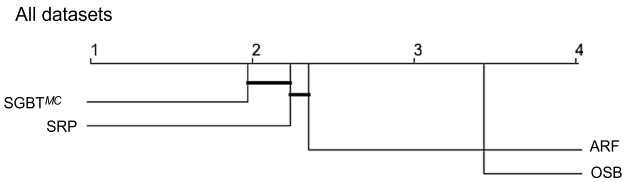

Fig. 8 Shaffer Post-hoc test with p-value 0.05 for **all datasets** (*KappaM*): SGBT^{MC} against other baselines (10 iterations with different random seeds). A **lower rank is better**. Table 9 contains the individual KappaM values for each algorithm on each dataset

Appendix D: Skip training on instances

Table 10 Average Accuracy and evaluation time(s) with Standard Deviation for SGBT $^{MC}_{SK\_1/k}$ [$S = 100$, $m = 75$, lr =1.25e−2] (values rounded to 2 decimals, 4 decimals were considered to select the winner)

	Accuracy (%)			Time (s)		
	SGBT$^{MC}_{SK_1/k}$		SGBT^{MC}	SGBT$^{MC}_{SK_1/k}$		SGBT^{MC}
k	2 (skip 1/2)	3 (skip 1/3)	1 (no skip)	2 (skip 1/2)	3 (skip 1/3)	1 (no skip)
Binary class						
AGR_a	94.32 ± 0.02	94.36 ± 0.01	**94.45 ± 0.01**	**719.16 ± 64.47**	818.16 ± 7.11	1,386.55 ± 51.40
AGR_g	91.81 ± 0.01	91.83 ± 0.03	**91.92 ± 0.01**	**663.56 ± 5.70**	810.28 ± 27.69	1,359.81 ± 74.20
RBF_B_m	90.55 ± 0.17	91.20 ± 0.15	**91.91 ± 0.19**	**983.85 ± 18.69**	1,218.42 ± 97.26	2,126.22 ± 299.86
RBF_B_f	78.11 ± 0.31	80.89 ± 0.20	**84.54 ± 0.67**	**891.39 ± 23.25**	1,061.86 ± 67.40	1,749.00 ± 251.29
RandomTree	84.73 ± 8.10	85.62 ± 8.99	**85.72 ± 9.40**	**74.62 ± 3.95**	93.09 ± 8.07	148.69 ± 6.50
Electricity	85.88 ± 0.05	87.10 ± 0.26	**88.54 ± 0.03**	**31.43 ± 2.72**	37.43 ± 0.67	54.75 ± 4.68
Airlines	67.99 ± 0.03	68.31 ± 0.07	**68.77 ± 0.03**	**309.32 ± 12.37**	378.30 ± 12.75	584.59 ± 47.24
Avg	84.77	85.62	**86.55**	**524.76**	631.07	1,058.52
Rank	3.00	2.00	**1.00**	**1.00**	2.00	3.00
Multi class						
LED_a	73.91 ± 0.01	73.97 ± 0.03	**74.05 ± 0.01**	**1,002.87 ± 17.89**	1,197.72 ± 101.18	1,747.52 ± 149.75
LED_g	73.18 ± 0.01	73.22 ± 0.01	**73.32 ± 0.01**	**977.22 ± 25.89**	1,217.96 ± 57.35	1,718.98 ± 128.82
RBF_m	86.17 ± 0.82	87.01 ± 0.73	**87.96 ± 0.63**	**1,441.83 ± 210.95**	1,875.82 ± 227.84	2,773.24 ± 164.02
RBF_f	68.75 ± 1.97	72.62 ± 1.58	**77.03 ± 1.39**	**1,395.00 ± 92.88**	1,683.68 ± 38.21	2,439.11 ± 200.90
LED	73.80 ± 0.20	73.76 ± 0.17	**73.81 ± 0.19**	**91.99 ± 6.92**	121.11 ± 9.15	162.38 ± 15.80
RBF5	88.51 ± 0.82	89.06 ± 0.79	**89.76 ± 0.72**	**153.97 ± 2.43**	201.30 ± 13.50	279.21 ± 32.25
Covtype	92.23 ± 0.00	93.15 ± 0.00	**94.28 ± 0.00**	**1,005.95 ± 49.29**	1,318.55 ± 48.46	1,944.42 ± 262.60
Avg	79.51	80.40	**81.46**	**866.97**	1,088.02	1,580.69
Rank	2.86	2.14	**1.00**	**1.00**	2.00	3.00
Avg (both)	82.14	83.01	**83.97**	**695.87**	859.55	1,319.61
Rank (both)	2.93	2.07	**1.00**	**1.00**	2.00	3.00

Funding Open Access funding enabled and organized by CAUL and its Member Institutions. NZ Tertiary Education Commission funded Real-time Analytics of Big Data Programme.

Data availability Open source

Declarations

Conflict of interest Not applicable

Ethical approval Not applicable

Consent to participate Not applicable

Consent for publication Not applicable

References

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences, 55*(1), 119–139.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of statistics, 29*, 1189–1232.

Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis, 38*(4), 367–378.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting (with discussion and a rejoinder by the authors). *The Annals of Statistics, 28*(2), 337–407.

Chen, T., & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM Sigkdd international conference on knowledge discovery and data mining*, pp. 785–794.

Gomes, H. M., Barddal, J. P., Enembreck, F., & Bifet, A. (2017). A survey on ensemble learning for data stream classification. *ACM Computing Surveys (CSUR), 50*(2), 1–36.

Bifet, A., Gavaldá, R., Holmes, G., & Pfahringer, B. (2018). *Machine learning for data streams: With practical examples in MOA* (pp. 52–96). Massachusetts: The MIT Press. https://doi.org/10.7551/mitpress/10654.001.0001

Montiel, J., Mitchell, R., Frank, E., Pfahringer, B., Abdessalem, T., & Bifet, A. (2020). Adaptive xgboost for evolving data streams. In *2020 international joint conference on neural networks (IJCNN)*, pp. 1–8. IEEE.

Wang, K., Lu, J., Liu, A., Song, Y., Xiong, L., & Zhang, G. (2022). Elastic gradient boosting decision tree with adaptive iterations for concept drift adaptation. *Neurocomputing, 491*, 288–304.

Gomes, H. M., Bifet, A., Read, J., Barddal, J. P., Enembreck, F., Pfahringer, B., Holmes, G., & Abdessalem, T. (2017). Adaptive random forests for evolving data stream classification. *Machine Learning, 106*(9), 1469–1495.

Gomes, H.M., Read, J., & Bifet, A. (2019). Streaming random patches for evolving data stream classification. In 2019 IEEE *International conference on data mining* (ICDM), pp. 240–249. IEEE.

Bifet, A., & Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM international conference on data mining*, pp. 443–448. SIAM.

Oza, N.C., & Russell, S.J. (2001). Online bagging and boosting. In *International workshop on artificial intelligence and statistics*, pp. 229–236. PMLR.

Servedio, R. A. (2003). Smooth boosting and learning with malicious noise. *The Journal of Machine Learning Research, 4*, 633–648.

Chen, S.-T., Lin, H.-T., & Lu, C.-J. (2012). An online boosting algorithm with theoretical justifications. arXiv preprint arXiv:1206.6422.

Ikonomovska, E., Gama, J., & Džeroski, S. (2011). Learning model trees from evolving data streams. *Data Mining and Knowledge Discovery, 23*(1), 128–168.

Gouk, H., Pfahringer, B., & Frank, E. (2019). Stochastic gradient trees. In *Asian conference on machine learning*, pp. 1094–1109. PMLR.

Mouss, H., Mouss, D., Mouss, N., & Sefouhi, L. (2004). Test of page-hinckley, an approach for fault detection in an agro-alimentary production system. In *2004 5th Asian control conference (IEEE Cat. No. 04EX904)*, vol. 2, pp. 815–818. IEEE.

Gama, J., Medas, P., Castillo, G., & Rodrigues, P. (2004). Learning with drift detection. In *Brazilian symposium on artificial intelligence*, pp. 286–295. Springer.

Pébay, P., Terriberry, T. B., Kolla, H., & Bennett, J. (2016). Numerically stable, scalable formulas for parallel and online computation of higher-order multivariate central moments with arbitrary weights. *Computational Statistics, 31*(4), 1305–1325.

Schubert, E., & Gertz, M. (2018). Numerically stable parallel computation of (co-) variance. In *Proceedings of the 30th international conference on scientific and statistical database management*, pp. 1–12

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). Data mining: Practical machine learning tools and techniques. *The Morgan Kaufmann Series in Data Management Systems* (pp. 322–328). San Francisco: Elsevier.

Gunasekara, N., Gomes, H.M., Pfahringer, B., & Bifet, A. (2022). Online hyperparameter optimization for streaming neural networks. In *2022 international joint conference on neural networks (IJCNN)*, pp. 1–9. IEEE.

Pavlovski, M., Zhou, F., Stojkovic, I., Kocarev, L., & Obradovic, Z. (2017). Adaptive skip-train structured regression for temporal networks. In *machine learning and knowledge discovery in databases: European conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part II 10*, pp. 305–321. Springer.