# Highlights

**Elastic Gradient Boosting Decision Tree with Adaptive Iterations for Concept Drift Adaptation**

Kun Wang, Jie Lu, Anjin Liu, Yiliao Song, Li Xiong, Guangquan Zhang

- Adaptive iterations (AdIter) method helps GBDT model to build several ensembles with different adaptative iterations and select the best prediction results by majority voting. This method helps the model to adapt to different drift severities.
- A bound analysis of model under concept drift shows how the concept drift occurs caused by data distribution changes and how the concept drift severity influences the model error.
- Performance of adaptive iterations (AdIter) method on synthetic and real-world datasets shows its efficiency.

# Elastic Gradient Boosting Decision Tree with Adaptive Iterations for Concept Drift Adaptation

Kun Wang[a,b], Jie Lu[a,*], Anjin Liu[a], Yiliao Song[a], Li Xiong[b], Guangquan Zhang[a]

[a] *University of Technology Sydney, Broadway, Sydney, 2007, NSW, Australia*
[b] *Shanghai University, Shangda Road 99, Shanghai, 200444, Shanghai, China*

**Abstract**

As an excellent ensemble algorithm, Gradient Boosting Decision Tree (GBDT) has been tested extensively with static data. However, real-world applications often involve dynamic data streams, which suffer from concept drift problems where the data distribution changes overtime. The performance of GBDT model is degraded when applied to predict data streams with concept drift. Although incremental learning can help to alleviate such degrading, finding a perfect learning rate (i.e., the iteration in GBDT) that suits all time periods with all their different drift severity levels can be difficult. In this paper, we convert the issue of determining an optimal learning rate into the issue of choosing the best adaptive iterations when tuning GBDT. We theoretically prove that drift severity is closely related to the convergence rate of model. Accordingly, we propose a novel drift adaptation method, called adaptive iterations (AdIter), that automatically chooses the number of iterations for different drift severities to improve the prediction accuracy for data streams under concept drift. In a series of comprehensive tests with seven state-of-the-art drift adaptation methods on both synthetic and real-world data, AdIter yielded superior accuracy levels.

*Keywords:* Concept drift, Ensemble learning, Data stream, Gradient boosting

*Corresponding author

*Email addresses:* kun.wang-2@student.uts.edu.au (Kun Wang), jie.lu@uts.edu.au (Jie Lu), anjin.liu@uts.edu.au (Anjin Liu), yiliao.song@uts.edu.au (Yiliao Song), xiongli8@shu.edu.cn (Li Xiong), guangquan.zhangu@uts.edu.au (Guangquan Zhang)

## 1. Introduction

Concept drift describes changes of data distribution in data streams [1]. When a significant drift occurs, learning models will no longer be able to perform the task it was designed for with sufficient accuracy [2]. The practical demand for models that are robust to concept drift is very broad, spanning applications from general recommender systems [3] to passenger demand predictors [4]. Further, there are several different types of drift and within each type, drift can occur to different degrees [5].

Ensemble learning method is a popular choice for concept drift handling [6]. It describes model training schemes that combine several weak learners to form one powerful predictive model [6]. Ensemble models tend to be very sensitive to drift, which makes them a good general solution, but they have not been equipped with the functions to recognize the subtler nuances of drift. Therefore, what is needed is a more proactive method, where the ensemble learning model is given this recognition capability and the skills to adapt to the drift more appropriately.

Of the different ensemble learning strategies, the boosting methods are known for their excellent performance. Gradient boosting is one of the most popular boosting methods. With gradient methods, all weak learners are initially given equal weight, which, with gradient descent, can be thought of the learning rate in gradient descent, which is the "shrinkage" parameter and minor in magnitude [7]. Weak learners are trained by minimizing the loss function of the entire model.

However, despite the growing throng of real-world applications that generate fast-moving and fast-evolving data streams, few studies could be found in the literature discussing how gradient boosting be applied to deal with concept drift [8]. Recently, an online gradient boosting model is proposed aiming at drifting data streams [9], that successfully applied the commonly-used drift-adaptation strategies (retraining and tuning) into the gradient boosting. A critical issue with the approach, though, is that it only provides for a fixed number of iterations in the incremental learning process. Thus, concept drift of different severity is not considered.

This paper proposes adaptive iterations (AdIter) to enhance the gradient boosting method so that it can be compatible with different drift severity. AdIter works with most ensemble learning methods but, in this paper,

we chose to implement it in the context of gradient boosting decision tree (GBDT) as an example of a gradient boosting method. Notably, AdIter does not induce increase higher, sometimes even reduce, the runtime complexity of conventional GBDT and, sometimes, it can even decrease it.

The main contributions of this paper include:

- A detailed description of a strategy to adaptively select GBDT models for concept drift adaption, called elastic gradient boosting decision tree (eGBDT) method. We accompany the description with a runtime complexity analysis is given to verify its efficiency.
- A bound analysis of a GBDT model under concept drift that shows the influence of drift severity on model errors.
- The adaptive iterations (AdIter) method that generates a set of eGBDT models with different tuning configurations that can be used to control the drift recovery speed. As a result, the system can adaptively choose the best tuning iterations as new data becomes available.

The rest of this paper is organized as follows. Related work is discussed in Section 2. Section 3 sets out the preliminary, problem statement, basic definitions. Our proposed methods for concept drift adaptation are presented in Section 4. The experiments and results are outlined in Section 5, and the paper concludes in Section 6 with a brief summary of material presented and our intentions for future work.

## 2. Related Work

This section summarizes the basic methods based on GBDT for data stream learning in addition to giving an overview of existing concept drift detection and adaptation methods.

### *2.1. Gradient Boosting Decision Tree for Data Stream Learning*

Boosting is a popular ensemble algorithm that linearly combines high-performing ensemble weak learners when dealing with learning tasks [10, 11]. GBDT is a typical type of boosting framework, it was proposed by [10] that turns a set of weak learners into a strong one through ensemble.

For handling data stream, many boosting-based methods have been proposed in recent years, such as: OnlineBoosting [12, 13], OnlineRUSBoost

[13]. But many of them are based on AdaBoost rather than gradient boosting. Besides, the Streaming Gradient Boosting (SGM) algorithm works well on stochastic data streams [11]. DART, which applies dropout method to iteratively reduce overfitting [14], is also an excellent method. However, although they are gradient boosting-based methods, neither of them have discuss the scenario when concept drift occurs in data stream in detail, let alone consider different drift severities.

Concept drift in data stream will affect model learning, that is, the learning efficiency of weak learners, which will lead to an increase in the cumulative loss of the weak learners. Although some gradient boosting-based methods to deal with data stream have been proposed, there are deficiencies in the self-adaptation of the GBDT model and the processing of different drift severities.

### *2.2. Concept Drift Learning Methods*

In data stream mining, learning model needs to be updated to adapt to the uncertain data distribution. Many researches focus on handling data stream [15, 16]. Methods for dealing with concept drift is not just limited to adjusting or updating a single classifier but will integrate multiple classifiers for an optimized learning result. Currently, the ensemble learning method has been used for stream mining tasks [17, 18]. In addition, the data stream mining method also has a certain extension in data sample filtering [19], clustering [20, 21], deep learning [22, 23], and class imbalance handling [24, 25].

Generally, there are two main categories of approaches for handling concept drift: active approaches and passive approaches [26, 6, 27]. For the active approaches, model updates are determined by the results of the drift detection mechanism it contains [28, 29]. And, the commonly used drift detection method is error-based, like the early proposed Drift Detection Method (DDM) [30] and ADaptive WINdowing (ADWIN) [31]. PSCCD [32], which detects changes using an exchangeable test, is also a good detection method. Furthermore, new detection methods have been proposed recently, such as EVLC [33] and MSFS [34]. However, although these methods help the learning model find out drift and trigger update in time, few of them discuss the different drift severities. On the contrary, passive approaches let the model keep continuous learning while less care about whether drift can be detected. According to the type of the learning model, passive approaches can also be

divided into two types: single classifier models and ensemble classifier models [35]. The very-fast decision tree (VFDT) [36] is a popular single model for learning under concept drift. In addition, aiming to deal with changing data stream, CVFDT [37] algorithm has been proposed and it also achieved a good performance. Popular ensemble models proposed recently are dynamic weighted majority (DWM) [38], accuracy updated ensemble (AUE) [39, 40]. Furthermore, as an excellent method, incremental learning can help model remember and combine the knowledge of new incoming data [41]. By applying this method, another ensemble method for handling concept drift has been proposed, called Learn++.NSE [42]. Furthermore, based on the ensemble network, pENsemble [43], an ensemble method, tried to vote the local experts and prunes the one with a lower weight.

Sufficient reading indicates that most of the current drift learning methods are either retrain and update the ensemble model on new concepts or incremental learning model. However, few of them specifically discuss how to handle different drift severities, since a fixed tuning setting is not suitable for all concept drifts. Therefore, aiming to consider the drift severity while adaptation for further generalization is necessary. In other words, when there are different severities of concept drift in the same data stream, it is essential to have a reasonable fine-tuning. Our research is based on this problem and designs a drift learning method that can automatically select retraining and tuning strategies, and find adaptive iterations to deal with different drift severities.

## 3. Gradient Boosting Decision Tree with Concept Drift

This section begins with an investigation of the GBDT model under different conditions of concept drift. We have summarized the notations, as shown in Table 1. Definitions of our identified "gradient-drift learner" and "local-minimum" learner then follow.

### *3.1. Preliminary and Problem Statement*

Given a training set with size $n$, denoted as $D_{\text{train}} = \{x_i, y_i\}_{i=1}^{n}$, a GBDT model $F(x)$ is trained with $M$ weak learners, where each learner is an iteration. For $m = 1$ to $M$, the pseudo-residuals [10] are calculated as follows,

$$r_{im} = -\left[\frac{\partial \ell_{MSE}(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}, i = 1, \ldots, n. \tag{1}$$

Train a base learner $h_m(x)$ with the value of $r_{im}$ and $\{x_i\}_{i=1}^n$ and calculate the multiplier $\gamma_m$ [10] as

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} \underbrace{l(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))}_{\text{loss function}}. \tag{2}$$

The GBDT model is then updated by

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x). \tag{3}$$

And the accumulate loss of current model is calculated as

$$l_m = l(y, F_m(x)) = |y - F_m(x)|. \tag{4}$$

We use the absolute residual to observe the difference between model prediction and the true label. $l_m$ reflects the performance of model after adding learner $h_m(x)$. It is an indicator for us to observe the performance of weak learner $h_m(x)$ in the model. As the iterations proceed, the loss of the model will gradually decrease until it converges.

The next step would be to test this well-trained model on the testing set $D_{\text{test}} = \{x_i', y_i'\}_{i=1}^n$ with the same chunk size and assess the resulting predictions. However, if $D_{\text{test}}$ contains concept drift, the GBDT model may not perform well, the loss will not drop during training as it normally does, resulting in poor performance. For streaming data, instances come at each time point, so this learning process is based on prequential-train-test procedure. For example, we train a GBDT model on data coming at time 0 to get loss $l^0$ from the true label and prediction results, then test the model on data coming at time 1 and get loss $l^1$ from the true label and prediction results. If the data at time 1 occurs concept drift, loss $l^1$ will increase. We refer to this problem as *rising loss affected by concept drift* and state it as follows.

**Problem 1.** (Rising Loss Affected by Concept Drift) Suppose a GBDT model $F_M(x)$ with $M$ initial weak learners trained on data chunk $D_{\text{train}}$ at time $t$ gets a loss $l^t(y, F_M(x))$ from true label and prediction results, or is simply denoted as $l^t$. When a new data chunk $D_{\text{test}}$ comes at time $t+1$ occurs concept drift, the loss $l^{t+1}$ of the model after testing on it is increased, i.e.,

$$l^t < l^{t+1}.$$

where $l^{t+1} = l^{t+1}(y, F_M(x))$. It is necessary to update the model in time to help the loss converge normally again. Assuming the new updated model is

Table 1: Notations

| Notations | Description |
|---|---|
| $x$, $X$ | Data instances. |
| $y$, $Y$ | The label of data instances. |
| $n$ | The number of data instances, chunk size. |
| $i$ | The data index. |
| $d$ | The number of features. |
| $D$ | Data chunk. |
| $F(x)$ | A GBDT model. |
| $F$ | The prediction of GBDT model. |
| $F'(x)$ | The updated GBDT model. |
| $h(x)$, $h$ | A weak learner (iteration) of GBDT model. |
| $M$ | The number of initial weak learners in GBDT model. |
| $m$ | The index of weak learner (iteration) in GBDT model. |
| $\mathcal{M}$ | The number of remaining learners after pruning. |
| $r$ | Pseudo residual. |
| $\gamma$ | Multiplier. |
| $l$ | Accumulate loss of GBDT model. |
| $t$, $T$ | $t$ is time point, $T$ is time-step of a stream. |
| $L$ | The number of added iterations. |
| $k$ | The index of local-minimum weak learner. |
| $E$ | The number of eGBDT models. |
| $\mathbb{E}$ | The expectation. |
| $V$ | Prediction vector in AdIter. |
| $\lambda$ | Learning rate. |
| $p$ | The p-value in Friedman test. |
| $P$, $P'$, $Q$ | Data distribution. |
| $d(Q, P)$ | The distance of data following $Q$ and $P$ distributions. |
| $\phi$ | The probability density function. |
| $\alpha$ | The portions of data following a distribution. |
| $\sigma$ | The ratio that the number of retraining triggered over total time-steps. |
| $\delta$ | The probability setting in bound theory analysis. |
| $R$, $\hat{R}$ | $R$ is the generalization error, $\hat{R}$ is the empirical error. |

$F'(x)$ with a loss of $l^{t+1'}(y, F'(x))$, the model should satisfy the following condition,

$$l^{t+1'}(y, F'(x)) - l^t \leq 0 < l^{t+1} - l^t. \tag{5}$$

This inequality shows the ideal state of the model we expect. As previously

mentioned, the initial trained model at time $t$ performs poorly at time $t+1$ due to concept drift, leading to a rising loss, that is, $l^{t+1} - l^t > 0$. Therefore, to help the model adapt to drift well, we update the initial model as $F'(x)$ and hope the loss $l^{t+1'}(y, F'(x))$ can decrease normally as time $t$. The best case is $l^{t+1'}(y, F'(x)) - l^t \leq 0$. However, how to update the model?

Tuning is one of the wise choices to help the model recover from this rising loss caused by concept drift, because adding in more iterations should help the model converges. The difficulty is that drift can occur at different levels of severity and, without knowing the type and extent of the current drift, we cannot know how many iterations to add. Add too many, and we risk overfitting the model. Too few and we underfit. We refer to this problem as *appropriate iterations under concept drift*, and state it as follows.

**Problem 2.** (Appropriate Iterations under Concept Drift) Consider a GBDT model $F_M(x)$ with $M$ weak learners trained at time $t$ that is tested on a data chunk $D_{\text{test}}$ with concept drift at time $t+1$. To allow the loss resulting from the concept drift to reduce to pre-drift levels, $L$ iterations (learners) will be added to help tune the model. The problem now is how to determine an appropriate number of iterations $L$. This problem is formulated as

$$L = \arg\min_{L} l(y, F_{M+L}(x)), \tag{6}$$

where $F_{M+L}(x)$ is the model after tuning.

Given these two problems, our main research objective is to devise a drift adaptation method that allows a GBDT model to suitably adapt to concept drift of different severities.

### *3.2. Learners in Drift Affected GBDT Model*

When concept drift occurs, performance of some weak learners will decrease, resulting in an increasing loss that, under normal circumstances, would have fallen. (For easy reference, we will refer to these learners as "gradient-drift learners".) To maintain model performance, these gradient-drift learners need to be identified and removed in time.

The first step is identification. Given a GBDT model $F_M(x)$ with $M$ weak learners $h_m(x)$, $1 \leq m \leq M$, we define a gradient-drift learner if its loss, i.e, $\mathbb{E}[l_m] = \mathbb{E}[l(y, h_m(x))]$, is increasing, as per Definition 1.

**Definition 1** (Gradient-Drift Learner)**.** *A gradient-drift learner is a weak learner $h_m(x)$ satisfying*

$$(\mathbb{E}[l_m^t] \leq \mathbb{E}[l_{m-1}^t]) \wedge (\mathbb{E}[l_m^{t+1}] > \mathbb{E}[l_{m-1}^{t+1}]). \tag{7}$$

As loss decreases before drift occurs and increases after, gradient-drift learners follow a local-minimum learner, i.e., learners with the minimum $\mathbb{E}(l(y, h_m(x))$. Hence, if we can find this local-minimum learner, we know the learners that follow will be the gradient-drift learners, and we can prune them. Formally, the local-minimum learner is defined as follows:

**Definition 2** (The Local-Minimum Learner)**.** *In a GBDT model* $F_M(x) = \sum_{m=1}^{M} h_m(x)$ *with* $M$ *weak learners,* $m \in [1, M]$*, the local-minimum learner is a weak learner* $h_m(x)$ *with the minimum* $\mathbb{E}(l(y, F_m(x))) \in [0, 1]$*, denoted as*

$$h_m(x) = \arg\min_m \mathbb{E}(l(y, F_m(x))), \tag{8}$$

*where* $F_m(x)$ *is the model at the* $m$*-th iteration,* $F_m(x) = F_{m-1}(x) + h_m(x)$*.*

Notably, the location, namely $m$, of the local-minimum learner also reflects the severity of concept drift. The smaller the value of $m$, the more gradient-drift learners there will be that follow, and the more severe the drift, vice versa.

With the drift severity identified, the next step is to help the model adapt. Strategies for accomplishing this are outlined in the next section, we choose adaptation strategies to handle gradient-drift learners, and discuss how to deal with different drift severities.

## 4. Elastic Gradient Boosting Decision Tree with Adaptive Iterations for Concept Drift Adaptation

In this section, we provide a detailed introduction of how GBDT model selects adaptation strategies, retraining and tuning, to response to concept drift. We then set out our adaptive iteration method, AdIter, to help the model deal with varying drift severities.

### *4.1. Selecting an Appropriate Adaptation Strategy (eGBDT)*

Definition 1 and 2 state that, to reduce the impact of concept drift so as to maintain model performance, we should prune the gradient-drift learners. To solve Problem 1, we proposed elastic gradient boosting decision tree (eGBDT) method [9]. The eGBDT can find an optimal location of pruning by a defined *local-minimum learner* and select an adaptive strategies between retraining and tuning after pruning. Next, we briefly introduce eGBDT.

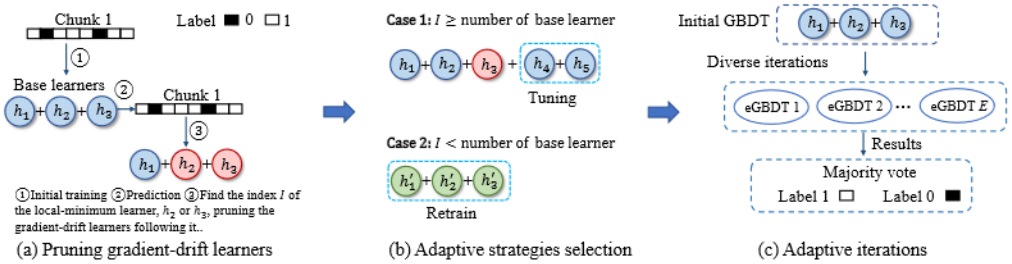



Figure 1: (a) The process for identifying the local-minimum learner. The learners that follow will be the gradient-drift learners and need to be pruned. Here, we assume $h_2$ or $h_3$ is the local-minimum learner. (b) Selecting the right adaptation strategy. If $h_3$ is the local-minimum learner, new iterations (learners) $h_4$ and $h_5$ are added incrementally. If $h_2$ is the local-minimum learner, a new GBDT model needs to be retrained. (c) The AdIter method. Dealing with drift of varying severities is a matter of tuning with the appropriate number of iterations. The final predictions are made via majority voting (Eq. (10)).

Consider two consecutive data chunks as $D_1 = \{x_i, y_i\}_{i=1}^n$, $D_2 = \{x_i', y_i'\}_{i=1}^n$, and a GBDT model $F_M(x) = \sum_{m=1}^M h_m(x)$ trained with data chunk $D_1$ and tested on data chunk $D_2$. Given the local-minimum learner $h_k(x)$, the GBDT model after pruning is $F_k(x) = \sum_{m=1}^k h_m(x)$. From here, there are two possible adaptation strategies to adapt $F_k(x)$: retraining or tuning. The following cases illustrate how to choose between the two strategies.

**Case 1:** $k \geq M$ denoting that no learners need to be pruned. However, this model may be under-fitting. Tuning could help to further decrease the loss of model. The GBDT model after tuning over $L$ iterations is $F_{k+L}$.

**Case 2:** $1 \leq k < M$ denoting the drift is too great for the initially trained model to combat. Therefore, we retrain a new model $F_M(x')$.

### *4.2. Adaptive Iterations (AdIter) for Varying Drift Severities Adaptation*

Section 4.1 shows how Problem 1 is solved by eGBDT, we now shift to solving Problem 2, i.e., deal with concept drift of varying severities. Inspired by the work in [44], a bound analysis of a GBDT model is provided in Appendix A to show how drift severity affects model error. The AdIter method follows from this analysis.

The main idea of AdIter is to select the most precise from many ensemble models rather than relying on only one to obtain the best result. Although eGBDT itself is already an ensemble of weak learners, it is still unable to handle concept drift with different severities as eGBDT uses a fixed iteration

**Algorithm 1** Adaptive Iterations (AdIter) Method

**Input**:
1) Stream data $D$

**Parameter**:
1) A set of eGBDT Config, $\{\text{eGBDT}^i_{\text{parm}}\}_{1,\dots,E}$.
2) Initial training chunk size, $\text{chunk}_{\text{ini}}$.
3) Sliding chunk size, $\text{chunk}_{\text{slide}}$.
4) The number of initially trained weak learner $M$.

**Output**:
1) Prediction result $\{\hat{Y}_{\text{chunk}_{\text{slide}}}\}$;

**for** $\text{eGBDT}^i_{\text{parm}}$ in $\{\text{GBDT}^i_{\text{parm}}\}_{1,\dots,E}$ **do**
  build eGBDT $F_i(X)$ on $D_{\text{chunk}_{\text{ini}}}$ # By Eq.(9)
**end for**
**while** $D$ has next chunk $D_{\text{chunk}_{\text{slide}}}$ **do**
  **for** $i = 1 : E$ **do**
    save prediction vector $V_i = F_i(D_{\text{chunk}_{\text{slide}}})$
    pruning eGBDT $F_i(X)$ on $D_{\text{chunk}_{\text{slide}}}$ and remain $\mathcal{M}$ learners.
    **if** $\mathcal{M} < M$ **then**
      retrain eGBDT $F_i(X)$ on $D_{\text{chunk}_{\text{slide}}}$
    **else**
      tuning eGBDT $F_i(X)$ on $D_{\text{chunk}_{\text{slide}}}$
    **end if**
  **end for**
  majority vote $\hat{Y}_{\text{chunk}_{\text{slide}}} = \text{MajorityVote}(\hat{Y}_i)$ # By Eq.(10)
**end while**
**return** prediction results of all chunks $\{\hat{Y}_{\text{chunk}_{\text{slide}}}\}$

for tuning.

Therefore, we propose AdIter, to broaden the tuning iterations setting of GBDT by majority voting. Algorithm 1 outlines the AdIter procedure. In more detail, the first step is to train several eGBDT models with different settings of tuning iterations. Given data sample $\{X, Y\}$, where $Y \in \{0, 1\}$. When the tuning iterations are $\{L_1, L_2, \cdots, L_E\}$, we will have $E$ numbers of eGBDT models, expressed as

$$F = \{F_1(X), F_2(X), \cdots, F_E(X)\}. \tag{9}$$

Denoting the outputs of $F_i(X)$ as $\hat{Y}_i$, we use the majority voting method to

compute final prediction of label $\hat{Y}$ by

$$\hat{Y} = \begin{cases} 0, & \text{if } \sum_i I_{\hat{Y}_i=0} \geq \sum_i I_{\hat{Y}_i=1} \\ 1, & \text{else} \end{cases}. \tag{10}$$

where $I$ is the characteristic function.

Clearly, we choose mode of all the predictions of $F_i(X)$ as the final prediction result. For example, for a binary classification task, assume we have $E = 3$ eGBDT models with tuning iterations $\{L_1, L_2, L_3\}$, and their prediction vector (votes) are $\hat{Y}_1 = 1, \hat{Y}_2 = 1, \hat{Y}_3 = 0$, then the final prediction is 1. If the ground truth is 1, we consider that $L_1$ and $L_2$ are appropriate iterations, but $L_3$ is not. The process of this AdIter method is shown in Figure 1.

It is worth noting that the AdIter is designed for enhancing the gradient boosting method so that it can be compatible with different drift severities. A fixed number of incrementally added iterations may not work well on data with different drift severities. Therefore, we first prune the gradient-drift learners. This pruning process is based on the loss of weak learner. Then, we aim to find the most appropriate adaptive iterations by voting to help the model become more robust on data with different drift severities.

### *4.3. Runtime Complexity Analysis*

In data stream learning, the runtime complexity is a critical evaluation criterion because of the real-time requirement of the application. In this section, we discuss the runtime complexity of our proposed algorithm. In general, the runtime complexity of eGBDT and AdIter methods depend on the frequency of tuning and retrain process.

For simplicity, we denote the runtime complexity of a weak learner as $\mathcal{O}(h)$, and for a base decision tree, the complexity is $\mathcal{O}(h) = \mathcal{O}(dn \log(n))$, where the $d$ is the number of features, $n$ is the training size. Since GBDT trains a new learner based on the residual of the sum of previous learners, for $M$ number of learns, the total runtime complexity is the linear sum of all learners, that is, $\mathcal{O}\left(\sum_{i=1}^{M} h\right) = \mathcal{O}(Mh)$. For streaming data, a new chunk of data is available at each time point (we use $n'$ to denote the chunk size), and $L$ new iterations (learners) are incrementally built for tuning. For $T$ time points, the overall complexity becomes $\mathcal{O}\left(\sum_{i=1}^{M+TL} h\right)$, namely $\mathcal{O}(Mh + TLh')$, where $h'$ has runtime complexity $\mathcal{O}(dn' \log(n'))$.

$M$ is the number of learners when initializing a GBDT; $T$ is the time-step of a stream; $L$ is the number of iterations appended at each time-step; $h$ is the complexity that training a learner on the initial training set with size $n$; $h'$ denotes training a learner on a new arrived data chunk with size $n'$; $h''$ denotes training on all available data with size $(n + Tn')$; $\delta$ denotes the ratio that the number of retraining triggered over the total number of time steps.

To build the GBDT on the same number of training samples $(n + Tn')$ with the same number of learners $(M + TL)$, we have the complexity of conventional GBDT as $\mathcal{O}\left((M + TL)\, h''\right)$, where each learner $h''$ is trained on the entire training set and has complexity

$$\begin{aligned} &\mathcal{O}\left((M + TL)\, d\, (n + Tn') \log\left(n + Tn'\right)\right) = \\ &\mathcal{O}\left(\underbrace{Md\,(n + Tn') \log\left(n + Tn'\right)}_{item_1} + \underbrace{TLd\,(n + Tn') \log\left(n + Tn'\right)}_{item_2}\right), \end{aligned} \tag{11}$$

and for tuning GBDT, the complexity is

$$\mathcal{O}\left(Mh + TLh'\right) = \mathcal{O}\left(\underbrace{Mdn \log\left(n\right)}_{item'_1} + \underbrace{TLdn' \log\left(n'\right)}_{item'_2}\right). \tag{12}$$

Apparently, $item'_1 < item_1$ and $item'_2 < item_2$, so we conclude that tuning GBDT has lower runtime complexity than retrain GBDT for data stream learning. This is because tuning GBDT only trains a new learner on a small subset of the data, while retrain GBDT trains each learner on the entire dataset.

For eGBDT with strategies selection, we denote the ratio that the number of retraining processes triggered by concept drift over the total number of time-steps as $\sigma$, and the retraining complexity as $\mathcal{O}\left(Mh'\right)$. Then, for $\sigma T$ times of retraining, the complexity will be $\mathcal{O}\left(\sigma TMh'\right)$, and for the rest tuning process, the complexity will be $\mathcal{O}\left((1 - \sigma)\, TLh'\right)$. Tree pruning is a linear search that finds the minimum residual of all the learners. Since the residuals are already calculated when making the predictions on new data, the searching complexity is $\mathcal{O}\left(1\right)$, and for $T$ time step, the pruning complexity is $\mathcal{O}\left(T\right)$. Therefore, the overall runtime complexity is

$$\begin{aligned} &\mathcal{O}\left(Mh + \sigma TMh' + (1 - \sigma)\, TLh' + T\right) \\ &= \mathcal{O}\left(\underbrace{Mh}_{item1} + \underbrace{\sigma(M - L)Th'}_{item2} + \underbrace{TLh'}_{item3}\right), \end{aligned} \tag{13}$$

Table 2: Runtime Complexity Analysis

| Algorithms | Runtime complexity |
|---|---|
| GBDT | $\mathcal{O}\left(\left(M+TL\right)h''\right)$ |
| Appending new trees at each time step | $\mathcal{O}\left(Lh'\right)$ |
| Tuning GBDT | $\mathcal{O}\left(Mh+TLh'\right)$ |
| Pruning at each time step | $\mathcal{O}\left(1\right)$ |
| Retraining GBDT | $\mathcal{O}\left(Mh'\right)$ |
| eGBDT | $\mathcal{O}\left(Mh+\sigma\left(M-L\right)Th'+TLh'\right)$ |
| AdIter | $\mathcal{O}\left(\sum_{i=1}^{E}\left(Mh+\sigma(M-L_i)Th'+TL_ih'\right)\right)$ |

where $0 \leq \sigma \leq 1$, $item_1$ is the initialization and $item_2$ with $item_3$ comprise the drift learning complexity. For the best case, where no concept drift occurs, namely $\sigma = 0$, eGBDT will continuous tuning and have a smaller runtime complexity. For the worst case, which has concept drift occurred and has GBDT retrained every time, the time complexity will be $\mathcal{O}\left(Mh+TMh'\right)$. Furthermore, since the AdIter method contains $E$ numbers of eGBDT models with different tuning iterations $\{L_1, L_2, \cdots, L_E\}$ in ensemble, so its runtime complexity can be expressed by

$$\mathcal{O}_{\text{AdIter}} = \mathcal{O}\left(\sum_{i=1}^{E}\left(Mh+\sigma(M-L_i)Th'+TL_ih'\right)\right). \tag{14}$$

Table 2 summarizes the runtime complexity of the proposed method for data stream learning.

## 5. Experiments

This section gives experiment settings, synthetic and real-world datasets description, and experiment results discussion.

### *5.1. Experiment Settings*

The experiment is based on prequential-train-test procedure [45]. Firstly, we compare the proposed AdIter method with four basic methods of using GBDT to handle concept drift, with their efficiencies being our main concern.

**Baseline**: initially train a GBDT model on the training set, then testing it on the rest of data without any model update.

**Retrain**: gives the pre-trained GBDT model after testing, and then retrain a new GBDT model for the next data chunk.

**Tuning**: incrementally adding fixed new iterations on new incoming data chunks based on the initially trained GBDT model.

**eGBDT**: adaptively selects retraining and tuning strategies to help GBDT model adapt to concept drift.

In addition, we also compare the performance of our AdIter method with seven state-of-the-art ensemble learning methods.

**ARF** [46]: is an classical ensemble-based concept drift learning method, which trains the Random Forest by using the bagging and embedding a ADWIN drift detector to deal with concept drift.

**Learn++.NSE** [42]: is an ensemble method for handling concept drift. By using the bagging, several learners are trained and combined by dynamically weighted majority voting.

**LeverageBag** [47]: aims to increase the accuracy and diversity. By combining the bagging principle and randomizing the input and output of learners, the adaptability of the model is improved.

**OnlineBoosting** [13]: transfers the conventional AdaBoost model into the online version for data stream learning. An ADWIN detector has been embedded to deal with concept drift.

**OnlineRUSBoost** [13]: is a boosting-based data stream learning method. It uses sampling technique iteratively to enhance the learning performance and also use ADWIN detector to handle concept drift.

**OnlineBagging** [12]: is a bagging-based data stream learning method, it uses the expected accuracy of learners on the testing data to weight them, this boosts the model robustness.

**Streaming Random Patches (SRP)** [48]: is an ensemble learning method which uses the basic idea of bagging and simulates the random subspace to deal with data streams.

We use the default parameter setting of the compared methods, which is a common practice. It has certain meaning since these methods are tested and packaged in Scikit-Multiflow. We have also summarized the parameters of the state-of-the-art methods, as shown in Table 3. The parameters for eGBDT were the same, i.e., $\text{GBDT}_{\text{parm}} = (M = 200, \text{Depth} = 4, \text{SampleRate} = 0.8, \lambda = 0.01)$, which are default parameters and will not change in our experiment. For tuning, we fixed add $L = 25$ iterations each time. For our AdIter method, we aim to built $E = 5$ numbers of eGBDT models, and the number of iterations of each of them were set as $\{L_1 = 25, L_2 = 50, L_3 =$

$75, L_4 = 100, L_5 = 125\}$. We use uniform parameters for all datasets without adjustment to verify the efficiency of our method. And we also conduct a parameter sensitivity analysis to show the robustness of our method.

Our AdIter method is implemented by Python 3.7 code, and we use the decision tree model in the sklearn package as the weak learner. The computational environment is: Red Hat Enterprise Linux Workstation release 7.9 (Maipo), Intel(R) Xeon(R) Gold 6238R CPU @ 2.20GHz. These ensemble learning methods selected for comparison are from the Scikit-Multiflow meta module [49]. To be fair, we use all these packages with their default parameter settings. And all methods have the same chunk size on the same bench mark datasets.

Table 3: Parameters of State-of-the-Art Methods

| Method | Weak learner | Number of learners |
|---|---|---|
| ARF | Hoeffding Tree Classifier | 10 |
| Learn++.NSE | Decision Tree Classifier | 10 |
| LeverageBag | KNN Classifier | 15 |
| OnlineBoosting | KNN ADWIN Classifier | 10 |
| OnlineRUSBoost | KNN ADWIN Classifier | 10 |
| OnlineBagging | KNN ADWIN Classifier | 10 |
| SRP | Hoeffding Tree Classifier | 10[1] |

[1] To improve the running efficiency, we manually set the number of learners of SRP method as 10, keeping the same as other methods.

### *5.2. Datasets*

The AdIter method mainly focuses on the binary classification task. We use the decision tree as the weak learner of the GBDT model. Multi-class classification can be extended by executing several binary classification tasks and getting the prediction results by softmax method. We tested the method on 12 binary classes datasets, including synthetic datasets and real-world datasets. The detailed information of these datasets are summarized in Table 4, 5, where the ratio indicates the class ratio. The synthetic datasets were:

**SEAa** [50]: contains 3 attributes and 2 classes. Data samples in each attribute are numberic between 0 and 10. There are 10,000 samples in this dataset that contains 3 times concept drift, 2,500 examples each, with different thresholds for the concept function.

Table 4: Datasets Statistics

| Synthetic | Sample | Feature | Class | Ratio | Chunk size | Drift type |
|---|---|---|---|---|---|---|
| SEAa | 10,000 | 3 | 2 | 1:1 | 100 | abrupt |
| RTG | 10,000 | 10 | 2 | 1:1 | 100 | no |
| RBF | 10,000 | 10 | 2 | 1:1 | 100 | incremental |
| RBFr | 10,000 | 10 | 2 | 1:1 | 100 | incremental |
| HYP | 10,000 | 10 | 2 | 1:1 | 100 | incremental |
| AGRa | 10,000 | 9 | 2 | 1:1 | 100 | abrupt |

Table 5: Real-World Datasets

| Real-world | Sample | Feature | Class | Ratio | Chunk size |
|---|---|---|---|---|---|
| Electricity | 45,312 | 8 | 2 | 0.73:1 | 100 |
| Weather | 25,626 | 8 | 2 | 0.49:1 | 365 |
| Spam | 9,324 | 500 | 2 | 0.34:1 | 100 |
| Usenet1 | 1,500 | 99 | 2 | 0.87:1 | 40 |
| Usenet2 | 1,500 | 99 | 2 | 0.50:1 | 40 |
| Airline | 539,383 | 7 | 2 | 0.80:1 | 100 |

**RTG** [51]: is generated by Random Tree Generator. By building a decision tree, the split nodes come from randomly selected attributes, and different classes are assigned to leaves. This dataset has 10,000 samples and 10 sample attributes.

**RBF, RBFr**: are generated using the Radial Basis Function (RBF) generator with 10 attributes. The parameter for the number of centroids is adjusted to generate different concept drift scenarios in the data. For RBF we generate 50 centroids and all of them are drifting with a margin equal to 0.0001, while RBF regional (RBFr) only has 10/50 drifting centroids with a margin equal to 0.01.

**HYP** [50]: is generated by a hyperplane generator, which can simulate the incremental drift. This dataset has 10,000 samples, 10 sample attributes, and 2 classes.

**AGRa** [52]: simulates the data stream with abrupt drift by using the the AGRAWAL generator. This dataset contains, 10,000 samples, 6 nominal and 3 continuous attributes, and 2 classes.

**Electricity**: describes the Australian New South Wales Electricity Market. It has 2 classes, which indicates the electricity price changes up or down

over time. There are totally 45,312 instances, in our experiment, we initially set the chunk size to 100.

**Weather**: was compiled by the US NOAA [1]. It contains 25,626 instances, 8 attributes, 2 classes. There are about 33% positive (rain) class, and 67% negative (no rain) class. Since this dataset has an observation period of one year, the chunk size is set to 365.

**Spam** [53]: is the dataset recorded by SpamAssasin which aims to deal with the spam emails. There total 9,324 instances and 500 attributes in this dataset. We initially set the chunk size as 100.

**Usenet1, Usenet2** [53]: are two real-world datasets with simulated interests drift. Each entity is a piece of news drawn from Twenty Newsgroups Dataset. Each consists of 1,500 samples with 99 attributes. Since this dataset has a smaller size, so we initially set the chunk size as 40.

**Airline**: is a prediction set of whether a given flight will be delayed from its scheduled departure. There are 539, 383 records with 7 attributes in this dataset. The class labels are: delayed or not delayed. The chunk size is initially set to 100.

The datasets and the python source code of AdIter method are available online[2].

### *5.3. Experiment and Discussion*

To assess AdIter, we ran the method on six synthetic datasets and six real-world datasets and compared the results with the state-of-the art benchmark methods. Details of the experiments follow.

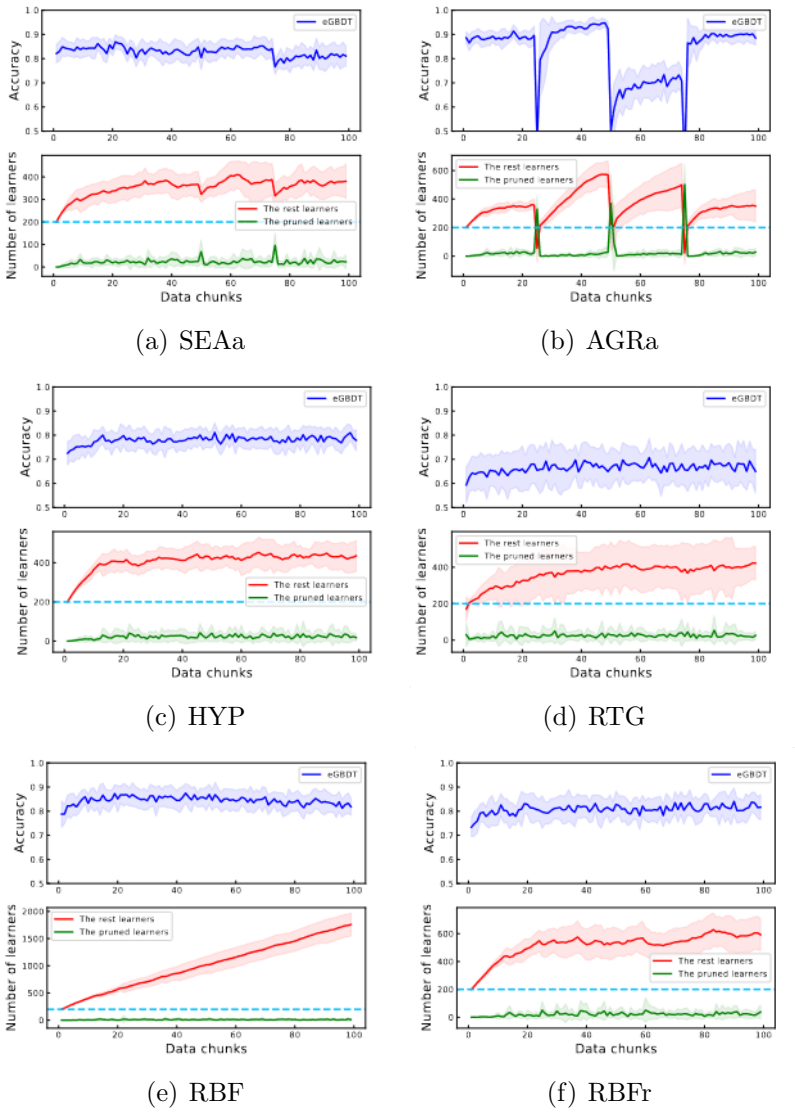
#### *5.3.1. Experiment 1: Retrain or tuning: Was the right strategy selected?*

In this experiment, we tested the eGBDT model on the synthetic datasets and simply ran our AdIter method. Figure 2 shows the number of learners in pruned and resulting accuracy for each dataset. The trend lines shown paint a good picture of eGBDT's underlying processes. abrupt drops in accuracy, for example, indicate a drift.

The gradient-drift learners are then pruned, which triggers the selection of an adaptation strategy – either retraining or tuning. During the learning process, the number of learners either increases or decreases depending on

[1] https://www1.ncdc.noaa.gov/pub/data/gsod/

[2] https://github.com/kunkun111/AdIter




(a) SEAa (b) AGRa

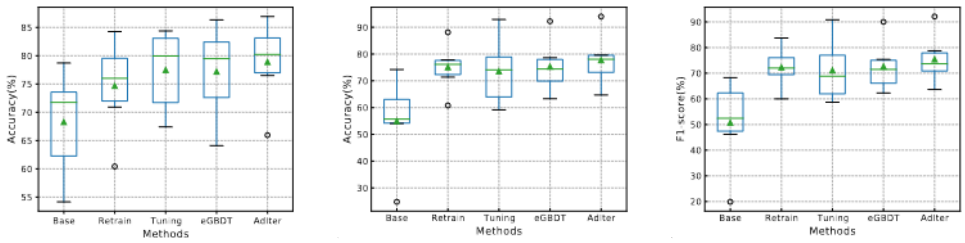
(c) HYP (d) RTG

(e) RBF (f) RBFr

Figure 2: A plot showing: the chunk accuracy, number of gradient-drift learners pruned, and count of the remaining learners. We can see that accuracy dropped and some learners were pruned to result in fewer learners overall. The SEAa and AGR datasets contain simulated abrupt drift at the 25th, 50th, and 75th chunks. Because the number of the learners remaining after the prune was less than $M = 200$, the retraining strategy was triggered. By contrast, the HYP, RTG, RBF, and RBFr datasets only contain a few small drifts. These were able to be handled by tuning.

Table 6: Accuracy of Six Synthetic Datasets (%)

| Methods | SEAa | RTG | RBF | RBFr | HYP | AGR | AvgRank | AvgScore |
|---|---|---|---|---|---|---|---|---|
| Base | 78.72±2.88(8) | 59.58±5.47(9) | 54.16±3.16(12) | 73.72±3.08(11) | 73.14±14.97(8) | 70.36±2.45(8) | 9.33 | 68.28 |
| Retrain | 80.47±0.49(7) | 60.42±5.37(8) | 76.75±1.36(6) | 75.25±1.84(9) | 84.27±3.32(5) | 70.91±0.94(6) | 6.83 | 74.67 |
| Tuning | 83.40±0.52(5) | 67.42±5.49(2) | 69.72±1.61(11) | 82.16±1.64(5) | 84.38±5.91(4) | 77.74±0.64(3) | 5 | 77.47 |
| eGBDT | 82.87±0.96(6) | 64.10±6.63(5) | 70.86±1.53(9) | 81.06±2.02(8) | 86.34±3.32(2) | 77.90±1.15(2) | 5.33 | 77.18 |
| AdIter | 83.51±0.86(4) | 65.96±6.33(3) | 76.50±1.47(7) | 82.03±1.76(6) | **86.93±2.56(1)** | **78.42±1.13(1)** | **3.66** | **78.89** |
| ARF | 84.72±0.36(3) | **67.78±0.5(1)** | 80.27±1.57(5) | 83.68±1.52(4) | 84.56±5.97(3) | 77.62±1.09(4) | **3.33** | **79.77** |
| Learn++.NSE | 76.47±0.7(12) | 61.16±6.6(7) | 71.69±1.17(8) | 73.58±1.74(12) | 82.11±2.92(7) | 65.43±1.26(12) | 9.66 | 71.74 |
| LeverageBag | 77.75±0.61(10) | 57.61±2.53(10) | 90.38±1.03(2) | 86.52±1.35(2) | 56.84±6.61(11) | 67.32±0.51(11) | 7.66 | 72.73 |
| OnlineBoosting | 78.66±0.74(9) | 57.44±2.82(12) | 84.42±1.14(4) | 81.28±1.79(7) | 57.11±6.79(9) | 70.19±0.55(9) | 8.33 | 71.51 |
| OnlineRusBoost | 77.64±0.78(11) | 57.58±2.74(11) | 87.32±1.22(3) | 83.90±1.35(3) | 57.10±6.6(10) | 69.24±0.48(10) | 8 | 72.13 |
| OnlineBagging | **84.96±0.44(1)** | 64.27±3.49(4) | **90.60±0.98(1)** | **87.56±1.32(1)** | 56.41±6.61(12) | 76.47±0.39(5) | 4 | 76.71 |
| SRP | 84.78±0.39(2) | 62.89±3.82(6) | 70.62±2.04(10) | 74.96±1.96(10) | 82.15±7.87(6) | 70.83±1.86(7) | 6.83 | 74.37 |




(a) Accuracy on synthetic datasets (b) Accuracy on real-world datasets (c) F1-score on real-world datasets

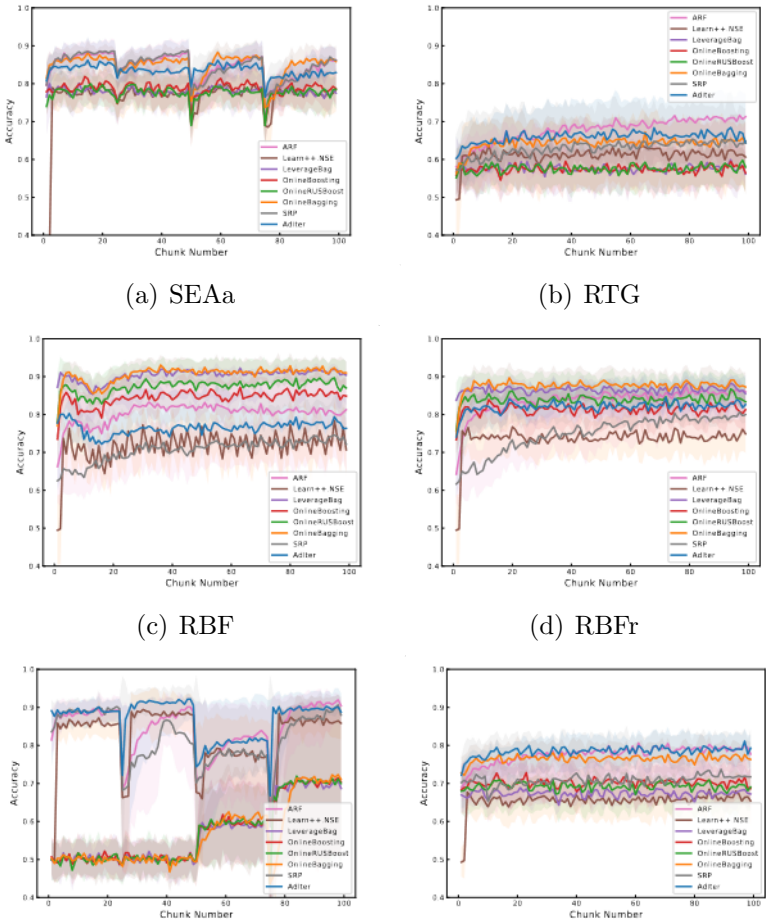
Figure 3: A box plot of the results of AdIter and four GBDT-based methods on synthetic datasets and real-world datasets. The AdIter method got higher average accuracy and F1-score among five methods.

which adaptation strategy is executed. After pruning and continuous learning, model accuracy begins to improve again. Take the SEAa and AGRa datasets – the abrupt drop in accuracy marks the onset of concept drift. A number of learners are quickly pruned when accuracy decreases. Figure 2 (b) shows that, because the number of learners remaining after the prune fell below the initial amount of $M = 200$, the decision was made to train a new model.

From this experiment, we can find the drift severity by observing the number of remaining learners after pruning. Also, as shown in Figure 2, the blue dashed line represents the initial number of learners in the model, and the red line represents the number of learners remaining after pruning. When the number of learners in the model is less than the initial number, this means that the model has not adapted to the new data, and we believe that a serious concept drift has occurred at this moment, which is obvious on the AGRa datasets. Otherwise, it means that the drift is relatively slight. Then, we simply ran our AdIter method and recorded the true and false rates of the results, as shown in Table 7.

*5.3.2. Experiment 2: How accurate is the AdIter method with synthetic drift?*

In this experiment, we compared AdIter's accuracy with the four GBDT-based baselines and the six synthetic datasets. We ran each method on each dataset 30 times with different random seeds. Table 6 records the mean accuracy and standard deviations. The average runtime of all the methods, without optimizing the code, was $2s$ (Baseline), $119s$ (Retrain), $31s$ (Tuning), $23s$ (eGBDT), and $264s$ (AdIter) as measured in WallTime.




(a) SEAa (b) RTG

(c) RBF (d) RBFr

(e) AGRa (f) HYP

Figure 4: Average chunk accuracy with the synthetic datasets. On the AGRa and HYP datasets, AdIter adapted well to both abrupt drift and incremental drift.

Table 7: True Positive Rate and False Positive Rate

| Datasets | True positive rate | False positive rate |
|---|---|---|
| SEAa | 90.42±1.1 | 27.64±2.1 |
| RTG | 64.60±14.5 | 37.54±15.9 |
| RBF | 75.69±5.8 | 24.65±6.5 |
| RBFr | 81.06±5.4 | 18.53±5.2 |
| AGRa | 87.00±3.8 | 13.13±2.1 |
| HYP | 78.09±1.7 | 21.23±1.8 |
| Electricity | 84.12±0.3 | 27.02±0.4 |
| Weather | 90.38±0.08 | 42.14±0.2 |
| Spam | 95.34±0.2 | 9.9±0.6 |
| Usenet1 | 74.09±0.6 | 29.91±0.4 |
| Usenet2 | 46.25±0.9 | 8.0±0.4 |
| Airline | 53.74±0.2 | 26.49±0.1 |

Obviousy, eGBDT saves more running time than tuning. It should be noted that the AdIter method is an ensemble of different tuning iterations of 5 eGBDT models, which consumes running time. However, if it is an ensemble of more tuning models, it will consume more time. Of these five methods, AdIter had the average ranking of 3.66 and accuracy of 78.89%, as shown in Figure 3(a). Therefore, our AdIter method has advantages in efficiency and performance.

The performance of AdIter is different for different drift types. In dealing with abrupt drift which occurs in the AGRa dataset, the efficiency of AdIter (78.42%) can be seen in comparison with the other methods. For incremental drift, AdIter gets the highest accuracy on HYP (86.93%) dataset. But for RBFr (82.03%) dataset, our method does not perform well compared to ARF and OnlineBagging, and needs further improvement. This is also reflected in the chunk accuracy comparison shown in Figure 4. The method performs relatively not well on synthetic data, mainly because the drift pattern on the data set is single. Although AdIter did not perform quite as well on the other datasets, it was still more accurate than most baselines.

*5.3.3. Experiment 3: How accurate is AdIter with real-world drift?*

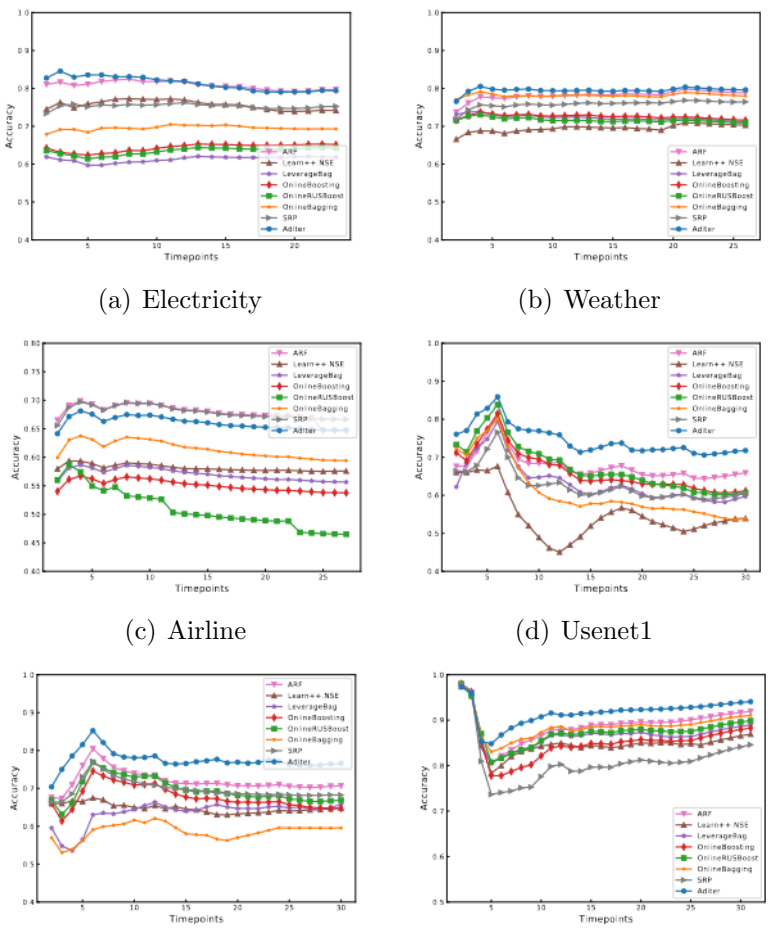
In this assessment, we used the six real-world datasets and evaluated the results in terms of accuracy, average macro F1-score, and MCC score. The experiments were implemented in the sklearn package, and the results

Table 8: Accuracy of Six Real-World Datasets (%)

| Methods | Electricity | Weather | Airline | Usenet1 | Usenet2 | Spam | AvgRank | AvgScore |
|---|---|---|---|---|---|---|---|---|
| Base | 65.16±0.07(10) | 74.20±0.17(8) | 56.76±0.42(10) | 54.06±1.07(12) | 24.79±0.0(12) | 54.68±0.13(12) | 10.66 | 54.94 |
| Retrain | 77.82±0.08(4) | 77.06±0.07(6) | 71.38±0.15(2) | 75.26±0.22(2) | 88.14±0.1(9) | 60.80±0.02(5) | 4.66 | 75.07 |
| Tuning | 78.13±0.2(3) | 79.15±0.1(2) | 61.99±0.5(5) | 70.03±0.45(5) | 92.93±0.22(2) | 59.16±0.3(7) | 4 | 73.56 |
| eGBDT | 76.01±0.37(5) | 78.66±0.17(4) | 68.90±0.58(3) | 72.85±0.71(3) | 92.26±0.47(3) | 63.33±0.11(4) | 3.66 | 75.33 |
| AdIter | 79.38±0.12(2) | **79.61±0.05(1)** | **71.97±0.38(1)** | **76.71±0.29(1)** | **94.02±0.21(1)** | 64.70±0.05(3) | **1.5** | **77.73** |
| ARF | **79.90±0.18(1)** | 78.79±0.26(3) | 66.06±1.05(4) | 70.73±0.58(4) | 91.76±0.25(4) | **66.81±0.07(1)** | 2.83 | 75.67 |
| Learn++.NSE | 74.34±0.34(7) | 70.23±0.1(12) | 53.83±0.44(11) | 64.97±0.35(9) | 86.88±0.75(10) | 57.58±0.03(8) | 9.5 | 67.97 |
| LeverageBag | 61.79±0.11(12) | 71.06±0.06(10) | 59.88±0.43(9) | 65.72±0.54(8) | 88.84±0.1(7) | 55.59±0.0(11) | 9.5 | 67.14 |
| OnlineBoosting | 65.25±0.22(9) | 71.56±0.28(9) | 61.15±1.3(6) | 64.71±1.78(10) | 88.17±0.43(8) | 55.70±0.6(10) | 8.66 | 67.75 |
| OnlineRusBoost | 64.34±0.25(11) | 70.85±0.24(11) | 60.62±1.49(7) | 66.89±1.18(7) | 89.75±0.3(6) | 56.20±1.12(9) | 8.5 | 68.10 |
| OnlineBagging | 69.55±0.09(8) | 77.88±0.11(5) | 53.78±1.24(12) | 59.55±1.0(11) | 90.88±0.11(5) | 59.32±0.03(6) | 7.83 | 68.49 |
| SRP | 75.31±0.81(6) | 76.46±0.85(7) | 60.51±1.94(8) | 68.28±0.76(6) | 84.56±0.57(11) | 66.56±0.07(2) | 6.66 | 71.94 |

Table 9: Average Macro F1-score of Six Real-World Datasets (%)

| Methods | Electricity | Weather | Airline | Usenet1 | Usenet2 | Spam | AvgRank | AvgScore |
|---|---|---|---|---|---|---|---|---|
| Base | 65.16±0.07(9) | 68.27±0.22(8) | 51.07±0.78(11) | 53.78±0.87(10) | 19.86±0.0(12) | 46.19±0.18(12) | 10.33 | 50.72 |
| Retrain | 77.17±0.09(4) | 72.70±0.08(6) | 71.32±0.16(2) | 68.83±0.22(2) | 83.71±0.14(8) | 59.99±0.02(5) | 4.5 | 72.28 |
| Tuning | 77.58±0.21(3) | 75.33±0.12(2) | 61.88±0.51(5) | 62.32±0.45(5) | 90.81±0.28(2) | 58.68±0.32(6) | 3.83 | 71.10 |
| eGBDT | 75.36±0.48(5) | 74.10±0.25(5) | 68.86±0.6(3) | 65.17±0.9(3) | 90.02±0.59(3) | 62.29±0.12(4) | 3.83 | 72.63 |
| AdIter | 78.74±0.13(2) | **75.42±0.08(1)** | **71.95±0.38(1)** | **70.51±0.42(1)** | **92.09±0.28(1)** | 63.67±0.06(3) | **1.5** | **75.39** |
| ARF | **79.22±0.19(1)** | 74.20±0.36(3) | 65.77±1.11(4) | 58.49±1.14(8) | 88.75±0.38(4) | **65.26±0.14(1)** | 3.5 | 71.94 |
| Learn++.NSE | 73.77±0.35(7) | 67.06±0.11(9) | 48.84±0.5(12) | 51.32±0.39(11) | 83.42±0.82(9) | 57.24±0.03(8) | 9.33 | 63.60 |
| LeverageBag | 60.24±0.11(12) | 66.25±0.06(11) | 59.78±0.43(9) | 62.72±0.45(4) | 84.37±0.11(7) | 54.99±0.0(9) | 8.66 | 64.72 |
| OnlineBoosting | 63.87±0.24(10) | 65.86±0.38(12) | 61.10±1.31(6) | 59.96±1.37(7) | 83.14±0.68(10) | 54.98±0.73(10) | 9.16 | 64.81 |
| OnlineRusBoost | 63.39±0.26(11) | 66.67±0.26(10) | 60.33±1.5(7) | 61.74±1.17(6) | 85.64±0.4(6) | 54.65±0.37(11) | 8.5 | 65.40 |
| OnlineBagging | 68.46±0.1(8) | 74.11±0.13(4) | 52.97±1.37(10) | 55.26±0.61(9) | 87.48±0.16(5) | 58.25±0.03(7) | 7.16 | 66.08 |
| SRP | 73.99±0.87(6) | 70.24±1.39(7) | 59.87±2.14(8) | 47.13±2.64(12) | 76.04±1.13(11) | 64.48±0.08(2) | 7.66 | 65.29 |




(a) Electricity (b) Weather

(c) Airline (d) Usenet1

(e) Usenet2 (f) Spam

Figure 5: AdIter's prequential accuracy with the six real-world datasets. AdIter was the most accurate method on all datasets except for Airline, where ARF showed the best performance.

Table 10: MCC Score of Six Real-World Datasets (%)

| Methods | Electricity | Weather | Airline | Usenet1 | Usenet2 | Spam | AvgRank | AvgScore |
|---|---|---|---|---|---|---|---|---|
| Base | 33.19±0.06(9) | 38.25±0.42(8) | 12.62±1.0(10) | 15.80±0.91(9) | 0.0±0.0(12) | 2.61±0.24(12) | 10 | 17.07 |
| Retrain | 54.40±0.19(4) | 46.14±0.16(6) | 42.65±0.32(2) | 40.68±0.55(2) | 67.51±0.28(8) | 20.12±0.04(5) | 4.5 | 45.25 |
| Tuning | 55.17±0.42(3) | 51.29±0.25(2) | 23.77±1.04(5) | 27.20±1.06(4) | 81.79±0.57(2) | 17.37±0.65(6) | 3.66 | 42.76 |
| eGBDT | 50.88±0.92(5) | 49.59±0.45(4) | 37.86±1.27(3) | 34.05±1.87(3) | 80.32±1.17(3) | 25.02±0.23(4) | 3.66 | 46.28 |
| AdIter | 57.58±0.26(2) | **52.02±0.14(1)** | **44.11±0.77(1)** | **44.41±0.78(1)** | **84.23±0.55(1)** | 27.81±0.12(3) | **1.5** | **51.69** |
| ARF | **58.61±0.38(1)** | 49.88±0.66(3) | 31.72±2.15(4) | 26.44±1.9(5) | 77.56±0.75(4) | **31.97±0.21(1)** | 3 | 46.03 |
| Learn++.NSE | 47.56±0.71(7) | 34.26±0.22(9) | 5.27±1.07(12) | 8.90±0.95(12) | 67.42±1.5(9) | 14.52±0.06(8) | 9.5 | 29.65 |
| LeverageBag | 20.77±0.23(12) | 32.77±0.13(11) | 19.59±0.87(9) | 25.81±0.85(6) | 69.05±0.26(7) | 9.98±0.01(9) | 9 | 29.66 |
| OnlineBoosting | 28.02±0.47(10) | 32.60±0.7(12) | 22.67±2.66(6) | 20.21±2.67(8) | 66.85±1.3(10) | 9.96±1.46(10) | 9.33 | 30.05 |
| OnlineRusBoost | 26.80±0.53(11) | 33.39±0.52(10) | 20.78±3.01(7) | 23.79±2.32(7) | 71.60±0.82(6) | 9.44±0.75(11) | 8.66 | 30.96 |
| OnlineBagging | 37.10±0.2(8) | 48.59±0.26(5) | 6.55±2.62(11) | 10.68±1.21(11) | 75.06±0.32(5) | 16.82±0.07(7) | 7.83 | 32.46 |
| SRP | 48.91±1.71(6) | 43.48±2.32(7) | 20.38±4.04(8) | 15.51±5.85(10) | 54.97±1.9(11) | 31.51±0.16(2) | 7.33 | 35.79 |

Table 11: Friedman Test of AdIter with Seven Methods

| Methods | $p$-value | Significance |
|---|---|---|
| ARF | 0.63735 | $p > 0.05$ |
| Learn++.NSE | 0.00002 | $p < 0.05$ |
| LeverageBag | 0.00097 | $p < 0.05$ |
| OnlineBoost | 0.00016 | $p < 0.05$ |
| OnlineRUSBoost | 0.00097 | $p < 0.05$ |
| OnlineBagging | 0.00468 | $p < 0.05$ |
| SRP | 0.00468 | $p < 0.05$ |

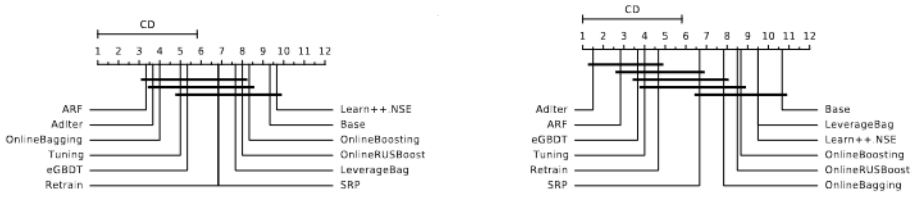
compared to seven benchmark methods, as listed in Tables 8, 9, 10.

In comparison to our proposed GBDT-based methods, Figure 3 (b)(c) shows AdIter's better average accuracy (77.73%), and average F1-score (75.39%). Further, among other seven ensemble learning methods, AdIter returned the highest average rank (1.5), and also yielded the highest MCC score, which indicates how well each method copes with class imbalances.

To examine AdIter's performance in greater detail, we then calculated the prequential accuracy of each method and plotted the results, as shown in Figure 5. Again, for F1-score and MCC score, the AdIter method also got the highest average ranking as 1.5, followed by ARF method. AdIter surpassed the rest with high prequential accuracy and robustness along the entire timeline.

Our next assessment metric for this experiment, was significance as calculated through the Friedman test. The results are shown in Table 11. Although there was no significant difference between AdIter and the ARF method, AdIter's bottom line results were better. In addition, according to the research in [54, 55], we draw a critical difference (CD) diagram to show the significance of our proposed method, as shown in Figure 6. After repeating 30 times experiments on each dataset. Our proposed AdIter may not perform ideal on synthetic datasets, but it shows a better performance on the real-world data.

Also, we provide the ROC-AUC analysis to demonstrate the efficiency of AdIter, as shown in Figure 7. The AUC score of our AdIter method is realtively lower than ARF method in some synthetic datasets. Although the AUC score of our AdIter method on synthetic datasets is lower than that of ARF method, it is relatively higher on real-world data, which also show




(a) Performance on synthetic data (b) Performance on real-world data

Figure 6: An illustrations of the significance of AdIter's performance. We choose 12 datasets and repeated 30 times experiments on each of them. On synthetic datasets, the performance of AdIter method does not have high significance. However, it shows a better performance on real-world datasets.

the performance of our method. Overall, these results shows that AdIter has certain potential and competitiveness, and is worthy of further research.

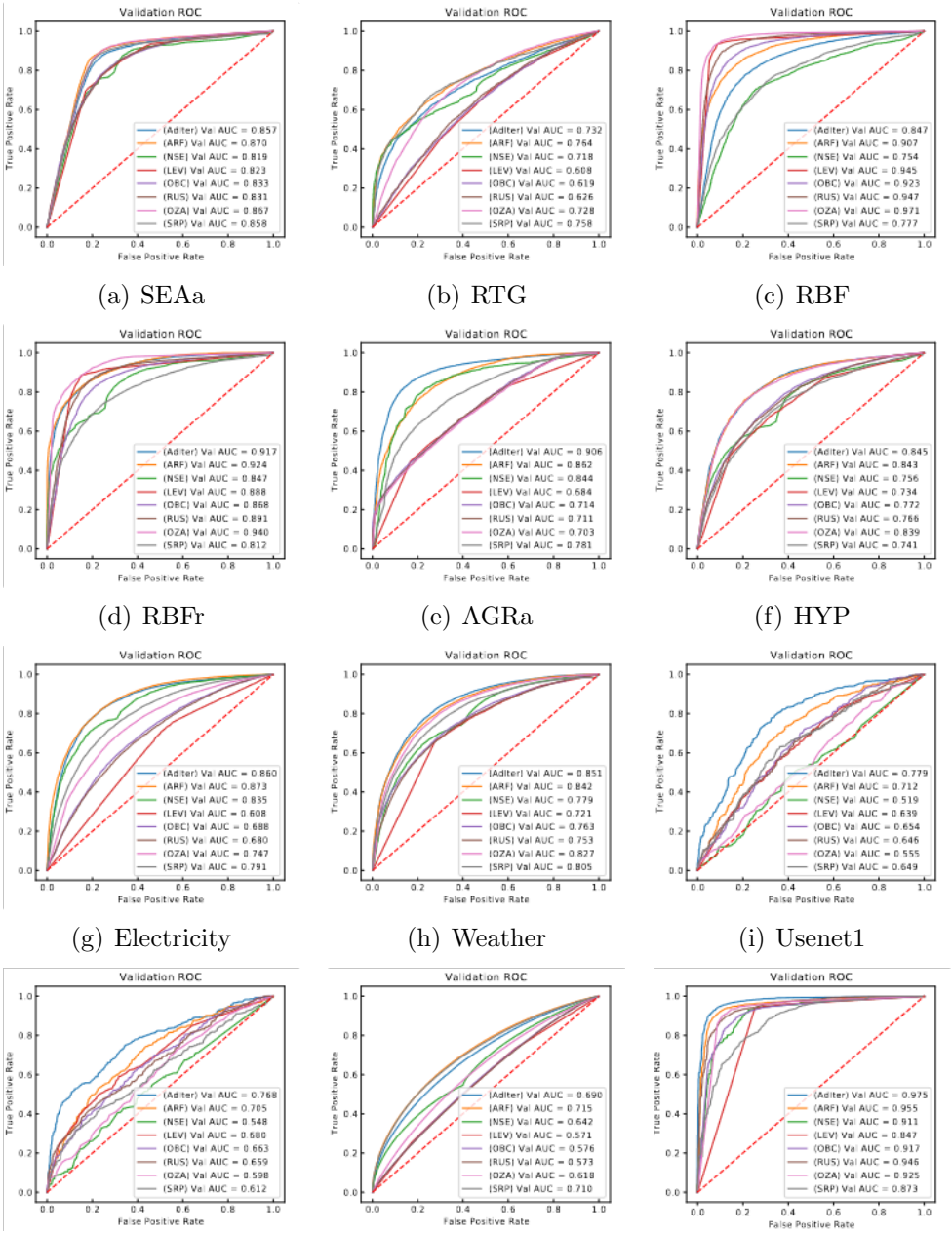
*5.3.4. Sensitivity Analysis*

As mentioned, the number of incremental iterations (learners) added by AdIter method was set to $\{L_1 = 25, L_2 = 50, L_3 = 75, L_4 = 100, L_5 = 125\}$, resulting in 5 eGBDT models. This parameter has an obvious impact on model performance, and therefore warrants a sensitivity analysis. We established five groups according to each of the above settings, as shown in Table 12, and tested the model on all benchmark datasets and calculate the prediction results. The resulting accuracy and F1-scores, pictured in Figure 8, show that values for both metrics rose as the number of iterations increased with most datasets. We used the parameter setting that gave the best results with the above experiments.

Table 12: Parameter Setting for Sensitivity Analysis I

| Index | Parameter setting ($M = 200$, $E$=5) |
|---|---|
| 1 | $\{L_1 = 5, L_2 = 10, L_3 = 15, L_4 = 20, L_5 = 25\}$ |
| 2 | $\{L_1 = 10, L_2 = 20, L_3 = 30, L_4 = 40, L_5 = 50\}$ |
| 3 | $\{L_1 = 15, L_2 = 30, L_3 = 45, L_4 = 60, L_5 = 75\}$ |
| 4 | $\{L_1 = 20, L_2 = 40, L_3 = 60, L_4 = 80, L_5 = 100\}$ |
| 5 | $\{L_1 = 25, L_2 = 50, L_3 = 75, L_4 = 100, L_5 = 125\}$ |

Besides, we have added another two groups of sensitivity analysis. In the first group, we fix the number of initial trained learners $M = 200$ and the




(a) SEAa (b) RTG (c) RBF

(d) RBFr (e) AGRa (f) HYP

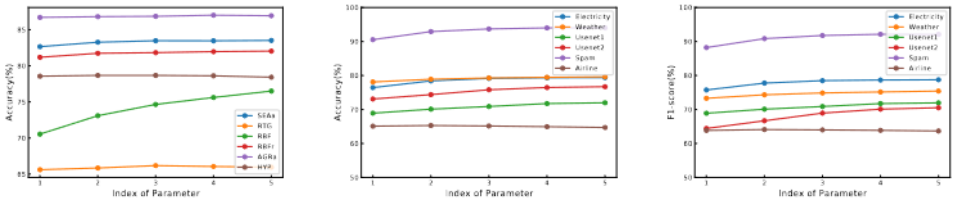
(g) Electricity (h) Weather (i) Usenet1

(j) Usenet2 (k) Airline (l) Spam

Figure 7: An illustration of ROC-AUC to show AdIter's efficiency. Our AdIter method achieves higher AUC scores on some synthetic and real-world datasets.

Table 13: Parameter Setting for Sensitivity Analysis II

| Index | Group 1 ($M = 200$, $L = 125$) | Group 2 ($L = 125$, $E = 5$) |
|---|---|---|
| 1 | $\{L_1 = 5, L_2 = 10, \cdots, L_{25} = 125\}$ | $M = 50$ |
| 2 | $\{L_1 = 10, L_2 = 20, \cdots, L_{12} = 120\}$ | $M = 100$ |
| 3 | $\{L_1 = 15, L_2 = 30, \cdots, L_8 = 120\}$ | $M = 150$ |
| 4 | $\{L_1 = 20, L_2 = 40, \cdots, L_6 = 120\}$ | $M = 200$ |
| 5 | $\{L_1 = 25, L_2 = 50, \cdots, L_5 = 125\}$ | $M = 250$ |



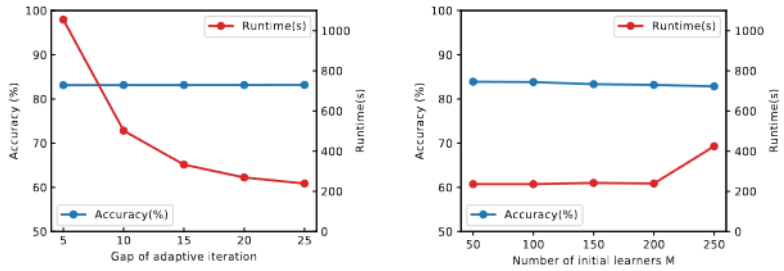

(a) Accuracy on synthetic datasets (b) Accuracy on real-world datasets (c) F1-score on real-world datasets

Figure 8: Sensitivity analysis for AdIter with all benchmark datasets, accuracy and F1-score are collected. Model performance improves as the number of iterations increase.

maximum adaptive iterations as $L = 125$. We adjust the number of eGBDTs by setting different gaps of adaptive iteration. In the second group, we fix the maximum adaptive iteration as $L = 125$ and the number of eGBDTs as $E = 5$. We adjust the number of initial trained learners. We adjust the number of initial trained learners. These two groups of parameter setting are shown in Table 13. We test the AdIter on the SEAa dataset and record the accuracy and runtime, as shown in Figure 9. From these figures, the runtime is sensitive by adjust the number of eGBDTs in the AdIter, but the accuracy has not change much. It reflects the robustness of our method. Therefore, in order to improve operation efficiency, we chose parameters that needs a relatively short runtime to set.

*5.3.5. Dimensionality and Structural Complexity Analysis*

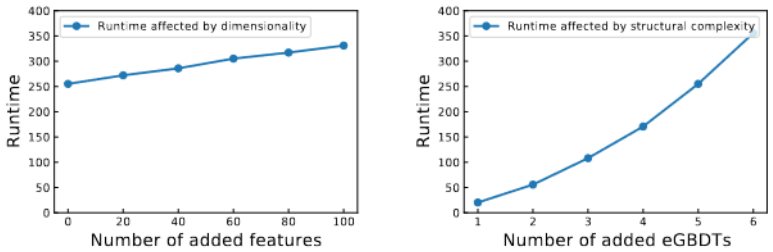
The Spam datasets, one of the real-world datasets, have 500 features. To further analyse the performance of AdIter, we also add new features to the SEAa dataset to simulate this experiment, and we have added an experiment to show the impact of dimensionality and structural complexity. First, we add new features to the SEAa dataset to simulate this experiment. We




(a) Group 1 (b) Group 2

Figure 9: Accuracy and runtime with parameter settings in Group 1 and Group 2.

sequentially add 20 features to the SEAa dataset to construct a new data set and conduct experiments in turn and record the running time. We use this to judge the impact of dimensionality on model performance. Second, we sequentially increase the number of eGBDT models in AdIter and conduct experiments to record the running time respectively. The results are shown in Figure 10.




(a) Runtime affected by dimensionality (b) Runtime affected by structural complexity

Figure 10: Analysis of AdIter affected by dimensionality and structural complexity. AdIter is more sensitive to the influence of structural complexity, but performs more efficiently for higher dimensional data.

### *5.3.6. Discussion of Limitations*

One of the main limitations of AdIter is its runtime. The eGBDT method is substantially faster, can adapt to drift, and it reduces memory consumption by pruning the learners after the last learner. By comparison, the AdIter method takes a relatively long run time. However, it delivers better performance.

In terms of concept drift adaptation, AdIter was ranked highest on average, which shows its potential and competitiveness. Further, AdIter is not designed specifically for GBDT models; it can be applied to any ensemble learning model. However, it did not deliver the best performance on every single datasets, especially flagging with the SEAa, RBF, and Airline datasets. With these, there was no significant difference to ARF. From this, we learned that it is not enough to simply adapt the model by setting the number of iterations. With drift severity of an uncertain nature, an outstanding strategy is still needed and time consumption is also a concern. In addition, extreme verification latency will also impact the performance of AdIter. Tackling these issues will be a priority as our follow-up research goals.

## 6. Conclusions and Further Study

In this paper, we propose the AdIter method for swift adaptation to concept drift. As the basis of AdIter, eGBDT helps GBDT model select the optimal adaptation strategy from either retraining or tuning. From a theoretical perspective, the greater the drift severity, the lower the model performance. Hence, to better recover from the accuracy losses associated with concept drift, AdIter helps models determine the number of additional iterations needed to return to pre-drift loss levels. Several eGBDT models are trained with some varying tuning iterations, and a final decision is made via a majority voting process. Overall, AdIter results in a more robust and accurate model, as demonstrated in a series of experiments with both real-world and synthetic datasets.

Although we set this paper in the context of a GBDT model, AdIter is compatible with many other ensemble learning models. For our future work, we will continue to improve the AdIter method to deal with data streams by optimizing some of the parameter settings and dealing with extreme verification latency to enhance its adaptability, efficiency, and robustness.

## Declaration of Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

**Acknowledgment**

This work was supported by the Australian Research Council through the Discovery Project under Grant No. DP190101733.

## Appendix A.

Consider three continuous data chunks $\{D_1, D_2, D_3\}$, the first data chunk $D_1$ follows the probability distribution $P$, which represents the old concept. However, in the second data chunk $D_2$, not only has $\alpha$ portion of data follow the probability distribution $P$; $1-\alpha$ portion of data that follow the probability distribution $Q$. Thus, the probability distribution of data chunk $D_2$ is expressed as

$$P' = \alpha P + (1-\alpha)\, Q. \tag{A.1}$$

This formulation indicates that concept drift has occurred.

The third data chunk $D_3$ reflects a totally new concept that fully follows the probability distribution $Q$. Further, assume $F(x)$ is the model prediction result, $y_P$ is the true label of data chunk $D_1$, $y_{P'}$ is the true label of data chunk $D_2$. We use $\phi_P(x), \phi_{P'}(x), \phi_Q(x)$ as the probability density function as distribution $P$, $P'$, $Q$ the data chunks follow, and the Eq. (A.1) can be written as

$$\int \phi_{P'}(x)\mathrm{d}x = \int \alpha\phi_P(x)\mathrm{d}x + \int (1-\alpha)\phi_Q(x)\mathrm{d}x. \tag{A.2}$$

Thus, the generalization error for $D_1$ is

$$R_P\left(F(x)\right) = \mathbb{E}_{x\sim P}\left[l_P\right] = \int l\left(x\right)\phi_P(x)\mathrm{d}x, \tag{A.3}$$

where $R_P\left(F(x)\right) = R_P(y_P, F(x))$, and $l_P = l\left(y_P, F(x)\right)$. The generalization error for $D_2$ is

$$R_{P'}\left(F(x)\right) = \mathbb{E}_{x\sim P'}\left[l_{P'}\right] = \int l\left(x\right)\phi_{P'}(x)\mathrm{d}x, \tag{A.4}$$

where $R_{P'}\left(F(x)\right) = R_{P'}(y_{P'}, F(x))$, and $l_{P'} = l\left(y_{P'}, F(x)\right)$. And The empirical error on $D_2$ is

$$\hat{R}_{P'}\left(F(x)\right) = \frac{1}{n}\sum_{i=1}^{n} l\left(h(x_i) - y_i\right). \tag{A.5}$$

Inspired by [44], rewriting the generalization error on $D_2$, we have

$$\begin{aligned}
R_{P'}\left(F(x)\right) =& R_{P'}\left(F(x)\right) + R_P\left(F(x)\right) - R_P\left(F(x)\right) + R_P\left(y_{P'}, F(x)\right) \\
& - R_P\left(y_{P'}, F(x)\right) \\
=& R_P\left(F(x)\right) + \left(R_{P'}\left(F(x)\right) - R_P\left(y_{P'}, F(x)\right)\right) \\
& + \left(-R_P\left(F(x)\right) + R_P\left(y_{P'}, F(x)\right)\right) \\
=& R_P\left(y_P, F(x)\right) + \int l\left(y_{P'}, F(x)\right)\left(\phi_{P'}(x) - \phi_P(x)\right) \mathrm{d}x \\
& + \int \left[l\left(y_{P'}, F(x)\right) - l\left(y_P, F(x)\right)\right] \phi_P(x) \mathrm{d}x \\
=& R_P\left(y_P, F(x)\right) + \int l\left(y_{P'}, F(x)\right)\left(\alpha\phi_P(x) + (1-\alpha)\,\phi_Q(x) - \phi_P(x)\right) \mathrm{d}x \\
& + \int \left[l\left(y_{P'}, F(x)\right) - l\left(y_P, F(x)\right)\right] \phi_P(x) \mathrm{d}x \\
=& R_P\left(y_P, F(x)\right) + (1-\alpha)\, d\left(Q, P\right) \\
& + \int \left[l\left(y_{P'}, F(x)\right) - l\left(y_P, F(x)\right)\right] \phi_P(x) \mathrm{d}x.
\end{aligned} \tag{A.6}$$

According to the bound theory of boosting [56], for any $\delta > 0$, with probability at least $1-\delta$, the bound of the GBDT model $F(x)$ with $M$ weak learners under concept drift can be written as

$$R_{P'}\left(y_{P'}, F(x)\right) \leq \hat{R}_P\left(y_P, F(x)\right) + \sqrt{\frac{\log\frac{1}{\delta}}{2n}} + (1-\alpha)\, d\left(Q, P\right) + \Delta, \tag{A.7}$$

where $n$ is the number of samples, $\Delta = \int \left[l(y_{P'}, F(x)) - l(y_P, F(x))\right]\phi_P(x)\mathrm{d}x$.

In this formula, $(1-\alpha)\, d\left(Q, P\right)$ represents the difference between the data distributions at two consecutive timestamps. The more obvious the difference, the greater the possibility of data drift. Moreover, these differences will affect $\Delta$, which directly reflects the extent to which the loss changes before and after drift. The greater the change, the greater the severity of the drift. Once determined, the learning strategy of the model needs to be adjusted accordingly to suit that level of drift severity.

## References


[1] A. Liu, J. Lu, G. Zhang, Diverse instance-weighting ensemble based on region drift disagreement for concept drift adaptation, IEEE Transactions on Neural Networks and Learning Systems 32 (1) (2020) 293–307.

[2] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, G. Zhang, Learning under concept drift: A review, IEEE Transactions on Knowledge and Data Engineering 31 (12) (2018) 2346–2363.

[3] J. Vinagre, A. M. Jorge, J. Gama, Online gradient boosting for incremental recommender systems, in: International Conference on Discovery Science, Springer, Limassol, Cyprus, 2018, pp. 209–223.

[4] A. Saadallah, L. Moreira-Matias, R. Sousa, J. Khiari, E. Jenelius, J. Gama, Bright-drift-aware demand predictions for taxi networks, IEEE Transactions on Knowledge and Data Engineering 32 (2) (2018) 234–245.

[5] Y. Song, J. Lu, A. Liu, H. Lu, G. Zhang, A segment-based drift adaptation method for data streams, IEEE Transactions on Neural Networks and Learning Systems (2021). doi:10.1109/tnnls.2021.3062062.

[6] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, M. Woźniak, Ensemble learning for data stream analysis: A survey, Information Fusion 37 (2017) 132–156.

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

[8] J. Lu, A. Liu, Y. Song, G. Zhang, Data-driven decision support under concept drift in streamed big data, Complex & Intelligent Systems 6 (1) (2020) 157–163.

[9] K. Wang, A. Liu, J. Lu, G. Zhang, L. Xiong, An elastic gradient boosting decision tree for concept drift learning, in: Australasian Joint Conference on Artificial Intelligence, Springer, 2020, pp. 420–432.

[10] J. H. Friedman, Greedy function approximation: A gradient boosting machine, Annals of Statistics (2001) 1189–1232.

[11] H. Hu, W. Sun, A. Venkatraman, M. Hebert, A. Bagnell, Gradient boosting on stochastic data streams, in: A. Singh, J. Zhu (Eds.), Proceedings of the 20th International Conference on Artificial Intelligence and Statistics, Vol. 54 of Proceedings of Machine Learning Research, PMLR, Fort Lauderdale, FL, USA, 2017, pp. 595–603.

[12] N. C. Oza, Online bagging and boosting, in: 2005 IEEE International Conference on Systems, Man and Cybernetics, Vol. 3, IEEE, Waikoloa, Hawaii, USA, 2005, pp. 2340–2345.

[13] B. Wang, J. Pineau, Online bagging and boosting for imbalanced data streams, IEEE Transactions on Knowledge and Data Engineering 28 (12) (2016) 3353–3366.

[14] R. K. Vinayak, R. Gilad-Bachrach, Dart: Dropouts meet multiple additive regression trees, in: Artificial Intelligence and Statistics, PMLR, 2015, pp. 489–497.

[15] H. Yu, J. Lu, G. Zhang, An online robust support vector regression for data streams, IEEE Transactions on Knowledge and Data Engineering 34 (1) (2020) 150–163.

[16] H. Yu, J. Lu, G. Zhang, Continuous support vector regression for non-stationary streaming data, IEEE Transactions on Cybernetics (2020) 1–14doi:10.1109/TCYB.2020.3015266.

[17] S. Pan, K. Wu, Y. Zhang, X. Li, Classifier ensemble for uncertain data stream classification, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, Hyderabad, India, 2010, pp. 488–495.

[18] S. Pan, J. Wu, X. Zhu, C. Zhang, Graph ensemble boosting for imbalanced noisy graph stream classification, IEEE Transactions on Cybernetics 45 (5) (2014) 954–968.

[19] F. Dong, J. Lu, Y. Song, F. Liu, G. Zhang, A drift region-based data sample filtering method, IEEE Transactions on Cybernetics (2021). doi:10.1109/TCYB.2021.3051406.

[20] J. Shao, Y. Tan, L. Gao, Q. Yang, C. Plant, I. Assent, Synchronization-based clustering on evolving data stream, Information Sciences 501 (2019) 573–587.

[21] Y. Song, J. Lu, H. Lu, G. Zhang, Fuzzy clustering-based adaptive regression for drifting data streams, IEEE Transactions on Fuzzy Systems 28 (3) (2019) 544–557.

[22] M. Pratama, C. Za'in, A. Ashfahani, Y. S. Ong, W. Ding, Automatic construction of multi-layer perceptron network from streaming examples, in: Proceedings of the 28th ACM International Conference on Information and Knowledge Management, ACM, Beijing, China, 2019, pp. 1171–1180.

[23] M. Pratama, A. Ashfahani, A. Hady, Weakly supervised deep learning approach in streaming environments, in: 2019 IEEE International Conference on Big Data, IEEE, Los Angeles, CA, USA, 2019, pp. 1195–1202.

[24] S. Wang, L. L. Minku, N. Chawla, X. Yao, Learning in the presence of class imbalance and concept drift, Neurocomputing 343 (2019) 1–2.

[25] S. Ren, B. Liao, W. Zhu, Z. Li, W. Liu, K. Li, The gradual resampling ensemble for mining imbalanced data streams with concept drift, Neurocomputing 286 (2018) 150–166.

[26] N. Lu, J. Lu, G. Zhang, R. L. De Mantaras, A concept drift-tolerant case-base editing technique, Artificial Intelligence 230 (2016) 108–133.

[27] V. Losing, B. Hammer, H. Wersing, KNN classifier with self adjusting memory for heterogeneous concept drift, in: 2016 IEEE 16th International Conference on Data Mining, IEEE, Barcelona, Spain, 2016, pp. 291–300.

[28] I. Žliobaitė, A. Bifet, B. Pfahringer, G. Holmes, Active learning with drifting streaming data, IEEE Transactions on Neural Networks and Learning Systems 25 (1) (2013) 27–39.

[29] J. Shan, H. Zhang, W. Liu, Q. Liu, Online active learning ensemble framework for drifted data streams, IEEE Transactions on Neural Networks and Learning Systems 30 (2) (2018) 486–498.

[30] J. Gama, P. Medas, G. Castillo, P. Rodrigues, Learning with drift detection, in: Brazilian Symposium on Artificial Intelligence, Springer, Sao Luis, Brazil, 2004, pp. 286–295.

[31] A. Bifet, R. Gavalda, Learning from time-changing data with adaptive windowing, in: Proceedings of the 2007 SIAM International Conference on Data Mining, SIAM, Minnesota, USA, 2007, pp. 443–448.

[32] N. Mozafari, S. Hashemi, A. Hamzeh, A precise statistical approach for concept change detection in unlabeled data streams, Computers & Mathematics with Applications 62 (4) (2011) 1655–1669.

[33] R. Razavi-Far, E. Hallaji, M. Saif, G. Ditzler, A novelty detector and extreme verification latency model for nonstationary environments, IEEE Transactions on Industrial Electronics 66 (1) (2019) 561–570.

[34] E. Hallaji, R. Razavi-Far, M. Saif, Detection of dalicious SCADA communications via multi-subspace feature selection, in: 2020 International Joint Conference on Neural Networks, IEEE, 2020, pp. 1–8.

[35] G. Ditzler, M. Roveri, C. Alippi, R. Polikar, Learning in nonstationary environments: A survey, IEEE Computational Intelligence Magazine 10 (4) (2015) 12–25.

[36] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2000, pp. 71–80.

[37] G. Hulten, L. Spencer, P. M. Domingos, Mining time-changing data streams, in: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2001, pp. 97–106.

[38] J. Z. Kolter, M. A. Maloof, Dynamic weighted majority: An ensemble method for drifting concepts, Journal of Machine Learning Research 8 (2007) 2755–2790.

[39] D. Brzezinski, J. Stefanowski, Accuracy updated ensemble for data streams eith concept drift, in: International Conference on Hybrid Artificial Intelligence Systems, Springer, Wroclaw, Poland, 2011, pp. 155–163.

[40] D. Brzezinski, J. Stefanowski, Reacting to different types of concept drift: The accuracy updated ensemble algorithm, IEEE Transactions on Neural Networks and Learning Systems 25 (1) (2013) 81–94.

[41] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Computing Surveys (CSUR) 46 (4) (2014) 1–37.

[42] R. Elwell, R. Polikar, Incremental learning of concept drift in nonstationary environments, IEEE Transactions on Neural Networks 22 (10) (2011) 1517–1531.

[43] M. Pratama, W. Pedrycz, E. Lughofer, Evolving ensemble fuzzy classifier, IEEE Transactions on Fuzzy Systems 26 (5) (2018) 2552–2567.

[44] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, J. W. Vaughan, A theory of learning from different domains, Machine Learning 79 (1-2) (2010) 151–175.

[45] Y. Song, J. Lu, H. Lu, G. Zhang, Fuzzy clustering-based adaptive regression for drifting data streams, IEEE Transactions on Fuzzy Systems 28 (3) (2020) 544–557.

[46] H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, T. Abdessalem, Adaptive random forests for evolving data stream classification, Machine Learning 106 (9-10) (2017) 1469–1495.

[47] A. Bifet, G. Holmes, B. Pfahringer, Leveraging bagging for evolving data streams, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, Berlin, Heidelberg, 2010, pp. 135–150.

[48] H. M. Gomes, J. Read, A. Bifet, Streaming random patches for evolving data stream classification, in: IEEE International Conference on Data Mining, IEEE, 2019, pp. 240–249.

[49] J. Montiel, J. Read, A. Bifet, T. Abdessalem, Scikit-multiflow: A multi-output streaming framework, Journal of Machine Learning Research 19 (72) (2018) 1–5.

[50] W. N. Street, Y. Kim, A streaming ensemble algorithm (SEA) for large-scale classification, in: Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, USA, 2001, pp. 377–382.

[51] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, New York, USA, 2000, pp. 71–80.

[52] R. Agrawal, T. Imielinski, A. Swami, Database mining: A performance perspective, IEEE Transactions on Knowledge and Data Engineering 5 (6) (1993) 914–925.

[53] I. Katakis, G. Tsoumakas, I. Vlahavas, Tracking recurring contexts using ensemble classifiers: An application to email filtering, Knowledge and Information Systems 22 (3) (2010) 371–391.

[54] E. Hallaji, R. Razavi-Far, M. Saif, Dlin: Deep ladder imputation network, IEEE Transactions on Cybernetics (2021) 1–13doi:10.1109/TCYB.2021.3054878.

[55] E. Hallaji, R. Razavi-Far, V. Palade, M. Saif, Adversarial learning on incomplete and imbalanced medical data for robust survival prediction of liver transplant patients, IEEE Access 9 (2021) 73641–73650.

[56] M. Mohri, A. Rostamizadeh, A. Talwalkar, Foundations of machine learning, MIT press, 2018.