Knowledge-Based Systems 201–202 (2020) 106058

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Online random forests regression with memories

Yuan Zhong [a,b], Hongyu Yang [a,*], Yanci Zhang [a,*], Ping Li [b]

[a] *College of Computer Science, Sichuan University, Chengdu, China*
[b] *School of Computer Science, Southwest Petroleum University, Chengdu, China*

ARTICLE INFO

Article history:
Received 3 December 2019
Received in revised form 18 May 2020
Accepted 19 May 2020
Available online 26 May 2020

Keywords:
Random forests regression
Long-term memory
Online weight learning
Leaf-level
Adaptive learning rate
Stochastic gradient descent

ABSTRACT

In recent years, the online schema of the conventional Random Forests(RFs) have attracted much attention because of its ability to handle sequential data or data whose distribution changes during the prediction process. However, most research on online RFs focuses on structural modification during the training stage, overlooking critical aspects of the sequential dataset, such as autocorrelation. In this paper, we demonstrate how to improve the predictive accuracy of the regression model by exploiting data correlation. Instead of modifying the structure of the off-line trained RFs, we endow RFs with memory during regression prediction through an online weight learning approach, which is called Online Weight Learning Random Forest Regression(OWL-RFR). Specifically, the weights of leaves are updated based on a novel adaptive stochastic gradient descent method, in which the adaptive learning rate considers the current and historical prediction bias ratios, compared with the static learning rate. Thus, leaf-level weight stores the learned information from the past data points for future correlated prediction. Compared with tree-level weight which only has immediate memory for current prediction, the leaf-level weight can provide long-term memory. Numerical experiments with OWL-RFR show remarkable improvements in predictive accuracy across several common machine learning datasets, compared to traditional RFs and other online approaches. Moreover, our results verify that the weight approach using the long-term memory of leaf-level weight is more effective than immediate dependency on tree-level weight. We show the improved effectiveness of the proposed adaptive learning rate in comparison to the static rate for most datasets, we also show the convergence and stability of our method.

1. Introduction

Random Forests(RFs) [1] are ensembles of randomized decision trees that use bagging for classifier or regression tasks. RFs are regularly used in machine learning applications, as well as some tasks demanding high real-time performance, such as computer vision [2,3]. Their main advantages embody the good performances in computing efficiency and prediction.

RFs are typically off-line methods [4]. Nevertheless, there are numerous situations in which applications require real-time processing of massive streaming data, such as prediction of network workload, traffic conditions during peak time, and users' evaluation of products. These types of extensive data are challenging to store in totality. Furthermore, the distribution of test samples may differ from that of training samples. Therefore, the evaluation of future testing instances using the models trained off-line directly may feature large prediction bias.

To address these issues, some efforts have been dedicated to online training [5–7]. However, to the best of our knowledge, most of the existing approaches suffer from frequent updating of the model, which generally leads to inefficient computation. Moreover, they often ignore the continuing dependencies between samples in the data stream.

In this paper, we propose a novel approach that updates the leaf-level weight of off-line trained base learners in an online manner and endows RFs with long-term memory to avoid the need for frequent model updating. More precisely, as testing data arrive sequentially, our method updates the leaf-level weight of RFs simultaneously, while keeping the structure of the trained RFs' model unchanged. In this way, the proposed approach achieves better prediction by such online weight learning, in addition to exploiting the superior efficiency of the off-line training approach. Moreover, our method avoids the low efficiency and poor accuracy of a pure online growth strategy during the beginning stages. Our contribution is three-fold:

(1) We propose an online weight learning approach that endows RFs with memory to improve regression prediction in cases of distribution is changed in streaming data.

* Corresponding authors.
E-mail addresses: yanghongyu@scu.edu.cn (H. Yang), yczhang@scu.edu.cn (Y. Zhang).

https://doi.org/10.1016/j.knosys.2020.106058
0950-7051/© 2020 Published by Elsevier B.V.

(2) We propose an adaptive learning rate for stochastic gradient descent (SGD) based on current and historical prediction bias ratios.
(3) We validate the proposed method with extensive experiments and show the convergence and stability of our method.

The rest of the paper is organized as follows. In Section 2 we discuss previous work in related areas. In Section 3 we introduce the main principles of our method and present the online weight learning algorithm for RFs. Section 4 details a series of experiments with our approach on regression tasks. Finally, the conclusion and future work are given in Section 5.

2. Related work

The conventional RFs are an ensemble of unpruned (classification or regression) decision trees, such as C4.5 [8] and CART [9], and are a type of extension of bagging [10]. Conventional RFs use random feature selection in tree induction procedures, through bootstrap sampling of training data. RFs have many advantages, such as low cost of computation, ease of implementation, and powerful performance. RFs' predictions are generated through aggregating predictions of trees, in the manner of either majority vote (for classification) or averaging (for regression). Hence, risks arise when RFs learn from an extremely imbalanced training dataset. The usual way to alleviate this problem is the use of weighted RFs.

2.1. Weighted random forests

The weighted random forests method is a well-studied technique in the research literature on RFs. Weighted averaging has served in ensemble learning since Perrone proposed ensemble learning in 1993 [11]. More recent research has proposed weighted approaches from different aspects based on the traditional RFs. Chen et al. [12] proposed that RFs classifiers would favor some classifications, and that each class could be assigned weight to deal with imbalances; that is, class weights could be used in node partitioning criteria and aggregation processes, to ensure that a few classes have equal weights. This method considers class-related weights. Additionally, there are weighted methods that are based on sample weight. Instead of simple random sampling, Amaratunga et al. [13] took the eligible subsets at each node via weighted random sampling, to achieve superior performance in several microarray datasets. Wei et al. [14] combined the support vector machine(SVM) and sample weighted RFs to resolve the negative impact of class imbalance. They used SVM model on all training samples to calculate the weight for all samples and produce the sample-weighted RFs. Maudes et al. [15] proposed the Random Feature Weights(RFW) method for constructing ensembles of decision trees similar to RFs: all the nodes in a tree have the same random weights for all attributes but vary from other trees, and features with higher weights are more likely to be used for tree construction. Wu et al. [16] proposed a new RFs based ensemble method, whose key idea is to split these features into two groups and create term weighting for the features. Other weighted approaches are based on tree weight. For example, Li et al. [17] proposed Trees Weighting Random Forest(TWRF), which weights the trees according to their classification ability, using out-of-bag(OOB) data to solve problems with datasets that have numerous dimensions and significant noisy features. Winham et al. [18] proposed the wRF method, which incorporates tree-level weight to support the accuracy of trees in classification prediction. They also used OOB training data to assess tree performance and calculate tree weight. All these methods determine weight by class, sample, feature, or tree for classification purpose. To the best of our knowledge, there has been little addressed on the leaf weight learning for RFs regression in the literature.

2.2. Online schema for random forests

Another extension of RFs involves online or dynamic schema. In the traditional batch learning paradigm, models cannot update while processing streaming or incremental data. These models can only be rebuilt periodically, which not only fails under real-time requirements but may result in outdated models. In recent years, online learning has grown to be an important domain in machine learning. The streaming computing paradigm of online learning matches well the dynamic changes of the characteristic of data. Therefore, there has been theoretical and practical research on online learning, including the classical Perceptron algorithm [19], and the more recent Forward–Backward Splitting (FOBOS) [20], Regularized Dual Averaging (RDA) [21], and Follow-The-Regularized-Leader (FTRL) [22] algorithms for online convex optimization.

Research has also focused on the online construction of trees to interface with big data or streaming data. Saffari et al. [4] proposed an online tree building approach, ORF, in which new sub-nodes are generated from the root node through continuous online sampling, achieving comparable performance to batch-trained RFs. Wang et al. [23] proposed the Incremental Extremely Random Forest(IERF) growing algorithm for online learning classification with streaming data. Cassidy et al. [24] analyzed the importance of features based on the flow learning method of ORF, in which ORF discards poorly performing trees that learned old concepts and removes their contributions to forest-wide mean decrease in accuracy and mean decrease in Gini impurity, thereby addressing large-volume data streams with concept drift. Lakshminarayanan et al. [25] used Mondrian processes to construct Mondrian forests, of which trees can grow in an incremental or online fashion. It can achieve prediction performance compared with periodically re-trained batch RFs, as well as existing online RFs. Ristin et al. [26] studied performance in two variants of RFs under four strategies for incrementally learning new classes, which avoids the need to retrain the RFs from scratch. Paul et al. [27] proposed an improved RFs with a minimum number of trees for classification tasks: starting from a forest with fewer trees, each iteration removes unimportant features by feature weight, until convergence.

All the above methods modify RFs models online. However, such modification consumes substantial time and storage space. Some researchers have adopted a dynamic or iterative approach to deal with data streams by adjusting the weight to improve RFs performance. For example, Kim et al. [28] presented the WAVE approach, which uses two weight vectors to compute classifiers and instances. They proved that the iterated weight vectors could converge on the optimal weights, with optimal weights obtaining a final prediction significantly better than simple majority voting. Zhukov et al. [29] presented a classification RFs built on a weighted majority voting ensemble rule for concept drift. Their base learner weight is computed for each sample evaluation using the base learner's accuracy and the intrinsic proximity measure of the RFs. The approach combines the temporal weighting of samples and ensemble pruning as a forgetting strategy.

2.3. Online regression

Most online RFs models have focused on the classification task [30–32]. Relatively few studies have focused on online regression. However, a series of improvements for the online regression task have been proposed in recent years, particularly in ensemble learning from a data stream. Kolter et al. [33] proposed Addictive Expert Ensembles(AddExp) for non-stationary data streams, for both classification and regression tasks. In regression prediction, a weight is associated with each base learner and the weight is always multiplied by $\beta^{|\hat{y}-y|}$, where $0 \leq \beta < 1$, $\hat{y}$ is the prediction given by the base learner and y is the actual value. Kadlec [34] developed the Incremental Local Learning Soft Sensing Algorithm (ILLSA). It used a different part of the training data to train several base models first, and then assigned weights to models during the online data stream stage, and weights were recalculated sample-by-sample. To handle various types of concept drifts, Gomes et al. [35] proposed an Online Weighted Ensemble(OWE) of regressor models, including two adaptive mechanisms: sample selecting and ensemble learning. The ensemble model is based on a sliding window, to collect new samples and keep removing redundant ones. A boosting similar approach is used for the weight calculation of base models in an ensemble, and OWE can consequently add or prune models to expand its structure. These online regression ensemble methods are not specific to tree-based methods, but contain types of model based-methods. Additionally, a few methods of learning online tree ensembles have emerged for regression tasks. For instance, Bifet et al. [36] used online bagging to create an ensemble of Hoeffding naive Bayes trees [37], and return a naive Bayes prediction from the leaf, when performing a prediction.

RFs fall into the ensemble learning category and can easily cope with the regression problem. To the best of our knowledge, however, only a few studies have investigated the online regression prediction of RFs. The aforementioned research findings inspire us to combine the superior aspects of the weighting pattern and online tactic approaches. Hence, our work focuses on online schema for the RFs regression task. Furthermore, we use leaf weight as a memory to provide long-term dependency in RFs regression prediction, which is not used in other approaches. In summary, we design a leaf-level weight updating algorithm that combines the conventional RFs model with dynamic online weight learning methods of prediction.

3. Proposed approach

3.1. Preliminaries

Online learning belongs to the online optimization problem, which attempts to overcome the shortcomings of statistical learning theory when addressing the dynamic aspects of large datasets. Online regression learning can be described formally as follows [38]. At every round $t = 1, 2, \dots, n$:

- receive question $x_t \in \mathcal{X}$ where $\mathcal{X} \in \mathbb{R}^d$ which corresponds to a set of features
- predict $p_t \in \mathcal{D}$ by $h_t \in \mathcal{H}$, $\mathcal{H}$ is the hypothesis set, $\mathcal{H}$=$\{x_t \mapsto \sum_{i=1}^{d} w_t[i]x_t[i]$: $\forall$ i, $w_t[i] \in \mathbb{R}\}$, where $w_t[i]$ is the ith element of w_t.
- receive true answer $y_t \in \mathcal{Y}$ where $\mathcal{Y} = \mathcal{D} = \mathbb{R}$.
- suffer loss $l(h_t, (p_t, y_t))$.

In regression problems, the most common loss function is the squared loss, $l(h_t, (p, y))$ =$(p$-$y)^2$ or the absolute loss, $l(h_t, (p, y))$= $|p - y|$, which is often termed instantaneous loss because it measures the discrepancy between the predicted answer and the correct answer. With online loss information, we can update the machine model and attempt to gain a better result in the next round. The objective of online learning is to minimize the cumulative regret, which describes the degree of regret stemming from not using the best model of the decision space. Formally, *Regret*[38] can be expressed as Eq. (1), where h^* denotes the static batch-trained model.

$$R(h, T) = \sum_{t=1}^{T} l(h_t, (p_t, y_t)) - \sum_{t=1}^{T} l(h^*, (p_t, y_t)) \tag{1}$$

Hence, our objective is to obtain the sub-linear regret between our online weight learning RFs and static batch-trained RFs.

3.2. Memory:leaf-level weight for long-term dependency

Although the above online regression looks on the surface similar to the weight learning of base learners, there is an essential difference between the two schemas. In our proposed approach, we aim to update the leaves' weights as model parameters, in each round. The leaves that are used for prediction of the current testing sample learn their weights from those of testing samples in the past, which endows the RFs with memory in the process of regression prediction. Therefore, in contrast to tree weight learning, which updates in each round and can only provide immediate (short-term) memory, our leaf-level weight learning, based on online regression, captures the long-term correlation of the data stream. We note that different from conventional RFs that assign weight for each tree, our approach only takes account of the weights on leaves, not using the trees' weights.

We use a diagram to illustrate the difference of leaf-level weight and tree-level weight update mechanisms. As illustrated in Fig. 1, suppose we have four samples in a temporal sequence, sample 3 is similar to sample 1 and sample 4 is similar to sample 2. Therefore, we can assume that sample 1 and sample 3 will both fall into leaf node n_1, and sample 2 and sample 4 will both fall into leaf node n_3. If we use the tree's weight, as shown in the figure, with test samples arriving, the weight of Tree-1 (w_{t1}^{t}) will update for the tth round of the test sample. For instance, the prediction of sample 2 depends on the weight learning from sample 1, which means that it provides an immediate memory for the current prediction according to the last update. Then, w_{t1}^{2} is updated based on w_{t1}^{1}, w_{t1}^{3} is updated based on w_{t1}^{2} and so on, this may cause the weight of tree to over-fit the most recent test sample. In contrast, the leaf node weight will update in the case where the current sample is similar to earlier samples, because both fall into the same leaf; e.g., as sample 3 falls into leaf node n_1, node n_1's weight($w_{n_1}^{3}$) for sample 3 is updated based on the weight($w_{n_1}^{1}$) for sample 1. Hence, leaf-level weight provides a long-term memory for current prediction based on earlier learning, which result in more reasonable predictions than from tree-level weight learning.

3.3. Online weight learning

Following the above ideas, first, we use the CART algorithm to construct the decision trees for RFs in a batch manner when pre-sampling is satisfied, with the least square residual as the criterion for the best hyperplane when nodes are split. We then iteratively update the weights of leaves, which indicate that new test instances fall into the same leaves during the prediction. For the sake of expression consistency, we define weighted averaging as Eq. (2), which is suited to the first prediction when the weight of all leaves is 1(the initial weight). This is also suited to the next prediction of unequal weight.

$$p_t = \frac{1}{T} \sum_{i=1}^{T} w_i y_i \tag{2}$$

where t is the sequence number of the test instance, p_t is the prediction of the tth test, T is the number of trees in the forest, and i is the sequence number of the tree in the RFs. We use one leaf's value to stand for one tree's value in prediction, with y_i being the value of a leaf used for the current prediction, and w_i being the weight of this leaf. While test samples continue to arrive, the weight of leaf that is used is updated based on its prior weight. More leaves can learn new weights as more test instances arrive. Clearly, it is hard to evaluate a learner on only a single test, and it is reasonable to give more weight to a learner that predicts more instances close to the true response. Hence, we use the Stochastic Gradient Descent(SGD) approach to review the

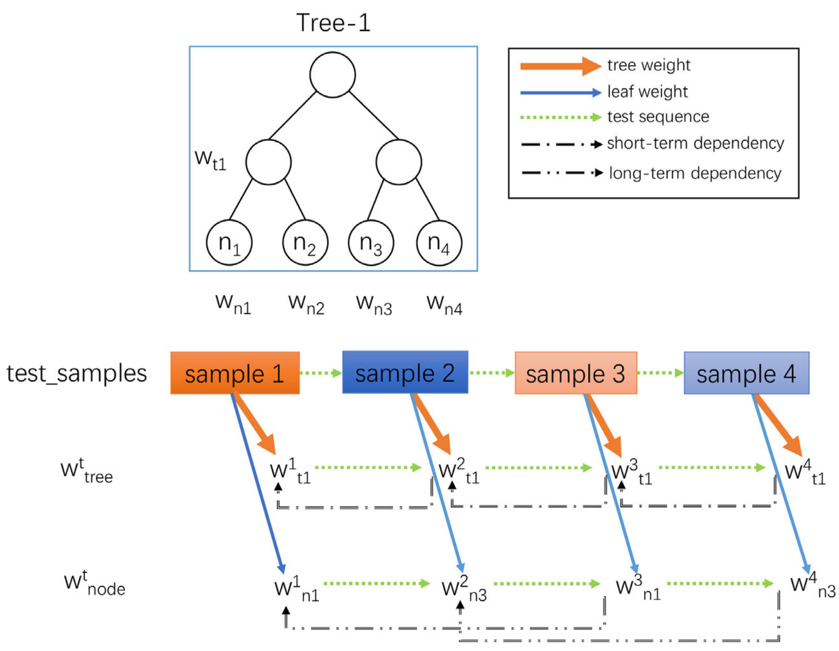


Fig. 1. Comparison of leaf-level weight and tree-level weight update mechanisms. Notably, test sample 1 is similar to sample 3 (their legends are similar in color). The same holds for sample 2 and sample 4.

variation of leaves' weights during prediction. For simplicity, we call our method Online Weight Learning for RFs Regression(OWL-RFR). The purpose of the online regression task is to reduce the bias error. Hence, we choose the squared error as the objective function, defined in Eq. (3).

$$E_t = \frac{1}{2}(p_t - y_t)^2 \tag{3}$$

where y_t is the true response for the tth test instance.

OWL-RFR is based on a gradient descent strategy: while testing samples fall into the leaf nodes of trees, we adjust the weights of those leaf nodes in the negative direction of the objective loss function. Consequently, we can cause the value of loss function to descend quickly. That is, we can acquire the appropriate weights to represent the base learner in forests with less bias of the prediction value, relative to the true value. Δw_i, which represents the incremental change to the original weight, can be calculated using Eq. (4).

$$\Delta w_i = -\eta_i \frac{\partial E_t}{\partial w_i} \tag{4}$$

Specifically, we can get Δw_i using Eq. (5):

$$\Delta w_i = -\eta_i (p_t - y_t)\frac{y_i}{T} \tag{5}$$

As an iterative learning algorithm, OWL-RFR computes the gradient repeatedly to acquire the local optimum for each stochastic testing instance. In the general sense, it updates leaves' weights over many iterations according to a test instance. More precisely, the weights of leaves, which the testing sample used for prediction, are continually updated until the iterative condition is no longer satisfied. The updating equation is given in Eq. (6):

$$\acute{w}_i = w_i + \Delta w_i \tag{6}$$

According to Eq. (5) and Eq. (6), the new weight, as described in Eq. (7) :

$$\acute{w}_i = w_i - \eta_i (p_t - y_t)\frac{y_i}{T} \tag{7}$$

3.4. Adaptive learning rate

The scheduling of learning rates often has a major impact on convergence speed in SGD training. It is well known, that the learning rate controls the descent velocity. Ultimately, the faster the speed, the faster the descent. However, certain situations must be considered. First, a faster velocity may cause an optimal path of descent being missed. Second, because one leaf node may be combined with other leaf nodes, to predict specific test instances during the prediction procedure, we should not consider only single testing. More specifically, we should consider the historical and current performance of the leaf node in addition to the current performance of the whole forests. Hence, we adopt a set of adaptive learning rate tactics for each leaf node, which are described in Eq. (8).

$$\eta_i^k = \frac{\eta_i}{\alpha k + N} \tag{8}$$

where k is the iteration number of weight learning for one testing sample, α is the decay rate of the iteration, N is the number of times that one leaf is used in the testing process, and η_i is the initial learning rate of a base learner. More specifically, η_i is the initial learning rate of the leaf currently used by this base learner, as described in Eq. (9).

$$\eta_i = \left|\eta_f - \eta_n\right| * \frac{\mu_i - \mu_{min}}{\mu_{min} - \mu_{max}} + \min\left(\eta_n, \eta_f\right) \tag{9}$$

where μ_i denotes the predictive ability of the base learner for the tth test instance (this is defined in Eq. (10)), μ_{min} is the minimum μ_i of all base learners, and μ_{max} is the maximum μ_i of all base learners. We preset **a** start learning rate η_n and final learning rate η_f. We then use min–max normalization to control the initial learning rate within the range[0, 1].

$$\mu_i = \min\left(\delta, \beta_j, \gamma\right) \tag{10}$$

where δ is the current predicted bias ratio for the whole forest, as described in Eq. (11), and β_j is the leaf node prediction bias ratio for the jth test sample when using the leaf node for prediction, this can be calculated using Eq. (12). Notably, $j \neq t$, where t is the sequence number of the test sample in all test sets, and j

is the sequence number of the test sample that uses the current leaf node to predict. γ refers to the whole historical bias ratio of this leaf node, across all tests using the current leaf node. This is calculated from the accumulated β_j, meaning that the leaves are used to predict for N test samples, so it is described by Eq. (13). We acquire a minimum of three values to represent the best performance of prediction, and we then convert that ability to the initial learning rate. Consequently, the worse the performance, the faster the learning rate. This complies with the notion that poor students need to work harder.

$$\delta = \frac{|p_t - y_t|}{y_t} \tag{11}$$

$$\beta_j = \frac{|y_i - y_t|}{y_t} \tag{12}$$

$$\gamma = \frac{\sum_{j=1}^{N} \beta_j}{N} \tag{13}$$

The rationale for this adaptive learning rate is choosing an appropriate bias ratio as a learning step size for each leaf according to its condition, and to prevent overfitting, as well as to accelerate convergence.

3.5. Online leaf-level weight learning algorithm

When a test instance arrives and starts the online weight update process, it will be in accordance with OWL-RFR, as described in Algorithm 1. At the end of the procedure, we will have new weights for the used leaf nodes from this kth prediction; some of the updated weights will then be used for the next prediction round.

To enhance readability, a table of symbols used in Algorithm 1, along with their meanings, is presented in Table 1.

Algorithm 1 OWL-RRF

```
Input:
    $T, \delta, \beta_j, \gamma, \eta_n, \eta_f, \epsilon, iters, \alpha, N, p_t, y_t, y_i, w_i$;
Output:
    $\acute{w}_i$;
1: $i = 1; k = 0; error = 0; error\_last = 0; loss = 0;$
2: for each $i \in [1, T]$ do
3:     $\mu_i = \min(\delta, \beta_j, \gamma)$;
4: end for
5: $\mu_{min} = \min(\mu_i)$;
6: $\mu_{max} = \max(\mu_i)$;
7: for each $i \in [1, T]$ do
8:     $\eta_i = |\eta_f - \eta_n| * \frac{\mu_i - \mu_{min}}{\mu_{min} - \mu_{max}} + \min(\eta_n, \eta_f)$;
9: end for
10: while $(loss \geq \epsilon) and (k \leq iters)$ do
11:     $p_t^k = 0$;
12:     $\eta_i^k = \frac{\eta_i}{\alpha k + N}$;
13:     for each $i \in [1, T]$ do
14:         $\Delta w_i = -\eta_i^k (p_t - y_t) \frac{y_i}{T}$;
15:         $\acute{w}_i = w_i + \Delta w_i$;
16:         $p_t^k += \frac{1}{T}(w_i y_i)$;
17:     end for
18:     $error = |p_t^k - y_t|$;
19:     $loss = |error\_last - error|$;
20:     $k = k + 1$;
21:     $error\_last = error$;
22: end while
```

Algorithm 1 describes how the weights of used leaves are updated, while aiming for less bias between the true value and

Table 1
List of Algorithm 1 symbols and meanings.

Symbol	Definition
i	The sequence number of the base learner
t	The sequence number of the test sample
k	The number of the iteration
j	The sequence number of the test sample that use the current leaf node to predict
T	The number of trees
δ	The predicted bias ratio of forests for current test
β_j	The predicted bias ratio of current leaf node for the jth test instance
γ	The historical bias ratio of this leaf node among all test instance that use this leaf node
μ_i	The predictive ability of the ith base learner for the tth test instance
μ_{min}	The minimum μ_i of all base learners
μ_{max}	The maximum μ_i of all base learners
η_n	The start learning rate
η_f	The final learning rate
η_i	The initial learning rate of a leaf node
p_t	The predicted value of RFs for the tth test instance
y_t	The true value of the tth test instance
y_i	The predicted value of the ith base learner for the tth test instance
N	The number of times a leaf node is used for prediction
α	The decay rate of iteration
ϵ	The accuracy of control iteration in the SGD process
$iters$	The maximum number of iterations in the SGD process
p_t^k	The predicted value in the kth iteration for the tth test instance
w_i	The weight of the used leaf node of the ith base learner for the current test
Δw_i	The incremental value of original weight
η_i^k	The learning rate of the used leaf node of the ith base learner in the kth interation
$\acute{w}_i$	The new weight of the used leaf node of the ith base learner

predicted value, during the prediction procedure. The updating process proceeds with test samples, until new samples no longer activate the weight updating algorithm, which has its threshold set at T_h. Naturally, the inner renewal process requires control parameters. Therefore, we set iteration control parameters, including accuracy of iteration: ϵ and the number of iterations $iters$. Line 1 initializes some parameters; Lines 2–9 compute the initial learning rate for each leaf node by Eq. (9); Lines 10–22 express the iterative process of weight learning from one test sample; Line 12 computes an adaptive learning rate of the leaf for the current iteration; Line 14 computes the incremental weight toward to the negative direction of the objective function; Line 15 updates the weight; Line 16 accumulates the prediction result from updated weights; Lines 18–21 recompute the control parameters.

4. Experiments

4.1. Benchmark datasets

The algorithms proposed in this paper are designed for scenarios in which data are delivered through continuous streaming. For comparative evaluation, we selected eight different sizes of benchmark datasets from the UCI [39] Machine Learning Repository and Kaggle data science community. These datasets, are Boston, California Housing(California), Concrete Compressive Strength (Concrete) [40], Airfoil Self-Noise(Airfoil), ParkinsonsTelemonitoring(Parkinsons) [41], Wine Quality(WineQuality) [42], SGEMM GPU kernel performance(Sgemm) [43] and Quality Prediction in a Mining Process(Mining)datasets from Kaggle. Their sizes, and numbers of features, are presented in Table 2.

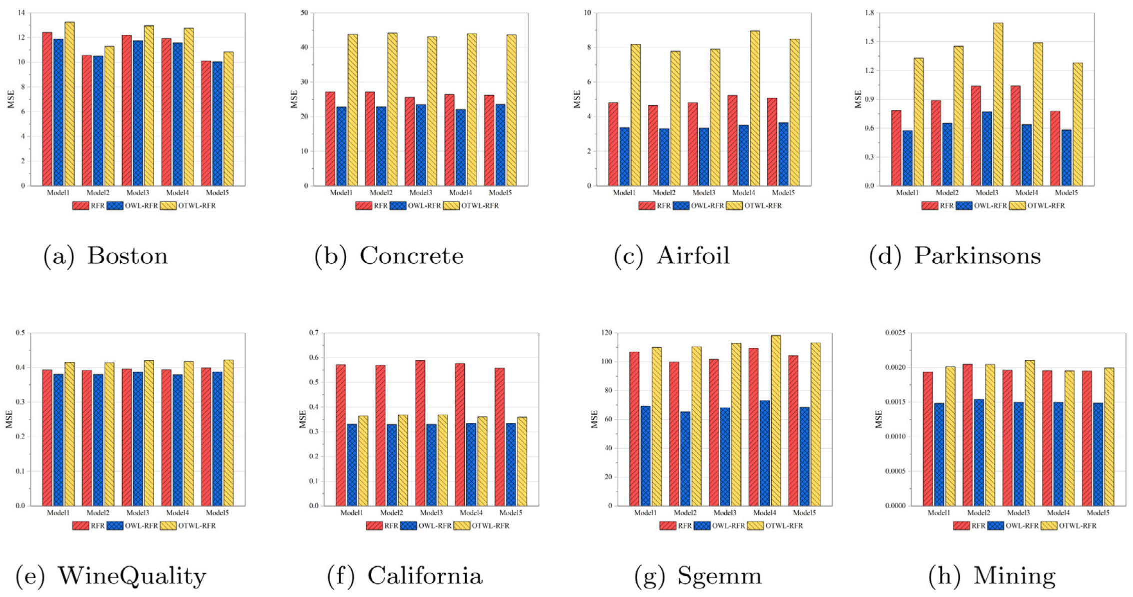


Fig. 2. The MSE of OWL-RFR vs. OTWL-RFR and RFR. Models 1 to 5 are the conventional RFs regression models trained through bagging sampling the same training dataset. The OWL-RFR and OTWL-RFR algorithms are executed based on these.

Table 2
Characteristics of the benchmark datasets used for the experimental evaluation.

Datasets	Size	#features
Boston	506	13
Concrete	1030	8
Airfoil	1502	5
WineQuality	4898	11
Parkinsons	5874	20
California	20640	8
Sgemm	241599	14
Mining	736282	22

Table 3
Parameters of learning threshold and rate.

Datasets	T_h	η_n	η_f	α
Boston	0.1	0.00001	0.0001	0.001
Concrete	0.001	0.01	0.1	1
Airfoil	0.001	0.0001	0.001	0.001
WineQuality	0.01	0.01	0.1	0.1
Parkinsons	0.01	0.01	0.06	0.00001
California	0.1	0.1	1	0.01
Sgemm	0.001	0.000001	0.00001	0.0001
Mining	0.001	0.01	0.1	0.003

4.2. Parameters

We implemented the traditional RFs regression(RFR) [1] in C++. Our OWL-RFR scripts were also implemented in C++, based on traditional RFR; we used the Mondrian Forests(MF) [25] implementation in Python. For each dataset, we randomly divided the dataset into a training set and testing set with a ratio of approximately 7:3, and we set the number of trees of RFR and MF to be 30, and the maximal depth of trees to be 100. In RFR, the maximum number of samples of leaf nodes was 5, with the candidate split features about one half of the datasets' features. In MF, the maximum number of samples of leaf nodes was set to 5 for the Mining dataset and 2 for the others, to obtain better results. In OWL-RFR, we set ϵ=1e−6, and *iters*=1e+4 for all datasets. Other key parameters of the learning threshold and rate are listed in Table 3. As shown in the table, η_f is ten times η_n in almost all datasets except for Parkinsons dataset.

4.3. Performance

We assume that the true response value is available immediately after prediction, to satisfy the online regression learning scenario. For performance evaluation, we trained five different RFR and MF models through bagging sampling the same training data in each dataset, and used them to predict the same testing data; we executed the OWL-RFR algorithm based on the five RFR models. We compared the performance of RFR, MF, and OWL-RFR using two indicators: Mean Squared Error(MSE) and R^2 Score coefficient of determination. An MSE close to 0 implies high accuracy, and R^2 Score close to 1 indicates high goodness of fit. As shown in Table 4, our OWL-RFR method attains results that are clearly superior to the simple averaging RFR and the online MF across most datasets, with Parkinsons the only exception, for which MF achieves a better result. This shows that the online weight learning schema of leaves is useful for improving the predictive performance of traditional RFs regression, and that learning while predicting can yield appropriate weight for streaming data.

4.4. Leaf-level weight vs. tree-level weight

In Section 3.2, we discussed how leaf-level weight includes long-term memory in our approach, in contrast to tree-level weight, which affords only immediate memory. We verified this theoretical analysis experimentally. In the experiment, we compared the MSE indicators of OWL-RFR, against RFR and Online Tree Weight Learning for RFs Regression(OTWL-RFR), which use the online weight learning method applied to trees. We conducted experiments based on the five models that described in Section 4.3 on all datasets. As Fig. 2 shows, the MSE of OWL-RFR was less than that of OTWL-RFR and RFR for all datasets. This proves that our approach, using weights on leaves instead of on trees, is better adapted to the online learning paradigm. It also proves that leaf-level weight can indeed serve as long-term memory, and the long-term memory of leaf-level weight is more effective than the immediate memory from tree-level weight.

Table 4
MSE and R2 comparisons (mean±sd) for different methods in benchmark datasets.

Datasets	Indicators	RFR	MF	OWL-RFR
Boston	MSE	11.44 ± 0.92	16.39 ± 0.26	**11.14 ± 0.74**
	R^2 Score	0.86 ± 0.01	0.80 ± 0.003	**0.87 ± 0.01**
Concrete	MSE	26.50 ± 0.60	46.53 ± 1.15	**22.93 ± 0.54**
	R^2 Score	0.90 ± 0.002	0.82 ± 0.004	**0.91 ± 0.002**
Airfoil	MSE	4.91 ± 0.21	4.54 ± 0.18	**3.44 ± 0.13**
	R^2 Score	0.88 ± 0.005	0.89 ± 0.004	**0.92 ± 0.003**
WineQuality	MSE	0.39 ± 0.002	0.43 ± 0.001	**0.38 ± 0.003**
	R^2 Score	0.52 ± 0.003	0.48 ± 0.001	**0.53 ± 0.004**
Parkinsons	MSE	0.92 ± 0.10	**0.25 ± 0.03**	0.66 ± 0.06
	R^2 Score	0.99 ± 0.002	**0.99 ± 0.0004**	0.99 ± 0.001
California	MSE	0.57 ± 0.01	0.87 ± 0.03	**0.33 ± 0.001**
	R^2 Score	0.60 ± 0.009	0.49 ± 0.02	**0.75 ± 0.003**
Sgemm	MSE	104.38 ± 3.39	793.48 ± 9.20	**68.78 ± 2.50**
	R^2 Score	0.99 ± 2.5E−5	0.99 ± 6.7E−5	**0.99 ± 1.8E−5**
Mining	MSE	1.97E−03 ± 4E−05	1.62E−2 ± 2E−4	**1.50E−3 ± 2E−5**
	R^2 Score	0.99 ± 3.1E−5	0.99 ± 1.6E−4	**0.99 ± 1.7E−5**

4.5. Adaptive vs. static learning rate

In the experiment, we used adaptive and static learning rates separately to calculate MSE and R^2 Score of the same test dataset on the five different RFs models. We set empirical value for the static learning rate in the different datasets, and set the start learning rate η_n of adaptive learning equal to the static learning rate. As Table 5 shows, our adaptive learning rate approach obtained better results for MSE than the static learning rate in most datasets, except for the Mining dataset. This indicates that our adaptive adjust strategy, employing leaves' current and historical predictive deviations, adapts effectively to the demands of streaming data. Leaves with good performance learn slowly because they already have a predictive capacity, to some extent. Conversely, leaves that perform poorly will learn to adapt more quickly as they encounter changes in the stream. However, a static learning rate cannot detect any changes in the performance of predictors in the data stream. Notably, the Mining dataset contains large amounts of training and testing data, so the prediction bias is very small during testing. Therefore, in this case, the adaptive learning rate is little different from the static one.

4.6. Convergence

To evaluate the convergence of OWL-RFR, we compared the MSE of our method with that of off-line RFs. We selected one model randomly from the five trained models and conducted testing. Fig. 3 shows the cumulative MSE of RFs, compared with that of OWL-RFR, as the number of testing samples increased, on different datasets. It shows that the cumulative MSE is positively correlated with the number of test samples, but we can find that the cumulative MSE of OWL-RFR is always less than that of off-line RFs. Therefore, the *Regret*(described in Section 3.1), between OWL-RFR and batch RFs with an increasing number of testing samples, should be sublinear. This result indicates that OWL-RFR is convergent.

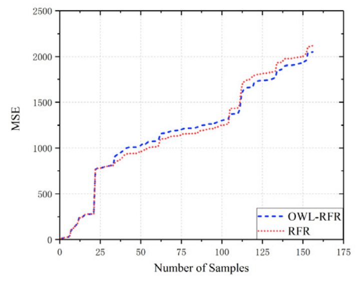
4.7. Stability

To prove that the OWL-RFR approach is stable, we compared the regret of the model on the same but disordered testing sets. We reordered the sequence of the same testing set of each dataset, and we obtained five disordered testing sets for every dataset. We then used the OWL-RFR algorithm to predict these testing sequences. According to Eq. (1), we used RFR as the target model, and obtained a regret value of OWL-RFR compared to RFR.

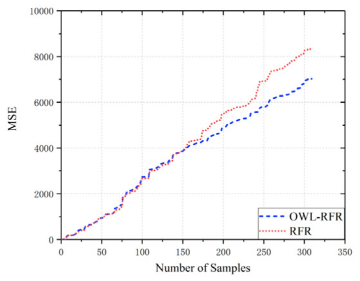
Table 5
Adaptive vs. static learning rate.

Dataset	Model	MSE		R^2 Score	
		Static	Adaptive	Static	Adaptive
Boston	Model1	10.150	**10.039**	0.877	**0.878**
	Model2	10.521	**10.501**	0.873	0.873
	Model3	11.853	**11.733**	0.856	**0.858**
	Model4	11.664	**11.565**	0.858	**0.860**
	Model5	12.277	**11.880**	0.851	**0.856**
Concrete	Model1	22.942	**22.769**	0.913	0.913
	Model2	22.862	**22.783**	0.913	0.913
	Model3	23.908	**23.504**	0.913	0.913
	Model4	22.297	**22.078**	0.915	**0.916**
	Model5	23.789	**23.541**	0.909	**0.910**
Airfoil	Model1	3.360	**3.358**	0.919	0.919
	Model2	3.355	**3.314**	0.919	**0.920**
	Model3	3.345	**3.340**	0.919	**0.920**
	Model4	3.570	**3.517**	0.914	**0.915**
	Model5	3.713	**3.646**	0.911	**0.912**
Parkinsons	Model1	0.586	**0.582**	0.991	0.991
	Model2	0.576	**0.577**	0.991	0.991
	Model3	0.654	**0.652**	0.990	0.990
	Model4	0.773	**0.768**	0.988	0.988
	Model5	0.639	**0.638**	0.990	0.990
WineQuality	Model1	0.387	**0.380**	0.528	**0.536**
	Model2	0.388	**0.380**	0.528	**0.537**
	Model3	0.396	**0.386**	0.517	**0.529**
	Model4	0.385	**0.379**	0.530	**0.537**
	Model5	0.394	**0.387**	0.520	**0.528**
California	Model1	0.335	**0.331**	0.765	**0.768**
	Model2	0.336	**0.330**	0.765	**0.769**
	Model3	0.334	**0.331**	0.766	**0.768**
	Model4	0.338	**0.333**	0.763	**0.766**
	Model5	0.337	**0.333**	0.764	**0.766**
Sgemm	Model1	70.923	**69.105**	0.999	0.999
	Model2	66.678	**65.210**	1.000	1.000
	Model3	69.468	**68.139**	0.999	**1.000**
	Model4	74.432	**73.006**	0.999	0.999
	Model5	70.272	**68.458**	0.999	**1.000**
Mining	Model1	0.001	0.001	0.999	0.999
	Model2	0.002	0.001	0.999	0.999
	Model3	0.001	0.001	0.999	0.999
	Model4	0.001	0.001	0.999	0.999
	Model5	0.001	0.001	0.999	0.999

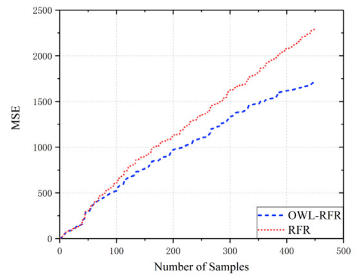
The regret is the difference in the accumulated loss of OWL-RFR and RFR. We obtained five *Regret* curves of testing sets. As Fig. 4 shows, the *Regret* curve of "TestingSet-x" reduces as the number of samples increases. This shows that the accumulated loss of



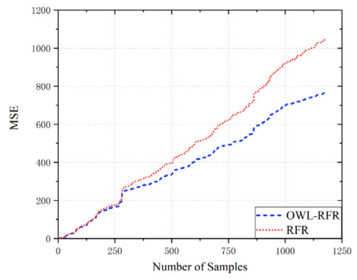
(a) Boston



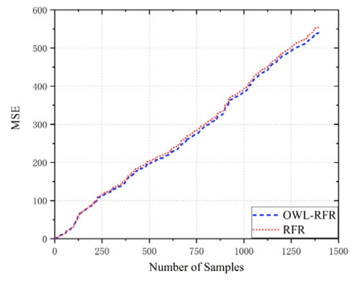
(b) Concert



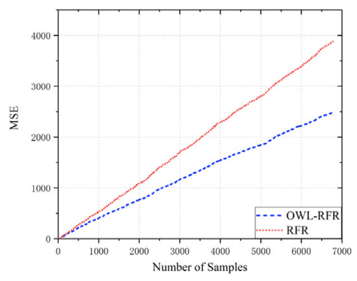
(c) Airfoil



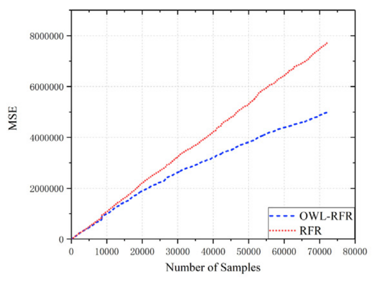
(d) Parkinsons



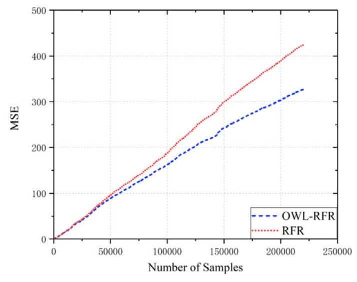
(e) WineQuality



(f) California



(g) Sgemm



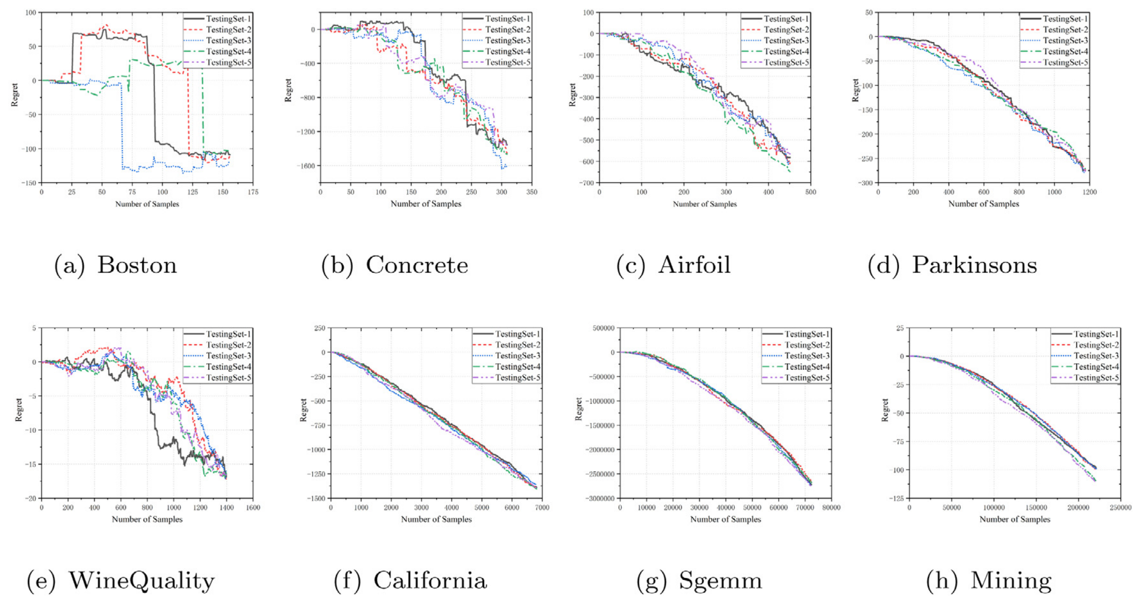
(h) Mining

Fig. 3. The cumulative MSE of RFs and that of OWL-RFR with the sequence of test samples.

OWL-RFR is less than that of RFR with the same number of testing samples, and further demonstrates that *Regret* is convergent with the number of samples. Indeed, its tendency is not affected by feeding the testing samples in different orders, which proves that our approach is always stable.

5. Conclusion

We introduce an novel online weight learning algorithm(OWL-RFR), the basis of which is more typically used for the off-line training of RFs for regression prediction. We endow the RFs with memory by weight when predicting. We analyze the difference between each leaf weight's long-term memory and the tree weight's immediate memory in streaming data, to prove the effectiveness of the former. Experiments on commonly used machine learning datasets show that OWL-RFR significantly outperforms traditional RF and online growing MF, in terms of predictive performance(MSE and R^2 Score) across most test datasets. Additionally, we prove through a comparative experiment that long-term memory support of leaf weight is indeed more useful.



(a) Boston (b) Concrete (c) Airfoil (d) Parkinsons

(e) WineQuality (f) California (g) Sgemm (h) Mining

Fig. 4. The Regret of five testing set based on the same trained model. The "TestingSet-1" to "TestingSet-5" are the regret curves based on these disordered testing sets.

Additionally, we find that our adaptive learning rate is more efficient than the static rate. Furthermore, we utilize *Regret* to prove that our approach is convergent and stable. In future work, we aim to extend this research to provide an online growing schema of RFs based on leaf weight. We also hope to extend the approach to the classification task.

CRediT authorship contribution statement

Yuan Zhong: Conceptualization, Methodology, Software, Writing - review & editing. **Hongyu Yang:** Supervision. **Yanci Zhang:** Validation, Investigation. **Ping Li:** Formal analysis, Writing - review & editing.

Declaration of competing 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.

Acknowledgments

The authors acknowledges the support from National Natural Science Foundation of China 61472261 and 61873218, Applied Basic Research Foundation of Sichuan Provincial Science and Technology Department, China (No. 18YYJC1147), Nanchong Science and Technology Department, China (18SXHZ0009) and Southwest Petroleum University innovation base, China (No. 642).

References

[1] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
[2] A. Bosch, A. Zisserman, X. Munoz, Image classification using random forests and ferns, in: 2007 IEEE 11th International Conference on Computer Vision, Ieee, 2007, pp. 1–8.
[3] C. Leistner, A. Saffari, J. Santner, H. Bischof, Semi-supervised random forests, in: 2009 IEEE 12th International Conference on Computer Vision, IEEE, 2009, pp. 506–513.
[4] A. Saffari, C. Leistner, J. Santner, M. Godec, H. Bischof, On-line random forests, in: 2009 ieee 12th international conference on computer vision workshops, iccv workshops, IEEE, 2009, pp. 1393–1400.
[5] P. Domingos, G. Hulten, Mining high-speed data streams, in: Proceeding of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002.
[6] Albert Bifet, Geoffrey Holmes, Bernhard Pfahringer, Richard Kirkby, Ricard Gavaldà, New ensemble methods for evolving data streams, in: Proceedings of 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2009, pp. 139–148.
[7] Elena Ikonomovska, João Gama, Sašo Džeroski, Learning model trees from evolving data streams, Data Mining Knowl. Discov. 23 (1) (2011) 128–168.
[8] J.R. Quinlan, C4.5: Programs for Machine Learning, Elsevier, 2014.
[9] L.I. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and regression trees (cart), Encycl. Ecol. 40 (3) (1984) 358.
[10] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.
[11] M.P. Perrone, L.N. Cooper, When Networks Disagree: Ensemble Methods for Hybrid Neural Networks, Tech. Rep., Brown Univ Providence RI Inst for Brain and Neural Systems, 1992.
[12] C. Chen, A. Liaw, L. Breiman, et al., Using random forest to learn imbalanced data, Univ. Calif. Publ. Bot. 110 (1-12) (2004) 24.
[13] D. Amaratunga, J. Cabrera, Y.-S. Lee, Enriched random forests, Bioinformatics 24 (18) (2008) 2010–2014.
[14] Z.S. Wei, K. Han, J.Y. Yang, H.B. Shen, D.J. Yu, Protein–protein interaction sites prediction by ensembling svm and sample-weighted random forests, Neurocomputing 193 (C) (2016) 201–212.
[15] J. Maudes, J.J. Rodríguez, C. García-Osorio, N. García-Pedrajas, Random feature weights for decision tree ensemble construction, Inform. Fusion 13 (1) (2012) 20–30.
[16] Q. Wu, Y. Ye, H. Zhang, M.K. Ng, S.-S. Ho, Forestexter: An efficient random forest algorithm for imbalanced text categorization, Knowl.-Based Syst. 67 (2014) 105–116.
[17] H.B. Li, W. Wang, H.W. Ding, J. Dong, Trees weighting random forest method for classifying high-dimensional noisy data, in: 2010 IEEE 7th International Conference on E-Business Engineering, IEEE, 2010, pp. 160–163.
[18] S.J. Winham, R.R. Freimuth, J.M. Biernacka, A weighted random forests approach to improve predictive performance, Stat. Anal. Data Mining: ASA Data Sci. J. 6 (6) (2013) 496–505.
[19] F. Rosenblatt, The perceptron: A probabilistic model for information storage and organization in the brain, Psychol. Rev. 65 (6) (1958) 386–408.
[20] Y. Singer, J.C. Duchi, Efficient learning using forward-backward splitting, in: Advances in Neural Information Processing Systems, 2009, pp. 495–503.
[21] L. Xiao, Dual averaging methods for regularized stochastic learning and online optimization, J. Mach. Learn. Res. 11 (Oct) (2010) 2543–2596.
[22] H.B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, et al., Ad click prediction: A view from the trenches, in: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2013, pp. 1222–1230.
[23] A.-P. Wang, G.-W. Wan, Z.-Q. Cheng, S.-K. Li, Incremental learning extremely random forest classifier for online learning, J. Softw. 22 (9) (2011) 2059–2074.
[24] A.P. Cassidy, F.A. Deviney, Calculating feature importance in data streams with concept drift using online random forest, in: 2014 IEEE International Conference on Big Data (Big Data), IEEE, 2014, pp. 23–28.

[25] B. Lakshminarayanan, D.M. Roy, Y.W. Teh, Mondrian forests: Efficient online random forests, Adv. Neural Inf. Process. Syst. 4 (2014) 3140–3148.

[26] M. Ristin, M. Guillaumin, J. Gall, L. Van Gool, Incremental learning of random forests for large-scale image classification, IEEE Trans. Pattern Anal. Mach. Intell. 38 (3) (2015) 490–503.

[27] A. Paul, D.P. Mukherjee, P. Das, A. Gangopadhyay, A.R. Chintha, S. Kundu, Improved random forest for classification, IEEE Trans. Image Process. 27 (8) (2018) 4012–4024.

[28] H. Kim, H. Kim, H. Moon, H. Ahn, A weight-adjusted voting algorithm for ensembles of classifiers, J. Korean Statist. Soc. 40 (4) (2011) 437–449.

[29] A.V. Zhukov, D.N. Sidorov, A.M. Foley, Random forest based approach for concept drift handling, in: International Conference on Analysis of Images, Social Networks and Texts, Springer, 2016, pp. 69–77.

[30] R. Elwell, R. Polikar, Incremental learning of concept drift in nonstationary environments, IEEE Trans. Neural Netw. 22 (10) (2011) 1517–1531.

[31] I. Frías-Blanco, J.d. Campo-ávila, G. Ramos-Jiménez, A.C. Carvalho, A. Ortiz-Díaz, R. Morales-Bueno, Online adaptive decision trees based on concentration inequalities, Knowl.-Based Syst. 104 (2016) 179–194.

[32] C. Zhang, J. Bi, S. Xu, E. Ramentol, G. Fan, B. Qiao, H. Fujita, Multi-imbalance: an open-source software for multi-class imbalance learning, Knowl.-Based Syst. 0 (2019).

[33] J.Z. Kolter, M.A. Maloof, Using Additive Expert Ensembles to Cope with Concept Drift, 2005.

[34] P. Kadlec, Local learning-based adaptive soft sensor for catalyst activation prediction, Aiche J. 57 (2011).

[35] S. Gomes Soares, R. Araújo, An on-line weighted ensemble of regressor models to handle concept drifts, Eng. Appl. Artif. Intell. 37 (2015) 392–406.

[36] A. Bifet, G. Holmes, B. Pfahringer, Leveraging Bagging for Evolving Data Streams, 2010.

[37] J. Gama, R. Rocha, P. Medas, Accurate decision trees for mining high-speed data streams, in: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24-27, 2003, p. 2003.

[38] S. Shalev-Shwartz, et al., Online learning and online convex optimization, Found. Trends Mach. Learn. 4 (2) (2012) 107–194.

[39] D. Dua, C. Graff, UCI machine learning repository, 2017, URL http://archive.ics.uci.edu/ml.

[40] I.-C. Yeh, Modeling of strength of high-performance concrete using artificial neural networks, Cement Concr. Res. 28 (12) (1998) 1797–1808.

[41] A. Tsanas, M.A. Little, P.E. McSharry, L.O. Ramig, Accurate telemonitoring of Parkinson's disease progression by noninvasive speech tests, IEEE Trans. Biomed. Eng. 57 (4) (2009) 884–893.

[42] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, J. Reis, Modeling wine preferences by data mining from physicochemical properties, Decis. Support Syst. 47 (4) (2009) 547–553.

[43] C. Nugteren, V. Codreanu, Cltune: a generic auto-tuner for opencl kernels, in: 2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip, IEEE, 2015, pp. 195–202.