BoostTree and BoostForest for Ensemble Learning

Changming Zhao, Dongrui Wu, *Fellow, IEEE*, Jian Huang, *Senior Member, IEEE*, Ye Yuan, *Senior Member, IEEE*, Hai-Tao Zhang, *Senior Member, IEEE*, Ruimin Peng, and Zhenhua Shi

Abstract—Bootstrap aggregating (Bagging) and boosting are two popular ensemble learning approaches, which combine multiple base learners to generate a composite model for more accurate and more reliable performance. They have been widely used in biology, engineering, healthcare, etc. This article proposes BoostForest, which is an ensemble learning approach using BoostTree as base learners and can be used for both classification and regression. BoostTree constructs a tree model by gradient boosting. It increases the randomness (diversity) by drawing the cut-points randomly at node splitting. BoostForest further increases the randomness by bootstrapping the training data in constructing different BoostTrees. BoostForest generally outperformed four classical ensemble learning approaches (Random Forest, Extra-Trees, XGBoost and LightGBM) on 35 classification and regression datasets. Remarkably, BoostForest tunes its parameters by simply sampling them randomly from a parameter pool, which can be easily specified, and its ensemble learning framework can also be used to combine many other base learners.

Index Terms—Bagging, BoostForest, BoostTree, boosting, ensemble learning

1 Introduction

Ensemble learning [1], [2] trains multiple base learners to exploit the relationship between a set of covariates (features) and a response (label), and then combines them to produce a strong composite learner with better generalization performance. It has been successfully used in biology [3], [4], [5], engineering [6], [7], [8], healthcare [9], [10], etc.

For example, in biology, Wang et al. [3] used an ensemble of neural networks to emulate mechanism-based biological models. They found that the ensemble is more accurate than an individual neural network, and the consistency among the individual models can indicate the error in prediction. In Moon exploration, Yang et al. [7] used ensemble transfer learning and Chang'E data for automatic Moon surface impact crater detection and age estimation. They successfully identified 109,956 new craters from 7,895 training samples. In healthcare, Agius et al. [10] developed a Chronic Lymphocytic Leukemia (CLL) treatment-infection model, an ensemble of 28 machine learning algorithms, to identify patients at risk of infection or CLL treatment within two years of diagnosis.

One of the most popular algorithms for constructing the base learners is the decision tree [11], [12], [13]. Two common approaches for constructing the composite learner are bootstrap aggregating (Bagging) and boosting.

Bagging [14] connects multiple base learners in parallel to reduce the variance of the ensemble. Each base learner is trained using the same learning algorithm on a bootstrap replica, which draws N (the size of the original training set) samples with replacement from the original training set. The outputs of these base learners are then aggregated by majority voting (for classification) or averaging (for regression) to obtain the final output. To achieve robust performance, the base learners in an ensemble should be both accurate and diverse [15], [16], [17].

There are many approaches to increase the accuracy of base learners in an ensemble. Combining the advantages of tree models and linear models can greatly improve the model's learning ability, which is the main idea of model trees. M5 [18] constructs a linear regression function at each

- *Changming Zhao, Dongrui Wu, and Ruimin Peng are with the Key Laboratory of the Ministry of Education for Image Processing and Intelligent Control, School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China, and also with the Shenzhen Huazhong University of Science and Technology Research Institute, Shenzhen, Guangdong 518029, China. E-mail: {cmzhao, rmpeng2019}@hust.edu.cn, drvu09@gmail.com.*
- *Jian Huang and Ye Yuan are with the Key Laboratory of the Ministry of Education for Image Processing and Intelligent Control, School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: {huang_jan, yye}@hust.edu.cn.*
- *Hai-Tao Zhang is with the Key Laboratory of the Ministry of Education for Image Processing and Intelligent Control, School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China, and with the Autonomous Intelligence Unmanned Systems Engineering Research Center of Ministry of Education of China, Beijing 100089, China, and also with the State Key Lab of Digital Manufacturing Equipment and Technology, Wuhan, Hubei 430074, China. E-mail: zht@mail.hust.edu.cn.*
- *Zhenhua Shi is with the Second Ship Design and Research Institute, Wuhan, Hubei 430205, China. E-mail: zhenhuashi95@foxmail.com.*

Manuscript received 18 December 2021; revised 9 August 2022; accepted 5 December 2022. Date of publication 7 December 2022; date of current version 5 June 2023.
This work was supported in part by the Science, Technology and Innovation Commission of Shenzhen Municipality under Grant 2021Szvup093, and in part by the National Natural Science Foundation of China under Grants 61873321, 62225306, and U2141235.
(Corresponding author: Dongrui Wu.)
Recommended for acceptance by M. Dudik.
This article has supplementary downloadable material available at https://doi.org/10.1109/TPAMI.2022.3227370, provided by the authors.
Digital Object Identifier no. 10.1109/TPAMI.2022.3227370

0162-8828 © 2022 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

leaf to approximate the target function for high fitting ability. When a new sample comes in, it is first sorted down to a leaf, then the linear model at that leaf is used to predict its output. M5P (aka M5′) [19] trains a linear model at each node of a pruned tree to reduce the risk of over-fitting. More sophisticated regression algorithms, e.g., Ridge Regression (RR) [2], Extreme Learning Machine (ELM) [20], Support Vector Regression (SVR) [21], and Neural Network [22], have rarely been used in model trees. A possible reason is that they have some hyper-parameters to tune, e.g., the regularization coefficient and the number of nodes in the hidden layers. It is an NP-complete problem to simultaneously determine the structure of the model tree and the parameters of each node model [23]. A common approach is cross-validation, but it is very time-consuming. Thus, it is desirable to develop a strategy that can make the model tree more compatible with these more sophisticated (and hence potentially better performing) regression models.

There are also many approaches to increase the diversity of base learners in an ensemble, which can be divided into three categories [24]: 1) Sample-based strategies, which train each base learner on a different subset of samples, and thus are scalable to Big Data. For example, Bagging uses bootstrap sampling to obtain different subsets to train the base learners, and AdaBoost [25] uses adaptive sample weights (larger weights for harder samples) in generating a new base learner. 2) Feature-based strategies, which train each base learner on different subsets of features, and thus are scalable to high dimensionality. For example, each decision tree in Random Forest (RandomForest) [26], [27], [28] selects the feature to be split from a random subset of features, instead of all available features. Similarly, each decision tree in Extremely Randomized Trees (Extra-Trees) [29] splits nodes by drawing the cut-points completely randomly. 3) Parameter-based strategies. If the base learners are sensitive to their parameters, then setting different parameters can improve the diversity. For example, different hidden layer weights can be used to initialize diverse neural networks. Interestingly, these three categories of diversity enhancement strategies are complementary; so, they can be combined for better performance.

Boosting [25], [30], [31], the driving force of Gradient Boosting Machine (GBM) [11], can be used to reduce the bias of an ensemble. It is an incremental learning process, in which a new base learner is built to compensate the error of previously generated learners. Each new base learner is added to the ensemble in a forward stage-wise manner. As the boosting algorithm iterates, base learners generated at later iterations tend to focus on the harder samples. Mason et al. [32] described boosting from the viewpoint of gradient descent and regarded boosting as a stage-wise learning scheme to optimize different objective functions iteratively. FilterBoost [33] is a variant of AdaBoost [25], which fits an additive logistic regression model and minimizes the negative log likelihood step-by-step. DeepBoosting [34] can use very deep decision trees as base learners. Popular implementations of GBM, e.g., XGBoost [12] and LightGBM [35], have been successfully used in many applications [36], [37], [38], [39]. However, traditional boosting approaches [11], [12], [35] often have many parameters and thus require cross-validation, which is unreliable on small datasets, and time-consuming on Big Data. It is desirable to develop an algorithm that has very few parameters to tune and is robust to them.

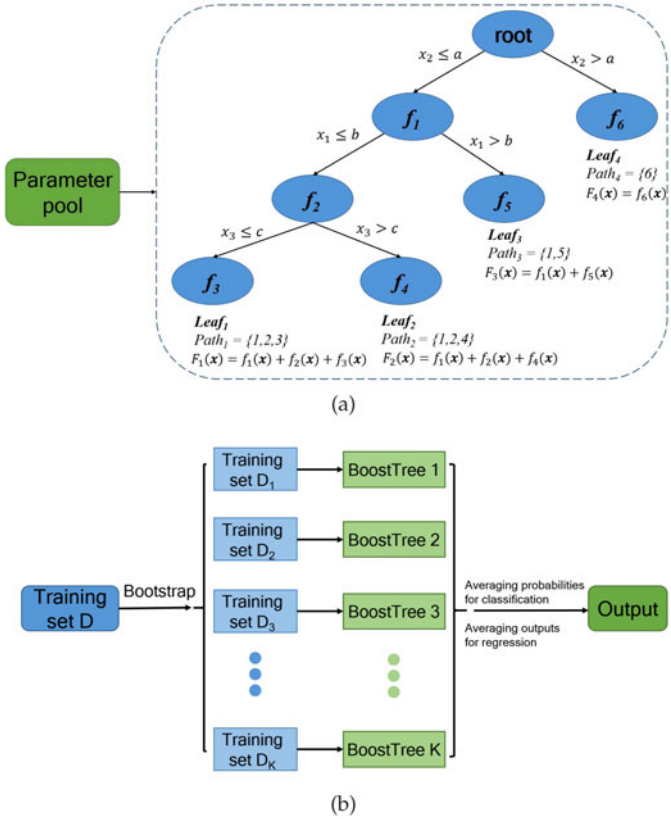


Fig. 1. (a) BoostTree with four leaves. For a given input, BoostTree first sorts it down to a leaf $q(\boldsymbol{x})$, then computes the final prediction by summing up the outputs of all node models along the path (given by $\mathrm{Path}_{q(\boldsymbol{x})}$) from the root to the leaf. The parameters of BoostTree are randomly selected from a parameter pool. (b) BoostForest with K BoostTrees. Bootstrap is used to obtain K replicas of the training set.

This paper proposes BoostForest, which integrates boosting and Bagging for both classification and regression. Our main contributions are:

1) We propose a novel decision tree model, BoostTree, that integrates GBM into a single decision tree, as shown in Fig. 1a. BoostTree uses the node function, e.g., RR, ELM, or SVR, to train a linear or nonlinear regression model at each node for regression or binary classification, or J regression models for J-class classification, where J is the number of classes. For a given input, BoostTree first sorts it down to a leaf, then computes the final prediction by summing up the outputs of all node models along the path from the root to that leaf. Similar to Extra-Trees, BoostTree increases the randomness (diversity) by drawing the cut-points randomly at node splitting.
2) We propose a novel parameter setting strategy, random parameter pool sampling, which makes BoostTree easier to tune its hyper-parameters than traditional approaches. In this strategy, the parameters of BoostTree are not specific values, but random samples from candidate sets stored in a parameter pool. Each time a root node is generated, BoostTree randomly selects its hyper-parameters from the parameter pool.

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

3) Using BoostTrees as base learners, we propose a novel ensemble learning approach, BoostForest, as shown in Fig. 1b. It first uses bootstrap to obtain multiple replicas of the original training set, and then trains a BoostTree on each replica. BoostForest uses the parameter pool sampling strategy to simplify its hyper-parameter tuning process, and outperforms several popular ensemble learning approaches. Moreover, it represents a very general ensemble learning framework, whose base learners can be any tree model, e.g., BoostTree, M5P, Extra-Tree (a single decision tree in Extra-Trees), or Logistic Model Tree (LMT) [40], or even a mixture of different models.

The remainder of this paper is organized as follows: Section 2 introduces some related work. Section 3 describes the details of our proposed BoostTree and BoostForest. Section 4 presents experimental results to demonstrate the superiority of BoostForest over several popular ensemble learning approaches on 35 classification and regression datasets. Finally, Section 5 draws conclusions and points out some future research directions.

2 Related Work

GBM [11] generates an ensemble via an iterative process. It uses Newton's method to decompose the original classification or regression problem into multiple sub-regression problems, and solves them iteratively. Given a dataset $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$ with N training samples, where $\boldsymbol{x}_n \in \mathbb{R}^{D\times 1}$ and D is the feature dimensionality, an ensemble ϕ generated by GBM uses K base learners $\{f_k\}_{k=1}^{K}$ to predict the output:

$$\hat{y}_n = \phi(\boldsymbol{x}_n) = \sum_{k=1}^{K} f_k(\boldsymbol{x}_n). \tag{1}$$

LogitBoost (Supplementary Algorithm 1, available online [41] is a popular implementation of GBM to optimize logistic regression. In each iteration, LogitBoost first computes the pseudo-labels and weights using the Newton (for binary classification) or quasi-Newton (for multi-class classification) method, and then updates the ensemble by adding a new regression model (for binary classification) or J new regression models [for J-class ($J > 2$) classification], which are trained to fit the pseudo-labels by weighted least-squares regression. If the original problem is regression, then in each GBM iteration, the pseudo-label is the residual between the true value and the prediction, and all samples have the same weight.

LogitBoost further converts the GBM output into a probability:

$$\begin{cases} p(\boldsymbol{x}_n) = \text{sigmoid}(\hat{y}_n), & J = 2 \\ \boldsymbol{p}(\boldsymbol{x}_n) = [p^1(\boldsymbol{x}_n), \ldots, p^J(\boldsymbol{x}_n)], & J > 2 \end{cases}, \tag{2}$$

where

$$\text{sigmoid}(\hat{y}_n) = \frac{1}{1+e^{-\hat{y}_n}}, \tag{3}$$

$$p^j(\boldsymbol{x}_n) = \text{softmax}^j(\hat{\boldsymbol{y}}_n) = \frac{e^{\hat{y}_n^j}}{\sum_{i=1}^{J} e^{\hat{y}_n^i}}, \tag{4}$$

in which $\hat{\boldsymbol{y}}_n = [\hat{y}_n^1, \hat{y}_n^2, \ldots, \hat{y}_n^J]^T \in \mathbb{R}^{J\times 1}$ is the output vector for J-class ($J > 2$) classification.

Malerba et al. [42] proposed Stepwise Model Tree Induction (SMOTI) to construct model trees gradually for regression. When the tree is growing, SMOTI adds more and more variables to the leaf model to approximate the target function. At each step, it adds either a splitting node or a regression node to the tree. A splitting node partitions the sample space, whereas a regression node removes the linear effect of the variables already included in its parent node and performs linear regression on one variable.

Different from GBM, LMT [40] generates only one tree instead of multiple trees for predicting. LMT extends model trees from regression to classification, by integrating LogitBoost into the tree model. The final model at a leaf consists of all node models along the path from the root to that leaf.

Assume an LMT has trained a function f_m for the m-th node. For an input $\boldsymbol{x}_n$, LMT first identifies $q(\boldsymbol{x}_n)$, the leaf node it belongs to, and then all f_m along the path from the root to that leaf node are summed up to predict the output, i.e.,

$$\hat{y}_n = F(\boldsymbol{x}_n) = \sum_{m\in \text{Path}_{q(\boldsymbol{x}_n)}} f_m(\boldsymbol{x}_n), \tag{5}$$

where $\text{Path}_{q(\boldsymbol{x}_n)}$ is the collection of the node indices along the path from the root to the leaf node $q(\boldsymbol{x}_n)$. Similar to LogitBoost, LMT can use (2) to output the probability.

SimpleLogistic [40] (a variant of LogitBoost) incrementally refines the linear logistic model. In each iteration, instead of using all features to perform linear regression, SimpleLogistic uses only one feature to train the model. In this way, only relevant features are selected, and the risk of over-fitting is reduced.

Gradient Boosting with Piece-wise Linear Regression Trees (GBDT-PL) [43] is a variant of XGBoost, which combines multiple Piece-wise Linear Regression Trees (PL-Trees) to predict the output. Similar to LMT, a PL-Tree fits a linear model at each node in an additive manner.

Webb [44] proposed MultiBoosting, which combines wagging (a variant of Bagging) and Adaboost [25] to reduce both the bias and the variance. It uses AdaBoost to generate a sub-committee of decision trees. By wagging a set of sub-committees, MultiBoosting may achieve lower error than AdaBoost.

RandomForest [26] uses two techniques to improve the diversity of each tree, and hence reduces over-fitting: 1) Bagging, i.e., each tree is trained with a bootstrap replica drawn from the original training set; and, 2) feature subsampling, i.e., for each node of the tree, a subset of the features is randomly selected from the complete feature set, then an optimal feature is selected from the subset to split the node.

Extra-Trees [29] splits nodes in a more random way than RandomForest. It first randomly draws one cut-point for each feature, and then selects the cut-point with the largest splitting gain to split a node. Extra-Trees reduces over-fitting by trading the base learners' accuracy for diversity.

TAO [45] first initializes the tree with a given depth by using CART [13] or random parameters, and then circularly

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

updates each decision or leaf node to optimize the objective function. It trains a binary classifier at each decision node to classify a sample into the left or right child node, and a constant label (TAO-c) or linear softmax classifier (TAO-l) at each leaf to predict the output. Its decision nodes are oblique nodes, which can split the samples more effectively than axis-aligned nodes. Bagged TAO-l (BaggedTAO-l) [46] further uses Bagging to integrate multiple TAO-l trees for better performance.

3 BoostTree and BoostForest

This section introduces the details of our proposed BoostTree and BoostForest. All source code is available at https://github.com/zhaochangming/BoostForest.

3.1 Motivation

Three problems need to be solved in model tree based ensemble learning:

1) How to design a model tree that is more compatible with Bagging? Generally, as the model complexity increases, the bias of the model decreases, but the variance increases. Between the two popular ensemble learning strategies, Bagging is suitable for integrating complex base learners to reduce the variance, whereas boosting for integrating simple base learners to reduce the bias. MultiBoosting combines AdaBoost and wagging to further improve the performance of a decision tree. However, how to combine GBM and Bagging to improve the performance of model trees has not been studied. LMT is a stable model with low randomness, so simply combining multiple LMTs with Bagging may not outperform a single LMT.
2) How to handle both classification and regression problems by using a single model tree structure? LMT applies LogitBoost to a decision tree to generate an accurate model tree; however, it cannot handle regression problems. GBDT-PL handles regression problems by using GBM to integrate multiple PL-Trees. However, performing boosting in a single tree is more efficient, and easier to be used in Bagging.
3) How to generate a model tree with easy parameter tuning, so that any regression algorithm, e.g., RR, ELM and SVR, can be used as its node function?

This section proposes BoostTree to solve the above three problems simultaneously. Moreover, multiple BoostTrees can be further integrated into a BoostForest for better performance.

3.2 General Idea of BoostTree

Our proposed BoostTree is inspired by LMT and GBM. Assume a BoostTree has M nodes, excluding the root. Then, it trains a function f_m for the m-th node, $m \in [1, M]$, and uses (5) to predict the output.

BoostTree minimizes the following regularized loss function:

$$\mathcal{L}(F) = \sum_{n=1}^{N} \ell(y_n, \hat{y}_n) + \sum_{m=1}^{M} \lambda\Omega(f_m), \tag{6}$$

where λ is the regularization coefficient. The second term above penalizes the complexity of BoostTree to reduce overfitting.

Different loss functions ℓ can be used to cope with regression and classification problems, as will be introduced in Sections 3.3, 3.4, and 3.5. For the ease of optimization, we require ℓ to be convex and differentiable.

In general, the objective function in (6) cannot be optimized directly. Inspired by LMT and GBM, BoostTree minimizes (6) in an additive manner. Assume a tree with T ($T \geq 2$) leaves has been generated after $T-1$ iterations. Then, there are $M = 2T-2$ nodes, excluding the root. We can rewrite (6) as:

$$\mathcal{L}(F) = \sum_{m=1}^{T} \text{LeafLoss}_m + \sum_{m=1}^{2T-2} \lambda\Omega(f_m), \tag{7}$$

where

$$\text{LeafLoss}_m = \sum_{n \in I_m} \ell\left(y_n, \sum_{i \in \text{Path}_m} f_i(\boldsymbol{x}_n)\right), \tag{8}$$

$$I_m = \{n | q(\boldsymbol{x}_n) = m\}, \tag{9}$$

i.e., I_m is the set of all training samples belonging to Leaf m. LeafLoss_m measures the loss of Leaf m. In each iteration, BoostTree uses a greedy learning scheme to add branches to the leaf with the highest loss.

Assume Node m is the leaf node with the highest loss. After the split, I_m is divided into two subsets: I_L (of the left node) and I_R (of the right node). Let f_L and f_R be the node models of the left and the right nodes trained separately using I_L and I_R, respectively. Then, the reduction of the loss in (7) is:

$$\begin{aligned}\delta_{\mathcal{L}} = &\sum_{n \in I_m} \ell(y_n, F_m(\boldsymbol{x}_n)) - \sum_{n \in I_L} \ell(y_n, F_m(\boldsymbol{x}_n) + f_L(\boldsymbol{x}_n)) \\ &- \sum_{n \in I_R} \ell(y_n, F_m(\boldsymbol{x}_n) + f_R(\boldsymbol{x}_n)) - \lambda\Omega(f_L) - \lambda\Omega(f_R), \quad (10)\\ = &\mathcal{C} - \mathcal{L}(f_L) - \mathcal{L}(f_R) \quad (11)\end{aligned}$$

where

$$\begin{aligned}\mathcal{C} &= \sum_{n \in I_m} \ell(y_n, F_m(\boldsymbol{x}_n)),\\ F_m(\boldsymbol{x}_n) &= \sum_{i \in \text{Path}_m} f_i(\boldsymbol{x}_n),\\ \mathcal{L}(f_L) &= \sum_{n \in I_L} \ell(y_n, F_m(\boldsymbol{x}_n) + f_L(\boldsymbol{x}_n)) + \lambda\Omega(f_L),\\ \mathcal{L}(f_R) &= \sum_{n \in I_R} \ell(y_n, F_m(\boldsymbol{x}_n) + f_R(\boldsymbol{x}_n)) + \lambda\Omega(f_R).\end{aligned}$$

Note that $\mathcal{C}$ is a constant, F_m is the ensemble of the models along the path from the root to Node m, and $\mathcal{L}(f_L)$ and $\mathcal{L}(f_R)$ are the loss functions for the left and right child nodes, respectively.

(11) can be optimized by minimizing $\mathcal{L}(f_L)$ and $\mathcal{L}(f_R)$ separately. More specifically, BoostTree uses two steps to optimize (11):

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

1) Split the node: BoostTree implements four criteria to select the cut-point:
   a) XGBoost Splitting Criterion (XGB-SC), which uses gradient decent to reduce the loss.
   b) Gini Splitting Criterion (Gini-SC), which tries to improve the purity of each leaf.
   c) Mean Squared Error (MSE), which tries to reduce the MSE of each leaf.
   d) C4.5 Splitting Criterion (C4.5-SC) [47], which selects the cut-point with the largest ratio of information gain.

Though all criteria are effective, XGB-SC achieved the best performance in our experiments, as will be demonstrated in Section 4.10.3; hence, we make it the default option in BoostTree.

XGB-SC uses the Euclidean norm of the leaf score to constrain the tree complexity. Ignoring the parent node loss and inspired by Equation (7) in XGBoost [12], we can calculate the splitting gain as:

$$\delta_{\text{gain}} = \frac{1}{2}\left[\frac{\left(\sum_{n\in I_L} g_n\right)^2}{\sum_{n\in I_L} h_n + \lambda} + \frac{\left(\sum_{n\in I_R} g_n\right)^2}{\sum_{n\in I_R} h_n + \lambda}\right], \tag{12}$$

where

$$g_n = \frac{\partial \ell(y_n, F_m(\boldsymbol{x}_n))}{\partial F_m(\boldsymbol{x}_n)} \tag{13}$$

$$h_n = \frac{\partial^2 \ell(y_n, F_m(\boldsymbol{x}_n))}{\partial F_m(\boldsymbol{x}_n)^2} \tag{14}$$

are the first and second order derivatives of the loss function w.r.t. $F_m(\boldsymbol{x}_n)$, respectively.

Similar to Extra-trees for increasing the randomness, BoostTree first draws a random cut-point uniformly for each feature, and then determines the best cut-point of the current node according to the maximum δ_{gain}.

2) Train the node model: BoostTree uses gradient boosting to decompose the original problem into multiple sub-regression problems, and uses a node function to solve a sub-regression problem in each node. It first calculates the pseudo-labels and sample weights to generate a temporary training set, and then trains a regression model for regression or binary classification, or J regression models for J-class ($J > 2$) classification. For simplicity, RR is used as the default node function. More details on training the node model for different tasks are introduced in Sections 3.3, 3.4, and 3.5.

For the ease of implementation, we can simply store all possible values of each parameter in a parameter pool, from which BoostTree randomly selects a combination of parameters, e.g., the minimum number of samples at a leaf $min_samples_leaf$, and the regularization coefficient λ. Robustness of using different parameters to train a BoostTree in BoostForest will be studied in Section 4.10.1.

Algorithm 1 shows the pseudo-code of BoostTree using RR as the node function, where the subfunction *FitModel* assumes different forms according to different learning tasks, as shown in Algorithms 2, 3, and 4.

Algorithm 1. BoostTree Using RR as the Node Function

Input: Data $= \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$, N training samples, where $\boldsymbol{x}_n \in \mathbb{R}^{D\times 1}$; Pool_{MSL}, candidate value pool of the minimum number of samples at a leaf min_samples_leaf; Pool_λ, candidate value pool of the ℓ_2 regularization parameter λ; (optional) MaxNumLeaf, the maximum number of leaves, default NULL.
Output: A BoostTree.
Randomly select λ and MSL from Pool_λ and Pool_{MSL}, respectively;
Initialize NumLeaf $= 1$, LeafList $= \{\}$ and $f(\boldsymbol{x}) = 0$;
root $\leftarrow \{\text{data} = \text{Data},\ \text{model} = f\}$;
`/* Initialize the root node. */`
Add root to LeafList;
BoostTree $\leftarrow$ split(root);
split(node){
Let m be the index of the current node;
$\{(\boldsymbol{x}_n, y_n)\}_{n\in I_m} \leftarrow$ node.data;
$F_m(\boldsymbol{x}) \leftarrow \sum_{i\in \text{Path}_m} \text{node}_i.\text{model}(\boldsymbol{x})$;
`/* Identify the cut-point. */`
Initialize $\delta_{\text{gain}}^{\max} = 0$ and $\text{split}_{\text{flag}} = \text{false}$;
for $d = 1 : D$ **do**
  Let $a_{\max}$ and $a_{\min}$ denote the maximal and minimal value of $\{x_{n,d} | n \in I_m\}$, respectively;
  Draw a random cut-point s uniformly in $[a_{\min},\ a_{\max}]$;
  $I_L = \{n | x_{n,d} \le s, n \in I_m\}$;
  $I_R = \{n | x_{n,d} > s, n \in I_m\}$;
  if $|I_L| \ge \text{MSL}$ and $|I_R| \ge \text{MSL}$ **then**
    Calculate δ_{gain} in (12);
    if $\text{split}_{\text{flag}} ==$ false or $\delta_{\text{gain}} > \delta_{\text{gain}}^{\max}$ **then**
      $\delta_{\text{gain}}^{\max} = \delta_{\text{gain}}$, $\text{split}_{\text{flag}} = \text{true}$, $I_L^* = I_L$, $I_R^* = I_R$;
    end
  end
end
if $\text{split}_{\text{flag}}$ **then**
  `/* Train the node model. */`
  $f_L^* = \text{FitModel}(\{(\boldsymbol{x}_n, y_n)\}_{n\in I_L^*}, F_m, \lambda)$;
  $f_R^* = \text{FitModel}(\{(\boldsymbol{x}_n, y_n)\}_{n\in I_R^*}, F_m, \lambda)$;
  `/* Generate the child nodes. */`
  node.leftChild $= \{\text{data} = \{(\boldsymbol{x}_n, y_n)\}_{n\in I_L^*}, \text{model} = f_L^*\}$;
  node.rightChild $= \{\text{data} = \{(\boldsymbol{x}_n, y_n)\}_{n\in I_R^*}, \text{model} = f_R^*\}$;
  NumLeaf $=$ NumLeaf $+ 1$;
  Add node.leftChild and node.rightChild to LeafList;
end
Pop node from LeafList;
`/* Split the leaf recursively. */`
if |LeafList| > 0 and (MaxNumLeaf == NULL or NumLeaf $<$ MaxNumLeaf) **then**
  Use (8) to calculate the leaf loss of each node in LeafList;
  split(node*), where node* is the leaf node with the highest loss;
end
return node;
}

3.3 BoostTree for Regression

For regression problems, we use

$$\ell(y_n, \hat{y}_n) = (y_n - \hat{y}_n)^2. \tag{15}$$

Assume Node c is the left or right child node of Node m. According to GBM, the loss function for Node c can be

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

expressed as:

$$\begin{aligned}\mathcal{L}(f_c) &= \sum_{n\in I_c}\{y_n - [F_m(\boldsymbol{x}_n) + f_c(\boldsymbol{x}_n)]\}^2 + \lambda\|\boldsymbol{a}_c\|_2^2 \\ &= \sum_{n\in I_c}\{f_c(\boldsymbol{x}_n) - [y_n - F_m(\boldsymbol{x}_n)]\}^2 + \lambda\|\boldsymbol{a}_c\|_2^2 \\ &= \sum_{n\in I_c}[f_c(\boldsymbol{x}_n) - \tilde{y}_n]^2 + \lambda\|\boldsymbol{a}_c\|_2^2, \end{aligned} \tag{16}$$

where $f_c(\boldsymbol{x}_n) = \boldsymbol{a}_c^T\boldsymbol{x}_n + b_c$, $\tilde{y}_n = y_n - F_m(\boldsymbol{x}_n)$ is the pseudo-label, i.e., the residual between the true value and the prediction, I_c is the set of all training samples belonging to Node c, $\boldsymbol{a}_c \in \mathbb{R}^{D\times 1}$ is a vector of the regression coefficients, and b_c is the intercept. Note that all samples have the same weight in regression.

MSE is sensitive to noise and outliers. To improve BoostTree's robustness, we save the lower (upper) bound lb (ub) of each node's pseudo-label at the training stage, and clip the corresponding node model's output at the prediction stage:

$$\text{Clip}(f_c(\boldsymbol{x}_n)) = \max(lb_c, \min(ub_c, f_c(\boldsymbol{x}_n))), \tag{17}$$

where $lb_c = \max_{n\in I_c}\tilde{y}_n$ and $ub_c = \min_{n\in I_c}\tilde{y}_n$.

When the tree is growing, the residuals of the nodes generated at later iterations approach zero, so lb and ub get closer. The effect of clipping will be demonstrated in Section 4.10.2.

Algorithm 2 shows the pseudo-code of BoostTree for regression.

Algorithm 2. FitModel for Regression

Input: $\{(\boldsymbol{x}_n, y_n)\}_{n\in I_c}$, sample set of the current node;
F_m, ensemble of the models along the path from the root to the parent node of the current node;
λ, the ℓ_2 regularization parameter.
Output: A regression model f_c for the current node.
$\tilde{y}_n = y_n - F_m(\boldsymbol{x}_n), n \in I_c$;
Fit $f_c = \text{RidgeRegression}(\{(\boldsymbol{x}_n, \tilde{y}_n)\}_{n\in I_c}, \lambda)$ using RR with regularization parameter λ;
Clip f_c using (17).

3.4 BoostTree for Binary Classification

In classification tasks, BoostTree is built using a LogitBoost-like algorithm, which iteratively updates the ensemble of the logistic linear models F by adding a new regression model f to it.

The cross-entropy loss is used in binary classification:

$$\begin{aligned}\ell(y_n, \hat{y}_n) = &-y_n\log\left(\text{sigmoid}(\hat{y}_n)\right) \\ &-(1-y_n)\log\left(1-\text{sigmoid}(\hat{y}_n)\right),\end{aligned} \tag{18}$$

where $\hat{y}_n = F_m(\boldsymbol{x}_n)$.

Assume Node c is the left or right child node of Node m. According to LogitBoost (Supplementary Algorithm 1, available online) [41], the second order Taylor expansion can be used to approximate the loss function for Node c:

$$\begin{aligned}\mathcal{L}(f_c) &= \sum_{n\in I_c}\ell(y_n, F_m(\boldsymbol{x}_n) + f_c(\boldsymbol{x}_n)) + \lambda\|\boldsymbol{a}_c\|_2^2 \\ &\approx \sum_{n\in I_c}\left[\ell(y_n, F_m(\boldsymbol{x}_n)) + g_n f_c(\boldsymbol{x}_n) + \frac{1}{2}h_n f_c(\boldsymbol{x}_n)^2\right] \\ &\quad + \lambda\|\boldsymbol{a}_c\|_2^2 \\ &= \frac{1}{2}\sum_{n\in I_c}h_n\left[f_c(\boldsymbol{x}_n) - (-\frac{g_n}{h_n})\right]^2 + \lambda\|\boldsymbol{a}_c\|_2^2 + \mathcal{C},\end{aligned} \tag{19}$$

where

$$\mathcal{C} = \sum_{n\in I_c}\ell(y_n, F_m(\boldsymbol{x}_n)) - \sum_{n\in I_c}\frac{1}{2}\frac{(g_n)^2}{h_n}$$

is a constant, and g_n and h_n are calculated by using (13) and (14), respectively. Note that g_n and h_n are irrelevant to f_c. Therefore, f_c can be any regression model.

Then, we can construct the pseudo-label $\tilde{y}_n$ and the sample weight w_n, as in LogitBoost:

$$\tilde{y}_n = -\frac{g_n}{h_n}, \tag{20}$$

$$w_n = h_n, \tag{21}$$

where

$$g_n = p(\boldsymbol{x}_n) - y_n, \tag{22}$$

$$h_n = p(\boldsymbol{x}_n)[1 - p(\boldsymbol{x}_n)], \tag{23}$$

in which $p(\boldsymbol{x}_n)$ is the estimated probability in (2).

To improve BoostTree's robustness to outliers, we follow LogitBoost [41] to limit the minimum weight to 2ϵ (ϵ is the machine epsilon), and clip the value of the pseudo-label $\tilde{y}$ to:

$$\text{Clip}(\tilde{y}) = \max(-y_{\max}, \min(y_{\max}, \tilde{y})), \tag{24}$$

where $y_{\max} \in [2, 4]$ (according to LogitBoost [41]). $y_{\max} = 4$ was used in our experiments.

Finally, we can remove $\mathcal{C}$ in (19) to simplify the loss function for child Node c:

$$\mathcal{L}(f_c) = \frac{1}{2}\sum_{n\in I_c}w_n[f_c(\boldsymbol{x}_n) - \tilde{y}_n]^2 + \lambda\|\boldsymbol{a}_c\|_2^2. \tag{25}$$

Algorithm 3 shows the pseudo-code of BoostTree for binary classification.

Algorithm 3. FitModel for Binary Classification

Input: $\{(\boldsymbol{x}_n, y_n)\}_{n\in I_c}$, sample set of the current node;
F_m, ensemble of the models along the path from the root to the parent node of the current node;
λ, the ℓ_2 regularization parameter.
Output: A regression model f_c for the current node.
$p(\boldsymbol{x}_n) = \text{sigmoid}(F_m(\boldsymbol{x}_n)), n \in I_c$;
$\tilde{y}_n = \dfrac{y_n - p(\boldsymbol{x}_n)}{p(\boldsymbol{x}_n)[1 - p(\boldsymbol{x}_n)]}, n \in I_c$;
$\tilde{y}_n = \text{Clip}(\tilde{y}_n)$ in (24), $n \in I_c$;
$w_n = p(\boldsymbol{x}_n)[1 - p(\boldsymbol{x}_n)], n \in I_c$;
$\mathcal{D}' = \{(\boldsymbol{x}_n, \tilde{y}_n, w_n) | \ n \in I_c\}$;
Fit $f_c = \text{WeightedRidgeRegression}(\mathcal{D}', \lambda)$ using weighted RR on $\mathcal{D}'$ with regularization parameter λ.

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

3.5 BoostTree for J-Class ($J > 2$) Classification

For J-class classification, we use

$$\ell(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n) = -\sum_{j=1}^{J} y_n^j \log\left(\text{softmax}^j(\hat{\boldsymbol{y}}_n)\right), \tag{26}$$

where $\boldsymbol{y}_n = [y_n^1, y_n^2, \ldots, y_n^J]^T \in \mathbb{R}^{J\times 1}$ is the one-hot encoding label vector, and $\hat{\boldsymbol{y}}_n = F_m(\boldsymbol{x}_n) = [\hat{y}_n^1, \hat{y}_n^2, \ldots, \hat{y}_n^J]^T \in \mathbb{R}^{J\times 1}$ is the output vector.

In each iteration, LogitBoost handles the J-class classification problem by decomposing it into J regression problems, so $\boldsymbol{f}_c$ becomes a set of linear models $\{f_c^1, f_c^2, \cdots, f_c^J\}$, where f_c^j calculates the output for Class j.

Similarly, the second order Taylor expansion can be used to approximate the loss function for Node c:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{f}_c) &= \sum_{n\in I_c} \ell(y_n, F_m(\boldsymbol{x}_n) + \boldsymbol{f}_c(\boldsymbol{x}_n)) + \lambda \sum_{j=1}^{J} \|\boldsymbol{a}_c^j\|_2^2 \\
&\approx \sum_{n\in I_c} \ell(y_n, F_m(\boldsymbol{x}_n)) + \lambda \sum_{j=1}^{J} \|\boldsymbol{a}_c^j\|_2^2 \\
&\quad + \sum_{j=1}^{J} \sum_{n\in I_c} \left[g_n^j f_c^j(\boldsymbol{x}_n) + \frac{1}{2} h_n^j f_c^j(\boldsymbol{x}_n)^2 \right] \\
&= \frac{1}{2} \sum_{j=1}^{J} \sum_{n\in I_c} h_n^j \left[f_c^j(\boldsymbol{x}_n) - (-\frac{g_n^j}{h_n^j}) \right]^2 + \lambda \sum_{j=1}^{J} \|\boldsymbol{a}_c^j\|_2^2 \\
&\quad + \mathcal{C},
\end{aligned} \tag{27}$$

where

$$\mathcal{C} = \sum_{n\in I_c} \ell(y_n, F_m(\boldsymbol{x}_n)) - \sum_{j=1}^{J} \sum_{n\in I_c} \frac{1}{2} \frac{(g_n^j)^2}{h_n^j} \tag{28}$$

is a constant, $\boldsymbol{a}_c^j$ is the coefficient vector of f_c^j, g_n^j is the j-th element of $\boldsymbol{g}_n$, and h_n^j is the j-th diagonal element of $\boldsymbol{h}_n$. $\boldsymbol{g}_n$ and $\boldsymbol{h}_n$ are again calculated by using (13) and (14), respectively.

Then, for the j-th class, we can calculate the pseudo-label $\tilde{y}_n^j$ and sample weight w_n^j, as in LogitBoost:

$$\tilde{y}_n^j = -\frac{g_n^j}{h_n^j}, \tag{29}$$

$$w_n^j = h_n^j, \tag{30}$$

where

$$g_n^j = p^j(\boldsymbol{x}_n) - y_n^j, \tag{31}$$

$$h_n^j = p^j(\boldsymbol{x}_n)[1 - p^j(\boldsymbol{x}_n)], \tag{32}$$

in which $p^j(\boldsymbol{x}_n)$ is the estimated probability of Class j in (4). To prevent too large pseudo-labels, we also use (24) to clip $\tilde{y}_n^j$.

Finally, we can remove $\mathcal{C}$ in (27) to simplify the loss function for Node c:

$$\mathcal{L}(\boldsymbol{f}_c) = \frac{1}{2} \sum_{j=1}^{J} \sum_{n\in I_c} w_n^j \left[f_c^j(\boldsymbol{x}_n) - \tilde{y}_n^j \right]^2 + \sum_{j=1}^{J} \lambda \|\boldsymbol{a}_c^j\|_2^2. \tag{33}$$

Algorithm 4 shows the pseudo-code of BoostTree for J-class ($J > 2$) classification.

Algorithm 4. FitModel for J-Class ($J > 2$) Classification

Input: $\{(\boldsymbol{x}_n, y_n)\}_{n\in I_c}$, sample set of the current node;
F_m, ensemble of the models along the path from the root to the parent node of the current node;
λ, the ℓ_2 regularization parameter.
Output: The regression model set $\boldsymbol{f}_c$ for the current node.
Compute $p^j(\boldsymbol{x}_n) = \text{softmax}^j(F_m(\boldsymbol{x}_n))$, $n \in I_c$, $j = 1, \ldots, J$;
for $j = 1 : J$ **do**
  $\tilde{y}_n^j = \dfrac{y_n^j - p^j(\boldsymbol{x}_n)}{p^j(\boldsymbol{x}_n)[1 - p^j(\boldsymbol{x}_n)]}$, $n \in I_c$;
  $\tilde{y}_n^j = \text{Clip}(\tilde{y}_n^j)$ in (24), $n \in I_c$;
  $w_n^j = p^j(\boldsymbol{x}_n)[1 - p^j(\boldsymbol{x}_n)]$, $n \in I_c$;
  $\mathcal{D}' = \{(\boldsymbol{x}_n, \tilde{y}_n^j, w_n^j) |\ n \in I_c\}$;
  Fit $f_c^j = \text{WeightedRidgeRegression}(\mathcal{D}', \lambda)$ using weighted RR on $\mathcal{D}'$ with regularization parameter λ;
end
`/* Center the outputs of the regression models, as in LogitBoost. */`
$f_c^j(\boldsymbol{x}) \leftarrow \dfrac{J-1}{J}\left[f_c^j(\boldsymbol{x}) - \dfrac{1}{J}\sum_{i=1}^{J} f_c^i(\boldsymbol{x})\right]$, $j = 1, \ldots, J$;
$\boldsymbol{f}_c = \{f_c^1, f_c^2, \ldots, f_c^J\}$.

3.6 Boostforest

BoostForest integrates multiple BoostTrees into a forest. It first uses bootstrap to generate K replicas of the original training set, and then trains a BoostTree on each replica.

For regression, the outputs predicted by all K BoostTrees are averaged as the final output:

$$\text{BoostForest}(\boldsymbol{x}) = \frac{1}{K} \sum_{k=1}^{K} \text{BoostTree}_k(\boldsymbol{x}). \tag{34}$$

For classification, the probabilities predicted by all K BoostTrees are averaged as the final probability:

$$\begin{aligned}
&\text{BoostForest}(\boldsymbol{x}) \\
&= \begin{cases} \frac{1}{K}\sum_{k=1}^{K} \text{sigmoid}(\text{BoostTree}_k(\boldsymbol{x})), & J = 2 \\ \frac{1}{K}\sum_{k=1}^{K} \text{softmax}(\text{BoostTree}_k(\boldsymbol{x})), & J > 2 \end{cases}.
\end{aligned} \tag{35}$$

Algorithm 5 gives the pseudo-code of BoostForest.

Algorithm 5. BoostForest Training Algorithm

Input: $\text{Data} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^{N}$, N training samples, where $\boldsymbol{x}_n \in \mathbb{R}^{d\times 1}$;
n_estimators, the number of BoostTrees;
Pool_{MSL}, parameter value pool of the minimum number of samples at a leaf;
Pool_{λ}, parameter value pool of the ℓ_2 regularization parameter λ.
Output: A BoostForest.
$\text{BoostForest} = \{\}$;
for $i = 1 :$ n_estimators **do**
Bootstrap Data' from Data;
Train BoostTree_i on Data' using Algorithm 1;
Add BoostTree_i to BoostForest;
end

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

3.7 Implementation Details

Inspired by the row sampling trick in XGBoost and lightGBM, we use two tricks to improve the speed of BoostTree:

1) Randomly select $BatchSize$ samples to identify the cut-point, if the number of samples belonging to the splitting node is larger than the batch size $BatchSize$.

2) When the number of samples belonging to Node m is larger than $BatchSize$, randomly select $BatchSize$ samples to train the node model, and approximate LeafLoss_m as:

$$\text{LeafLoss}_m = \frac{|I_m|}{|I_{\text{batch}}|} \sum_{n \in I_{\text{batch}}} \ell\left(y_n, \sum_{i \in \text{Path}_m} f_i(\boldsymbol{x}_n)\right), \quad (36)$$

where I_{batch} is the batch index set of I_m.

$BatchSize = 1,000$ was used in our experiments.

To ensure BoostTree's number of leave nodes is larger than or equal to two, BoostTree tries to split the root node until it is split successfully.

3.8 Discussions

To clarify the novelties of BoostForest, this subsection first discusses the differences and connections between it and some related approaches, e.g., LMT, GBDT-PL, MultiBoosting and TAO, and then summarizes approaches which motivated BoostForest.

There are four main differences between LMT and BoostForest:

1) LMT uses SimpleLogistic to train the node model, and only handles classification. BoostTree can use any regression algorithm to train the node model, and can handle both classification and regression.
2) LMT computes the splitting gains on all values of the attributes to select the cut-point, whereas BoostTree draws the cut-points completely randomly, which increases its diversity and makes BoostTrees more suitable for Bagging.
3) LMT prunes the tree to reduce over-fitting and uses only one tree to predict the output. However, a single learner can easily be affected by noise and outliers, leading to poor generalization. BoostForest combines multiple BoostTrees with Bagging to reduce over-fitting.
4) LMT's asymptotic complexity is $O(d \cdot N \cdot \log N + N \cdot d^2 \cdot Depth)$ (ignoring the pruning operation), where $Depth$ is the depth of the tree, N is the sample size, and D is the feature dimensionality. BoostTree's complexity is $O[d \cdot N + (N \cdot d^2 + d^3) \cdot Depth]$ (using RR as the node function), where the first part is the cost of building an Extra-tree, and the second is the cost of building the node models. If down-sampling is performed before fitting the RR model, BoostTree's asymptotic complexity is less than $O[d \cdot N + (BatchSize \cdot d^2 + d^3) \cdot M]$, where M is the number of nodes.

M5 and M5P train a linear model at each node during the training process. Their smoothing operations use all node models along the path from the root to a leaf to predict the output. M5 and M5P construct model trees for regression, whereas LMT constructs them for classification. LMT is the most relevant approach to BoostTree. They both integrate GBM into one single tree, train a base learner at each node to perform boosting, and combine all node models along the path from the root to the corresponding leaf to predict the output. BoostTree may be viewed as an upgraded LMT.

BoostTree improves LMT from three aspects:

1) Model structure: BoostTree expands LMT's node models to non-linear models.
2) Application scenario: BoostTree further extends LMT to regression problems.
3) Training process: BoostTree optimizes the cut-point selection strategy by combining XGB-SC and Extra-trees' cut-point drawing strategy, and also the parameter setting strategy. These improvements make it easier to tune the hyper-parameters in BoostTree. Note that LMT can automatically perform cross-validation within each node to determine the number of training iterations for SimpleLogistic, and prune the tree to further reduce over-fitting. However, when the node models are replaced by other more complex models, the computational cost of cross-validation at all nodes may be very high. To alleviate this problem, BoostTree develops a random parameter pool sampling strategy and combines it with Bagging.

There are four main differences between GBDT-PL and BoostForest:

1) GBDT-PL trains a linear model at each node, and does not evaluate its performance on multi-class classification. BoostTree can use any regression algorithm to train the node model, and can handle binary and multi-class classifications, and regression.
2) GBDT-PL splits the node and trains the node models simultaneously, whereas BoostTree first splits the node, and then trains the node models. When the node models are replaced by other complex models, it's very time-consuming for GBDT-PL to select the cut-point, because it needs to train the left and right node models at all candidate cut-points.
3) GBDT-PL uses GBM to train multiple PL-Trees, whereas BoostTree integrate GBM into one single tree.
4) GBDT-PL needs a validation set to select its hyper-parameters, which include the tree structure and node model parameters. Its computational cost increases with the number of hyper-parameter combinations. BoostTree optimizes its hyper-parameter setting strategy by integrating random parameter pool sampling and Bagging.

When each node uses a linear model, BoostTree has a similar structure as a tree in GBDT-PL, because all linear node models along the path to the corresponding leaf can be combined into one single linear model. However, when each node uses a non-linear model, the node models cannot be easily combined, which makes the structures of BoostTree and GBDT-PL tree different.

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

TABLE 1
Approaches That Motivated BoostForest

BoostForest choice	Motivated by
Tree structure	LMT [40]
Splitting criteria	XGBoost [12], CART [13] and C4.5 [47]
Cut-point drawing	Extra-trees [29]
Node model training	LMT [40], GBM [11], LogitBoost [41] and stochastic gradient boosting [31]
Tree ensemble generation	RandomForest [26]
Random parameter pool sampling	Empirical
Clipping operation in regression	Empirical

There are three main differences between MultiBoosting and BoostForest:

1) MultiBoosting uses traditional decision trees as the base learners, whereas BoostForest uses BoostTrees. Each BoostTree is trained by gradient boosting, so itself can be viewed as an ensemble model.
2) MultiBoosting uses first AdaBoost to train multiple decision trees sequentially to reduce the bias, and then wagging (a variant of Bagging) to further reduce the variance. BoostForest first generates multiple BoostTrees independently, and then uses the idea of Random Forest to reduce their variance.
3) MultiBoosting can only be used for classification, whereas BoostForest can handle both classification and regression.

MultiBoosting and BoostForest both combine boosting and Bagging, but in different ways: MultiBoosting combines them serially, whereas BoostForest combines them in parallel.

TAO is a general and efficient optimization algorithm, which can train many types of decision trees, e.g., both TAO and BoostTree can train the base learners in a tree ensemble. However, there are three main differences between TAO and BoostTree:

1) TAO trains a binary classifier at each decision node to classify a sample into the left or right child node. Its decision nodes are oblique nodes, whereas BoostTree's decision nodes are axis-aligned nodes.
2) TAO uses only the leaf model to predict the output, whereas BoostTree combines all node models along the path from the root to the corresponding leaf to predict the output.
3) TAO circularly updates each decision or leaf nodes to optimize the objective function, whereas BoostTree optimizes it by using a greedy learning scheme to add new branches to the leaf.

Table 1 summarizes the approaches that motivated BoostForest. Essentially, BoostForest integrates five existing strategies and develops two new empirical ones.

4 Experimental Results

Extensive experiments were carried out to verify the effectiveness of BoostForest in both classification and regression. The following seven questions were examined:

1) What is the generalization performance of BoostForest, compared with several popular ensemble learning approaches, e.g., RandomForest [26], Extra-Trees [29], XGBoost [12], LightGBM [35], MultiBoosting [44], GBDT-PL [43], FilterBoost [33] and DeepBoosting [34]?
2) How fast does BoostForest converge, as the number of base learners increases?
3) How does the base learner model complexity affect the generalization performance of BoostForest?
4) Can BoostForest handle datasets with a large number of samples or features?
5) Can our proposed approach for constructing BoostForest, i.e., data replica by bootstrapping and random parameter selection from the parameter pool, also be used to integrate other tree models, e.g., Extra-Tree [29], M5P [19], and LMT [40]?
6) How does the performance of BoostForest change when different node functions, e.g., RR, ELM or SVR, are used in BoostTrees?
7) How robust is BoostForest to its hyper-parameters and the node splitting criterion?

4.1 Datasets

We performed experiments on 37 real-world datasets[1] (20 for classification and 17 for regression), summarized in Supplementary Table 1, available online. They covered a wide range of feature dimensionalities (between 4 and 1,024) and sample sizes (between 103 and 11,000,000). For each dataset, categorical features were converted to numerical ones by one-hot encoding. Unless stated otherwise, each feature's distribution was scaled to a standard normal distribution, and the labels in the regression datasets were z-normalized.

4.2 Algorithms

Supplementary Table 2, available online, shows the hyper-parameters of the 11 baselines and BoostForest used in our experiments. More details about the baseline parameters can be found online.[2] Unless stated otherwise, BoostForest used default parameter values.

4.3 Experimental Setting

We performed experiments on 30 small to medium sized datasets (15 for classification and 15 for regression) in Sections 4.4, 4.5, and 4.6 and 4.10, 35 datasets (30 small to medium sized ones and five large sized ones) in Section 4.7, 32 datasets (30 small to medium sized ones and two high dimensional ones) in Section 4.8, and 17 datasets (15 small to medium sized ones and two large sized ones) in Section 4.9.

All algorithms were repeated 10 times on each dataset (except MNIST and HIGGS). For each experiment, we first randomly divided the data into 60% training, 20%

1. http://archive.ics.uci.edu/ml/datasets.php; http://www.cad.zju.edu.cn/home/dengcai/Data/data.html; http://yann.lecun.com/exdb/mnist/

2. https://scikit-learn.org; https://xgboost.readthedocs.io/en/latest/parameter.html; https://lightgbm.readthedocs.io/en/latest/Parameters.html; https://github.com/GBDT-PL/GBDT-PL; https://github.com/dmarcous/deepboost; https://www.cs.waikato.ac.nz/ml/weka/

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

TABLE 2
Performances of the Seven Ensemble Learning Approaches on the 30 Datasets

Classification Dataset	RandomForest	Extra-Trees	XGBoost	LightGBM	MultiBoosting	BoostForest
	Mean and standard deviation (in parentheses) of the classification accuracy					
SON	0.8500 (0.0500)	0.8595 (0.0607)	**0.8619** (0.0551)	0.8405 (0.0648)	0.8429 (0.0641)	0.8500 (0.0574)
SEE	0.9143 (0.0265)•	0.9310 (0.0225)•	0.9238 (0.0233)•	0.9238 (0.0333)•	0.9500 (0.0225)•	**0.9667** (0.0117)
QB	**1.0000** (0.0000)	0.9980 (0.0060)	0.9880 (0.0098)•	0.9880 (0.0098)•	0.9940 (0.0092)	**1.0000** (0.0000)
VC2	0.8097 (0.0455)•	0.8355 (0.0493)	0.8242 (0.0429)	0.8290 (0.0416)	0.8226 (0.0346)•	**0.8500** (0.0361)
VC3	0.8129 (0.0383)•	0.8065 (0.0279)•	0.8226 (0.0495)	0.8161 (0.0480)	0.8032 (0.0426)•	**0.8371** (0.0255)
MV1	0.8812 (0.0310)	0.9010 (0.0243)	0.9042 (0.0298)	0.8969 (0.0341)	**0.9094** (0.0255)	0.8917 (0.0284)
BCD	0.9605 (0.0153)•	0.9623 (0.0192)•	0.9632 (0.0238)•	0.9640 (0.0173)•	0.9781 (0.0137)	**0.9798** (0.0136)
ILP	0.7051 (0.0390)•	**0.7333** (0.0278)	0.6872 (0.0349)•	0.7128 (0.0297)	0.7000 (0.0394)•	0.7274 (0.0347)
BD	0.7827 (0.0272)	0.7813 (0.0139)	**0.7833** (0.0201)	0.7773 (0.0205)	0.7660 (0.0282)	0.7807 (0.0156)
PID	0.7662 (0.0186)	**0.7682** (0.0205)	0.7604 (0.0305)	0.7617 (0.0317)	0.7519 (0.0223)	0.7682 (0.0197)
VS	0.7400 (0.0162)•	0.7376 (0.0241)•	0.7565 (0.0286)•	0.7594 (0.0247)•	0.7529 (0.0207)•	**0.8429** (0.0215)
QSAR	0.8801 (0.0221)	0.8754 (0.0188)•	0.8673 (0.0246)•	0.8758 (0.0229)	0.8735 (0.0249)•	**0.8934** (0.0225)
DRD	0.6654 (0.0273)•	0.6814 (0.0252)•	0.6745 (0.0282)•	0.6779 (0.0233)•	0.6870 (0.0227)•	**0.7442** (0.0192)
BA	0.9942 (0.0033)•	0.9993 (0.0015)	0.9953 (0.0040)•	0.9949 (0.0044)•	0.9985 (0.0018)•	**1.0000** (0.0000)
WDG	0.8579 (0.0075)•	0.8648 (0.0063)•	0.8609 (0.0086)•	0.8611 (0.0087)•	0.8572 (0.0077)•	**0.8716** (0.0102)
Average accuracy	0.8413 (0.0079)	0.8490 (0.0077)	0.8449 (0.0122)	0.8453 (0.0086)	0.8458 (0.0097)	**0.8669** (0.0064)
Average rank	4.4667	3.0667	3.8667	3.8000	4.1333	**1.6667**
Average time	10.6116 (0.1849)	11.0003 (0.0768)	17.1842 (0.1357)	7.5803 (0.5097)	**3.8320** (0.2296)	4.7953 (0.0896)
Average model size	3.5578×10^5	6.4825×10^5	1.0264×10^5	$\mathbf{5.9302 \times 10^4}$	1.3214×10^5	5.7421×10^6
Regression Dataset	**RandomForest**	**Extra-Trees**	**XGBoost**	**LightGBM**	**GBDT-PL**	**BoostForest**
	Mean and standard deviation (in parentheses) of the regression RMSE					
CS	0.4041 (0.0684)•	0.3506 (0.0629)•	0.3500 (0.0994)•	0.3913 (0.0597)•	0.3420 (0.1044)•	**0.2284** (0.0331)
CF	0.7748 (0.1174)•	0.7461 (0.0940)•	0.8291 (0.1136)•	0.7847 (0.0895)•	0.8493 (0.1406)•	**0.6991** (0.0721)
AMPG	0.3621 (0.0587)•	0.3520 (0.0694)	0.3734 (0.0533)•	0.3662 (0.0600)	0.3733 (0.0585)•	**0.3422** (0.0706)
REV	**0.5087** (0.0960)	0.5093 (0.0938)	0.5485 (0.0910)	0.5261 (0.0880)	0.5628 (0.0904)•	0.5161 (0.1005)
NO	**0.6375** (0.0628)	0.6527 (0.0777)	0.6412 (0.0664)	0.6499 (0.0906)	0.6436 (0.0727)	0.6422 (0.0750)
PM	0.8051 (0.0478)•	0.8024 (0.0457)•	0.7654 (0.0598)	0.7493 (0.0522)	**0.7422** (0.0604)	0.7786 (0.0427)
BH	0.4009 (0.0828)•	0.3895 (0.0742)•	0.3962 (0.0844)•	0.4002 (0.0821)•	0.4119 (0.0968)•	**0.3593** (0.0705)
CPS	0.8354 (0.0646)	0.8295 (0.0768)	**0.8273** (0.0606)	0.8346 (0.0665)	0.8489 (0.0598)	0.8682 (0.0745)
CCS	0.2808 (0.0167)•	0.2763 (0.0191)•	0.2472 (0.0256)	0.2390 (0.0209)	**0.2367** (0.0225)	0.2529 (0.0192)
ASN	0.2486 (0.0164)	0.2333 (0.0194)	0.2239 (0.0164)	0.2283 (0.0124)	**0.2177** (0.0318)	0.2478 (0.0193)
ADS	0.6552 (0.0286)	0.6598 (0.0300)	0.6564 (0.0286)	0.6586 (0.0267)	0.6494 (0.0284)	**0.6493** (0.0187)
WQW	0.6972 (0.0379)•	0.6924 (0.0365)	0.7108 (0.0337)•	0.7073 (0.0352)•	0.6918 (0.0320)	**0.6874** (0.0324)
AQ	0.0032 (0.0030)•	0.0038 (0.0037)•	0.0030 (0.0030)	0.0074 (0.0031)•	0.0031 (0.0011)•	**0.0016** (0.0013)
CCPP	0.1933 (0.0051)•	0.1972 (0.0053)•	0.1792 (0.0053)	0.1784 (0.0061)	**0.1773** (0.0051)	0.1902 (0.0062)
EGSS	0.3188 (0.0051)•	0.3036 (0.0040)•	0.2133 (0.0036)	0.2073 (0.0042)	**0.1894** (0.0042)	0.2195 (0.0040)
Average RMSE	0.4750 (0.0122)	0.4666 (0.0114)	0.4643 (0.0137)	0.4619 (0.0134)	0.4626 (0.0216)	**0.4455** (0.0109)
Average rank	4.2000	3.9333	3.4000	3.7333	3.0667	**2.6667**
Average time	10.1125 (0.2117)	10.9757 (0.1039)	19.9508 (0.1587)	12.1506 (0.4130)	75.4997 (0.2494)	**4.8862** (0.1055)
Average model size	4.0171×10^6	5.1793×10^6	5.0651×10^5	$\mathbf{3.8676 \times 10^5}$	1.1150×10^6	9.1213×10^6

The best performance is marked in bold. • *indicates a statistically significant win for boostforest.*

validation and 20% test. For MNIST or HIGGS, we randomly divided the data into 80% training, 20% validation, and used the original test data for testing. Next, we used the validation set to select the best parameters. Then, the training set and validation set were combined to train a model with these parameters. Finally, we verified its performance on the test set. For RandomForest and Extra-Trees, we used the out-of-bag error to select their parameters.

We used the classification accuracy and the root mean squared error (RMSE) as the main performance measure for classification and regression, respectively. We also computed the average rank and training time (including the process of parameter selection) for each algorithm on each dataset. For K algorithms, the best one has rank 1, and the worst has rank K. We used seconds as the unit of time in our experiments and 50 processing cores to run the experiments on a server with two Intel(R) Xeon(R) Platinum 8276 CPUs. Additionally, we used bytes as the unit of model size in our experiments.

To validate if BoostForest significantly outperformed the baselines ($\alpha = 0.05$), we first calculated the p-values using the standard t-test, and then performed Benjamini Hochberg False Discovery Rate (BH-FDR) correction [48] to adjust them. The statistically significant ones are marked by •.

4.4 Generalization Performance of BoostForest

First, we compared the generalization performance of BoostForest with eight baselines. The results are shown in Tables 2 and 3.

Table 2 shows that BoostForest achieved the best generalization performance on 17 out of the 30 datasets, the fastest

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

TABLE 3
Performances of the Four Ensemble Learning Approaches on the 11 Binary Classification Datasets

Classification Dataset	GBDT-PL	DeepBoosting	FilterBoost	BoostForest
	Mean and standard deviation (in parentheses) of the classification accuracy			
SON	0.8143 (0.0810)	0.8071 (0.1024)	0.8452 (0.0586)	**0.8500** (0.0574)
QB	0.9980 (0.0060)	0.9880 (0.0098)•	0.9880 (0.0098)•	**1.0000** (0.0000)
VC2	0.7968 (0.0383)•	0.8097 (0.0594)•	0.8048 (0.0435)•	**0.8500** (0.0361)
MV1	0.8958 (0.0349)	0.8583 (0.0393)	**0.8990** (0.0361)	0.8917 (0.0284)
BCD	0.9632 (0.0348)	0.9456 (0.0321)•	0.9693 (0.0119)•	**0.9798** (0.0136)
ILP	0.6761 (0.0339)•	0.6897 (0.0342)•	0.6932 (0.0252)•	**0.7274** (0.0347)
BD	**0.7813** (0.0217)	0.7773 (0.0225)	0.7780 (0.0215)	0.7807 (0.0156)
PID	0.7591 (0.0272)	0.7442 (0.0250)•	0.7571 (0.0210)	**0.7682** (0.0197)
QSAR	0.8664 (0.0358)•	0.8735 (0.0220)•	0.8640 (0.0222)•	**0.8934** (0.0225)
DRD	0.7316 (0.0361)	0.6775 (0.0228)•	0.6710 (0.0301)•	**0.7442** (0.0192)
BA	0.9967 (0.0025)•	0.9895 (0.0084)•	0.9971 (0.0022)•	**1.0000** (0.0000)
Average accuracy	0.8436 (0.0115)	0.8328 (0.0137)	0.8424 (0.0076)	**0.8623** (0.0076)
Average rank	2.6364	3.4545	2.6364	**1.2727**
Average time	96.0279 (0.6398)	11.2469 (0.1279)	**1.2163** (0.1139)	2.9423 (0.0529)
Average model size	1.0386×10^6	$\mathbf{1.9835 \times 10^3}$	7.0042×10^3	2.7449×10^6

The best performance is marked in bold. • *indicates a statistically significant win for boostforest.*

average time in regression, the second fastest average time in classification, and the best average performance, standard deviation and rank in both classification and regression.

Table 3 shows that BoostForest achieved the best generalization performance on nine out of the 11 datasets, the best average performance, standard deviation and rank, and the second fastest average time.

On average, BoostForest achieved the best classification accuracy or regression RMSE. It significantly outperformed RandomForest on 19 datasets, Extra-Trees on 15 datasets, XGBoost on 14 datasets, LightGBM on 12 datasets, MultiBoosting on nine datasets, GBDT-PL on 10 datasets, DeepBoosting on eight datasets, and FilterBoost on seven datasets.

It is also important to note that BoostForest used the default parameter settings on all datasets, and did not use a validation set to select the best parameters. So, it is very convenient to use. Additionally, each BoostTree is trained with a different training set, so BoostForest can be easily parallelized to further speed it up.

One limitation of BoostForest is that its average model size was the largest, so it may not be easily deployed to low storage devices. One of our future research directions is to reduce its model size.

4.5 Generalization Performance W.r.t. The Number of Base Learners

BoostForest needs to specify the number of BoostTrees in it. It is important to study how its performance changes with this number.

On each dataset, we gradually increased the number of base learners from three to 250, and tuned other parameters of the four baselines by using the validation set (for XGBoost and LightGBM) or the out-of-bag error (for RandomForest and Extra-Trees).

Fig. 2a shows the accuracies of the five algorithms on the last four classification datasets. Complete results on all 15 classification datasets are shown in Supplementary Fig. 1, available online. Generally, as the number of base learners increased, the performances of all ensemble learning approaches quickly converged. BoostForest achieved the highest classification accuracy on 11 of the 15 datasets, and the second highest on another three (MV1, BD, and PID).

Fig. 2b shows the RMSEs of the five algorithms on the last four regression datasets. Complete results on all 15 regression datasets are shown in Supplementary Fig. 2, available online. Again, as the number of base learners increased, generally the performances of all algorithms quickly converged. BoostForest achieved the smallest RMSE on seven of the 15 datasets.

In summary, BoostForest has very good generalization performance w.r.t. the number of base learners, and it generally converges faster than the four baselines (RandomForest, Extra-Trees, XGBoost, and LightGBM).

4.6 Generalization Performance W.r.t. The Base Learner Model Complexity

We also evaluated the generalization performance of the five ensemble approaches, as the base learner model complexity increases.

The base learner model complexity was controlled by the maximum number of leaves $MaxNumLeaf$ per tree, which was gradually increased from two to 32 for classification, and two to 256 for regression. We fixed the number of base learners at 250, and tuned other parameters of the four baselines by using the validation set (for XGBoost and LightGBM) or the out-of-bag error (for RandomForest and Extra-Trees).

Fig. 3a shows the accuracies of the five algorithms on the last four classification datasets. Complete results on all 15 classification datasets are shown in Supplementary Fig. 3, available online. BoostForest achieved the highest classification accuracy on 12 of the 15 datasets.

Fig. 3b shows the average RMSEs of the five algorithms on the last four regression datasets. Complete results on all 15 regression datasets are shown in Supplementary Fig. 4,

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

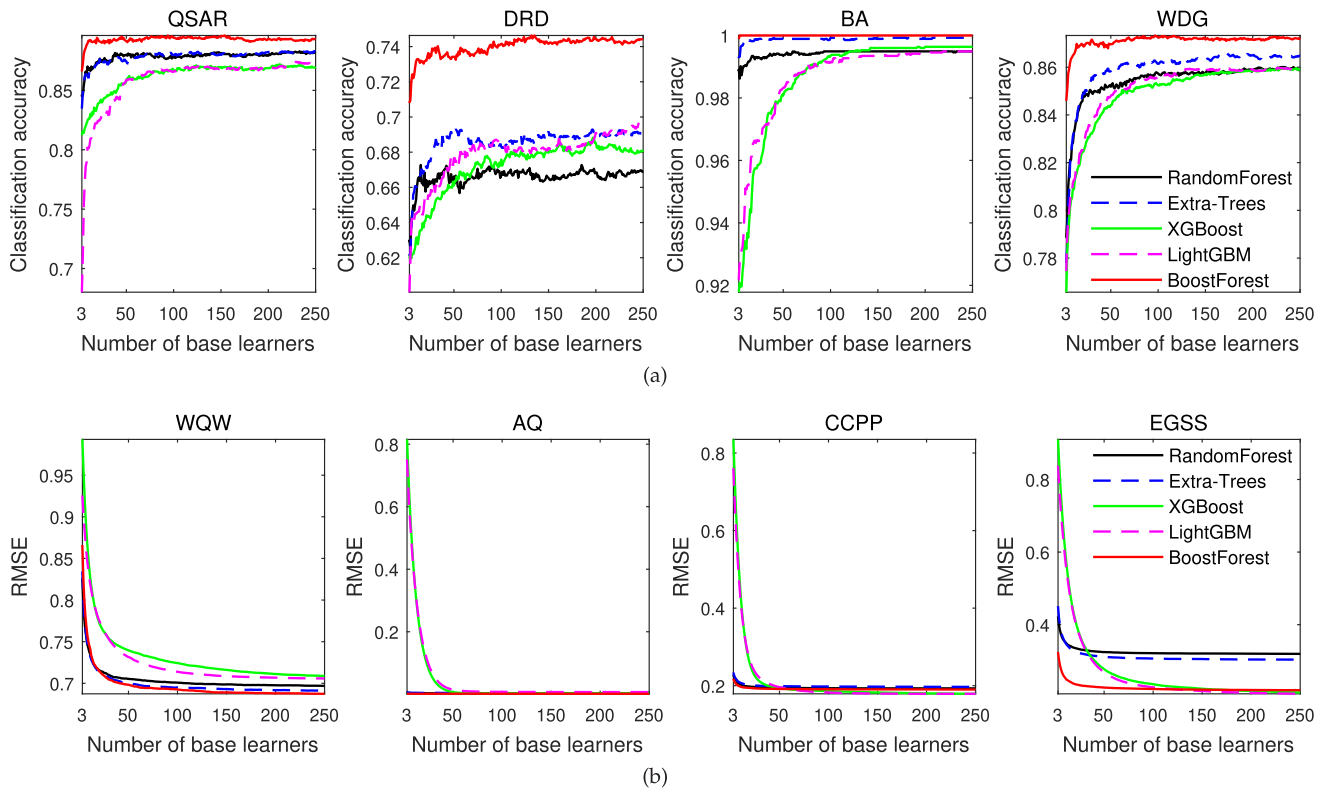


Fig. 2. Generalization performance w.r.t. the number of base learners, averaged over 10 repeats. (a) Average classification accuracies on the last four classification datasets, with different number of base learners. Complete results on the 15 classification datasets are shown in Supplementary Fig. 1, available online. (b) Average RMSEs on the last four regression datasets, with different number of base learners. Complete results on the 15 regression datasets are shown in Supplementary Fig. 2, available online.

available online. On most datasets, the performances of all algorithms increased as the maximum number of leaves per tree increased. BoostForest achieved the smallest RMSE on seven of the 15 datasets, and the second smallest RMSE on the NO dataset.

These results suggest that BoostForest generalize well with the base learner model complexity.

4.7 Generalization Performance on Large Datasets

Previous experiments have shown the superiority of BoostForest on small to medium sized datasets with not very high dimensionalities. This subsection investigates its performance on four datasets with a large number of samples and/or features.

Table 4 compares the performance of BoostForest with four classical and popular ensemble methods on three classification datasets and two regression datasets. For the SUSY dataset, we set $min_samples_leaf = 2$ and $MaxNumLeaf = 20,000$ in each BoostTree.

BoostForest still demonstrated superior and consistent performance: it achieved the highest average accuracy and the second smallest average time in classification, and the lowest average RMSE and the third smallest average time in regression.

Considering that BoostForest's average model size was the largest and its base learners were more complex than baselines,' we also compared BoostForest with Bagging-LightGBM, which uses Bagging to integrate 600 LightGBMs, and BaggedTAO-l, which also uses Bagging to integrate multiple complex trees. To increase BaggedTAO-l's training speed on YB and LR, we selected the cut-point from the set of each feature's {10, 20, ..., 90} quantiles to initialize each TAO-l in BaggedTAO-l. LMT and M5P are also complex trees, which are compared with BoostForest in Section 4.8.

Supplementary Table 3, available online, compares BoostForest with Bagging-LightGBM. BoostForest significantly outperformed Bagging-LightGBM on 15 datasets, and achieved higher accuracies on 14 of the 18 classification datasets, and lower RMSEs on 11 of the 17 regression datasets. These results indicated that BoostForest can still achieve promising performance when its average model size was smaller than Bagging-LightGBM.

Supplementary Table 4, available online, compares BoostForest with BaggedTAO-l. BoostForest significantly outperformed BaggedTAO-l on four datasets, and achieved higher accuraies on 16 of the 17 classification datasets. BaggedTAO-l had smaller average model size, because TAO can use the sparsity penalty to remove unnecessary parameters. This idea may also be used to reduce BoostForest's model size.

4.8 Use Other Base Learners in BoostForest

Next, we studied if the strategy that BoostForest uses to combine multiple BoostTrees (data replica by bootstrapping, and random parameter selection from a parameter pool) can also be extended to other tree models, i.e., whether we can still achieve good ensemble learning performance when BoostTree is replaced by another tree model.

When Extra-tree, LMT and M5P were used as the tree model, the resulting forests were denoted as ETForest,

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

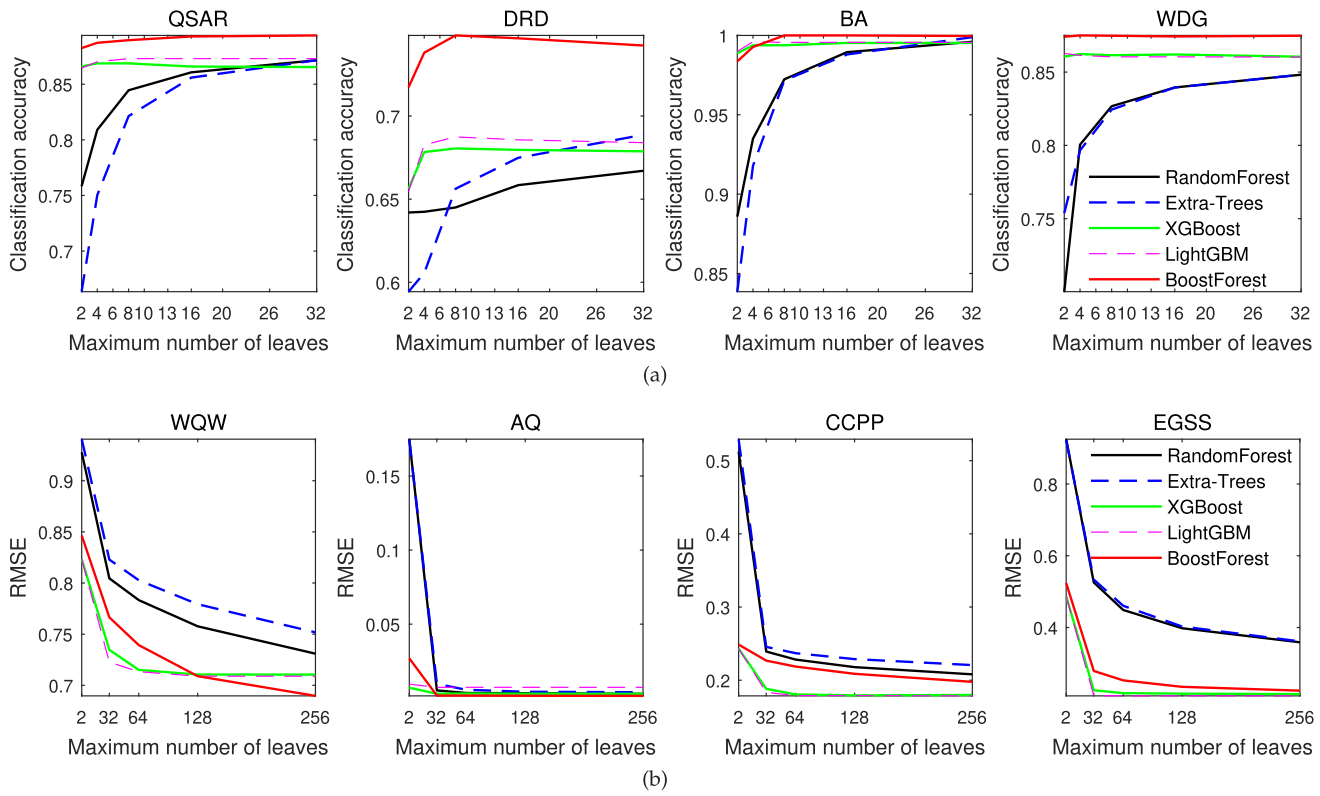


Fig. 3. Generalization performance w.r.t. the base learner model complexity, averaged over 10 repeats. (a) Average classification accuracies on the last four classification datasets, with different maximum number of leaves. Complete results on the 15 classification datasets are shown in Supplementary Fig. 3, available online. (b) Average RMSEs on the last four regression datasets, with different maximum number of leaves. Complete results on the 15 regression datasets are shown in Supplementary Fig. 4, available online.

LMForest and ModelForest, respectively. Individual M5P in ModelForest, LMT in LMForest, and Extra-tree in ETForest, was not pruned, and $min_samples_leaf$ was randomly sampled from {5, 6, ..., 15}. The parameters tuned for the baseline Extra-tree were max_depth and $min_samples_leaf$, whose candidate values were $\{1, 2, 4, 6, 8, 10\}$ and $\{5, 10, 15\}$, respectively. We set $min_samples_leaf = 10$ and $\lambda = 0.1$ in BoostTree.

Supplementary Table 5, available online, shows the results. ETForest outperformed Extra-tree on 30 of the 32 datasets. ModelForest outperformed M5P on all 16 regression datasets. BoostForest outperformed BoostTree on all 32 datasets. These results indicated that our proposed strategy for integrating BoostTrees into BoostForest can also be used to integrate Extra-tree and M5P into a composite learner with improved performance.

TABLE 4
Performances of the Five Ensemble Learning Approaches on Datasets With a Large Number of Samples and/or Features

Dataset	RandomForest	Extra-Trees	XGBoost	LightGBM	BoostForest
	Mean and standard deviation (in parentheses) of the classification accuracy				
YB	0.9913 (0.0033)	0.9797 (0.0085)•	0.9890 (0.0051)	0.9874 (0.0049)•	**0.9917** (0.0023)
LR	0.9662 (0.0018)•	0.9706 (0.0029)•	0.9637 (0.0026)•	0.9682 (0.0016)•	**0.9763** (0.0028)
SUSY	0.8026 (0.0004)•	0.8019 (0.0003)•	0.8039 (0.0003)•	0.8038 (0.0004)•	**0.8040** (0.0003)
Average accuracy	0.9200 (0.0015)	0.9174 (0.0030)	0.9189 (0.0023)	0.9198 (0.0018)	**0.9240** (0.0014)
Average rank	3.3333	4.0000	3.3333	3.3333	**1.0000**
Average time	4586.9995 (622.2597)	**2514.5877** (580.9703)	13698.1652 (2257.4636)	5552.3772 (107.2198)	2698.2770 (90.0081)
Average model size	3.2834×10^8	5.9279×10^8	3.0507×10^6	$\mathbf{1.0188 \times 10^6}$	5.1207×10^9
	Mean and standard deviation (in parentheses) of the regression RMSE				
PTS	0.5709 (0.0048)•	0.5685 (0.0048)•	0.5713 (0.0050)•	0.5577 (0.0062)•	**0.5504** (0.0065)
RLCT	0.0617 (0.0054)•	**0.0442** (0.0030)	0.0630 (0.0026)•	0.0605 (0.0026)•	0.0463 (0.0019)
Average RMSE	0.3163 (0.0038)	0.3063 (0.0035)	0.3172 (0.0027)	0.3091 (0.0035)	**0.2983** (0.0029)
Average rank	4.0000	2.0000	5.0000	2.5000	**1.5000**
Average time	183.7901 (2.9807)	**97.3998** (2.9675)	1550.5950 (551.1156)	1678.8888 (74.8529)	964.5815 (68.9347)
Average model size	1.1778×10^8	1.1787×10^8	$\mathbf{5.0945 \times 10^6}$	5.1119×10^6	2.5022×10^9

The best performance is marked in bold. • *indicates statistically significant win for boostforest.*

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

TABLE 5
Performances of the Seven Ensemble Learning Approaches on MNIST and HIGGS

MNIST				
Approach	#Estimators	Accuracy	Time	Model size
RandomForest	500	0.9724	1.7557×10^2	2.3051×10^8
Extra-Trees	500	0.9717	1.7513×10^2	3.3620×10^8
Bagging-XGBoost	500	0.9800	4.1711×10^5	2.9380×10^9
Bagging-LightGBM	500	0.9817	7.2468×10^4	7.3000×10^8
Bagging-CNN_{150}	100	0.9871	$[7.5563 \times 10^3]$	1.2560×10^6
Bagging-CNN_3	6300	0.9775	$[9.3731 \times 10^3]$	7.9128×10^7
BoostForest-CNN	100	**0.9887**	$[7.4071 \times 10^3]$	5.4808×10^7
HIGGS				
Approach	#Estimators	Accuracy	Time	Model size
RandomForest	500	0.7633	2.6710×10^4	1.1634×10^{10}
Extra-Trees	500	0.7532	2.3760×10^4	2.7498×10^{10}
Bagging-XGBoost	500	0.7705	1.7212×10^5	1.2034×10^{10}
Bagging-LightGBM	500	0.7740	2.8322×10^5	1.2034×10^{10}
Bagging-DNN_{150}	100	0.7873	$[3.6404 \times 10^4]$	1.4009×10^8
Bagging-DNN_3	6300	0.7712	$[4.6521 \times 10^4]$	8.8257×10^9
BoostForest-DNN	100	**0.7886**	$[2.6441 \times 10^4]$	7.9734×10^9

The highest classification accuracy is marked in bold. [·] means the training time was recorded by running the approach on a NVIDIA RTX 3090 GPU. CNN_{150} and DNN_{150} were trained with 150 epochs, and CNN_3 and DNN_3 were trained with three epochs.

However, LMForest only outperformed LMT on six of the 16 classification datasets, though it achieved better average performance than LMT, i.e., our proposed strategy is not very effective in combining multiple LMTs to further improve their performance. Additionally, using Python to call LMT's and M5P's Java APIs to generate the base learners in parallel is not very efficient, so the average time of LMForest and ModelForest was relatively long.

Finally, BoostForest outperformed LMForest on 14 datasets, among which five were statistically significant; BoostForest outperformed ETForest on 31 datasets, among which 24 were statistically significant; and, BoostForest outperformed ModelForest on 14 datasets, among which 11 were statistically significant.

In summary, BoostForest achieved better average classification performance than LMForest and ETForest, and also better average regression performance than ModelForest and ETForest, indicating that BoostTree is a more effective base learner for our proposed ensemble strategy than ExtraTree, LMT and M5P.

4.9 Use Other Regression Models in BoostTree

We also studied if other more complex and nonlinear regression algorithms, e.g., ELM and SVR, can be used to replace RR as the node function in BoostTree. The resulting trees are denoted as BoostTree-ELM and BoostTree-SVR, respectively, and the corresponding forests as BoostForest-ELM and BoostForest-SVR.

ELM [20] is a single hidden layer neural network. It randomly generates the hidden nodes, and analytically determines the output weights through generalized inverse or RR. Its model complexity can be controlled by the number of hidden nodes *NumHiddenNodes* and the regularization coefficient λ of RR. We set their candidate values to {10, 20, 30, 40} and {0.0001, 0.001, 0.01, 0.1, 1}, respectively, to construct the parameter pool. The sigmoid activation function was used in the hidden layer.

Linear SVR [21] was used in BoostTree-SVR. The parameter pool for the regularization parameter C and the slack variable ϵ was {0.01, 0.1, 1.0, 10, 100} and {0.1, 0.2, 0.4, 0.8, 1.0}, respectively. We set $min_samples_leaf = 10$ in BoostTree-ELM and BoostTree-SVR, and BoostForest-ELM and BoostForest-SVR to {5, 6, ..., 15}, respectively. Details of BoostTree-ELM, BoostForest-ELM, BoostTree-SVR and BoostForest-SVR are described in Supplementary Algorithms 2-7, available online.

Supplementary Table 6, available online, compares the performances of ELM and BoostTree-ELM with BoostForest-ELM (SVR and BoostTree-SVR with BoostForest-SVR) on the 15 regression datasets. BoostForest-ELM statistically significantly outperformed ELM (BoostTree-ELM) on 13 (15) datasets. BoostForest-SVR statistically significantly outperformed SVR (BoostTree-SVR) on 14 (15) datasets. BoostTree-ELM and BoostTree-SVR were more likely to overfit, because of their high model complexity. So, it is necessary to combine multiple BoostTrees into BoostForest to reduce over-fitting.

These results showed that when more complex and nonlinear regression algorithms, such as ELM and SVR, are used to replace RR as the node function in BoostTree, the performance may degrade due to overfit; however, integrating the corresponding BoostTrees into a BoostForest can always improve the performance.

We also explored if convolutional neural network (CNN) and deep neural network (DNN) can be used as the node function in BoostForest. The resulting forests are denoted as BoostForest-CNN and BoostForest-DNN, respectively. Supplementary Fig. 5, available online, shows the structures of CNN and DNN used in our experiments. We initialized each node model of BoostForest-CNN or BoostForest-DNN by using its parent node model's weights, scaled its output values to $(-4, 4)$, to be consistent with the range of pseudo-label in (24), and trained it with 20 epochs. For the MNIST dataset, we set $min_samples_leaf = 1,024$, batch size to 1,024, and tree depth to 5 in each BoostTree, and scaled the

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

minimum and maximum values of each feature to 0 and 1, respectively. For the HIGGS dataset, we set $min_samples_leaf = 2,048$, batch size to 2,048, and tree depth to 5 in each BoostTree, and randomly selected 2,000,000 samples to train each base learner in BoostForest-DNN and Bagging-DNN, respectively. If the number of samples belonging to the splitting node was larger than 100,000, we randomly selected 100,000 samples to identify the cut-point. When training CNN or DNN, we adopted AdamW[3] with betas of $(0.9, 0.999)$, initial learning rate 0.01, and weight decay 0.0001.

Table 5 compares the performances of BoostForest-CNN and BoostForest-DNN with six approaches on MNIST and HIGGS, respectively. With less training time, BoostForest-CNN and BoostForest-DNN achieved higher accuracies than Bagging-CNN_{150} and Bagging-DNN_{150}, respectively. With smaller model size, BoostForest-CNN (BoostForest-DNN) achieved higher accuracies than RandomForest, Extra-Trees, Bagging-XGBoost, Bagging-lightGBM, and Bagging-CNN_3 (Bagging-DNN_3). These results indicated that complex CNN and DNN can also be used as the node function in BoostForest.

4.10 Robustness of BoostForest

To investigate the robustness of BoostForest, we performed extensive experiments to study how its performance changed with different hyper-parameters. Additionally, we also studied the effect of the clipping operation in (17) and the splitting criterion.

4.10.1 Effect of $min_samples_leaf$ and λ

There are mainly two hyper-parameters in our proposed BoostForest: $min_samples_leaf$ and λ. Supplementary Table 7, available online, shows the average performances of BoostForest, as $min_samples_leaf$ increased from 5 to 15, and λ increased from 0.0001 to 1. Randomly selecting each BoostTree's parameters from the parameter pool to form a BoostForest achieved comparable average performance with using the best parameters in both regression and classification, indicating that we can use random parameter pool sampling to simplify BoostForest's parameter selection process.

4.10.2 Effect of the Clipping Operation in Regression

Clipping makes BoostForest more robust to noise and outliers in regression. To verify the effect of the clipping operation, we compared the performance of "BoostForest (w/o clipping)" with "BoostForest (w/ clipping)".

Supplementary Table 8, available online, shows the results. BoostForest (w/ clipping) outperformed BoostForest (w/o clipping) on most datasets. When the data noise was small, which means BoostForest can achieve a small RMSE, clipping made BoostForest too conservative to achieve better performance, e.g., on the AQ dataset.

Compared with the baselines in Table 2, even BoostForest (w/o clipping) achieved better average RMSE than RandomForest, Extra-Trees, XGBoost, LightGBM and GBDT-PL, indicating that BoostForest (w/o clipping) is also an effective model, though BoostForest (w/ clipping) is more effective.

4.10.3 Effect of the Splitting Criterion

We also studied if Gini-SC, C4.5-SC (usually used in C4.5 and LMT) or MSE-SC (usually used in CART and Extra-Trees), can replace XGB-SC in BoostForest.

Supplementary Table 9, available online, shows BoostForest's performances of using different splitting criteria. Using Gini-SC, C4.5-SC and XGB-SC achieved comparable average accuracies in classification. Using XGB-SC achieved better average RMSE than using MSE-SC in regression.

5 Conclusions and Future Research

This article has proposed a new tree model, BoostTree, that integrates GBM into a single model tree. BoostTree trains a regression model (for regression or binary classification) or multiple regression models (for multi-class classification) at each node. For a given input, BoostTree first sorts it down to a leaf, then computes the final prediction by summing up the outputs of all node models along the path from the root to that leaf.

Using BoostTrees as base learners, we also proposed a new ensemble learning approach, BoostForest. It first uses bootstrap to obtain multiple replicas of the training set, and then trains a BoostTree on each replica. Its hyper-parameters are easy to tune. Moreover, it represents a very general ensemble learning framework, whose base learners can be any tree model, e.g., BoostTree, Extra-Tree, M5P, or LMT, or even a mixture of different models.

BoostForest performs favorably over several ensemble learning approaches, e.g., RandomForest, Extra-Trees, XGBoost, LightGBM, and GBDT-PL, in both classification and regression, and also MultiBoosting, FilterBoost and DeepBoosting in classification. BoostForest simultaneously uses two randomness injection strategies: 1) data sample manipulation through bootstrapping; and, 2) input feature manipulation through randomly drawing cut-points at node splitting. Output representation manipulation will be considered in our future research.

Zhou and Feng [24] showed that Random Forests can be assembled into a Deep Forest to achieve better performance. As we have demonstrated that BoostForest generally outperforms Random Forest, it is also expected that replacing Random Forests in Deep Forest by BoostForests may result in better performance. This is also one of our future research directions.

BoostForest has two main limitations:

1) BoostForest cannot handle NULL values, because it needs all features to train a node model.
2) BoostForest may not be deployed to low storage devices, due to its large model size.

Therefore, our future research will:

1) Improve BoostTree to handle NULL values.
2) Reduce BoostForest's model size by using the sparsity penalty in TAO [45].

3. https://pytorch.org/docs/stable/generated/torch.optim.AdamW

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] Z. H. Zhou, *Ensemble Methods: Foundations and Algorithms*. Boca Raton, FL, USA: CRC, 2012.

[2] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY, USA: Springer, 2009.

[3] S. Wang et al., "Massive computational acceleration by using neural networks to emulate mechanism-based biological models," *Nature Commun.*, vol. 10, no. 1, pp. 1–9, 2019.

[4] X. Sun, Y. Liu, and L. An, "Ensemble dimensionality reduction and feature gene extraction for single-cell RNA-seq data," *Nature Commun.*, vol. 11, no. 1, pp. 1–9, 2020.

[5] Y. Cao, T. A. Geddes, J. Y. H. Yang, and P. Yang, "Ensemble deep learning in bioinformatics," *Nature Mach. Intell.*, vol. 2, no. 9, pp. 500–508, 2020.

[6] S. Paisitkriangkrai, C. Shen, and A. V. D. Hengel, "Pedestrian detection with spatially pooled features and structured ensemble learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 6, pp. 1243–1257, Jun. 2016.

[7] C. Yang et al., "Lunar impact crater identification and age estimation with chang'E data by deep and transfer learning," *Nature Commun.*, vol. 11, no. 1, pp. 1–15, 2020.

[8] Y. F. Li, L. Z. Guo, and Z. H. Zhou, "Towards safe weakly supervised learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 1, pp. 334–346, Jan. 2021.

[9] S. S. Kim, K. K. Dey, O. Weissbrod, C. Marquez-Luna, S. Gazal, and A. L. Price, "Improving the informativeness of mendelian disease pathogenicity scores for common disease," *Nature Commun.*, vol. 11, pp. 1–15, 2020.

[10] R. Agius et al., "Machine learning can identify newly diagnosed patients with CLL at high risk of infection," *Nature Commun.*, vol. 11, no. 1, pp. 1–17, 2020.

[11] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, 2001.

[12] T. Q. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 785–794.

[13] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Boca Raton, FL, USA: CRC, 1984.

[14] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996.

[15] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Mach. Learn.*, vol. 51, no. 2, pp. 181–207, 2003.

[16] E. K. Tang, P. N. Suganthan, and X. Yao, "An analysis of diversity measures," *Mach. Learn.*, vol. 65, no. 1, pp. 247–271, 2006.

[17] G. Brown, J. Wyatt, R. Harris, and X. Yao, "Diversity creation methods: A survey and categorisation," *Inf. Fusion*, vol. 6, no. 1, pp. 5–20, 2005.

[18] J. R. Quinlan, "Learning with continuous classes," in *Proc. 5th Australian Joint Conf. Artif. Intell.*, 1992, pp. 343–348.

[19] Y. Wang and I. H. Witten, "Induction of model trees for predicting continuous classes," in *Proc. 9th Eur. Conf. Mach. Learn.*, 1997.

[20] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1–3, pp. 489–501, 2006.

[21] H. Drucker, C. J. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines," in *Proc. Adv. Neural Inf. Process. Syst.*, 1997, pp. 155–161.

[22] A. Jain, J. Mao, and K. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, 1996.

[23] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is NP-complete," *Inf. Process. Lett.*, vol. 5, no. 1, pp. 15–17, 1976.

[24] Z. H. Zhou and J. Feng, "Deep forest," *Nat. Sci. Rev.*, vol. 6, no. 1, pp. 74–86, 2018.

[25] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *Proc. 13th Int. Conf. Mach. Learn.*, 1996, pp. 148–156.

[26] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[27] I. Barandiaran, "The random subspace method for constructing decision forests," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 20, no. 8, pp. 1–22, Aug. 1998.

[28] Y. Amit and D. Geman, "Shape quantization and recognition with randomized trees," *Neural Computation*, vol. 9, no. 7, pp. 1545–1588, 1997.

[29] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," *Mach. Learn.*, vol. 63, no. 1, pp. 3–42, 2006.

[30] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-class AdaBoost," *Statist. Interface*, vol. 2, no. 3, pp. 349–360, 2009.

[31] J. H. Friedman, "Stochastic gradient boosting," *Comput. Statist. Data Anal.*, vol. 38, no. 4, pp. 367–378, 2002.

[32] L. Mason, J. Baxter, P. L. Bartlett, and M. R. Frean, "Boosting algorithms as gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2000, pp. 512–518.

[33] J. K. Bradley and R. E. Schapire, "Filterboost: Regression and classification on large datasets," in *Proc. 20th Int. Conf. Neural Inf. Process. Syst.*, 2007, pp. 185–192.

[34] C. Cortes, M. Mohri, and U. Syed, "Deep boosting," in *Proc. 31st Int. Conf. Mach. Learn.*, 2014, pp. 1179–1187.

[35] G. Ke et al., "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 3146–3154.

[36] T. Q. Chen and T. He, "Higgs boson discovery with boosted trees," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 69–80.

[37] A. Rakhlin, A. Shvets, V. Iglovikov, and A. A. Kalinin, "Deep convolutional neural networks for breast cancer histology image analysis," in *Proc. 15th Int. Conf. Image Anal. Recognit.*, 2018, pp. 737–744.

[38] L. Liu, M. Ji, and M. Buchroithner, "Combining partial least squares and the gradient-boosting method for soil property retrieval using visible near-infrared shortwave infrared spectra," *Remote Sens.*, vol. 9, no. 12, pp. 1299–1317, 2017.

[39] J. Walsh, I. T. Heazlewood, and M. Climstein, "Regularized linear and gradient boosted ensemble methods to predict athletes' gender based on a survey of masters athletes," *Model Assist. Statist. Appl.*, vol. 14, no. 1, pp. 47–64, 2019.

[40] N. Landwehr, M. Hall, and E. Frank, "Logistic model trees," *Mach. Learn.*, vol. 59, no. 1–2, pp. 161–205, 2005.

[41] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: A statistical view of boosting," *Ann. Statist.*, vol. 28, no. 2, pp. 337–407, 2000.

[42] D. Malerba, A. Appice, M. Ceci, and M. Monopoli, "Trading-off local versus global effects of regression nodes in model trees," in *Proc. 13th Int. Symp. Foundations Intell. Syst.*, 2002, pp. 393–402.

[43] Y. Shi, J. Li, and Z. Li, "Gradient boosting with piece-wise linear regression trees," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, 2019, pp. 3432–3438.

[44] G. I. Webb, "Multiboosting: A technique for combining boosting and wagging," *Mach. Learn.*, vol. 40, no. 2, pp. 159–196, 2000.

[45] M. A. Carreira-Perpinan and P. Tavallali, "Alternating optimization of decision trees, with application to learning sparse oblique trees," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 1211–1221.

[46] M. A. Carreira-Perpinan and A. Zharmagambetov, "Ensembles of bagged TAO trees consistently improve over random forests, AdaBoost and gradient boosting," in *Proc. ACM-IMS Foundations Data Sci. Conf.*, 2020, pp. 35–46.

[47] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Mateo, CA, USA: Morgan Kaufmann, 1994.

[48] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *J. Roy. Statist. Soc.*, vol. 57, no. 1, pp. 289–300, 1995.



Changming Zhao received the BE degree in automation from the Huazhong University of Science and Technology, Wuhan, China, in 2018. He is currently working toward the PhD degree in artificial intelligence. His research interest includes machine learning.

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.



Dongrui Wu (Fellow, IEEE) received the BE degree in automation from the University of Science and Technology of China, in 2003, the ME degree in electrical engineering from the National University of Singapore in 2005, and the PhD degree in electrical engineering from the University of Southern California, in 2009. He is now professor with the School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, China. His research interests include affective computing, brain-computer interface, computational intelligence, and machine learning. He has more than 200 publications. He received the IEEE CIS Outstanding PhD Dissertation Award in 2012, the NAFIPS Early Career Award in 2014, the IEEE SMC Society Early Career Award in 2017, the USERN Prize in Formal Sciences in 2020, and the Chinese Association of Automation Early Career Award in 2021. He is a BoG member and an associate vice president of the IEEE SMC Society. He will be the editor-in-chief of IEEE TFS in 2023.



Jian Huang (Senior Member, IEEE) received the BE, ME, and PhD degrees from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1997, 2000, and 2005, respectively. He was a postdoctoral researcher and research fellow with Nagoya University, Japan. He is now professor with the School of Artificial Intelligence and Automation, HUST, and a guest professor with Nagoya University and University Paris-Est Creteil. His research interests include rehabilitation robot, robotic assembly, networked control systems and bioinformatics. He has more than 300 papers. He is an associate editor of *IEEE Transactions on Fuzzy Systems* and Springer ROBOMECH.



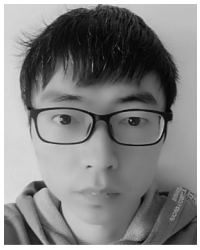
Ye Yuan (Senior Member, IEEE) received the BE degree (valedictorian) in automation from Shanghai Jiao Tong University, China, in 2008, and the MPhil and PhD degrees in engineering from the University of Cambridge, U.K., in 2009 and 2012, respectively. He was a postdoctoral researcher with UC Berkeley, and a junior research fellow with the University of Cambridge. Since 2016, he has been professor with the Huazhong University of Science and Technology, Wuhan, China. His research interests include system identification, machine learning, and control theory with applications to manufacturing and healthcare.



Hai-Tao Zhang (Senior Member, IEEE) received the BE and PhD degrees from the University of Science and Technology of China, Hefei, China, in 2000 and 2005, respectively. During January-December 2007, he was a postdoctoral researcher with the University of Cambridge, Cambridge, U.K. Since 2005, he has been with Huazhong University of Science and Technology, Wuhan, China, where he was an associate professor, from 2005 to 2010, and has been a full professor, since 2010. His research interests include swarming intelligence, model predictive control, and unmanned system cooperation control. He was the recipient of the National Science Fund for Distinguished Young Scholars of China. He is/was an associate editor of *IEEE Transactions on Systems, Man, and Cybernetics*, *IEEE Transactions on Circuits and Systems II: Express Briefs*, *Engineering*, and *Asian Journal of Control*.



Ruimin Peng received the BE degree in software engineering from Northwestern Polytechnical University, Xi'an, China, in 2019. She is currently working toward the PhD degree with the School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, China. Her research interests include brain-computer interfaces and machine learning.



Zhenhua Shi received the BE and PhD degrees in automation from the Huazhong University of Science and Technology, Wuhan, China, in 2017 and 2022, respectively. He is currently with the Second Ship Design and Research Institute, Wuhan, Hubei, China. His research interests include machine learning and brain-computer interfaces.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/csdl.

Authorized licensed use limited to: Reykjavik University. Downloaded on May 22,2024 at 13:17:10 UTC from IEEE Xplore. Restrictions apply.