BoostTree and BoostForest for Ensemble Learning

Changming Zhao[1], Dongrui Wu[1,*], Jian Huang[1], Ye Yuan[1], Hai-Tao Zhang[1], Ruimin Peng[1], Zhenhua Shi[1] and Chenfeng Guo[1]

[1]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, China. *e-mail: drwu@hust.edu.cn

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 achieves high randomness (diversity) by sampling its parameters randomly from a parameter pool, and selecting a subset of features randomly at node splitting. BoostForest further increases the randomness by bootstrapping the training data in constructing different BoostTrees. BoostForest outperformed four classical ensemble learning approaches (Random Forest, Extra-Trees, XGBoost and LightGBM) on 34 classification and regression datasets. Remarkably, BoostForest has only one hyper-parameter (the number of BoostTrees), which can be easily specified. Our code is publicly available, and the proposed ensemble learning framework can also be used to combine many other base learners.

Ensemble learning[1,2] trains multiple base learners to explore 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[3–7], climate prediction[8], healthcare[9,10], materials design[11,12], Moon exploration[13], etc.

For example, in biology, Wang et al.[4] 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 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 2 years of diagnosis. To accelerate materials design, Lansford and Vlachos[11] used a neural network ensemble to characterize surface microstructure of complex materials. In Moon exploration, Yang et al.[13] 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.

One of the most popular algorithms for constructing the base learners is the decision tree[14–17]. Two common approaches for constructing the composite learner are bootstrap aggregating (Bagging) and boosting.

Bagging[18] 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[19–21].

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[22] constructs a linear regression function at each 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′)[23]

trains linear models at each leaf of a pruned tree to reduce the risk of over-fitting. Any regression model, e.g., Ridge Regression[2] (RR), Extreme Learning Machine[24] (ELM), Support Vector Regression[25] (SVR), and Neural Network, can be used as the node model. These regression models have some hyper-parameters to tune, such as the regularization coefficient and the number of nodes in the hidden layer. It is an NP-hard problem to simultaneously determine the structure of the model tree and parameters of each node model. A common approach is cross-validation, but it is very time-consuming. It is desirable to develop a strategy that can make the model tree more compatible with these regression models.

Approaches to increase the diversity of base learners in an ensemble. Diversity enhancement strategies can be divided into three categories[26]: 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[27] 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[28–31] 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[32] (Extra-Trees) splits nodes by choosing the cut-points completely randomly. 3) Parameter-based strategies. If the base learners are sensitive to the parameters, 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, their combination may lead to better performance.

Boosting[27,33,34], the driving force of Gradient Boosting Machine[14] (GBM), 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.[35] described boosting from the viewpoint of gradient descent and regarded boosting as a stage-wise learning scheme to optimize different objective functions iteratively. Popular implementations of GBMs, e.g., XGBoost[15] and LightGBM[17], have been successfully used in many applications[36–39].

Friedman et al.[40] proposed LogitBoost (Supplementary Algorithm 1) to optimize logistic regression by maximum likelihood. It generates the ensemble by performing Newton updates iteratively. In each iteration, LogitBoost first computes the working response and weights using Newton (for two-class) or quasi-Newton (for multi-class) method, and then the ensemble is updated by adding a new model, which is trained to fit the working response by a weighted least-squares regression. However, traditional boosting approaches[14,15,17] 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 and is robust to them.

This article 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 GBMs into a single decision tree, as shown in Figure 1a. BoostTree trains a linear or nonlinear function, e.g., RR, ELM, or SVR, 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. BoostTree achieves high randomness (diversity) by selecting a subset of features randomly at node splitting.

2. We propose a novel diversity enhancement strategy, random parameter pool sampling, which makes BoostTree more robust to hyper-parameters. In this strategy, the parameters of the BoostTree are not specific values, but random samples from candidate sets stored in a parameter pool. Each time a node model is generated, its parameters are randomly selected from the parameter pool. This further improves the diversity of BoostTrees.

3. Using BoostTrees as base learners, we propose

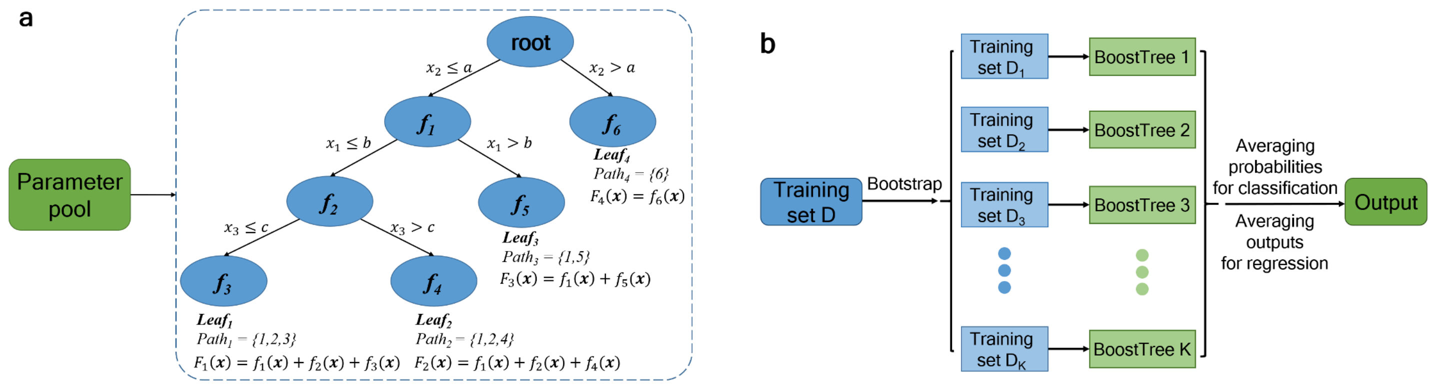

Figure 1: **BoostTree and BoostForest. a** BoostTree with four leaves. BoostTree uses GBM to train a linear or nonlinear function at each node. For a given input, BoostTree first sorts it down to a leaf $q(\boldsymbol{x})$, then computes the final prediction by summing up the outputs of all node models along the path (given by $Path_{q(\boldsymbol{x})}$) 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.

a novel ensemble learning approach, BoostForest (Figure 1b). It uses bootstrap to obtain replicas of the training set, and trains a BoostTree on each replica. It has only one hyper-parameter (the number of BoostTrees), and outperforms several classical ensemble learning approaches. Moreover, it represents a very general ensemble learning framework, whose base learners can be any model, e.g., BoostTree, decision tree, or neural network, or even a mixture of different models.

The details of BoostTree and BoostForest are described in Supplementary Algorithms 2-6.

Results

Experiments were carried out to verify the effectiveness of BoostForest in both classification and regression. For simplicity, RR was used as the default node function.

The following six questions were examined:

1. What is the generalization performance of BoostForest, compared with classical ensemble learning approaches, e.g., RandomForest[28], Extra-Trees[32], XGBoost[15] and LightGBM[17]?

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 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 base learners, e.g., classification and regression tree[16] (CART), model tree[22], and logistic model tree[41] (LMT)?

5. How does the performance of BoostForest change with different node functions, e.g., RR, ELM or SVR, are used in BoostTrees?

6. Can BoostForest handle large datasets?

Datasets. We performed experiments on 34 real-world datasets[1] (17 for classification and 17 for regression), summarized in Supplementary Table 1. They cover a wide range of conditions in terms of the number of features (between 4 and 784) and the sample size (between 103 and 70,000).

For each dataset, categorical features were converted to numerical ones by one-hot encoding. Unless stated otherwise, the numerical features were scaled to $[0, 1]$, and the labels were z-normalized for regression datasets.

Algorithms and parameters. BoostForest was compared with two classical Bagging approaches, Random Forest[28] and Extra-Trees[32], and also two popular boosting approaches, XGBoost[15] and LightGBM[17].

[1]http://archive.ics.uci.edu/ml/datasets.php; http://yann.lecun.com/exdb/mnist/

Hyper-parameters of the four baselines are summarized in Supplementary Table 2. When the number of samples was smaller than 10,000, the best parameter combination was determined by grid-search using inner 5-fold cross-validation; otherwise, 30% samples were reserved from the training set to form a validation set for parameter selection. We used early-stopping to control the number of boosting iterations for XGBoost and LightGBM, and out-of-bag error to select the parameters of Random Forest and Extra-Trees. After parameter selection, all samples in the training set were used to train the model.

The number of base learners in BoostForest and its variants was set to 100. Its parameter pool consisted of the minimum number of samples on each leaf *MinSamplesLeaf* and the regularization coefficient λ. We set their candidate set to $\{5, 6, \ldots, 15\}$ and $\{0.0001, 0.001, 0.01, 0.1\}$, respectively.

Performance measures. 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 a rank for each algorithm on each dataset. For K algorithms, the best one has rank 1, and the worst rank K.

Generalization performance of BoostForest. First, we compared the generalization performance of BoostForest with the four baselines. Table 1 shows the results, averaged over five repeats of 2-fold cross-validations. BoostForest achieved the best generalization performance on 25 out of the 30 datasets, and comparable performance with the best baseline on another dataset (BH).

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[42] to adjust them. The statistically significant ones are marked by $\bullet$ in Table 1. BoostForest significantly outperformed RandomForest on 25 datasets, Extra-Trees on 25 datasets, XGBoost on 22 datasets, and LightGBM on 22 datasets.

Note that BoostForest has only one hyper-parameter (the number of BoostTrees), which can be easily specified. Each BoostTree is trained with a different training set, so BoostForest can be easily parallelized. So, BoostForest is very easy to use in practice.

Generalization performance w.r.t. the number of base learners. As mentioned above, BoostForest only 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 3 to 100, and tuned other parameters of the four baselines by grid-search using inner 5-fold cross-validation.

Figure 2a shows the accuracies of the five algorithms on the last four classification datasets, averaged over two repeats of 5-fold cross-validations. Complete results on all 15 classification datasets are shown in Supplementary Figure 1. Generally, as the number of base learners increased, the performances of all ensemble learning approaches first quickly increased and then converged. BoostForest achieved the highest classification accuracy on 14 of the 15 datasets, and the second highest on the remaining one (ILP).

Figure 2b shows the RMSEs of the five algorithms on the last four regression datasets, averaged over two repeats of 5-fold cross-validations. Complete results on all 15 regression datasets are shown in Supplementary Figure 2. Again, as the number of base learners increased, generally the performances of all algorithms rapidly increased and then converged. BoostForest achieved the smallest RMSE on 10 datasets, and the second smallest RMSE on another three (BH, ASN and EGSS).

Generally, BoostForest converges within 50 BoostTrees.

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.

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 of the ensemble, whereas boosting for integrating simple base learners to reduce the bias of the ensemble. In this study, the base learner model complexity was controlled by the maximum number of leaves per tree, which was gradually increased from 2 to 30 for classification, and 2 to 256

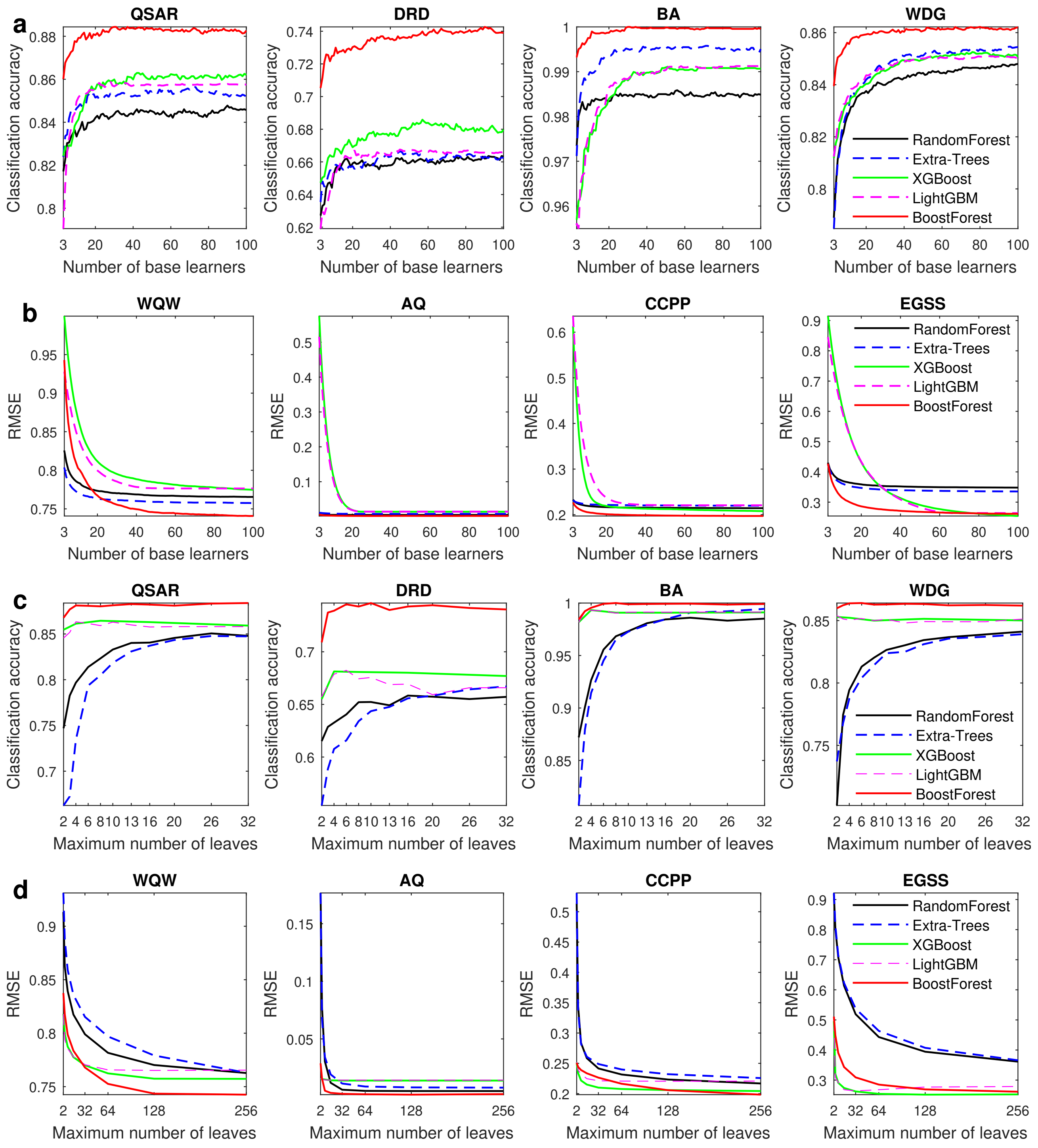

Figure 2: **Generalization performance w.r.t. the number of base learners and the base learner model complexity**, averaged over five repeats of 2-fold cross-validation. **a** average classification accuracies on the last four classification datasets, with different number of base learners. Complete results on the 15 classification datasets are shown in Supplementary Figure 1. **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 Figure 2. **c** 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 Figure 3. **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 Figure 4.

for regression. We fixed the number of base learners at 100, and tuned other parameters of the four baselines by grid-search using inner 5-fold cross-validation.

Figure 2c shows the accuracies of the five algorithms on the last four classification datasets, averaged over five repeats of 2-fold cross-validation. Complete results on all 15 classification datasets are shown in Supplementary Figure 3. On most datasets, the performances of all algorithms increased as the maximum number of leaves per tree increased. BoostForest always achieved the highest classification accuracy on all 15 datasets.

Figure 2d 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 Figure 4. Again, for most datasets, the performances of all algorithms increased as the maximum number of leaves per tree increased. BoostForest achieved the the smallest RMSE on 11 datasets, and the second smallest RMSE on another two (ASN and EGSS).

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 parameters 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 base learner, e.g., CART. The resulting forest is denoted as CARForest. The parameters to be tuned for the baseline CART were *maxDepth* and *minSamplesLeaf*, whose candidate value set was $\{4, 6, 8\}$ and $\{5, 10, 15\}$, respectively.

Table 2 compares the performances of CART with CARForest on the 30 datasets. The best parameter combination of CART was determined by grid-search using inner 5-fold cross-validation. All results were averaged over five repeats of 2-fold cross-validations. CARForest outperformed CART on 28 of the 30 datasets, among which 27 were statistically significant. More experimental results on using LMT and M5P as the base learner are given in Supplementary Tables 3 and 4. All tables show that our strategy for integrating BoostTrees into BoostForest can also be used to integrate other base learners into a composite learner for improved performance.

Tables 1 and 2, and Supplementary Tables 3 and 4, together show that BoostForest achieved better average classification performance than LMForest and CARForest, and also better average regression performance than ModelForest and CARForest, indicating that BoostTree is a more effective base learner than CART, LMT, and M5P.

Use other regression models in BoostTree. We also studied if other more complex and nonlinear regression models, 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[24] 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 $\{0.001, 0.01, 0.1\}$, respectively, to construct the parameter pool. Sigmoid activation functions were used in the hidden layer.

Linear SVR[25] was used in BoostTree-SVR. The parameter pool for the regularization parameter C and the slack variable ϵ was $\{0.1, 1, 2, 5, 10\}$ and $\{0.1, 0.2, 0.4, 0.8\}$, respectively.

A hyper-parameter of BoostTree-ELM and BoostTree-SVR was the maximum number of leaves *MaxNumLeaf*, whose candidate values were $\{5, 10, 15, 20\}$. The best *MaxNumLeaf* of BoostTree-ELM and BoostTree-SVR was determined by grid-search from its candidate values using inner 5-fold cross-validation. Details of BoostTree-ELM, BoostForest-ELM, BoostTree-SVR and BoostForest-SVR are described in Supplementary Algorithms 7-12.

Table 3 compares the generalization performance of ELM and BoostTree-ELM with BoostForest-ELM (SVR and BoostTree-SVR with BoostForest-SVR) on the 15 regression datasets. Their results were averaged over five repeats of 2-fold cross-validations. We also used the t-test adjusted by BH-FDR to check if BoostForest-ELM or BoostForest-SVR significantly outperformed the baselines ($\alpha = 0.05$). BoostForest-ELM statistically significantly outperformed ELM (BoostTree-ELM) on 13 (14) datasets. BoostForest-SVR statistically significantly outperformed SVR and

BoostTree-SVR on 14 datasets. When the number of samples is small, BoostTree-ELM and BoostTree-SVR are more likely to overfit, because of their high model complexity and random parameters. So, it is necessary to combine multiple BoostTrees into BoostForest to reduce over-fitting.

Generalization performance on large datasets. Previous experiments have shown the superiority of BoostForest on small to medium sized datasets with not too high dimensionalities. Next, we investigated its performance on large datasets with high dimensionalities.

Table 4 compares the performance of BoostForest with four classical and popular ensemble methods on two large classification datasets and two large regression datasets. Their results were averaged over five repeats of 2-fold cross-validations. In order to observe the effect of feature dimensions on BoostForest, we also tested how BoostForest performed on MNIST-PCA and RLCT-PCA [the feature dimensionality of MNIST and RLCT were reduced to 10 using principal component analysis[43] (PCA)].

BoostForest still demonstrated good and consistent performance: it achieved the highest average accuracy in classification, and the lowest average RMSE in regression. However, on MNIST and RLCT, BoostForest did not achieve the best performance unless PCA was used to reduce the feature dimensionality. Our future research will make BoostForest more suitable for high-dimensional data.

Discussion

This article has proposed a new tree model, BoostTree, that integrates GBMs into a single decision tree. BoostTree trains a linear or nonlinear function 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 uses bootstrap to obtain replicas of the training set, and trains a BoostTree on each replica. It has only one hyper-parameter (the number of BoostTrees). Moreover, it represents a very general ensemble learning framework, whose base learners can be any model, e.g., BoostTree, decision tree, or neural network, or even a mixture of different models.

BoostForest performs favorably over classical ensemble learning approaches, e.g., Random Forest, Extra-Trees, XGBoost and LightGBM, in both classification and regression, and also MultiBoosting[44] in classification (Supplementary Materials), because it simultaneously uses three of the four randomness injection strategies[1]: 1) data sample manipulation through bootstrapping; 2) input feature manipulation through random feature subset selection at BoostTree node splitting; and, 3) learning parameter manipulation through random selection from the parameter pool. The fourth strategy, output representation manipulation, will be considered in our future research.

Recently, Zhou and Feng[26] 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.

Finally, we will also apply BoostForest to challenging problems in biology, engineering, healthcare, etc., in which ensemble learning has found many successful applications[3–13].

Methods

Given a dataset $\mathcal{D} = \{(\boldsymbol{x}_n, y_n)\}_{n=1}^N$ with N training examples, where $\boldsymbol{x}_n \in \mathbb{R}^{D\times 1}$ and D is the feature dimensionality, an ensemble ϕ generated by gradient boosting[14,15] uses K base learners to predict the output:

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

where each f_k is a base learner (usually a decision tree). GBM[14] generates the ensemble via an iterative process. In each iteration, gradient boosting learning first trains a new base learner according to the negative gradient direction, and then performs line search to determine the optimal step size.

Different from GBM, LMT[41] for classification generates only one tree instead of multiple trees. It integrates logistic regression into a decision tree, and uses LogitBoost[40] to train a set of linear models iteratively at each node.

Our proposed BoostTree is inspired by LMT. Assume a BoostTree has M nodes, excluding the root. Then, we train a function $f_m(\boldsymbol{x})$ for the m-th node, $m \in [1, M]$. For an input $\boldsymbol{x}$, BoostTree first determines $q(\boldsymbol{x})$, the leaf node it belongs to, and then all $f_m(\boldsymbol{x})$ along the path from the root to that leaf node is summed up to predict the output, i.e.,

$$\hat{y} = F(\boldsymbol{x}) = \sum_{m \in \mathrm{Path}_{q(\boldsymbol{x})}} f_m(\boldsymbol{x}), \tag{2}$$

where $\mathrm{Path}_{q(\boldsymbol{x})}$ is the collection of the node indices along the path from the root to the leaf node $q(\boldsymbol{x})$.

BoostTree minimizes the following regularized loss function:

$$\mathcal{L}(F) = \sum_{n=1}^{N} \ell(y_n, \hat{y}_n) + \sum_{m=1}^{M} \lambda_m \Omega(f_m), \tag{3}$$

where λ_m is the regularization coefficient of f_m. The second term above penalizes the complexity of BoostTree to reduce over-fitting.

Different loss functions ℓ can be used to deal with regression and classification problems. For the ease of optimization, we require ℓ to be convex and differentiable.

In general, the objective function in (3) cannot be optimized directly. Inspired by LMT and GBM, BoostTree is constructed in an additive manner. Assume a tree with T ($T \geq 2$) leaves have been generated after $T-1$ iterations. Then, there are $M = 2T-2$ nodes, excluding the root. We can rewrite (3) as:

$$\mathcal{L}(F) = \sum_{t=1}^{T} \mathrm{LeafLoss}_t + \sum_{m=1}^{2T-2} \lambda_m \Omega(f_m), \tag{4}$$

where

$$\mathrm{LeafLoss}_t = \sum_{n \in I_t} \ell \left[y_n, \sum_{m \in \mathrm{Path}_t} f_m(\boldsymbol{x}_n) \right], \tag{5}$$

$$I_t = \{n | q(\boldsymbol{x}_n) = t\}, \tag{6}$$

i.e., I_t is the set of all training samples belonging to Leaf t. $\mathrm{LeafLoss}_t$ measures the impurity score of Leaf t. In each iteration, the leaf with the highest impurity score is selected to be split. Then, a greedy learning scheme is used to add branches to that leaf.

Let I_m be the set of all training samples belonging to node m to be split. 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 linear models of the left and the right nodes trained separately using I_L and I_R, respectively. Then, the reduction of the loss in equation (3) is:

$$\begin{aligned}\delta_{\mathcal{L}} = & \sum_{n \in I_m} \ell[y_n, F_{I_m}(\boldsymbol{x}_n)] - \sum_{n \in I_L} \ell[y_n, F_{I_m}(\boldsymbol{x}_n) + f_L(\boldsymbol{x}_n)] \\ & - \sum_{n \in I_R} l[y_n, F_{I_m}(\boldsymbol{x}_n) + f_R(\boldsymbol{x}_n)] - \lambda_L \Omega(f_L) \\ & - \lambda_R \Omega(f_R),\end{aligned} \tag{7}$$

where

$$F_{I_m}(\boldsymbol{x}_n) = \sum_{i \in \mathrm{Path}_m} f_i(\boldsymbol{x}), \tag{8}$$

is the ensemble of the models along the path from the root node to node m, and λ_L (λ_R) represents the regularization coefficient of f_L (f_R) trained in the left (right) child node.

Considering using RR as the node function,which uses the Euclidean norm of coefficients to constrain the model complexity, and using Taylor's theorem. Referring to equation (7) in XGBoost[15], (7) can be approximated as follows:

$$\begin{aligned}\delta_{\mathcal{L}} \approx & \frac{1}{2} \left[\frac{\left(\sum_{n \in I_L} g_n\right)^2}{\sum_{n \in I_L} h_n + \lambda_L} + \frac{\left(\sum_{n \in I_R} g_n\right)^2}{\sum_{n \in I_R} h_i + \lambda_R} \right. \\ & \left. - \frac{\left(\sum_{n \in I} g_n\right)^2}{\sum_{n \in I} h_n + 0.0001} \right] \\ & - \lambda_L * \|\boldsymbol{w}_L\|_2^2 - \lambda_R * \|\boldsymbol{w}_R\|_2^2,\end{aligned} \tag{9}$$

where

$$g_n = \frac{\partial \ell(y_n, F_{I_m}(\mathbf{x}_n))}{\partial F_{I_m}(\mathbf{x}_n)}, \tag{10}$$

$$h_n = \frac{\partial^2 \ell(y_n, F_{I_m}(\mathbf{x}_n))}{\partial F_{I_m}(\mathbf{x}_n)^2}, \tag{11}$$

are the first and second order derivatives of loss function with respect to $F_{I_m}(\mathbf{x}_n)$, respectively. 0.0001 in the denominator is used to prevent the situation that the denominator is 0. $\boldsymbol{w}_L$ and $\boldsymbol{w}_R$ is the coefficients of RR model.

The splitting algorithm of BoostTree is shown in Supplementary Algorithm 2, where the subfunction *FitModel* assumes different forms according to different learning tasks, as shown in Supplementary Algorithms 3-5. We use gradient boosting to train the linear models for f_L and f_R in both regression and classification. In Supplementary Algorithm 2, we optimize (9) in two steps:

1. Ignoring the coefficients of the linear model, only consider the bias to calculate $\delta_{\mathcal{L}}$ in all possible splits:

$$\begin{aligned}\delta_{\mathcal{L}} \approx & \frac{1}{2} \left[\frac{\left(\sum_{n \in I_L} g_n\right)^2}{\sum_{n \in I_L} h_n + \lambda_L} + \frac{\left(\sum_{n \in I_R} g_n\right)^2}{\sum_{n \in I_R} h_i + \lambda_R} \right. \\ & \left. - \frac{\left(\sum_{n \in I} g_n\right)^2}{\sum_{n \in I} h_n + 0.0001} \right],\end{aligned} \tag{12}$$

and then determine the best splitting feature of the current node according to the maximum $\delta_{\mathcal{L}}$.

2. Fit the coefficients of the linear model.

BoostTree for regression. For regression problems, we use

$$\ell(y_n, \hat{y}_n) = (y_n - \hat{y}_n)^2, \tag{13}$$

and linear $f_m(\boldsymbol{x})$:

$$f_m(\boldsymbol{x}) = \boldsymbol{w}_m^T \boldsymbol{x} + b_m, \quad m = 1, \ldots, M \tag{14}$$

where $\boldsymbol{w}_m \in \mathbb{R}^{D\times 1}$ is a vector of the regression coefficients, and b_m is the intercept.

The loss function for the m-th node is:

$$\begin{aligned}\mathcal{L}(f_m) = &\sum_{n\in P_m} \ell[y_n, F_{P_m}(\boldsymbol{x}_n) + f_m(\boldsymbol{x}_n)] \\ &+ \lambda_m \|\boldsymbol{w}_m\|_2^2,\end{aligned} \tag{15}$$

where P_m is the parent node of the m-th node. In each iteration, GBM fits the pseudo-response $\tilde{y}_n = y_n - F_I(\boldsymbol{x}_n)$, which is the residual between the true value and the prediction, to minimize the above loss.

Supplementary Algorithm 3 shows the details of BoostTree for regression.

BoostTree for binary classification. In classification tasks, BoostTree is built using a LogitBoost-like algorithm, which iteratively updates the logistic linear models $F(\boldsymbol{x})$ by adding a new model $f_m(\boldsymbol{x})$ to $F(\boldsymbol{x})$. We perform Newton updates to fit the linear model at each node.

The cross-entropy loss is used in binary classification:

$$\begin{aligned}\ell(y_n, \hat{y}_n) = &- y_n \log[\text{sigmod}(\hat{y}_n)] \\ &- (1 - y_n)\log[1 - \text{sigmod}(\hat{y}_n)],\end{aligned} \tag{16}$$

where

$$\text{sigmod}(\hat{y}_n) = \frac{1}{1 + e^{-\hat{y}_n}}. \tag{17}$$

$f_m(\boldsymbol{x})$ is again linear, as in (14). The loss function for the m-th node can still be expressed by (15).

To improve the robustness of BoostTree to outliers, we (optionally) filter out samples whose weights are smaller than 5% quantile of all weights, limit the minimum weight to 2ϵ (ϵ is the machine epsilon), and clip the value of the pseudo-response $\tilde{y}$ to:

$$\text{Clip}(\tilde{y}) = \begin{cases} y_{\max}, & \tilde{y} > y_{\max} \\ -y_{\max}, & \tilde{y} < -y_{\max} \end{cases}, \tag{18}$$

where $y_{\max} \in [2, 4]$ (according to Friedman et al.[40]). $y_{\max} = 4$ was used in our experiments.

Supplementary Algorithm 4 shows the details of BoostTree for binary classification.

BoostTree for J-class ($J > 2$) classification. For J-class classification, we use

$$\ell(\boldsymbol{y}_n, \hat{\boldsymbol{y}}_n) = -\sum_{j=1}^{J} y^j \log[\text{softmax}^j(\hat{\boldsymbol{y}}_n)], \tag{19}$$

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

$$\text{softmax}^j(\hat{\boldsymbol{y}}_n) = \frac{e^{\hat{y}_n^j}}{\sum_{i=1}^J e^{\hat{y}_n^i}} \tag{20}$$

is the estimated probability of Class j for an input $\boldsymbol{x}_n$.

$\mathbf{f}_m(\boldsymbol{x})$ becomes a set of linear models $\{f_m^1(\boldsymbol{x}), f_m^2(\boldsymbol{x}), \cdots, f_m^J(\boldsymbol{x})\}$, where $f_m^j(\boldsymbol{x})$ is used to calculate the output for Class j.

The loss function for the m-th node then becomes:

$$\mathcal{L}(\mathbf{f}_m) = \sum_{n\in P_m} \ell[y_n, F_{P_m}(\boldsymbol{x}_n) + \mathbf{f}_m(\boldsymbol{x}_n)] + \sum_{j=1}^{J} \lambda_m \|\boldsymbol{w}_m^j\|_2^2, \tag{21}$$

where $\mathbf{f}_m = \left\{f_m^1, f_m^2, \cdots, f_m^J\right\}$ is a set of linear models, and $\boldsymbol{w}_m^j$ is the coefficient vector of f_m^j.

Supplementary Algorithm 5 shows the details of BoostTree for J-class ($J > 2$) classification.

BoostForest. Two techniques are used in Random Forest to improve the diversity of each tree, and hence to reduce overfitting: 1) Bagging, i.e., each tree is trained with a bootstrap replica drawn from the original training set; and, 2) feature sub-sampling, i.e., for each node of the tree, a subset of k features is randomly selected from the complete feature set, then an optimal feature is selected from the subset to split the node. k is usually set to $\text{ceil}(\sqrt{D})$ or $\text{ceil}(\log_2 D + 1)$. In this way, the computational cost of training a base learner is greatly reduced.

BoostForest integrates multiple BoostTrees into a forest. It does not require cross-validation to select the parameters for each BoostTree. We simply put all possible parameter values into a *parameter pool*, from which each BoostTree randomly selects its parameters, e.g., the minimum number of samples at a leaf $N_{\min}$, and the regularization coefficient λ. This increases the diversity of BoostTrees.

Supplementary Algorithm 6 gives the detailed BoostForest training algorithm.

Implementation details. A trick to reduce the computational cost is to reduce the number of times to calculate (12). For each numerical feature selected to be split, we first find its minimum and maximum, and extract 100 evenly spaced values between them. Then, we find the optimal split in these 100 feature values.

The loss function of BoostTree-ELM is the same as the original BoostTree's loss function, because the objective functions of ELM and RR are the same. The loss function of BoostTree-SVR needs to be modified according to the loss function of SVR. We set λ_m in (3) to $\frac{1}{2C_m}$, where C_m is the regularization coefficient of the m-th SVR model. Then, the loss function in (3) can be rewritten as:

$$\mathcal{L}(F) = \sum_{n=1}^{N} \ell\left(y_n, \hat{y}_n\right) + \sum_{m=1}^{M} \frac{1}{2C_m} \Omega\left(f_m\right), \tag{22}$$

and (7) as:

$$\begin{aligned}
\delta_{\mathcal{L}} = & \sum_{n \in I_m} \ell[y_n, F_{I_m}(\boldsymbol{x}_n)] - \sum_{n \in I_L} \ell[y_n, F_{I_m}(\boldsymbol{x}_n) + f_L(\boldsymbol{x}_n)] \\
& - \sum_{n \in I_R} l[y_n, F_{I_m}(\boldsymbol{x}_n) + f_R(\boldsymbol{x}_n)] - \frac{1}{2C_L}\Omega(f_L) \\
& - \frac{1}{2C_R}\Omega(f_R) \\
\approx & \frac{1}{2}\left[\frac{\left(\sum_{n \in I_L} g_n\right)^2}{\sum_{n \in I_L} h_n + \lambda_L} + \frac{\left(\sum_{n \in I_R} g_n\right)^2}{\sum_{n \in I_R} h_i + \lambda_R}\right. \\
& \left. - \frac{\left(\sum_{n \in I} g_n\right)^2}{\sum_{n \in I} h_n + 0.0001}\right] \\
& - \frac{1}{2C_L} * \|\boldsymbol{w}_L\|_2^2 - \frac{1}{2C_R} * \|\boldsymbol{w}_R\|_2^2,
\end{aligned} \tag{23}$$

where C_L and C_R are the regularization coefficient of SVR trained in the left and right child node, respectively, and $\boldsymbol{w}_L$ and $\boldsymbol{w}_R$ are the corresponding SVR coefficients.

Data availability

33 publicly available datasets from the UCI Machine Learning Repository (`https://archive.ics.uci.edu/ml/index.php`) and MNIST (`http://yann.lecun.com/exdb/mnist/`) were used in this study.

Code availability

All source code is openly available on GitHub (`https://github.com/zhaochangming/BoostForest`).

Acknowledgements

This research was supported by the Technology Innovation Project of Hubei Province of China under Grant 2019AEA171, the Hubei Province Funds for Distinguished Young Scholars under Grant 2020CFA050, the National Natural Science Foundation of China under Grants 61873321 and U1913207, the International Science and Technology Cooperation Program of China under Grant 2017YFE0128300, CCF-BAIDU Open Fund under Grant OF2020006, and the Key Laboratory of Brain Machine Collaborative Intelligence of Zhejiang Province under Grant 2020E10010-01.

Author contributions

C.Z. conceived the concept, performed the analyses, interpreted the results and wrote the manuscript. D.W. conceived the concept, improved the analyses and the manuscript, and supervised the work. J.H., Y.Y., H.Z., R.P., Z.S. and C.G. improved the analyses and proofread the manuscript.

Competing interests

The authors declare no competing interests.

References

1. Zhou, Z. H. *Ensemble Methods: Foundations and Algorithms* (CRC Press, Boca Raton, FL, 2012).
2. Hastie, T., Tibshirani, R. & Friedman, J. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Springer, New York, NY, 2009).
3. Singh, J., Hanson, J., Paliwal, K. & Zhou, Y. RNA secondary structure prediction using an ensemble of two-dimensional deep neural networks and transfer learning. *Nature Communications* **10**, 1–13 (2019).
4. Wang, S. *et al.* Massive computational acceleration by using neural networks to emulate mechanism-based biological models. *Nature Communications* **10**, 1–9 (2019).
5. Radivojević, T., Costello, Z., Workman, K. & Martin, H. G. A machine learning automated recommendation tool for synthetic biology. *Nature Communications* **11**, 1–14 (2020).
6. Sun, X., Liu, Y. & An, L. Ensemble dimensionality reduction and feature gene extraction for single-cell RNA-seq data. *Nature Communications* **11**, 1–9 (2020).
7. Cao, Y., Geddes, T. A., Yang, J. Y. H. & Yang, P. Ensemble deep learning in bioinformatics. *Nature Machine Intelligence* **2**, 500–508 (2020).
8. Strobach, E. & Bel, G. Learning algorithms allow for improved reliability and accuracy of global mean surface temperature projections. *Nature Communications* **11**, 1–7 (2020).
9. Kim, S. S. *et al.* Improving the informativeness of Mendelian disease pathogenicity scores for common disease. *Nature Communications* **11**, 1–15 (2020).
10. Agius, R. *et al.* Machine learning can identify newly diagnosed patients with CLL at high risk of infection. *Nature Communications* **11**, 1–17 (2020).

11. Lansford, J. L. & Vlachos, D. G. Infrared spectroscopy data- and physics-driven machine learning for characterizing surface microstructure of complex materials. *Nature Communications* **11**, 1–12 (2020).

12. Goodall, R. E. & Lee, A. A. Predicting materials properties without crystal structure: Deep representation learning from stoichiometry. *Nature Communications* **11**, 1–9 (2020).

13. Yang, C. *et al.* Lunar impact crater identification and age estimation with Chang'E data by deep and transfer learning. *Nature Communications* **11**, 1–15 (2020).

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

15. Chen, T. Q. & Guestrin, C. XGBoost: A scalable tree boosting system. In *Proc. 22nd ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, 785–794 (San Francisico, CA, 2016).

16. Breiman, L., Friedman, J., Olshen, R. & Stone, C. *Classification and Regression Trees* (CRC Press, Boca Raton, FL, 1984).

17. Ke, G. *et al.* LightGBM: A highly efficient gradient boosting decision tree. In *Proc. Advances in Neural Information Processing Systems*, 3146–3154 (Long Beach, CA, 2017).

18. Breiman, L. Bagging predictors. *Machine Learning* **24**, 123–140 (1996).

19. Kuncheva, L. I. & Whitaker, C. J. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning* **51**, 181–207 (2003).

20. Tang, E. K., Suganthan, P. N. & Yao, X. An analysis of diversity measures. *Machine Learning* **65**, 247–271 (2006).

21. Brown, G., Wyatt, J., Harris, R. & Yao, X. Diversity creation methods: A survey and categorisation. *Information Fusion* **6**, 5–20 (2005).

22. Quinlan, J. R. Learning with continuous classes. In *Proc. 5th Australian Joint Conf. on Artificial Intelligence*, 343–348 (Tasmania, Australia, 1992).

23. Wang, Y. & Witten, I. H. Induction of model trees for predicting continuous classes. In *Proc. 9th European Conf. on Machine Learning* (Prague, Czech Republic, 1997).

24. Huang, G.-B., Zhu, Q.-Y. & Siew, C.-K. Extreme learning machine: Theory and applications. *Neurocomputing* **70**, 489–501 (2006).

25. Drucker, H., Burges, C. J., Kaufman, L., Smola, A. J. & Vapnik, V. Support vector regression machines. In *Proc. Advances in Neural Information Processing Systems*, 155–161 (Cambridge, MA, 1997).

26. Zhou, Z. H. & Feng, J. Deep forest. *National Science Review* **6**, 74–86 (2018).

27. Freund, Y. & Schapire, R. E. Experiments with a new boosting algorithm. In *Proc. 13th Int'l Conf. on Machine Learning*, 148–156 (Bari, Italy, 1996).

28. Breiman, L. Random forests. *Machine Learning* **45**, 5–32 (2001).

29. Ho, T. K. Random decision forests. In *Proc. 3rd Int'l Conf. on Document Analysis and Recognition*, 278–282 (Montreal, Canada, 1995).

30. Barandiaran, I. The random subspace method for constructing decision forests. *IEEE Trans. on Pattern Analysis and Machine Intelligence* **20**, 1–22 (1998).

31. Amit, Y. & Geman, D. Shape quantization and recognition with randomized trees. *Neural Computation* **9**, 1545–1588 (1997).

32. Geurts, P., Ernst, D. & Wehenkel, L. Extremely randomized trees. *Machine Learning* **63**, 3–42 (2006).

33. Hastie, T., Rosset, S., Zhu, J. & Zou, H. Multi-class AdaBoost. *Statistics and Its Interface* **2**, 349–360 (2009).

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

35. Mason, L., Baxter, J., Bartlett, P. L. & Frean, M. R. Boosting algorithms as gradient descent. In *Proc. Advances in Neural Information Processing Systems*, 512–518 (Breckenridge, Colorado, 2000).

36. Chen, T. Q. & He, T. Higgs boson discovery with boosted trees. In *Proc. Advances in Neural Information Processing Systems*, 69–80 (Montreal, Canada, 2015).

37. Rakhlin, A., Shvets, A., Iglovikov, V. & Kalinin, A. A. Deep convolutional neural networks for breast cancer histology image analysis. In *Proc. 15th Int'l Conf. on Image Analysis and Recognition*, 737–744 (Povoa de Varzim, Portugal, 2018).

38. Liu, L., Ji, M. & Buchroithner, M. Combining partial least squares and the gradient-boosting method for soil property retrieval using visible near-infrared shortwave infrared spectra. *Remote Sensing* **9**, 1299–1317 (2017).

39. Walsh, J., Heazlewood, I. T. & Climstein, M. Regularized linear and gradient boosted ensemble methods to predict athletes' gender based on a survey of masters athletes. *Model Assisted Statistics and Applications* **14**, 47–64 (2019).

40. Friedman, J., Hastie, T. & Tibshirani, R. Additive logistic regression: A statistical view of boosting. *Annals of Statistics* **28**, 337–407 (2000).

41. Landwehr, N., Hall, M. & Frank, E. Logistic model trees. *Machine Learning* **59**, 161–205 (2005).

42. Benjamini, Y. & Hochberg, Y. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *Journal of the Royal Statistical Society* **57**, 289–300 (1995).

43. Wold, S., Esbensen, K. & Geladi, P. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems* **2**, 37–52 (1987).

44. Webb, G. I. Multiboosting: A technique for combining boosting and wagging. *Machine Learning* **40**, 159–196 (2000).

List of Figures

Table 1: Performances of the five ensemble learning approaches on the 30 datasets. The best performance is marked in bold. ● indicates statistically significant win for BoostForest.

Dataset	RandomForest	Extra-Trees	XGBoost	LightGBM	BoostForest
	Mean and standard deviation (in parenthesis) of the classification accuracy				
SON	0.769 (0.038)●	0.763 (0.034)●	0.810 (0.035)	0.795 (0.037)●	**0.822** (0.026)
SEE	0.896 (0.027)●	0.904 (0.028)●	0.910 (0.023)●	0.908 (0.025)●	**0.940** (0.022)
QB	0.990 (0.011)	0.996 (0.004)	0.990 (0.010)	0.988 (0.011)	**0.998** (0.005)
VC2	0.831 (0.021)●	0.754 (0.079)●	0.829 (0.029)●	0.837 (0.026)	**0.853** (0.026)
VC3	0.830 (0.026)●	0.754 (0.044)●	0.825 (0.030)●	0.837 (0.024)	**0.850** (0.027)
MV1	0.803 (0.043)●	0.820 (0.037)●	0.821 (0.038)●	0.810 (0.031)●	**0.843** (0.025)
BCD	0.944 (0.011)●	0.949 (0.010)●	0.960 (0.008)●	0.954 (0.013)●	**0.972** (0.009)
ILP	0.697 (0.013)●	0.706 (0.019)	0.691 (0.010)●	0.708 (0.014)	**0.709** (0.019)
BD	0.783 (0.009)	0.766 (0.009)●	0.775 (0.017)	0.769 (0.008)●	**0.785** (0.007)
PID	0.759 (0.014)	0.744 (0.013)●	0.746 (0.018)●	0.747 (0.014)●	**0.762** (0.011)
VS	0.728 (0.021)●	0.724 (0.017)●	0.763 (0.019)●	0.748 (0.023)●	**0.830** (0.016)
QSAR	0.852 (0.012)●	0.852 (0.016)●	0.862 (0.009)●	0.858 (0.007)●	**0.882** (0.006)
DRD	0.659 (0.011)●	0.668 (0.015)●	0.676 (0.015)●	0.664 (0.009)●	**0.740** (0.008)
BA	0.985 (0.008)●	0.995 (0.003)●	0.991 (0.008)●	0.991 (0.006)●	**0.998** (0.003)
WDG	0.847 (0.005)●	0.855 (0.005)●	0.851 (0.006)●	0.850 (0.007)●	**0.861** (0.005)
Average accuracy	0.825	0.817	0.833	0.831	**0.856**
Average rank	4.067	3.800	2.867	3.267	**1.000**
	Mean and standard deviation (in parenthesis) of the regression RMSE				
CS	0.621 (0.103)●	0.608 (0.088)●	0.479 (0.069)●	0.524 (0.084)●	**0.307** (0.024)
CF	0.826 (0.104)●	0.810 (0.128)	0.791 (0.069)	0.830 (0.110)●	**0.758** (0.064)
AMPG	0.375 (0.019)●	0.376 (0.018)●	0.381 (0.020)●	0.386 (0.022)●	**0.359** (0.017)
REV	**0.564** (0.072)	0.574 (0.076)	0.575 (0.067)	0.575 (0.071)	0.575 (0.070)
NO	0.667 (0.037)●	0.692 (0.040)●	0.653 (0.034)●	0.660 (0.033)●	**0.640** (0.036)
PM	0.830 (0.093)●	0.851 (0.085)●	**0.785** (0.061)	0.798 (0.078)	0.812 (0.080)
BH	0.443 (0.063)●	0.419 (0.071)	**0.396** (0.054)	0.436 (0.057)●	0.400 (0.055)
CPS	0.882 (0.135)	0.901 (0.137)●	0.897 (0.125)●	0.886 (0.137)	**0.877** (0.130)
CCS	0.382 (0.019)●	0.388 (0.014)●	0.327 (0.022)●	0.356 (0.025)●	**0.304** (0.016)
ASN	0.400 (0.023)●	0.418 (0.026)●	**0.314** (0.015)●	0.390 (0.015)●	0.340 (0.015)
ADS	0.671 (0.024)●	0.675 (0.026)●	0.674 (0.025)●	0.676 (0.023)●	**0.664** (0.022)
WQW	0.762 (0.028)●	0.760 (0.026)●	0.779 (0.020)●	0.776 (0.025)●	**0.744** (0.022)
AQ	0.005 (0.002)●	0.008 (0.002)●	0.014 (0.002)●	0.014 (0.002)●	**0.002** (0.001)
CCPP	0.214 (0.004)●	0.220 (0.004)●	0.207 (0.004)●	0.221 (0.004)●	**0.197** (0.004)
EGSS	0.347 (0.007)●	0.335 (0.005)●	**0.250** (0.003)	0.264 (0.003)	0.261 (0.005)
Average RMSE	0.533	0.536	0.501	0.519	**0.483**
Average rank	3.333	3.800	2.600	3.667	**1.600**

Table 2: Performances on the 30 datasets, when CART is used to replace BoostTree in BoostForest. The best performance is marked in bold. ● indicates statistically significant win for CARForest.

Dataset	CART	CARForest
	Mean and standard deviation (in parenthesis) of the classification accuracy	
SON	0.707 (0.042)●	**0.767** (0.033)
SEE	0.896 (0.025)	**0.890** (0.030)
QB	0.984 (0.006)●	**0.990** (0.011)
VC2	0.798 (0.029)●	**0.841** (0.023)
VC3	0.805 (0.035)●	**0.839** (0.031)
MV1	0.737 (0.030)●	**0.797** (0.034)
BCD	0.932 (0.017)●	**0.944** (0.011)
ILP	0.671 (0.023)●	**0.701** (0.013)
BD	0.768 (0.011)●	**0.784** (0.012)
PID	0.730 (0.012)●	**0.754** (0.017)
VS	0.657 (0.035)●	**0.727** (0.016)
QSAR	0.798 (0.014)●	**0.846** (0.015)
DRD	0.622 (0.024)●	**0.665** (0.016)
BA	0.971 (0.010)●	**0.982** (0.010)
WDG	0.756 (0.007)●	**0.848** (0.005)
Average accuracy	0.789	**0.825**
Average rank	1.933	**1.067**
	Mean and standard deviation (in parenthesis) of the regression RMSE	
CS	**0.672** (0.160)	0.745 (0.130)
CF	0.870 (0.088)●	**0.826** (0.092)
AMPG	0.447 (0.010)●	**0.402** (0.024)
REV	0.627 (0.070)●	**0.565** (0.076)
NO	0.758 (0.037)●	**0.692**(0.041)
PM	0.919 (0.089)●	**0.840** (0.086)
BH	0.534 (0.074)●	**0.468** (0.077)
CPS	0.934 (0.138)●	**0.870** (0.146)
CCS	0.491 (0.027)●	**0.429** (0.024)
ASN	0.519 (0.023)●	**0.451** (0.026)
ADS	0.724 (0.025)●	**0.670** (0.025)
WQW	0.856 (0.023)●	**0.761** (0.024)
AQ	**0.005** (0.002)	0.014 (0.002)
CCPP	0.244 (0.004)●	**0.214** (0.004)
EGSS	0.544 (0.006)●	**0.386** (0.008)
Average RMSE	0.610	**0.556**
Average rank	1.867	**1.133**

Table 3: Mean and standard deviation (in parenthesis) of the regression RMSE, when ELM or SVR is used to replace RR in BoostTree. The best performance is marked in bold. ● indicates statistically significant win for BoostForest-ELM or BoostForest-SVR.

Dataset	ELM	BoostTree-ELM	BoostForest-ELM
CS	**0.294** (0.071)	0.386 (0.092)●	0.300 (0.025)
CF	0.799 (0.093)●	0.879 (0.136)●	**0.740** (0.069)
AMPG	0.369 (0.020)●	0.408 (0.038)●	**0.355** (0.010)
REV	0.608 (0.072)●	0.627 (0.092)●	**0.569** (0.070)
NO	0.699 (0.033)●	0.752 (0.057)●	**0.644** (0.033)
PM	0.915 (0.086)●	0.966 (0.074)●	**0.840** (0.080)
BH	0.517 (0.076)●	0.517 (0.106)●	**0.380** (0.061)
CPS	0.922 (0.150)●	1.029 (0.122)●	**0.881** (0.130)
CCS	0.550 (0.048)●	0.443 (0.068)●	**0.361** (0.012)
ASN	0.621 (0.040)●	0.580 (0.049)●	**0.495** (0.023)
ADS	0.688 (0.047)●	0.674 (0.025)●	**0.659** (0.025)
WQW	0.837 (0.021)●	0.831 (0.021)●	**0.775** (0.021)
AQ	**0.005** (0.001)	0.007 (0.005)	**0.005** (0.000)
CCPP	0.247 (0.004)●	0.238 (0.006)●	**0.215** (0.004)
EGSS	0.537 (0.014)●	0.402 (0.038)●	**0.269** (0.005)
Average RMSE	0.574	0.583	**0.499**
Average rank	2.267	2.600	**1.133**
Dataset	SVR	BoostTree-SVR	BoostForest-SVR
CS	**0.380** (0.033)	0.462 (0.071)	0.430 (0.065)
CF	0.833 (0.119)●	0.861 (0.101)●	**0.761** (0.068)
AMPG	0.445 (0.027)●	0.414 (0.039)●	**0.361**(0.014)
REV	0.672 (0.084)●	0.618 (0.059)●	**0.561** (0.073)
NO	0.723 (0.038)●	0.734 (0.056)●	**0.641** (0.033)
PM	0.936 (0.089)●	0.924 (0.073)●	**0.823** (0.083)
BH	0.559 (0.065)●	0.489 (0.082)●	**0.409** (0.061)
CPS	0.876 (0.133)●	0.947 (0.132)●	**0.862** (0.139)
CCS	0.641 (0.020)●	0.445 (0.037)●	**0.377** (0.017)
ASN	0.703 (0.036)●	0.565 (0.041)●	**0.461** (0.020)
ADS	0.698 (0.027)●	0.694 (0.030)●	**0.663** (0.027)
WQW	0.855 (0.022)●	0.837 (0.025)●	**0.766** (0.020)
AQ	0.054 (0.002)●	0.037 (0.033)●	**0.008** (0.002)
CCPP	0.267 (0.003)●	0.260 (0.013)●	**0.221** (0.003)
EGSS	0.596 (0.007)●	0.424 (0.025)●	**0.289** (0.007)
Average RMSE	0.616	0.581	**0.509**
Average rank	2.667	2.267	**1.067**

Table 4: Performances of the five ensemble learning approaches on large datasets. The best performance is marked in bold. ● indicates statistically significant win for BoostForest.

Dataset	RandomForest	Extra-Trees	XGBoost	LightGBM	BoostForest
	Mean and standard deviation (in parenthesis) of the classification accuracy				
LR	0.923 (0.004)●	0.914 (0.005)●	0.928 (0.002)●	0.935 (0.003)●	**0.964** (0.001)
MNIST	0.957 (0.001)	0.955 (0.001)	0.964 (0.001)	**0.970** (0.001)	0.957 (0.001)
MNIST-PCA	0.897 (0.001)●	0.893 (0.002)●	0.902 (0.002)●	0.919 (0.002)●	**0.923** (0.002)
Average accuracy	0.926	0.921	0.931	0.941	**0.948**
Average rank	4.000	5.000	2.667	**1.667**	**1.667**
	Mean and standard deviation (in parenthesis) of the regression RMSE				
PTS	0.623 (0.003)●	0.654 (0.004)●	0.670 (0.004)●	0.636 (0.002)●	**0.590** (0.004)
RLCT	0.094 (0.005)	**0.082** (0.002)	0.130 (0.003)	0.091 (0.003)	0.109 (0.005)
RLCT-PCA	0.177 (0.009)●	0.171 (0.003)●	0.214 (0.005)●	0.168 (0.005)●	**0.140** (0.006)
Average RMSE	0.298	0.302	0.338	0.298	**0.280**
Average rank	3.000	2.667	5.000	2.333	**2.000**