BoostTree and BoostForest for Ensemble Learning: Supplementary Materials

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

1 Supplementary Algorithms

This section presents pseudo-code of all algorithms introduced in this article.

Supplementary Algorithm 1: LogitBoost for classification[1].

Input: K, the maximum number of iterations;
$\{\boldsymbol{x}_n, y_n\}_{n=1}^N$, N labeled training samples.
Output: The ensemble $F(\boldsymbol{x})$.

if $J == 2$ **then**
  Initialize $F(\boldsymbol{x}) = 0$, $w_n = \frac{1}{N}$, and $p(\boldsymbol{x}_n) = \frac{1}{2}$, $n = 1, 2, \ldots, N$
  for $k = 1 : K$ **do**
    $z_n = \dfrac{y_n - p\left(\boldsymbol{x}_n\right)}{p\left(\boldsymbol{x}_n\right)\left(1 - p\left(\boldsymbol{x}_n\right)\right)}$, $n = 1, ..., N$;
    $w_n = p\left(\boldsymbol{x}_n\right)\left(1 - p\left(\boldsymbol{x}_n\right)\right)$, $n = 1, ..., N$;
    Fit a function $f_k(\boldsymbol{x})$ by weighted least-squares regression, using $\{\boldsymbol{x}_n, z_n\}_{n=1}^N$ and weights $\{w_n\}_{n=1}^N$;
    $F(\boldsymbol{x}) \leftarrow F(\boldsymbol{x}) + f_k(\boldsymbol{x})$;
    $p(\boldsymbol{x}_n) = \dfrac{1}{1 + e^{-F(\boldsymbol{x}_n)}}$, $n = 1, ..., N$;
  end
else
  Initialize $F_j(\boldsymbol{x}) = 0$, $w_{nj} = \frac{1}{N}$, and $p_j(\boldsymbol{x}_n) = \frac{1}{J}$, $n = 1, 2, \ldots, N$, $j = 1, 2, \ldots, J$;
  for $k = 1 : K$ **do**
    for $j = 1 : J$ **do**
      $z_{nj} = \dfrac{y_{nj} - p_j\left(\boldsymbol{x}_n\right)}{p_j\left(\boldsymbol{x}_n\right)\left(1 - p_j\left(\boldsymbol{x}_n\right)\right)}$, $n = 1, ..., N$;
      $w_{nj} = p_j\left(\boldsymbol{x}_n\right)\left(1 - p_j\left(\boldsymbol{x}_n\right)\right)$, $n = 1, ..., N$;
      Fit a function $f_{kj}(\boldsymbol{x})$ by weighted least-squares regression, using $\{\boldsymbol{x}_n, z_{nj}\}_{n=1}^N$ and weights $\{w_{nj}\}_{n=1}^N$;
    end
    $f_{kj}(\boldsymbol{x}) \leftarrow \dfrac{J-1}{J}\left[f_{kj}(\boldsymbol{x}) - \dfrac{1}{J}\sum_{i=1}^{J} f_{ki}(\boldsymbol{x})\right]$;
    $F_j(\boldsymbol{x}) \leftarrow F_j(\boldsymbol{x}) + f_{kj}(\boldsymbol{x})$;
    $p_j(\boldsymbol{x}_n) = \dfrac{e^{F_j(\boldsymbol{x}_n)}}{\sum_{i=1}^{J} e^{F_i(\boldsymbol{x}_n)}}$, $n = 1, ..., N$;
  end
end

Supplementary Algorithm 2: BoostTree using ridge regression (RR) as the node function.

Input: $\mathrm{Data} = \{\mathbf{x}_n, y_n\}_{n=1}^N$, N training samples, where $\mathbf{x}_n \in \mathbb{R}^D$;
$\mathrm{Pool}_{\mathrm{MinSamplesLeaf}}$, candidate value pool of the minimum number of samples at a leaf;
Pool_λ, candidate value pool of the ℓ_2 regularization parameter λ;
(optional) $\mathrm{MaxNumLeaf}$, the maximum number of leaves.
Output: A BoostTree.

$\mathrm{NumLeaf} = 1$;
$f(\boldsymbol{x}) \equiv 0$;
root $\leftarrow \{\mathrm{data} = \mathrm{Data}, \mathrm{model} = f, \mathrm{leftChild} = \mathrm{None}, \mathrm{rightChild} = \mathrm{None}\}$;
$\mathrm{BoostTree} \leftarrow \mathrm{split}(\mathrm{root})$;

$\mathrm{split}(\mathrm{node})\{$
Let t be the index of the current node;
$\{\boldsymbol{x}_n, y_n\}_{n \in I} \leftarrow \mathrm{node.data}$;
$F_I(\boldsymbol{x}) \leftarrow \sum_{m \in \mathrm{Path}(t)} \mathrm{node}_m.\mathrm{model}(\boldsymbol{x})$;
Randomly select $N_{\min}^L$ and $N_{\min}^R$ from $\mathrm{Pool}_{\mathrm{MinSamplesLeaf}}$;
$\delta_{\mathcal{L}}^{\max} = 0$;
Randomly select $\sqrt{D}$ features to create a feature subset $\mathcal{F}$;
for $d \in \mathcal{F}$ **do**
  $S_d = \{x_{n,d} | n \in I\}$;
  for s *in* S_d **do**
    $I_L = \{n | x_{n,d} \le s, n \in I\}$;
    $I_R = \{n | x_{n,d} > s, n \in I\}$;
    if $|I_L| \ge N_{\min}^L$ *and* $|I_R| \ge N_{\min}^R$ **then**
      Randomly select λ_L and λ_R from Pool_λ;
      Calculate $\delta_{\mathcal{L}}$ in (12);
      if $\delta_{\mathcal{L}} > \delta_{\mathcal{L}}^{\max}$ **then**
        $\delta_{\mathcal{L}}^{\max} = \delta_{\mathcal{L}}, \quad d^* = d, \quad s^* = s, \quad \lambda_L^* = \lambda_L, \quad \lambda_R^* = \lambda_R$;
      end
    end
  end
end
if $\delta_{\mathcal{L}}^{\max} > 0$ **then**
  $I_L^* = \{n | x_{n,d^*} \le s^*, n \in I\}$;
  $I_R^* = \{n | x_{n,d^*} > s^*, n \in I\}$;
  $f_L^* = \mathrm{FitModel}(\{\boldsymbol{x}_n, y_n\}_{n \in I_L^*}, F_I, \lambda_L^*)$;
  $f_R^* = \mathrm{FitModel}(\{\boldsymbol{x}_n, y_n\}_{n \in I_R^*}, F_I, \lambda_R^*)$;
  Calculate $\delta_{\mathcal{L}}$ in (9);
  if $\delta_{\mathcal{L}} > 0$ **then**
    $\mathrm{node.leftChild} = \{\mathrm{data} = \{\boldsymbol{x}_n, y_n\}_{n \in I_L^*}, \ \mathrm{model} = f_L^*, \ \mathrm{leftChild} = \mathrm{None}, \ \mathrm{rightChild} = \mathrm{None}\}$;
    $\mathrm{node.rightChild} = \{\mathrm{data} = \{\boldsymbol{x}_n, y_n\}_{n \in I_R^*}, \ \mathrm{model} = f_R^*, \ \mathrm{leftChild} = \mathrm{None}, \mathrm{rightChild} = \mathrm{None}\}$;
    $\mathrm{NumLeaf} = \mathrm{NumLeaf} + 1$;
  end
  if $\mathrm{MaxNumLeaf}$ *is not supplied, or* $\mathrm{NumLeaf} \le \mathrm{MaxNumLeaf}$ **then**
    For each leaf, the cross entropy loss (classification) or RMSE (regression) multiplied by the number of samples at that leaf is used as the impurity score;
    Identify node^*, the leaf node with the highest impurity score;
    $\mathrm{split}(\mathrm{node}^*)$;
  end
end

Supplementary Algorithm 3: $\mathrm{FitModel}$ for linear regression.

Input: $\{\boldsymbol{x}_n, y_n\}_{n\in I}$, sample set of the current node;
F_I, ensemble of the models along the path from the root node to the parent node of the current node;
λ, the ℓ_2 regularization parameter.

Output: Linear regression model f_m for the current node.

$\tilde{y}_n = y_n - F_I(\boldsymbol{x}_n)$, $n \in I$;
Fit $f_m = \mathrm{RidgeRegression}(\{\boldsymbol{x}_n, \tilde{y}_n\}_{n\in I}, \lambda)$ using ridge regression with regularization parameter λ.

Supplementary Algorithm 4: $\mathrm{FitModel}$ for binary classification.

Input: $\{\boldsymbol{x}_n, y_n\}_{n\in I}$, sample set of the current node;
F_I, ensemble of the models along the path from the root node to the parent node of the current node;
λ, the ℓ_2 regularization parameter.

Output: Linear classifier f_m for the current node.

$p(\boldsymbol{x}_n) = \mathrm{sigmod}[F_I(\boldsymbol{x}_n)]$, $n \in I$;
$\tilde{y}_n = \dfrac{y_n - p(\boldsymbol{x}_n)}{p(\boldsymbol{x}_n)(1-p(\boldsymbol{x}_n))}$, $n \in I$;
$\tilde{y}_n = \mathrm{Clip}(\tilde{y}_n)$ in (18), $n \in I$;
$w_n = p(\boldsymbol{x}_n)(1-p(\boldsymbol{x}_n))$, $n \in I$;
$q_{5\%} = \mathrm{Quantile}(\{w_n \mid n \in I\}, 5)$;
$\mathcal{D}' = \{(\boldsymbol{x}_n, \tilde{y}_n, w_n) \mid w_n > q_{5\%},\ n \in I\}$;
Fit $f_m = \mathrm{WeightedRidgeRegression}(\mathcal{D}', \lambda)$ using weighted ridge regression on $\mathcal{D}'$ with regularization parameter λ.

Supplementary Algorithm 5: FitModel for J-class ($J > 2$) classification.

Input: $\{\boldsymbol{x}_n, y_n\}_{n \in I}$, sample set of the current node;
F_I, ensemble of the models along the path from the root node to the parent node of the current node;
λ, the ℓ_2 regularization parameter.
Output: The linear classifier set $\boldsymbol{f}_m$ for the current node.

Compute $p_j(\boldsymbol{x}_n) = \mathrm{softmax}^j(\hat{\boldsymbol{y}}_n)$, $n \in I$, $j = 1, ..., J$;
for $j = 1 : J$ **do**
  $\tilde{y}_{nj} = \dfrac{y_{nj} - p_j(\boldsymbol{x}_n)}{p_j(\boldsymbol{x}_n)(1 - p_j(\boldsymbol{x}_n))}$, $n \in I$;
  $\tilde{y}_{nj} = \mathrm{Clip}(\tilde{y}_{nj})$ in (18), $n \in I$;
  $w_{nj} = p_j(\boldsymbol{x}_n)(1 - p_j(\boldsymbol{x}_n))$, $n \in I$;
  $q_{5\%} = \mathrm{Quantile}(\{w_{nj} \mid n \in I\}, 5)$;
  $\mathcal{D}' = \{(\boldsymbol{x}_n, \tilde{y}_{nj}, w_{nj}) \mid w_{nj} > q_{5\%},\ n \in I\}$;
  Fit $f_j = \mathrm{WeightedRidgeRegression}(\mathcal{D}', \lambda)$ using weighted ridge regression on $\mathcal{D}'$ with regularization parameter λ;
end
$f_j(\boldsymbol{x}) \leftarrow \dfrac{J-1}{J}\left[f_j(\boldsymbol{x}) - \dfrac{1}{J}\sum_{i=1}^{J} f_i(\boldsymbol{x})\right], \quad j = 1, ..., J;$
$\boldsymbol{f}_m = \{f_1, f_2, \ldots, f_J\}$.

Supplementary Algorithm 6: BoostForest training algorithm.

Input: $\mathrm{Data} = \{\mathbf{x}_n, y_n\}_{n=1}^{N}$, N training samples, where $\mathbf{x}_n \in \mathbb{R}^D$;
$\mathrm{NumEstimators}$, the number of BoostTrees;
$\mathrm{Pool}_{\mathrm{MinSamplesLeaf}}$, candidate value pool of the minimum number of samples at a leaf;
Pool_{λ}, candidate value pool of the regularization parameter λ.
Output: A BoostForest

$\mathrm{BoostForest} = \{\}$;
for $i = 1 : \mathrm{NumEstimators}$ **do**
  Bootstrap Data' from Data;
  Train $\mathrm{BoostTree}_i$ on Data' using Supplementary Algorithm 2;
  Add $\mathrm{BoostTree}_i$ to $\mathrm{BoostForest}$;
end

Supplementary Algorithm 7: BoostTree-ELM for regression.

Input: $\text{Data} = \{\mathbf{x}_n, y_n\}_{n=1}^N$, N training samples, where $\mathbf{x}_n \in \mathbb{R}^D$;
$\text{Pool}_{\text{MinSamplesLeaf}}$, candidate value pool of the minimum number of samples at a leaf;
$\text{Pool}_{\text{NumHiddenNodes}}$, candidate value pool of the number of hidden nodes;
Pool_λ, candidate value pool of the regularization parameter λ;
(optional) MaxNumLeaf, the maximum number of leaves.

Output: A BoostTree-ELM.

$\text{NumLeaf} = 1$;
$f(\boldsymbol{x}) \equiv 0$;
$\text{root} \leftarrow \{\text{data} = \text{Data},\ \text{model} = f,\ \text{leftChild} = \text{None},\ \text{rightChild} = \text{None}\}$;
$\text{BoostTree-ELM} \leftarrow \text{split}(\text{root})$;

$\text{split}(\text{node})\{$
Let t be the index of the current node;
$\{\boldsymbol{x}_n, y_n\}_{n \in I} \leftarrow \text{node.data}$;
$F_I(\boldsymbol{x}) \leftarrow \sum_{m \in \text{Path}(t)} \text{node}_m.\text{model}(\boldsymbol{x})$;
Randomly select $N_{\min}^L$ and $N_{\min}^R$ from $\text{Pool}_{\text{MinSamplesLeaf}}$;
$\delta_{\mathcal{L}}^{\max} = 0$;
Randomly select $\sqrt{D}$ features to create a feature subset $\mathcal{F}$;
for $d \in \mathcal{F}$ **do**
  $S_d = \{x_{n,d} | n \in I\}$;
  for $s\ in\ S_d$ **do**
    $I_L = \{n | x_{n,d} \le s, n \in I\}$;
    $I_R = \{n | x_{n,d} > s, n \in I\}$;
    if $|I_L| \ge N_{\min}^L$ *and* $|I_R| \ge N_{\min}^R$ **then**
      Randomly select λ_L and λ_R from $Pool_\lambda$;
      Calculate $\delta_{\mathcal{L}}$ in (12);
      if $\delta_{\mathcal{L}} > \delta_{\mathcal{L}}^{\max}$ **then**
        $\delta_{\mathcal{L}}^{\max} = \delta_{\mathcal{L}}, \quad d^* = d, \quad s^* = s, \quad \lambda_L^* = \lambda_L, \quad \lambda_R^* = \lambda_R$;
      end
    end
  end
end
if $\delta_{\mathcal{L}}^{\max} > 0$ **then**
  Randomly select M_L and M_R from $\text{Pool}_{\text{NumHiddenNodes}}$;
  $I_L^* = \{n | x_{n,d^*} \le s^*, n \in I\}$;
  $I_R^* = \{n | x_{n,d^*} > s^*, n \in I\}$;
  $f_L^* = \text{FitModelELM}(\{\boldsymbol{x}_n, y_n\}_{n \in I_L^*},\ F_I,\ \lambda_L^*,\ M_L)$;
  $f_R^* = \text{FitModelELM}(\{\boldsymbol{x}_n, y_n\}_{n \in I_R^*},\ F_I,\ \lambda_R^*,\ M_R)$;
  Calculate $\delta_{\mathcal{L}}$ in (9);
  if $\delta_{\mathcal{L}} > 0$ **then**
    $\text{node.leftChild} = \{\text{data} = \{\boldsymbol{x}_n, y_n\}_{n \in I_L^*},\ \text{model} = f_L^*,\ \text{leftChild} = \text{None},\ \text{rightChild} = \text{None}\}$;
    $\text{node.rightChild} = \{\text{data} = \{\boldsymbol{x}_n, y_n\}_{n \in I_R^*},\ \text{model} = f_R^*,\ \text{leftChild} = \text{None}, \text{rightChild} = \text{None}\}$;
    $\text{NumLeaf} = \text{NumLeaf} + 1$;
  end
  if MaxNumLeaf *is not supplied, or* $\text{NumLeaf} \le \text{MaxNumLeaf}$ **then**
    For each leaf, the cross entropy loss (classification) or RMSE (regression) multiplied by the number of samples at that leaf is used as the impurity score;
    Identify node^*, the leaf node with the highest impurity score;
    $\text{split}(\text{node}^*)$;
  end
end

Supplementary Algorithm 8: $\mathrm{FitModelELM}$ for regression.

Input: $\{\boldsymbol{x}_n, y_n\}_{n\in I}$, sample set of the current node;
F_I, ensemble of the models along the path from the root node to the parent node of the current node;
λ, the regularization parameter;
M, the number of hidden nodes.

Output: ELM model f_m for the current node.

$\tilde{y}_n = y_n - F_I(\boldsymbol{x}_n), n \in I$;
Fit $f_m = \mathrm{ELM}(\{\boldsymbol{x}_n, \tilde{y}_n\}_{n\in I}, \lambda, M)$ using ELM with regularization parameter λ and M hidden nodes.

Supplementary Algorithm 9: BoostForest-ELM for regression.

Input: $\mathrm{Data} = \{\mathbf{x}_n, y_n\}_{n=1}^N$, N training samples, where $\mathbf{x}_n \in \mathbb{R}^D$;
$\mathrm{NumEstimators}$, the number of trees;
$\mathrm{Pool}_{\mathrm{MinSamplesLeaf}}$, candidate value pool of the minimum number of samples at a leaf;
$\mathrm{Pool}_{\mathrm{NumHiddenNodes}}$, candidate value pool of the number of hidden nodes;
Pool_{λ}, candidate value pool of the regularization parameter λ.

Output: A BoostForest-ELM.

$\mathrm{BoostForest\text{-}ELM} = \{\}$;
for $i = 1 : \mathrm{NumEstimators}$ **do**
  Bootstrap Data' from Data;
  Train $\mathrm{BoostTree\text{-}ELM}_i$ on Data' using Supplementary Algorithm 7;
  Add $\mathrm{BoostTree\text{-}ELM}_i$ to $\mathrm{BoostForest\text{-}ELM}$;
end

Supplementary Algorithm 10: BoostTree-SVR for regression.

Input: $\mathrm{Data} = \{\mathbf{x}_n, y_n\}_{n=1}^{N}$, N training samples, where $\mathbf{x}_n \in \mathbb{R}^D$;
$\mathrm{Pool}_{\mathrm{MinSamplesLeaf}}$, candidate value pool of the minimum number of samples at a leaf;
Pool_C, candidate value pool of the regularization parameter;
Pool_ϵ, candidate value pool of the slack variable;
(optional) $\mathrm{MaxNumLeaf}$, the maximum number of leaves.

Output: A BoostTree-SVR

$\mathrm{NumLeaf} = 1$;
$f(\boldsymbol{x}) \equiv 0$;
$\mathrm{root} \leftarrow \{\mathrm{data} = \mathrm{Data},\ \mathrm{model} = f,\ \mathrm{leftChild} = \mathrm{None},\ \mathrm{rightChild} = \mathrm{None}\}$;
$\mathrm{BoostTree\text{-}SVR} \leftarrow \mathrm{split}(\mathrm{root})$;

$\mathrm{split}(\mathrm{node})\{$
Let t be the index of the current node;
$\{\boldsymbol{x}_n, y_n\}_{n \in I} \leftarrow \mathrm{node.data}$;
$F_I(\boldsymbol{x}) \leftarrow \sum_{m \in \mathrm{Path}(t)} \mathrm{node}_m.\mathrm{model}(\boldsymbol{x})$;
Randomly select $N_{\min}^L$ and $N_{\min}^R$ from $\mathrm{Pool}_{\mathrm{MinSamplesLeaf}}$;
$\delta_{\mathcal{L}}^{\max} = 0$;
Randomly select $\sqrt{D}$ features to create a feature subset $\mathcal{F}$;
for $d \in \mathcal{F}$ **do**
  $S_d = \{x_{n,d} | n \in I\}$;
  for $s\ in\ S_d$ **do**
    $I_L = \{n | x_{n,d} \le s, n \in I\}$;
    $I_R = \{n | x_{n,d} > s, n \in I\}$;
    if $|I_L| \ge N_{\min}^L$ *and* $|I_R| \ge N_{\min}^R$ **then**
      Randomly select C_L and C_R from $Pool_C$;
      Calculate $\delta_{\mathcal{L}}$ in (12);
      if $\delta_{\mathcal{L}} > \delta_{\mathcal{L}}^{\max}$ **then**
        $\delta_{\mathcal{L}}^{\max} = \delta_{\mathcal{L}}, \quad d^* = d, \quad s^* = s, \quad C_L^* = C_L, \quad C_R^* = C_R$;
      end
    end
  end
end
if $\delta_{\mathcal{L}}^{\max} > 0$ **then**
  Randomly select ϵ_L and ϵ_R from $Pool_\epsilon$;
  $I_L^* = \{n | x_{n,d^*} \le s^*, n \in I\}$;
  $I_R^* = \{n | x_{n,d^*} > s^*, n \in I\}$;
  $f_L^* = \mathrm{FitModelSVR}(\{\boldsymbol{x}_n, y_n\}_{n \in I_L^*}, F_I, C_L^*, \epsilon_L)$;
  $f_R^* = \mathrm{FitModelSVR}(\{\boldsymbol{x}_n, y_n\}_{n \in I_R^*}, F_I, C_R^*, \epsilon_R)$;
  Calculate $\delta_{\mathcal{L}}$ in (23);
  if $\delta_{\mathcal{L}} > 0$ **then**
    $\mathrm{node.leftChild} = \{\mathrm{data} = \{\boldsymbol{x}_n, y_n\}_{n \in I_L^*},\ \mathrm{model} = f_L^*,\ \mathrm{leftChild} = \mathrm{None},\ \mathrm{rightChild} = \mathrm{None}\}$;
    $\mathrm{node.rightChild} = \{\mathrm{data} = \{\boldsymbol{x}_n, y_n\}_{n \in I_R^*},\ \mathrm{model} = f_R^*,\ \mathrm{leftChild} = \mathrm{None}, \mathrm{rightChild} = \mathrm{None}\}$;
    $\mathrm{NumLeaf} = \mathrm{NumLeaf} + 1$;
  end
  if $\mathrm{MaxNumLeaf}$ *is not supplied, or* $\mathrm{NumLeaf} \le \mathrm{MaxNumLeaf}$ **then**
    For each leaf, the cross entropy loss (classification) or RMSE (regression) multiplied by the number of samples at that leaf is used as the impurity score;
    Identify node^*, the leaf node with the highest impurity score;
    $\mathrm{split}(\mathrm{node}^*)$;
  end
end

Supplementary Algorithm 11: $\mathrm{FitModelSVR}$ for regression.

Input: $\{\boldsymbol{x}_n, y_n\}_{n\in I}$, sample set of the current node;
F_I, ensemble of the models along the path from the root node to the parent node of the current node;
C, the regularization parameter of SVR;
ϵ, the slack variable of SVR.

Output: The SVR model f_m for the current node.

$\tilde{y}_n = y_n - F_I(\boldsymbol{x}_n), n \in I$;
Fit $f_m = \mathrm{SVR}(\{\boldsymbol{x}_n, \tilde{y}_n\}_{n\in I}, C, \epsilon)$ using SVR.

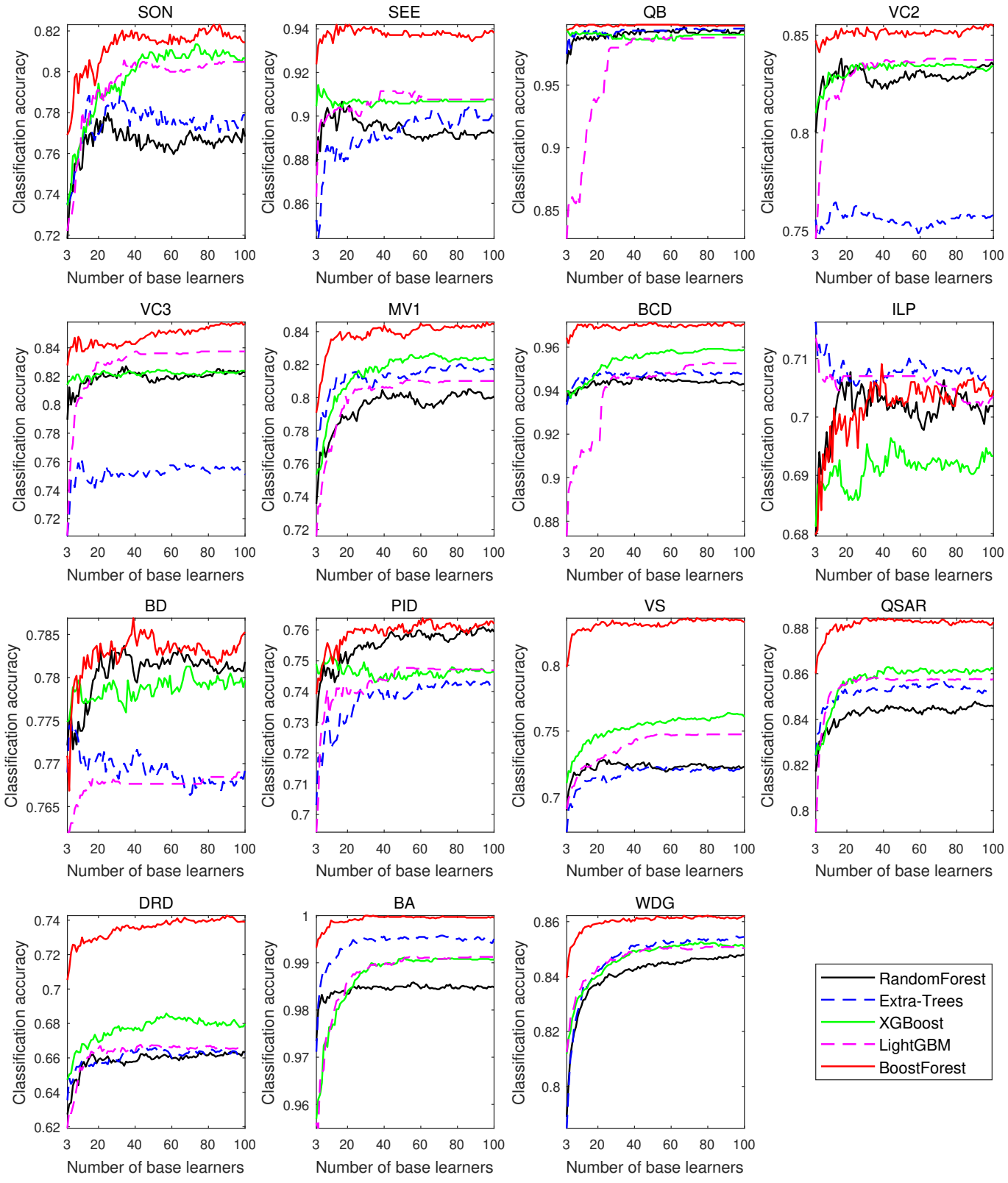
Supplementary Algorithm 12: BoostForest-SVR for regression.

Input: $\mathrm{Data} = \{\mathbf{x}_n, y_n\}_{n=1}^N$, N training samples, where $\mathbf{x}_n \in \mathbb{R}^D$;
$\mathrm{NumEstimators}$, the number of trees;
$\mathrm{Pool}_{\mathrm{MinSamplesLeaf}}$, candidate value pool of the minimum number of samples at a leaf;
Pool_ϵ, candidate value pool of the slack variable;
Pool_C, candidate value pool of the regularization parameter.

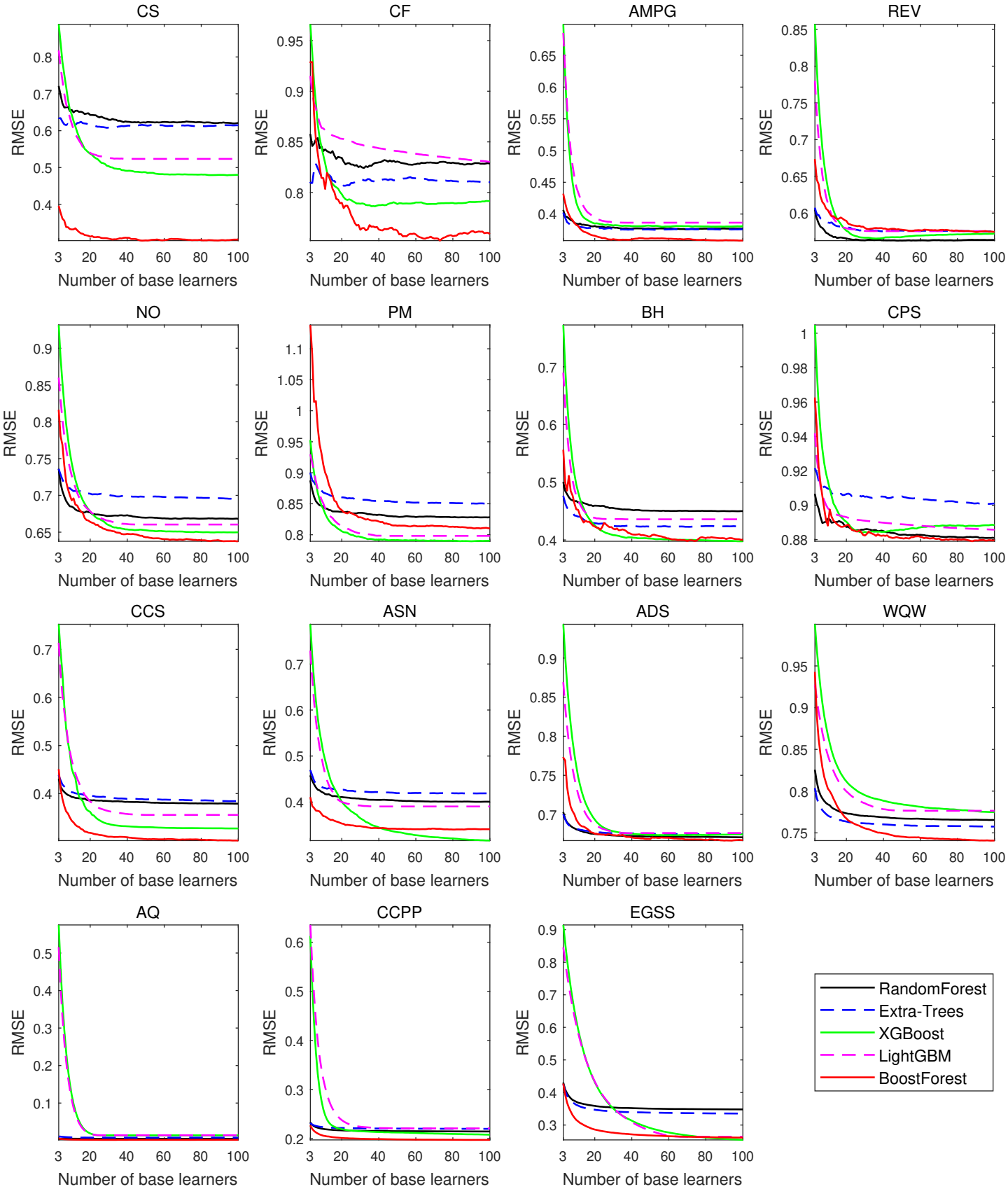
Output: A BoostForest-SVR

$\mathrm{BoostForest\text{-}SVR} = \{\}$;
for $i = 1 : \mathrm{NumEstimators}$ **do**
  Bootstrap Data' from Data;
  Train $\mathrm{BoostTree\text{-}SVR}_i$ on Data' using Supplementary Algorithm 10;
  Add $\mathrm{BoostTree\text{-}SVR}_i$ to $\mathrm{BoostForest\text{-}SVR}$;
end

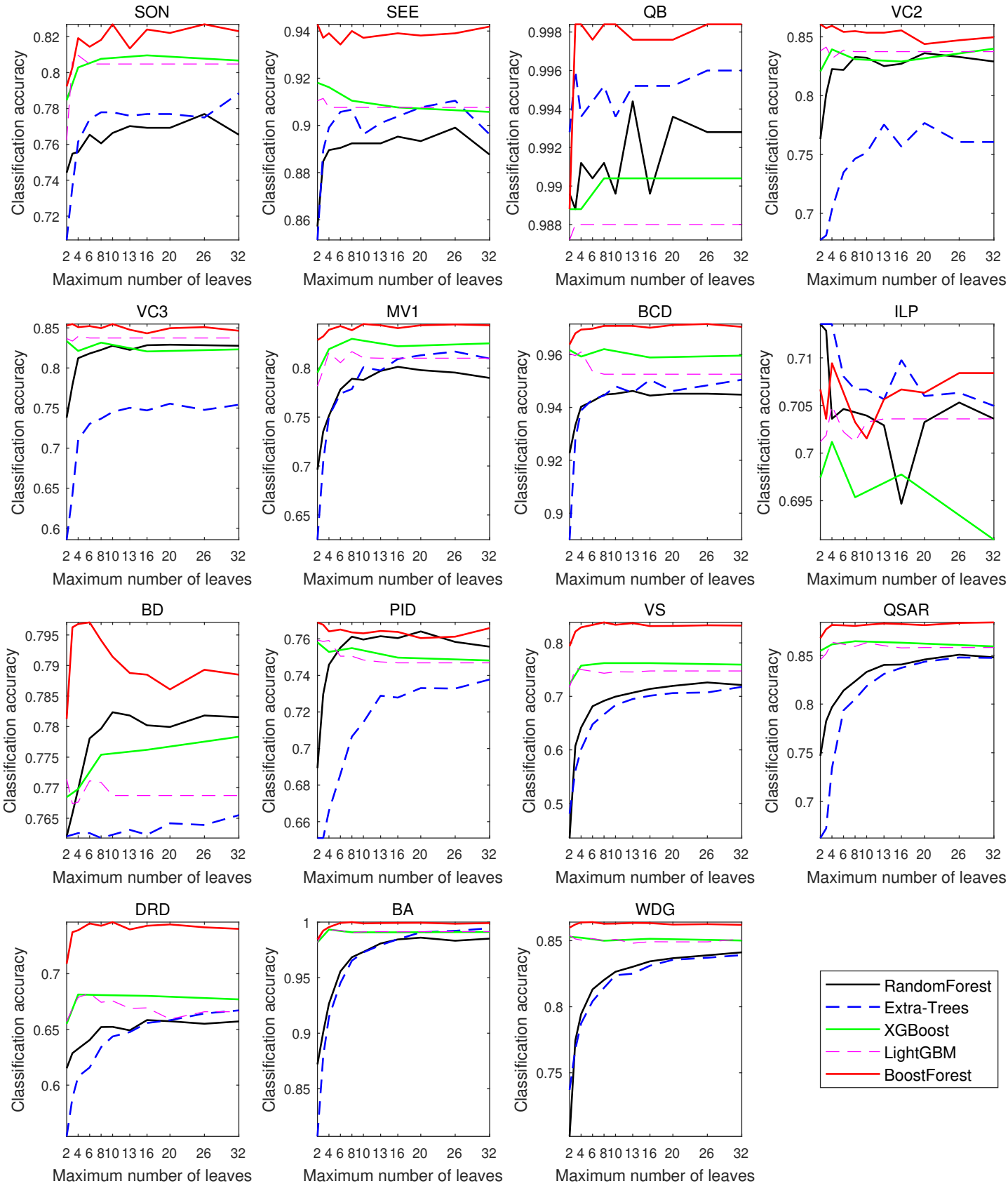
2 Supplementary Figures



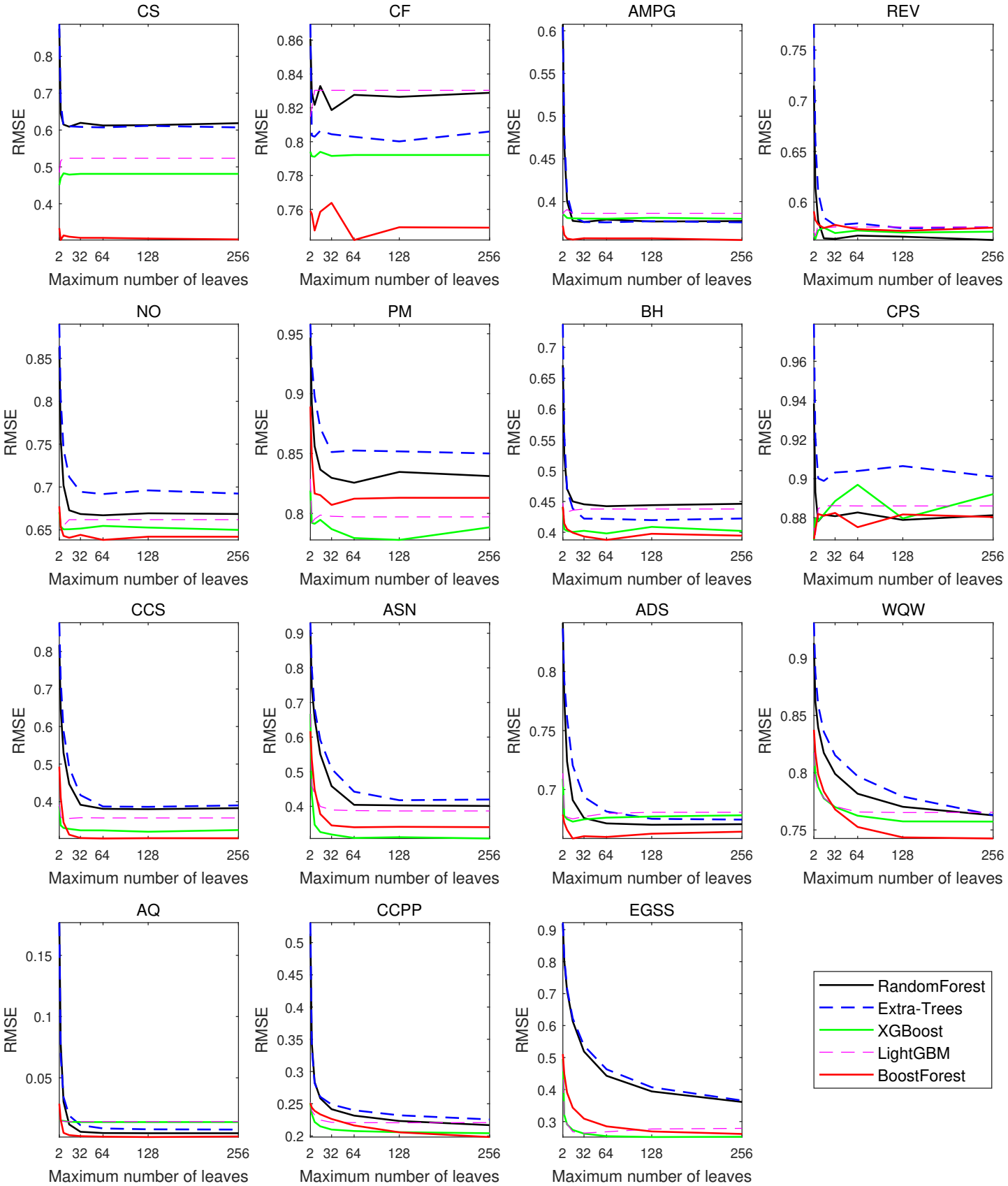
Supplementary Figure 1: Average classification accuracies on the 15 classification datasets, with different number of base learners.



Supplementary Figure 2: Average RMSEs on the 15 regression datasets, with different number of base learners.



Supplementary Figure 3: Average classification accuracies on the 15 classification datasets, with different maximum number of leaves.



Supplementary Figure 4: Average RMSEs on the 15 regression datasets, with different maximum number of leaves.

3 Supplementary Tables

This section introduces the statistics of the 34 datasets used in our experiments, and parameters of the ensemble algorithms.

Supplementary Table 1: The 34 datasets used in our experiments.

Task	Dataset	Abbreviation	#Samples	#Features	#Classes
Classification	Sonar	SON	208	60	2
	Seeds	SEE	210	7	3
	Qualitative Bankruptcy	QB	250	6	2
	Vertebral Column-2	VC2	310	6	2
	Vertebral Column-3	VC3	310	6	3
	Musk Version 1	MV1	476	166	2
	Breast Cancer Diagnosis	BCD	569	30	2
	Indian Liver Patient	ILP	583	11	2
	Blood Donations	BD	748	4	2
	Pima Indians Diabetes	PID	768	8	2
	Vehicle Silhouettes	VS	846	18	4
	QSAR Biodegradation	QSAR	1,055	41	2
	Diabetic Retinopathy Debrecen	DRD	1,151	19	2
	Banknote Authentication	BA	1,372	4	2
	Waveform Database Generator	WDG	5,000	21	3
	Letter Recognition	LR	20,000	16	26
	MNIST	MNIST	70,000	784	10
Regression	Concrete Slump	CS	103	7	
	Concrete Flow	CF	103	7	
	autoMPG	AMPG	392	7	
	Real Estate Valuation	REV	414	6	
	NO2	NO	500	7	
	PM10	PM	500	7	
	Boston Housing	BH	506	13	
	CPS	CPS	534	11	
	Concrete Compressive Strength	CCS	1,030	8	
	Airfoil Self-Noise	ASN	1,503	5	
	Abalone Data Set	ADS	4,177	10	
	Wine Quality White	WQW	4,898	11	
	Air Quality	AQ	9,357	8	
	Combined Cycle Power Plant	CCPP	9,568	4	
	Electrical Grid Stability Simulated	EGSS	10,000	12	
	Protein Tertiary Structure	PTS	45,730	9	
	Relative Location of CT	RLCT	53,500	384	

Supplementary Table 2: Parameter settings for RandomForest, Extra-Trees, XGBoost and LightGBM. More details about the parameters for RandomForest and Extra-Trees can be found at https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. More details about the parameters for XGBoost and LightGBM can be found at https://xgboost.readthedocs.io/en/latest/parameter.html.

Parameter	RandomForest Extra-Trees	XGBoost LightGBM
If the number of samples in the dataset is $< 10,000$		
nEstimators (number of trees)	$\{80, 100, 120\}$	$\{80, 100, 120\}$
minSamplesLeaf (minmum number of samples at a leaf node)	$[5, 15]$	
η (learning rate)		$\{0.01, 0.1, 0.2\}$
γ (minimum loss reduction required to further split a leaf node)		$\{0.25, 0.5, 1\}$
α (L1 regularization coefficient on weights)		$\{0.1, 0.5, 1\}$
λ (L2 regularization coefficient on weights)		$\{0.1, 0.5, 1\}$
subsample (subsample ratio of the training samples)		$\{0.8, 1\}$
colsample (subsample ratio of columns)		$\{0.8, 1\}$
maxDepth (maximum depth of a tree)		$\{4, 5, 6\}$
minChildWeight (minimum sum of sample weight needed in a child)		$\{0.5, 1, 3\}$ (classification) $\{5, 10, 15\}$ (regression)
If the number of samples in the dataset is $\geq 10,000$		
nEstimators (number of trees)	$\{80, 100, 120\}$	$\{1000\}$
minSamplesLeaf (minmum number of samples at a leaf node)	$\{5, 10, 15\}$	
η (learning rate)		$\{0.01\}$
γ (minimum loss reduction required to further split a leaf node)		default
α (L1 regularization coefficient on weights)		$\{0.1, 1\}$
λ (L2 regularization coefficient on weights)		$\{0.1, 1\}$
subsample (subsample ratio of the training samples)		default
colsample (subsample ratio of columns)		default
maxDepth (maximum depth of a tree)		$\{5, 6\}$
minChildWeight (minimum sum of sample weight needed in a child)		$\{5\}$
early_stopping_rounds		$\{50\}$

4 Supplementary Experiments

This section presents additional experimental results on using other base learners in BoostForest, and comparisons with MultiBoosting[2].

Use other base learners in BoostForest. 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., LMT or Model Tree. The resulting forests are denoted as LMForest and ModelForest, respectively.

LMT[3] extends model tree from regression to classification by integrating logistic regression into the tree model. It uses Stepwise Model Tree Induction[4] to construct the tree. The final logistic regression model at a leaf consists of all linear models at the nodes in the path from the root to that leaf. SimpleLogistic[3] (a variant of LogitBoost) is used to incrementally refine 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.

LMT[1] and M5P[2] implementations in Weka were used in our experiments. We stopped SimpleLogistic training if the minimum error on the validation set had not changed for 20 iterations. For each M5P model tree in ModelForest, the number of samples at a leaf node should not be too small, so the minimum number of samples at its leaf was randomly sampled from the parameter pool $\{10, 20, 30\}$. To improve its stability, RR models with regularization coefficient $\lambda = 0.001$ were trained at every leaf.

Supplementary Table 3 compares the performances of LMT with LMForest on the 15 classification datasets. Supplementary Table 4 compares the performances of M5P with ModelForest on the 15 regression datasets. All results were averaged over five repeats of 2-fold cross-validations. LMForest outperformed LMT on 10 of the 15 datasets, among which three were statistically significant. ModelForest outperformed M5P on 14 of the 15 datasets, among which 12 were statistically significant.

Comparison with MultiBoosting. Considering that BoostForest is an ensemble method integrating Bagging and Boosting, it is necessary to compare its performance with other algorithms using similar strategies.

MultiBoosting[2] is a representative method that combines waggings and Adaboost[5] to reduce both the bias and the variance. It adopts AdaBoost as the sub-committee classifiers' producer and decision tree (CART in our experiment) as the base learner to decide committees. By wagging a set of sub-committees, MultiBoosting achieves a lower error than AdaBoost. Its model complexity can be controlled by the number of estimators, the maximum depth of each tree, and MinSampleLeaf. We set their candidate values to $\{80, 100, 120\}$, $\{5, 7, 10\}$ and $\{5, 10, 15\}$, respectively. The best combination of parameters was determined by grid-search from its candidate values using inner 5-fold cross-validation.

There are three main differences between MultiBoosting and BoostForest:

1. MultiBoosting uses AdaBoost to train multiple tree models for boosting, and can only be used for classification, whereas BoostForest uses gradient boosting to train the base models, which can handle both classification and regression. BoostForest optimizes the loss function using gradient descent, and only one tree model is generated for boosting.

2. MultiBoosting first uses AdaBoost to train tree models to reduce the bias, and then wagging to train tree models to reduce the variance, when the training error is relatively small. BoostForest generates only one

[1] http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/LMT.html

[2] http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/M5P.html

Supplementary Table 3: Mean and standard deviation (in parenthesis) of the classification accuracy, when LMT is used to replace BoostTree in BoostForest. The best performance is marked in bold. • indicates statistically significant win for LMForest.

Dataset	LMT	LMForest
SON	0.762 (0.043)	**0.796** (0.030)
SEE	**0.941** (0.020)	0.933 (0.020)
QB	0.986 (0.013)	**0.994** (0.012)
VC2	**0.847** (0.027)	0.846 (0.021)
VC3	**0.857** (0.023)	0.850 (0.026)
MV1	0.794 (0.026)•	**0.822** (0.026)
BCD	0.967 (0.007)	**0.971** (0.007)
ILP	**0.706** (0.016)	0.700 (0.021)
BD	0.778 (0.016)	**0.787** (0.009)
PID	0.761 (0.019)	**0.762** (0.010)
VS	0.792 (0.017)	**0.800** (0.012)
QSAR	0.862 (0.010)•	**0.871** (0.009)
DRD	0.702 (0.016)	**0.707** (0.013)
BA	0.999 (0.002)•	**0.999** (0.002)
WDG	**0.866** (0.005)	0.865 (0.007)
Average ACC	0.841	**0.847**
Average rank	1.667	**1.333**

Supplementary Table 4: Mean and standard deviation (in parenthesis) of the regression RMSE, when M5P is used to replace BoostTree in BoostForest. The best performance is marked in bold. • indicates statistically significant win for ModelForest.

Dataset	M5P	ModelForest
CS	0.384 (0.039)•	**0.330** (0.030)
CF	0.782 (0.084)	**0.763** (0.076)
AMPG	0.379 (0.020)•	**0.365** (0.015)
REV	0.594 (0.067)•	**0.568** (0.069)
NO	0.697 (0.030)•	**0.652** (0.038)
PM	0.908 (0.075)•	**0.833** (0.082)
BH	0.459 (0.077)•	**0.389** (0.071)
CPS	**0.871** (0.132)	0.913 (0.122)
CCS	0.398 (0.020)•	**0.336** (0.017)
ASN	0.495 (0.060)•	**0.342** (0.016)
ADS	0.672 (0.029)	**0.667** (0.024)
WQW	0.832 (0.021)•	**0.748** (0.020)
AQ	0.089 (0.075)•	**0.004** (0.001)
CCPP	0.239 (0.003)•	**0.210** (0.003)
EGSS	0.373 (0.006)•	**0.281** (0.005)
Average RMSE	0.545	**0.493**
Average rank	1.933	**1.067**

Supplementary Table 5: Mean and standard deviation (in parenthesis) of the classification accuracy, when MultiBoosting is compared with BoostForest. The best performance is marked in bold. • indicates statistically significant win for BoostForest.

Dataset	MultiBoosting	BoostForest
SON	0.804 (0.026)•	**0.822** (0.026)
SEE	0.919 (0.029)•	**0.940** (0.022)
QB	0.991 (0.009)•	**0.998** (0.005)
VC2	0.839 (0.025)•	**0.853** (0.026)
VC3	0.841 (0.039)	**0.850** (0.027)
MV1	0.837 (0.030)	**0.843** (0.025)
BCD	0.966 (0.008)	**0.972** (0.009)
ILP	0.697 (0.020)	**0.709** (0.019)
BD	0.766 (0.018)•	**0.785** (0.007)
PID	0.748 (0.008)•	**0.762** (0.011)
VS	0.755 (0.020)•	**0.830** (0.016)
QSAR	0.868 (0.007)•	**0.882** (0.006)
DRD	0.679 (0.014)•	**0.740** (0.008)
BA	0.997 (0.003)	**0.998** (0.003)
WDG	0.845 (0.007)•	**0.861** (0.005)
Average accuracy	0.837	0.856
Average rank	2.000	**1.000**

tree for boosting, and uses Random Forest to reduce the variance, which is less likely to be trapped in a local minimum.

3. MultiBoosting uses traditional decision trees as the base learners, whereas BoostForest uses BoostTrees.

Table 5 compares the performances of BoostForest and MultiBoosting on the 15 classification datasets. BoostForest achieved higher accuracy on all datasets, among which 10 were statistically significant.

References

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

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

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

4. Malerba, D., Appice, A., Ceci, M. & Monopoli, M. Trading-off local versus global effects of regression nodes in model trees. In *Proc. 13th Int'l Symposium on Foundations of Intelligent Systems*, 393–402 (Lyon, France, 2002).

5. 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).