Supplementary material for: Smaller, more accurate regression forests using tree alternating optimization

Arman Zharmagambetov Miguel Á. Carreira-Perpiñán
Dept. of Computer Science & Engineering, University of California, Merced
http://eecs.ucmerced.edu

June 29, 2020

Abstract

We provide the following. 1) Proofs for the theorems in the main paper (section 1). 2) A set of experiments exploring single-output vs multi-output regression (section 2). 3) Additional experiments that give further evidence to the claims in the main paper (section 3); these include additional datasets, as well as training time. 4) An evaluation of different ways to solve the weighted 0/1 loss binary classification problem in the decision node optimization (section 4). 5) Description of the experiments' setup, for reproducibility: datasets, comparison methods, hyperparameters, etc. (section 5).

1 Formal theorem statements and proofs

A more general version of the Tree Alternating Optimization (TAO) algorithm, for arbitrary loss and regularization functions and for arbitrary types of trees and nodes, is given by Carreira-Perpiñán [5]. The results there are specialized here for regression.

1.1 Tree definition and optimization problem

Consider a rooted directed binary tree (each decision node has two children) of a given, predetermined structure (of depth Δ, not necessarily complete) with nodes indexed in set $\mathcal{N}$ and parameters $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_i\}_{i\in\mathcal{N}}$. Each decision node i has a decision function $f_i(\mathbf{x};\boldsymbol{\theta}_i)$: $\mathbb{R}^D \rightarrow \mathcal{C}_i$, where $\mathcal{C}_i = \{\texttt{left}_i, \texttt{right}_i\} \subset \mathcal{N}$, sending instance $\mathbf{x}$ to the corresponding child of i. We consider oblique trees, having hyperplane decision functions "go to right if $\mathbf{w}_i^T\mathbf{x} + w_{i0} \geq 0$" (where $\boldsymbol{\theta}_i = \{\mathbf{w}_i, w_{i0}\}$); axis-aligned (univariate) trees are a special case where $\mathbf{w}_i$ is an indicator vector for a single feature. Each leaf i has a predictor function $\mathbf{g}_i(\mathbf{x};\boldsymbol{\theta}_i)$: $\mathbb{R}^D \rightarrow \mathbb{R}^K$ that produces the actual output. We consider constant predictors $\mathbf{g}_i(\mathbf{x};\boldsymbol{\theta}_i) = \mathbf{w}_i$ and linear predictors $\mathbf{g}_i(\mathbf{x};\boldsymbol{\theta}_i) = \mathbf{W}_i\mathbf{x} + \mathbf{w}_i$ (where $\boldsymbol{\theta}_i = \{\mathbf{W}_i, \mathbf{w}_i\}$). The tree's prediction $\mathbf{T}(\mathbf{x};\boldsymbol{\Theta})$ for an instance $\mathbf{x}$ is obtained by routing $\mathbf{x}$ from the root to exactly one leaf and applying its predictor. We do not consider soft trees where $\mathbf{x}$ is routed to each leaf with a certain probability.

We consider the problem of learning the parameters of a regression tree of given structure by minimizing:

$$E(\boldsymbol{\Theta}) = \sum_{n=1}^{N} L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n;\boldsymbol{\Theta})) + \alpha \sum_{i\in\mathcal{N}} \phi_i(\boldsymbol{\theta}_i) \tag{1}$$

given a training set $\{(\mathbf{x}_n, \mathbf{y}_n)\}_{n=1}^N \subset \mathbb{R}^D \times \mathbb{R}^K$. The loss function $L(\mathbf{y}, \mathbf{z})$ measures the disagreement between two vectors $\mathbf{y}$ (ground-truth label) and $\mathbf{z}$ (tree prediction); we use the squared error $\|\mathbf{y} - \mathbf{z}\|_2^2$ (although it is possible to use other losses, such as the least absolute deviation or a robust loss). The regularization term penalizes the parameters $\boldsymbol{\theta}_i$ of each node, where ϕ_i is e.g. a norm such as ℓ_1 or ℓ_2. The hyperparameter $\alpha \geq 0$ controls the tradeoff between the loss and the regularization. We define the *reduced set* $\mathcal{R}_i \subset \{1, \dots, N\}$ of node i (decision node or leaf) as the training instances that reach i given the current tree parameters.

We say that $\mathcal{S} \subset \mathcal{N}$ is a set of non-descendant nodes if $\forall i, j \in \mathcal{S}$ neither i is a descendant of j nor j is a descendant of i in the tree graph. We assume that the parameters are not shared across nodes: $i, j \in \mathcal{N},\ i \neq j \Rightarrow \boldsymbol{\theta}_i \cap \boldsymbol{\theta}_j = \varnothing$.

Theorem 1.1 (Separability)**.** *Let $\mathbf{T}(\mathbf{x}; \boldsymbol{\Theta})$ be the predictive function of a rooted directed decision tree and $\mathcal{S} \subset \mathcal{N}$ a nonempty set of non-descendant nodes in the tree. Then, as a function of the parameters $\{\boldsymbol{\theta}_i\colon\ i \in \mathcal{S}\}$ (i.e., fixing all other parameters $\boldsymbol{\Theta}_{rest} = \boldsymbol{\Theta} \setminus \{\boldsymbol{\theta}_i\colon\ i \in \mathcal{S}\}$), the function $E(\boldsymbol{\Theta})$ of eq.* (1) *can be equivalently written as*

$$E(\boldsymbol{\Theta}) = \sum_{i \in \mathcal{S}} E_i(\boldsymbol{\theta}_i, \boldsymbol{\Theta}_{rest}) + E_{rest}(\boldsymbol{\Theta}_{rest}) \tag{2}$$

where $\{E_i\colon\ i \in \mathcal{S}\}$ and E_{rest} are certain functions.

Proof. Since the nodes in $\mathcal{S}$ are non-descendants, we have that $\forall i, j \in \mathcal{S},\ i \neq j \Rightarrow \mathcal{R}_i \cap \mathcal{R}_j = \varnothing$. Then we can write the objective function of eq. (1) as the sum of two terms:

$$E(\boldsymbol{\Theta}) = \sum_{i \in \mathcal{S}} E_i(\boldsymbol{\Theta}) + E_{\text{rest}}(\boldsymbol{\Theta})$$

where

$$E_i(\boldsymbol{\Theta}) = \sum_{n \in \mathcal{R}_i} L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})) + \alpha\, \phi_i(\boldsymbol{\theta}_i)$$

and

$$E_{\text{rest}}(\boldsymbol{\Theta}) = \sum_{n \in \overline{\mathcal{R}}} L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})) + \alpha \sum_{i \in \overline{\mathcal{S}}} \phi_i(\boldsymbol{\theta}_i)$$

where $\overline{\mathcal{S}} = \mathcal{N} \setminus \mathcal{S}$ and $\overline{\mathcal{R}} = \{1, \dots, N\} \setminus \cup_{i \in \mathcal{S}} \mathcal{R}_i$.

Each term $E_i(\boldsymbol{\Theta})$ does not depend on $\boldsymbol{\theta}_j$ for any $j \in \mathcal{S} \setminus \{i\}$, so we can write $E_i(\boldsymbol{\Theta}) = E_i(\boldsymbol{\theta}_i, \boldsymbol{\Theta}_{\text{rest}})$, because changing the value of $\boldsymbol{\theta}_j$ only affects $\mathcal{R}_j$ and L for $n \in \mathcal{R}_j$.

Likewise, term $E_{\text{rest}}(\boldsymbol{\Theta})$ does not depend on $\boldsymbol{\theta}_i$ for any $i \in \mathcal{S}$, so calling $\boldsymbol{\Theta}_{\text{rest}} = \{\boldsymbol{\theta}_i\colon\ i \in \overline{\mathcal{S}}\}$ we can write $E_{\text{rest}}(\boldsymbol{\Theta}) = E_{\text{rest}}(\boldsymbol{\Theta}_{\text{rest}})$, because changing the value of $\{\boldsymbol{\theta}_i\colon\ i \in \mathcal{S}\}$ only affects $\{\mathcal{R}_i\colon\ i \in \mathcal{S}\}$ and L for $n \in \cup_{i \in \mathcal{S}} \mathcal{R}_i$. □

Theorem 1.2 (Reduced problem over a decision node)**.** *Consider the objective function $E(\boldsymbol{\Theta})$ of eq.* (1) *and a decision node i. Assume the parameter values $\boldsymbol{\Theta} \setminus \{\boldsymbol{\theta}_i\}$ of all the nodes except i are fixed. Then, as a function of $\boldsymbol{\theta}_i$, we can write eq.* (1) *equivalently as:*

$$E(\boldsymbol{\Theta}) = E_i(\boldsymbol{\theta}_i) + E_{rest}(\boldsymbol{\Theta} \setminus \{\boldsymbol{\theta}_i\}) \qquad \textit{with} \qquad E_i(\boldsymbol{\theta}_i) = \sum_{n \in \mathcal{R}_i} l_{in}(f_i(\mathbf{x}_n; \boldsymbol{\theta}_i)) + \alpha\, \phi_i(\boldsymbol{\theta}_i) \tag{3}$$

where $\mathcal{R}_i$ is the reduced set of node i, and we define the function $l_{in}\colon\ \mathcal{C}_i \to \mathbb{R}$ as $l_{in}(z) = L(\mathbf{y}_n, \mathbf{T}_z(\mathbf{x}_n; \boldsymbol{\Theta}_z))$ for any $z \in \mathcal{C}_i$ (child of i), where $\mathbf{T}_z(\cdot; \boldsymbol{\Theta}_z)$ is the predictive function for the subtree rooted at node z.

Hence, the optimization problem $\min_{\boldsymbol{\theta}_i} E(\boldsymbol{\Theta})$ is equivalent to the following optimization problem:

$$\min_{\boldsymbol{\theta}_i} \overline{E}_i(\boldsymbol{\theta}_i) = \sum_{n \in \mathcal{R}_i} \overline{L}_{in}(\overline{y}_{in}, f_i(\mathbf{x}_n; \boldsymbol{\theta}_i)) + \alpha\, \phi_i(\boldsymbol{\theta}_i) \tag{4}$$

where the weighted 0/1 loss $\overline{L}_{in}(\overline{y}_{in}, \cdot)\colon\ \mathcal{C}_i \to \mathbb{R}^+ \cup \{0\}$ for instance $n \in \mathcal{R}_i$ is defined as $\overline{L}_{in}(\overline{y}_{in}, y) = l_{in}(y) - l_{in}(\overline{y}_{in})\ \forall y \in \mathcal{C}_i$, where $\overline{y}_{in} = \arg\min_{y \in \mathcal{C}_i} l_{in}(y)$ is the "best" child of i for n (or any $\overline{y}_{in} \in \arg\min_{y \in \mathcal{C}_i} l_{in}(y)$ in case of ties).

Proof. Applying theorem 1.1 (separability) to the particular case $\mathcal{S} = \{i\}$, we can write

$$E(\boldsymbol{\Theta}) = E_i(\boldsymbol{\theta}_i, \boldsymbol{\Theta}_{\text{rest}}) + E_{\text{rest}}(\boldsymbol{\Theta}_{\text{rest}})$$

where $\boldsymbol{\Theta}_{\text{rest}} = \boldsymbol{\Theta} \setminus \{\boldsymbol{\theta}_i\} = \{\boldsymbol{\theta}_j\colon\ j \in \mathcal{N} \setminus \{i\}\}$,

$$E_i(\boldsymbol{\theta}_i, \boldsymbol{\Theta}_{\text{rest}}) = \sum_{n \in \mathcal{R}_i} L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})) + \alpha\, \phi_i(\boldsymbol{\theta}_i)$$

and E_{rest} is defined as in the proof of theorem 1.1. To prove the first part of the theorem (eq. (3)), we just need to prove that, when $\boldsymbol{\Theta}_{\text{rest}}$ is constant, then for each $n \in \mathcal{R}_i$ we have $L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})) = l_{in}(f_i(\mathbf{x}_n; \boldsymbol{\theta}_i))$ for a certain function l_{in} which only depends on $\boldsymbol{\theta}_i$. In plain English, "$L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta}))$" is the loss of instance $\mathbf{x}_n$ under the current tree, which will depend on which child of node i we send $\mathbf{x}_n$ to; and "$l_{in}(f_i(\mathbf{x}_n; \boldsymbol{\theta}_i))$" is precisely the loss incurred by sending $\mathbf{x}_n$ down the child predicted by i's decision function.
Let us prove that. Refer to fig. 1, which illustrates a binary tree, but the argument works for any tree, not necessarily binary or complete. When we fix all parameters except $\boldsymbol{\theta}_i$, to understand the resulting objective function we need only consider the reduced set $\mathcal{R}_i$, the parameters and decision function of node i, and the subtree predictive function defined for each child of i. In the figure, where $i = 2$ and $\mathcal{C}_i = \{4, 5\}$, these are $\mathbf{T}_4(\mathbf{x}; \boldsymbol{\Theta}_4)$ and $\mathbf{T}_5(\mathbf{x}; \boldsymbol{\Theta}_5)$ for the left and right child of node i, respectively. Thus, the subtree predictive function $\mathbf{T}_4(\mathbf{x}; \boldsymbol{\Theta}_4)$ computes the prediction of the left subtree when node i sends an instance $\mathbf{x}$ to the left child. This is done by routing $\mathbf{x}$ down the left subtree to one leaf and applying the leaf's predictor to compute the output. The function $\mathbf{T}_4(\mathbf{x}; \boldsymbol{\Theta}_4)$ does not depend on $\boldsymbol{\theta}_i$. A similar argument holds for the right child. Therefore, the loss term $L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta}))$ can take only one of two possible values for an instance $\mathbf{x}_n \in \mathcal{R}_i$: $l_{in,4} = L(\mathbf{y}_n, \mathbf{T}_4(\mathbf{x}_n; \boldsymbol{\Theta}_4)) \in \mathbb{R}$ if $f_i(\mathbf{x}_n; \boldsymbol{\theta}_i) = 4$ (the left child) and $l_{in,5} = L_n(\mathbf{T}_5(\mathbf{x}_n; \boldsymbol{\Theta}_5)) \in \mathbb{R}$ if $f_i(\mathbf{x}_n; \boldsymbol{\theta}_i) = 5$ (the right child), where the decision function for node i is f_i: $\mathcal{X} \to \mathcal{C}_i$ (with parameters $\boldsymbol{\theta}_i$). Then, define a function l_{in}: $\mathcal{C}_i \to \mathbb{R}$ as $l_{in}(z) = L(\mathbf{y}_n, \mathbf{T}_z(\mathbf{x}_n; \boldsymbol{\Theta}_z))$, where $z \in \mathcal{C}_i$ is any child of i and $\mathbf{T}_z(\cdot; \boldsymbol{\Theta}_z)$ is the predictive function for the subtree rooted at z. The function l_{in} gives the loss value incurred by each of the two children of node i. Hence we can write $L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})) = l_{in}(f_i(\mathbf{x}_n; \boldsymbol{\theta}_i))$, which only depends on the parameters $\boldsymbol{\theta}_i$ of node i.

We now prove the second part of the theorem, eq. (4). This follows because $\overline{E}_i(\boldsymbol{\theta}_i) = E_i(\boldsymbol{\theta}_i) + \sum_{n \in \mathcal{R}_i} l_{in}(\overline{y}_{in})$ and the latter term is constant (independent of $\boldsymbol{\theta}_i$). Furthermore, the loss function $\overline{L}_{in}$ satisfies $\overline{L}_{in}(\overline{y}_{in}, y) = 0$ if $y = \overline{y}_{in}$ (or $y \in \arg\min_{y \in \mathcal{C}_i} l_{in}(y)$ in case of ties), and $\overline{L}_{in}(\overline{y}_{in}, y) > 0$ otherwise, hence it is a weighted 0/1 loss function with "ground-truth" label $\overline{y}_{in}$. For example, if $\mathcal{C}_i = \{\texttt{left}, \texttt{right}\}$ and $l_{in}(\texttt{left}) > l_{in}(\texttt{right})$ then $\overline{y}_{in} = \texttt{right}$, $\overline{L}_{in}(\texttt{right}, \texttt{right}) = 0$ and $\overline{L}_{in}(\texttt{right}, \texttt{left}) > 0$. Note that, although $\overline{L}_{in}(\cdot, \cdot)$ has two arguments, the first one is fixed: $\overline{L}_{in}(\overline{y}_{in}, \cdot)$. □

Theorem 1.3 (Reduced problem over a leaf)**.** *Consider the objective function $E(\boldsymbol{\Theta})$ of eq.* (1) *and a leaf node i. Assume the parameter values $\boldsymbol{\Theta} \setminus \{\boldsymbol{\theta}_i\}$ of all the nodes except i are fixed. Then, as a function of $\boldsymbol{\theta}_i$, we can write eq.* (1) *equivalently as:*

$$E(\boldsymbol{\Theta}) = E_i(\boldsymbol{\theta}_i) + E_{rest}(\boldsymbol{\Theta} \setminus \{\boldsymbol{\theta}_i\}) \qquad \textit{with} \qquad E_i(\boldsymbol{\theta}_i) = \sum_{n \in \mathcal{R}_i} L(\mathbf{y}_n, \mathbf{g}_i(\mathbf{x}_n; \boldsymbol{\theta}_i)) + \alpha\, \phi_i(\boldsymbol{\theta}_i) \tag{5}$$

where $\mathcal{R}_i$ is the reduced set of node i.

Proof. Applying theorem 1.1 (separability) to the particular case $\mathcal{S} = \{i\}$, we can write

$$E(\boldsymbol{\Theta}) = E_i(\boldsymbol{\theta}_i, \boldsymbol{\Theta}_{\text{rest}}) + E_{\text{rest}}(\boldsymbol{\Theta})$$

where $\boldsymbol{\Theta}_{\text{rest}} = \{\boldsymbol{\theta}_i\colon\ i \in \mathcal{N} \setminus \{i\}\}$,

$$E_i(\boldsymbol{\theta}_i, \boldsymbol{\Theta}_{\text{rest}}) = \sum_{n \in \mathcal{R}_i} L(\mathbf{y}_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})) + \alpha\, \phi_i(\boldsymbol{\theta}_i)$$

and $\boldsymbol{\Theta}_{\text{rest}}$ is defined as in the proof of theorem 1.1. The result follows by noting that, for a leaf i, $\mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})) = \mathbf{g}_i(\mathbf{x}_n; \boldsymbol{\theta}_i)$. □

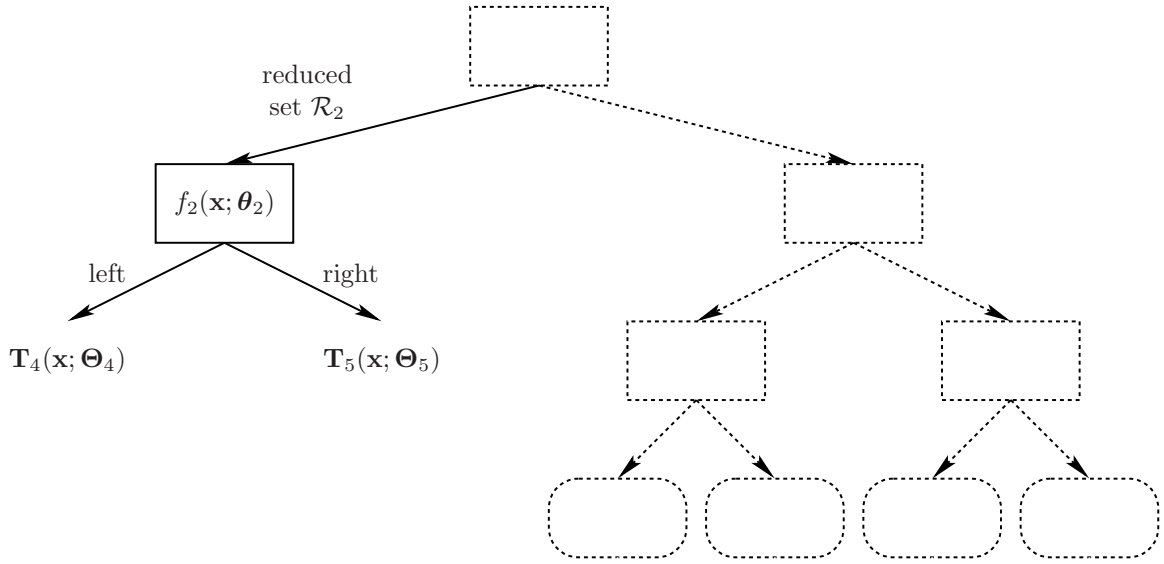

Figure 1: Schematic representation of the optimization over node 2 (a decision node, with parameters $\boldsymbol{\theta}_2$) in an example tree. The left and right subtrees of node 2 behave like two fixed predictor functions $\mathbf{T}_4(\mathbf{x};\boldsymbol{\Theta}_4)$ and $\mathbf{T}_5(\mathbf{x};\boldsymbol{\Theta}_5)$ which produce an output for an input $\mathbf{x}$ when going left or right in node 2, respectively. Only the training instances that reach node 2 under the current tree (the reduced set $\mathcal{R}_2$ of node 2) participate in the optimization.

2 Single-output vs multi-output regression

A regression problem with a real-valued output vector of dimension K can be considered as a single regression problem with a K-dimensional output, as K regression problems each with a one-dimensional output (scalar), or as any combination in between, such as 4 regression problems each with a $(K/4)$-dimensional output, and in general M models each with K/M outputs. The final, K-dimensional output is obtained by concatenating the individual output vectors of the M models.

In linear regression (which includes as particular case a constant predictor, for $\mathbf{W} = \mathbf{0}$) all of these are equivalent (using the squared error loss):

$$\min_{\mathbf{W},\mathbf{b}} \sum_{n=1}^{N} \|\mathbf{y}_n - \mathbf{W}\mathbf{x}_n - \mathbf{b}\|^2 = \min_{\substack{\mathbf{w}_1,\dots\mathbf{w}_K \\ b_1,\dots,b_K}} \sum_{k=1}^{K}\sum_{n=1}^{N} (y_{nk} - \mathbf{w}_k^T\mathbf{x}_n - b_k)^2 \Leftrightarrow \begin{cases} \min\limits_{\mathbf{w}_1,b_1} \sum\limits_{n=1}^{N} (y_{n1} - \mathbf{w}_1^T\mathbf{x}_n - b_1)^2 \\ \dots \\ \min\limits_{\mathbf{w}_K,b_K} \sum\limits_{n=1}^{N} (y_{nK} - \mathbf{w}_K^T\mathbf{x}_n - b_K)^2 \end{cases}$$

where $\mathbf{W} = (\mathbf{w}_1, \dots, \mathbf{w}_K)^T \in \mathbb{R}^{K\times D}$ and $\mathbf{b} \in \mathbb{R}^K$. That is, the K-dimensional output problem over $\mathbf{W}$ and $\mathbf{b}$ is equivalent to one separate, one-dimensional output problem for each row of $\mathbf{W}$ and $\mathbf{b}$.

However, with other models, such as neural nets or—our focus here—regression trees or forests, the above equivalence does not hold, and the resulting model (hence the error and model size) depends on which formulation we use. The reason is as follows. Consider a K-dimensional output. If we use a single tree having an output size K (i.e., each leaf outputs a K-dimensional vector), then all the decision nodes' parameters are shared across the K dimensions. If we use K trees each with an output size of 1 (whose concatenation is the desired K-dimensional output), then each tree has its own decision nodes' parameters, not shared with other trees. The same happens with neural nets, where the weight layers before the output layer are shared in a single, K-output net but not shared across K, single-output nets. In practice, with neural nets it is common to use a single net with a K-dimensional output layer rather than K nets each with a one-dimensional output layer.

Here, we evaluate empirically how the different formulations affect TAO and other regression forest algorithms. We consider the two regression tasks on MNIST of section 5.3 of the main paper: "full-image rotation" (with output dimension $K = D = 784$) and "patch selection and rotation" ($K = 64$). In each task, we consider 4 output sizes (1, $K/16$, $K/4$, K), by partitioning the original output image into blocks. For example, for the full-image rotation, we have the following: 784 blocks of size 1×1, 16 blocks of 7×7, 4 blocks of 14×14, and 1 block of 28×28. For each output size, we train 4 methods: Random Forests (RF), Extra Trees (ET), XGBoost, TAO-c (constant predictor at the leaves) and TAO-l (linear predictor at the leaves). Note that XGBoost is only applicable for an output dimension of one. For each method, we tune the forest hyperparameters (number of trees T and depth Δ) to achieve the lowest error possible.

Table 1 and figures 2–3 show the results. Firstly, the order of the different methods in terms of test error remains the same in all conditions and tasks. In decreasing error, we have XGBoost, RF, ET, TAO-c and (at a distance) TAO-l. The performance of the single-tree TAO-l (forest with $T = 1$) is already impressive: its test error is comparable to that of RF, ET and TAO-c forests, but the single TAO-l tree is of course far smaller and faster.

Second, as a function of the output size, all methods except TAO-l improve their error monotonically as the output size decreases and is lowest at an output size of 1 (i.e., concatenating K scalar regression forests). However, the model size increases (almost monotonically) as the output size decreases, and becomes huge if using an output size of 1. Hence, for these methods, the operating point that optimally trades off error vs model size will depend on the application. For TAO-l, remarkably, the error remains almost constant regardless of the output size (although, if we were to limit strongly its forest size, the error would naturally increase). The model size does vary: as the output size increases, the inference FLOPS decrease monotonically and the number of parameters decreases monotonically in the "patch" task but not in the "full-image" task (where it first decreases then increases). Hence, the optimal operating point in this case only depends on model size, but it is optimal or near-optimal for a single forest that outputs a K-dimensional vector. This makes the choice easy in practice for TAO-l: just use the K-dimensional output vector directly in a single forest.

	Forest, output size	$E_{\text{train}} \times 10^{-2}$	$E_{\text{test}} \times 10^{-2}$	#pars.	FLOPS	T	Δ
MNIST rotated	**TAO-c,** K	1.89±0.04	20.97±0.09	1 317 514	1 014	1	27
	TAO-c, $K/4$	1.51±0.05	19.31±0.07	1 276 663	2 367	4	79
	TAO-c, $K/16$	1.93±0.07	15.15±0.10	4 689 817	12 484	16	88
	RF, K	5.18±0.24	14.08±0.25	67 926 130	(28 259)	1000	40
	ET, K	0.00±0.00	13.72±0.13	108 999 700	(3 360)	1000	38
	TAO-c, K	2.03±0.06	13.29±0.11	9 102 171	34 738	30	31
	TAO-c, 1	1.21±0.06	12.98±0.05	21 735 871	1 090 147	784	9
	RF, $K/4$	4.75±0.11	12.55±0.13	29 729 538	(28 079)	400	121
	ET, $K/4$	0.00±0.00	11.97±0.01	47 105 192	(25 378)	400	117
	TAO-c, $K/4$	1.87±0.04	11.51±0.10	25 533 241	47 263	80	84
	RF, $K/16$	4.28±0.07	11.27±0.09	100 352 380	(143 991)	1600	152
	ET, $K/16$	0.00±0.00	10.87±0.01	159 845 216	(136 717)	1600	133
	XGBoost, 1	1.18±0.00	10.35±0.00	179 897 510	(612 924)	39200	25
	TAO-c, $K/16$	1.14±0.03	9.93±0.09	91 412 361	241 421	320	91
	TAO-l, 1	4.33±0.13	9.91±0.14	16 361 403	1 814 351	784	7
	TAO-l, $K/4$	4.44±0.09	9.79±0.11	189 984	10 217	4	7
	TAO-l, $K/16$	4.31±0.10	9.74±0.13	115 936	12 288	16	7
	TAO-l, K	4.28±0.11	9.63±0.17	288 342	4 491	1	7
	RF, 1	3.71±0.03	9.54±0.05	1 150 765 542	(2 777 482)	39200	114
	ET, 1	0.17±0.04	8.97±0.06	1 930 748 364	(2 613 017)	39200	127
	TAO-c, 1	1.01±0.07	8.21±0.09	267 124 048	8 917 431	7840	9
	TAO-l, 1	3.62±0.06	6.84±0.08	171 843 689	17 947 520	7840	7
	TAO-l, $K/4$	3.77±0.08	6.75±0.12	3 745 743	206 180	80	7
	TAO-l, $K/16$	3.59±0.09	6.73±0.09	2 318 723	245 761	320	7
	TAO-l, K	3.81±0.07	6.59±0.11	7 715 182	126 212	30	7
MNIST rotated patch	**TAO-c,** K	13.13±0.03	21.17±0.02	1 218 643	2 501	1	14
	TAO-c, $K/4$	12.91±0.02	20.23±0.07	113 956	8 651	4	9
	TAO-c, $K/16$	11.95±0.03	19.68±0.06	334 301	25 924	16	9
	TAO-c, 1	11.10±0.06	18.39±0.07	1 546 846	113 397	64	9
	RF, K	6.31±0.01	17.18±0.03	71 301 822	(42 078)	1000	61
	XGBoost, 1	0.55±0.00	16.79±0.00	44 099 280	(84 578)	3200	25
	ET, K	0.00±0.00	16.69±0.01	113 745 228	(41 079)	1000	64
	TAO-c, K	12.19±0.01	16.61±0.03	35 412 132	70 432	30	14
	TAO-l, $K/4$	6.81±0.03	16.46±0.06	112 435	9 517	4	7
	TAO-l, 1	6.90±0.08	16.42±0.07	1 511 332	156 182	64	7
	TAO-l, $K/16$	6.79±0.07	16.21±0.05	351 643	35 761	16	7
	TAO-l, K	6.76±0.02	16.13±0.05	65 462	2 674	1	7
	RF, $K/4$	6.06±0.08	16.01±0.16	29 628 978	(25 573)	400	66
	RF, $K/16$	5.92±0.09	15.57±0.10	103 251 968	(110 612)	1600	81
	RF, 1	6.06±0.02	15.51±0.03	146 529 256	(191 787)	3200	98
	ET, $K/4$	0.00±0.00	15.36±0.01	46 982 328	(24 832)	400	65
	TAO-c, $K/4$	11.82±0.02	15.22±0.04	2 284 919	163 351	80	9
	ET, 1	0.76±0.03	14.98±0.03	242 321 102	(180 537)	3200	88
	ET, $K/16$	0.00±0.00	14.64±0.05	165 132 006	(99 096)	1600	76
	TAO-c, $K/16$	10.63±0.02	14.22±0.08	6 413 743	514 430	320	9
	TAO-c, 1	10.21±0.04	13.91±0.09	13 678 720	1 064 160	640	9
	TAO-l, 1	4.41±0.02	10.26±0.03	14 897 960	1 603 227	640	7
	TAO-l, $K/4$	4.64±0.02	10.12±0.04	2 431 017	184 140	80	7
	TAO-l, $K/16$	4.25±0.03	10.03±0.05	7 310 176	719 964	320	7
	TAO-l, K	4.14±0.01	9.91±0.03	1 984 143	77 936	30	7

Table 1: Single-output vs multi-output regression. Like tables 7 and 9 for the MNIST rotation tasks, but where we consider regression problems of varying output size. For the "full-image" rotation $K = 784$; for the "patch selection and rotation" $K = 64$. The rows are sorted in decreasing test error.

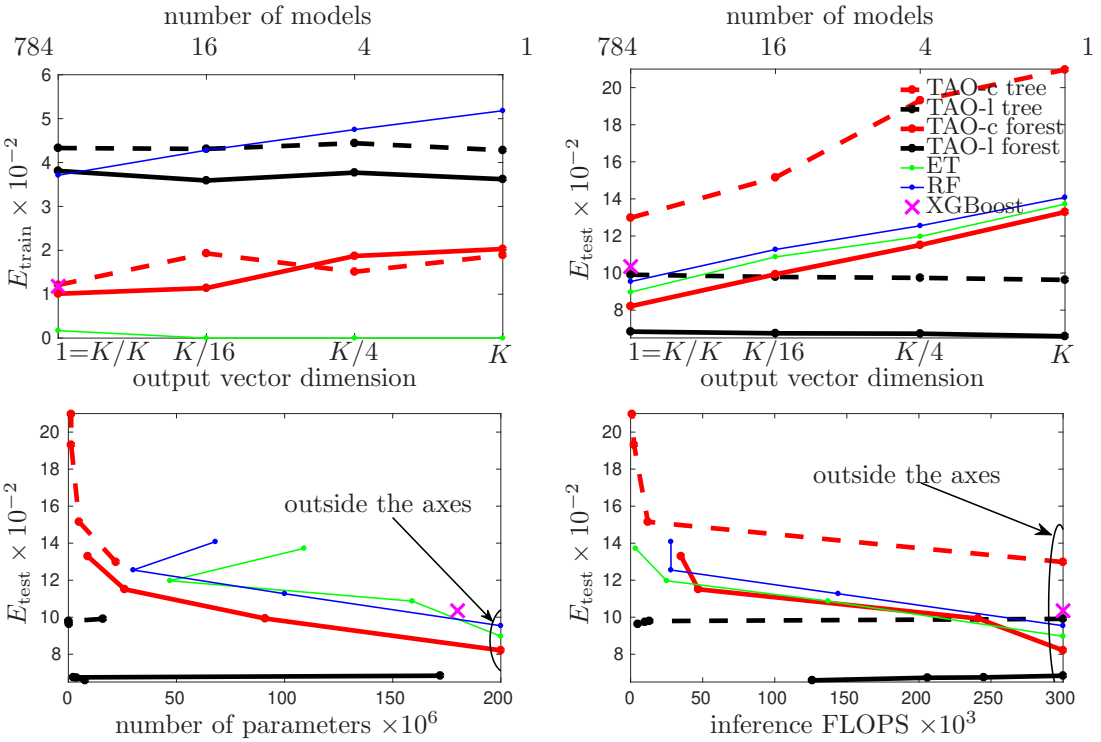

Figure 2: The results of table 1 but as a figure, for the MNIST "full-image rotation" task. *Top*: error in the training set (left) and test set (right). The X axis shows the output vector dimension (below) and number of models to be concatenated (above). *Bottom*: test error vs number of parameters (left) and inference FLOPS (right), showing the tradeoff between accuracy and size for each forest method and output vector size. Note some points do not fit within the figure axes and have been projected on the axes boundaries.

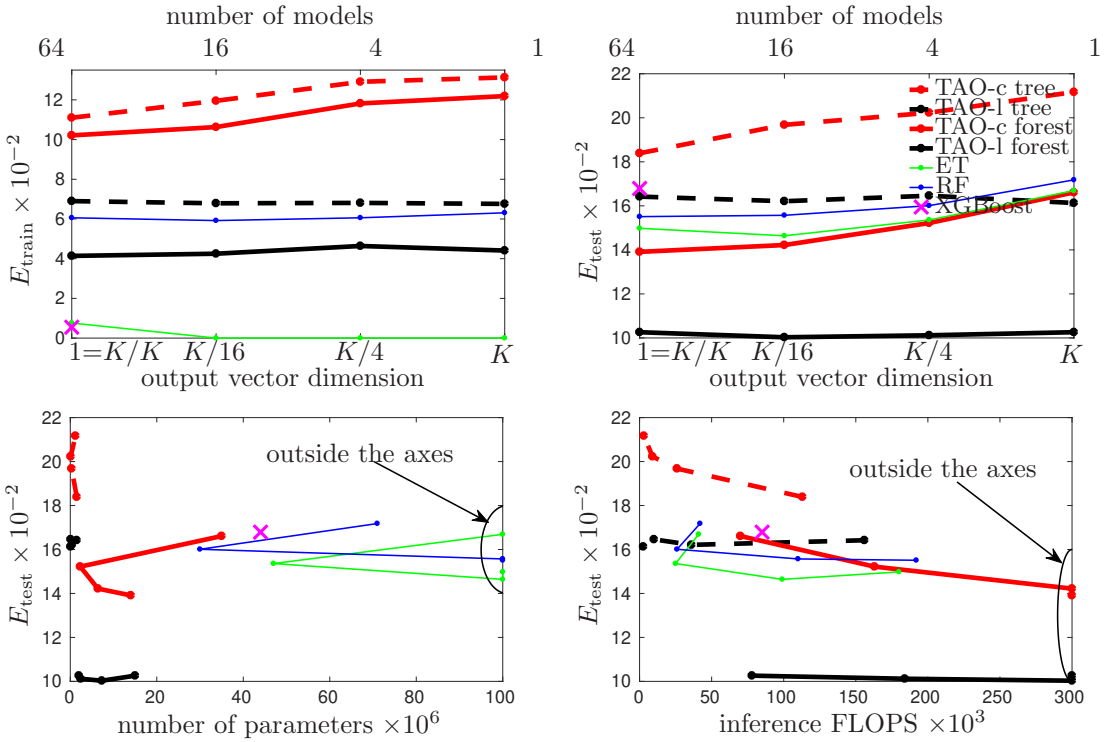

Figure 3: Like fig. 2 but for the MNIST "patch selection and rotation" task.

Dataset	RF	QRF	Huber	Tukey	**TAO-c**	**TAO-l**
CCS	6.1	5.9	5.7	5.9	5.7	5.2
airfoil	4.3	3.2	3.8	4.1	2.6	2.2

Table 2: Comparison with the forest model from [12] and other methods (taken from [12]) on the Concrete Compressive Strength (CCS) and Airfoil datasets.

3 Additional experimental results

We repeat the tables and figures in the main paper but with additional information (including training time in the MNIST tasks). We also include another dataset (APPA-REAL, for an application of human age regression) and a comparison with another method (Robust Forest [12]). The conclusions are as in the main paper.

Table 2 shows the results of comparison against [12] which uses robust loss functions (e.g. Huber) to develop better performing regression forests. We compare with their published results on two UCI repository datasets: concrete compressive strength (CCS) and airfoil, both with scalar outputs. All forest models in [12] use 1 000 trees with depth 10, while both TAO-c and TAO-l use 30 trees of depth 7. Other than that, we follow the same experimental settings as in the other experiments: we chose 2/3 observations for training and the rest for testing, and we report the RMSE over 5 independent runs. TAO forests are the clear winners in both datasets by a considerable margin.

	Forest	E_{train}	E_{test}	#pars.	FLOPS	T	Δ
abalone	CART	0.13±0.00	3.01±0.01	2 891	20	1	20
	XGBoost	0.52±0.00	2.22±0.00	31 532	(1089)	100	10
	XGBoost	0.01±0.00	2.20±0.00	219 586	(9 349)	1000	10
	GIF [2]	–	2.18	(50 120)	–	10	–
	TAO-c	2.11±0.02	2.18±0.05	287	41	1	6
	AdaBoost	0.86±0.01	2.16±0.01	52 758	(1 000)	100	10
	AdaBoost	0.84±0.00	2.15±0.00	497 507	(10 000)	1000	10
	ET	0.00±0.00	2.14±0.00	443 809	(3 011)	100	36
	RF	0.83±0.00	2.12±0.01	230 806	(2 473)	100	29
	rRF [17]	–	2.10±0.01	(100 000)	(1 000)	100	10
	ARF [20]	–	2.10±0.03	(100 000)	(1 000)	100	10
	RF	0.80±0.00	2.10±0.00	2 306 668	(24 736)	1000	34
	TAO-c	1.98±0.00	2.08±0.01	8 940	1 307	30	6
	TAO-l	2.00±0.01	2.07±0.01	303	40	1	5
	TAO-c	1.94±0.00	2.05±0.01	33 217	1 718	30	8
	TAO-l	1.98±0.01	2.04±0.01	8 328	1 204	30	5
ailerons ($E \times 10^{-4}$)	CART	2.80±0.00	2.88±0.00	103	9	1	9
	RF	0.65±0.01	1.84±0.02	875 264	(3 512)	100	45
	ET	0.00±0.00	1.84±0.00	1 426 004	(4 068)	100	49
	ARF [20]	–	1.78±0.01	(35 770)	(750)	50	15
	AdaBoost	1.32±0.00	1.77±0.01	18 215	(1 175)	100	15
	TAO-c	1.65±0.02	1.76±0.02	681	87	1	6
	rRF [17]	–	1.75±0.02	(71 540)	(1 000)	100	10
	RF	0.70±0.00	1.75±0.00	8 740 049	(35 042)	1000	47
	AdaBoost	1.27±0.00	1.75±0.00	200 028	(12 184)	1000	15
	XGBoost	1.63±0.00	1.74±0.00	1 792	(300)	100	7
	TAO-l	1.64±0.02	1.74±0.01	447	93	1	5
	XGBoost	1.62±0.00	1.72±0.00	3 592	(1 264)	1000	7
	TAO-c	1.55±0.03	1.67±0.04	21 107	2 513	30	6
	TAO-l	1.53±0.02	1.66±0.04	26 712	2 611	30	5
cpuact	CART	0.03±0.00	3.63±0.32	9 691	25	1	25
	TAO-c	2.47±0.07	2.71±0.04	498	51	1	6
	RF	0.95±0.01	2.62±0.04	620 272	(2 842)	100	36
	ARF [20]	–	2.62±0.01	(98 300)	750	50	15
	AdaBoost	1.20±0.01	2.61±0.16	72 320	(1 000)	100	10
	RF	0.91±0.00	2.60±0.01	6 204 052	(28 401)	1000	37
	XGBoost	0.93±0.00	2.60±0.00	40 174	(1 000)	100	10
	ET	0.00±0.00	2.58±0.03	982 620	(3 733)	100	45
	TAO-l	2.36±0.03	2.58±0.02	246	41	1	5
	XGBoost	0.00±0.00	2.57±0.00	293 788	(8 780)	1000	10
	AdaBoost	1.14±0.00	2.56±0.11	701 208	(10 000)	1000	10
	ET	0.00±0.00	2.49±0.03	9 825 562	(37 605)	1000	50
	TAO-c	2.11±0.03	2.39±0.05	24 090	1 590	30	7
	TAO-l	2.21±0.01	2.35±0.01	8 133	1 179	30	5

Table 3: Like table 1 in the main paper, but also including the training set error.

	Forest	E_{train}	E_{test}	#pars.	FLOPS	T	Δ
CT slice	CART	0.01±0.00	2.71±0.06	84 967	51	1	51
	TAO-c	1.42±0.04	1.54±0.05	7 167	1 123	1	7
	AdaBoost	1.05±0.01	1.48±0.03	122 484	(1 000)	100	10
	XGBoost	0.73±0.00	1.45±0.00	71 298	(1 000)	100	10
	AdaBoost	0.98±0.00	1.31±0.01	1 084 836	(10 000)	1000	10
	XGBoost	0.08±0.00	1.18±0.00	464 474	(10 000)	1000	10
	TAO-l	0.76±0.02	1.16±0.02	5 445	768	1	5
	ET	0.00±0.00	1.06±0.01	85 416 322	(62 433)	100	82
	RF	0.41±0.00	1.03±0.01	5 393 612	(5 818)	100	71
	cRF [7]	–	1.00	(17M)	–	1000	–
	RF	0.37±0.00	0.97±0.01	53 895 970	(57 299)	1000	78
	TAO-c	0.84±0.02	0.89±0.02	213 721	30 743	30	7
	TAO-l	0.52±0.01	0.71±0.02	164 647	23 070	30	5
	TAO-l	0.49±0.01	0.58±0.03	241 687	24 661	30	6
YearPredictionMSD	CART	0.01±0.00	13.41±0.11	620 539	49	1	49
	RF	3.40±0.00	9.31±0.00	40 115 501	(5 237)	100	68
	ET	0.01±0.00	9.31±0.00	76 827 734	(6 091)	100	73
	AdaBoost	5.45±0.02	9.25±0.01	2 527 629	(1 500)	100	15
	RF	3.33±0.00	9.23±0.00	401 180 944	(52 066)	1000	73
	AdaBoost	5.21±0.01	9.21±0.03	24 428 596	(15 000)	1000	15
	TAO-c	8.91±0.03	9.11±0.05	6 808	448	1	8
	TAO-l	8.89±0.02	9.08±0.03	2 361	388	1	6
	XGBoost	7.74±0.00	9.04±0.00	102 843	(1 000)	100	10
	XGBoost	3.79±0.00	9.01±0.00	1 145 460	(10 000)	1000	10
	cRF [7]	–	8.90	(184M)	–	1000	–
	TAO-c	8.87±0.01	8.90±0.01	186 423	13 437	30	7
	TAO-l	8.83±0.01	8.87±0.01	73 412	12 061	30	6
	TAO-c	8.61±0.01	8.85±0.01	246 187	13 948	30	9
	TAO-l	8.59±0.01	8.83±0.01	148 171	12 487	30	7
SARCOS	CART	0.11±0.00	3.62±0.00	83 577	23	1	21
	TAO-c	2.37±0.03	2.44±0.04	71 263	202	1	14
	RF	1.54±0.01	1.56±0.01	5 054 601	(2 997)	100	30
	RF	1.53±0.01	1.54±0.01	50 548 077	(29 991)	1000	30
	MLP (from [21])	–	1.46	139 015	–	1	–
	AdaBoost	1.40±0.01	1.40±0.01	1 217 090	(7 000)	700	10
	TAO-c	1.35±0.02	1.36±0.03	1 993 254	5 796	30	14
	AdaBoost	1.33±0.01	1.33±0.03	11 875 582	(70 000)	7000	10
	TAO-c	1.25±0.03	1.30±0.03	6 536 343	10 257	50	15
	XGBoost	1.24±0.00	1.24±0.00	773 682	(7 000)	700	10
	ANT [21]	–	1.18	103 823	61 640	1	–
	ANT [21]	–	1.11	598 280	360 766	8	–
	XGBoost	1.10±0.00	1.10±0.00	3 942 708	(70 000)	7000	10
	TAO-l	1.01±0.02	1.04±0.02	18 211	151	1	10
	TAO-l	0.84±0.02	0.85±0.02	694 117	4 833	30	10

Table 4: Like table 2 in the main paper, but also including the training set error.

3.1 Application: human age regression

We study the performance of TAO forests in a real-world application. We use the APPA-REAL [1] dataset, which consists of 7 591 annotated real human images. Each image has two labels: real age and apparent age. The apparent age is calculated based on 38 votes (on average) per image, which makes the average apparent age very stable (0.3 standard deviation). We test all methods on both of the tasks. The output is a scalar in [1 100]. We use the same train/test splitting as in the original paper: 4 113 train, 1 500 validation and 1 978 test images. All forest-based methods use as features the output of 2 048 neurons from a pretrained deep net. Specifically, we fine-tune a ResNeXt50_32x4d architecture on human face images (RGB) and use the output of the last convolutional layer as features. The human face images were provided by the authors [1] and they were obtained using a face detector from [14]. Also, following [18] we report the results of neural networks as well. Specifically, the network architecture is the same as in [23], except the last fully connected layer is replaced by another 101-dimensional linear layer and the final prediction is the expected value of the softmax probabilities. We train our network using ADAM [11] for 80 epochs with a learning rate of 0.001 which decays each 20 epochs by 0.2. This network achieves 2.79% train error and 7.89% test error (both RMSE) on apparent age prediction and 5.66% train error and 10.15% test error (both RMSE) on real age prediction. We use the PyTorch [15] implementation of the ResNeXt (available inside the Torchvision package) pretrained on ImageNet. Table 5 shows the results (sorted by decreasing test error).

	Forest	E_{train}	E_{test}	#pars.	FLOPS	T	Δ
Apparent age	ResNeXt	2.79	7.89			–	–
	AdaBoost	0.79±0.01	7.89±0.03	237 342	(1 500)	100	15
	AdaBoost	0.48±0.00	7.83±0.01	2 373 962	(15 000)	1000	15
	TAO-c	4.11±0.04	7.83±0.08	44 526	1 803	1	8
	XGBoost	0.05±0.00	7.82±0.00	109 014	(1 500)	100	15
	XGBoost	0.00±0.00	7.80±0.00	236 637	(15 000)	1000	15
	ET	0.00±0.00	7.76±0.02	684 236	(3 360)	100	44
	RF	1.54±0.01	7.73±0.01	436 104	(2 967)	100	37
	RF	1.53±0.00	7.72±0.01	4 308 285	(28 481)	1000	40
	TAO-c	3.85±0.04	7.60±0.04	1 611 405	34 919	30	8
	TAO-l	3.74±0.03	7.54±0.04	1 011	387	1	4
	TAO-l	3.53±0.01	7.48±0.01	33 171	13 029	30	4
Real age	ResNeXt	5.66	10.15			–	–
	AdaBoost	0.78±0.01	7.90±0.02	238 503	(1 500)	100	15
	TAO-c	4.16±0.06	7.86±0.11	47 421	2 379	1	8
	AdaBoost	0.48±0.00	7.85±0.01	2 382 924	(15 000)	1000	15
	XGBoost	0.05±0.00	7.83±0.00	107 019	(1 500)	100	15
	XGBoost	0.00±0.00	7.81±0.00	234 336	(15 000)	1000	15
	RF	1.57±0.01	7.78±0.02	430 463	(2 843)	100	36
	ET	0.00±0.00	7.77±0.01	683 706	(3 389)	100	41
	RF	1.55±0.01	7.74±0.01	4 306 096	(28 460)	1000	38
	TAO-c	3.58±0.04	7.65±0.06	1 714 103	71 023	30	8
	TAO-l	3.85±0.09	7.59±0.03	1 161	574	1	4
	TAO-l	3.56±0.02	7.50±0.02	31 374	12 213	30	4

Table 5: Like table 3 but for APPA-REAL human age regression. The ResNeXt rows are not forests but a deep neural net; we omit its number of parameters and FLOPS. For all forest methods, when calculating a model size we ignore the cost of feature extraction from the deep net.

3.2 MNIST regression

3.2.1 Predicting a rotated MNIST image: "full image rotation"

We can construct a challenging regression problem as described below, to define a mapping that is locally linear but globally severely nonlinear and where both the input and output are high-dimensional. For each image $\mathbf{x}_n \in \mathbb{R}^D$, we generate its ground-truth output $\mathbf{y}_n \in \mathbb{R}^D$ by rotating the image by an angle which depends on the digit class of $\mathbf{x}_n$. The rotation is a linear transformation $\mathbf{A}\mathbf{x}_n$ (slightly nonlinear because of cropping effects in the image borders), where each output pixel is a linear combination of some of the input pixels (with convex coefficients, so each output pixel is in [0 1]). However, the overall mapping is nonlinear because the rotation angle depends on the digit's ground-truth class.

We assign an angle in $[-90° \; 90°]$ at random to each class (see table 6). We implement the rotation using the `ndimage` package in the `scipy` Python library. We set its parameters as follows: `order` = 1 (to use linear spline interpolation, which ensures the output pixels are in [0 1], just like the input pixels); `reshape` = `false` (to apply cropping if the image exceeds the borders). All other parameters are set to their default values. We run TAO for 20 iterations with $\alpha = 0.01$.

Table 7 shows the results of different forest methods. Fig. 4 visualizes the (input,output,predicted) images for some sample instances.

For the constant-leaf forests (all except TAO-l), we clearly see that the predictions look blurry. This is because the forest output is the average of one leaf's output per tree, and each leaf's output is itself the average of the instances reaching that leaf.

class	0	1	2	3	4	5	6	7	8	9
angle	8°	49°	−57°	−63°	16°	−18°	−10°	−32°	−71°	58°

Table 6: Rotation angle of each digit class.

	Forest	$E_{\text{train}} \times 10^{-2}$	$E_{\text{test}} \times 10^{-2}$	#pars.	FLOPS	T	Δ	Time
MNIST rotated	AdaBoost	>24 hours runtime				39200	25	
	CART	0.37±0.00	23.08±0.12	119 937	28	1	28	3.5
	TAO-c	13.31±0.27	21.10±0.37	16 941 530	5 108	1	16	411
	RF	5.21±0.25	14.38±0.23	7 587 246	(2 830)	100	39	5.2
	RF	5.18±0.24	14.08±0.25	67 926 130	(28 259)	1000	40	18.7
	ET	0.00±0.00	13.83±0.12	11 999 900	(3 091)	100	35	3.6
	TAO-c	2.03±0.10	13.76±0.09	9 174 514	41 808	30	25	291
	ET	0.00±0.00	13.72±0.13	108 999 700	(3 360)	1000	38	8.2
	XGBoost	1.18±0.00	10.35±0.00	179 897 510	(612 924)	39200	25	238
	TAO-l	4.28±0.11	9.63±0.17	288 342	4 491	1	7	186
	TAO-l	3.81±0.07	6.59±0.11	7 715 182	126 212	30	7	257

Table 7: Like table 3 but for the MNIST regression task "full-image rotation". Additionally, we report the training set error and the training time of the entire forest (minutes). Note: for the TAO-c forests we used a CART tree as initial structure rather than a complete tree so that the tree could be deeper (which worked better in this particular dataset) yet computationally feasible. Everywhere else in this and other tables we always use a complete tree structure as initial tree for TAO.

Figure 4: Selected MNIST test images, corresponding ground-truth output (class-dependent full-image rotation) and predicted output by different forest algorithms. You may want to zoom in.

3.2.2 Predicting a rotated MNIST image: "patch selection and rotation"

Another challenging regression problem is defined as follows. We can use the previous MNIST experiment to make the output depend on only a subset of the pixels, hence do feature selection (separately for each class). For each image $\mathbf{x}_n \in \mathbb{R}^D$, we generate its ground-truth output $\mathbf{y}_n \in \mathbb{R}^K$ by rotating an 8×8 patch ($K = 64$ pixels) of the image by an angle, where both the patch location and the rotation angle depend on the digit class of $\mathbf{x}_n$.

We assign an angle in $[-90° \; 90°]$ and a patch location at random to each class (see table 8 and fig. 5). We run TAO for 30 iterations with $\alpha = 0.01$.

Table 9 shows the results of different forest methods. Fig. 6 visualizes the (input,output,predicted) images for some sample instances.

We also evaluate how well each forest is able to select features (pixels) that are necessary to predict the ground truth, by comparing the set of features used by the forest (in any node of any tree) with the set of ground-truth features (the set of input pixels that appear in any patch). As can be seen from the Jaccard, precision and recall scores, TAO trees and forests are better than other forests at discarding features that are not useful for the prediction.

class	0	1	2	3	4	5	6	7	8	9
angle	$-86°$	$-10°$	$-29°$	$16°$	$21°$	$-23°$	$-2°$	$-52°$	$-90°$	$-5°$
bbox	[6 6 14 14]	[10 11 18 19]	[5 19 13 27]	[15 14 23 22]	[6 9 14 17]	[3 17 11 25]	[13 10 21 18]	[12 6 20 14]	[9 8 17 16]	[5 10 13 18]

Table 8: Rotation angle and image patch location (as a bounding box containing the X,Y coordinates of the lower left and upper right corners, respectively) for each class.

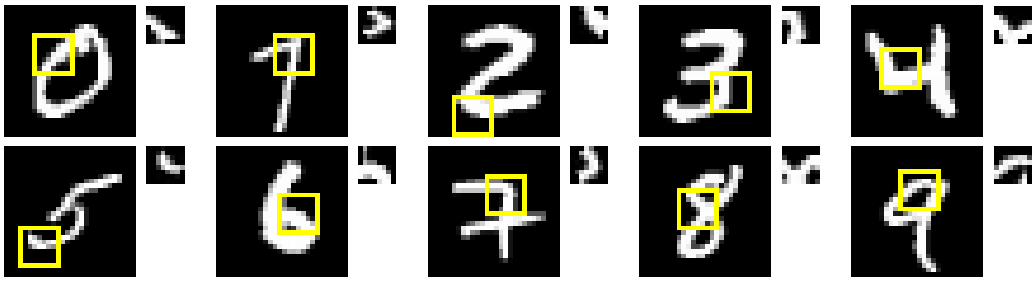

Figure 5: Image patch location for each class, shown on sample images, as well as the rotated patch.

	Forest	$E_{\text{train}} \times 10^{-2}$	$E_{\text{test}} \times 10^{-2}$	#pars.	FLOPS	T	Δ	Time	J(%)	P(%)	R(%)
MNIST rotated patch	CART	0.07±0.00	28.51±0.11	118 493	49	1	49	1.5	54	54	100
	TAO-c	13.13±0.02	21.17±0.02	1 218 643	2 501	1	14	155	62	62	100
	RF	6.77±0.01	17.87±0.04	7 561 030	(4 669)	100	59	2.0	45	45	100
	RF	6.31±0.01	17.18±0.03	71 301 822	(42 078)	1000	61	4.7	44	44	100
	ET	0.00±0.00	16.97±0.01	11 970 720	(4 595)	100	57	1.1	44	44	100
	XGBoost	0.55±0.00	16.79±0.00	44 099 280	(84 578)	3200	25	24.8	45	45	100
	ET	0.00±0.00	16.69±0.01	113 745 228	(41 079)	1000	64	3.2	44	44	100
	AdaBoost	0.53±0.05	16.65±0.09	108 774 252	(79 688)	3200	25	517	44	44	100
	TAO-c	12.19±0.02	16.61±0.03	35 412 132	70 432	30	14	218	52	52	100
	TAO-l	6.76±0.02	16.13±0.05	65 462	2 674	1	7	119	58	58	100
	TAO-l	4.14±0.01	9.91±0.03	1 984 143	77 936	30	7	157	51	51	100

Table 9: Like table 7 but for predicting MNIST rotated patch. Additionally, we report the training set error and the training time of the entire forest (minutes), and the Jaccard score J, precision P and recall R of the pixels used by each forest compared to the ground-truth pixels that are necessary to generate the output.

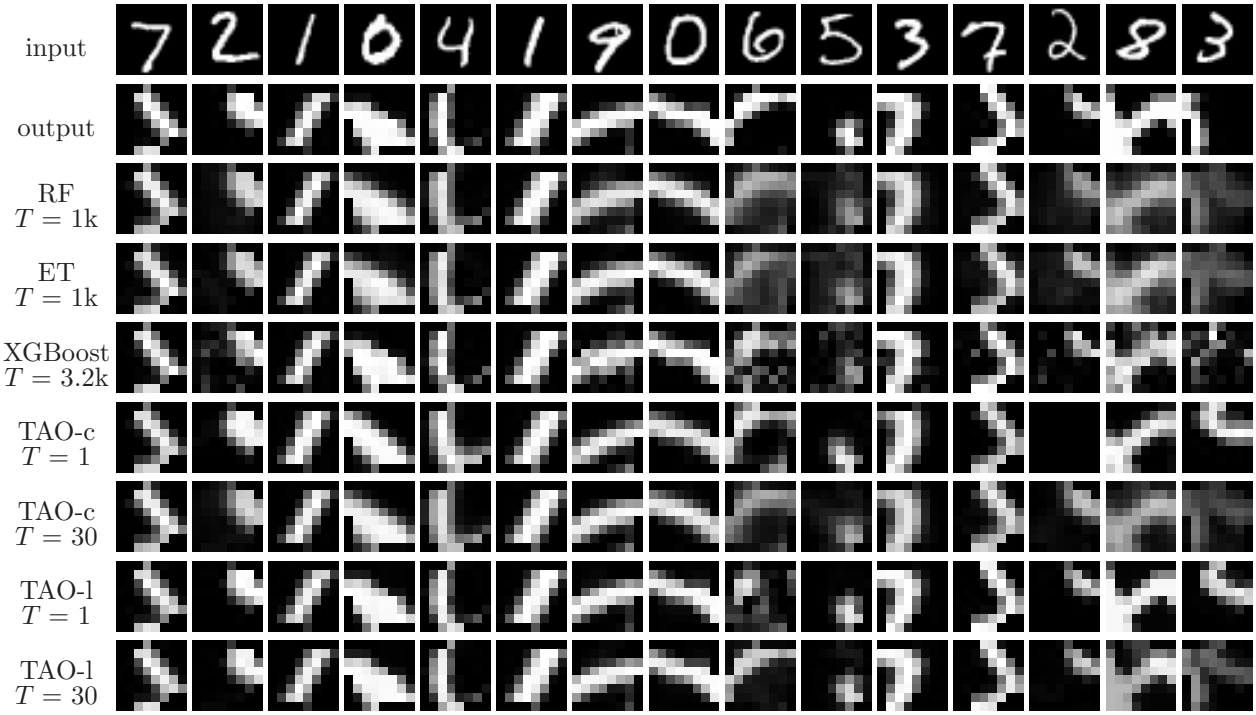

Figure 6: Like fig. 4 but for predicting the MNIST rotated patch.

3.3 Study of different diversity mechanisms

We amplify the results of section 5.5 in the main paper on additional datasets (the conclusions are the same):

Different training samples and different initialization Fig. 7 on the ailerons dataset.

Feature subsets Fig. 10 on the CT slide dataset.

Forest hyperparameters: tree depth Δ, number of trees T and number of TAO iterations I Fig. 8 on the ailerons dataset.

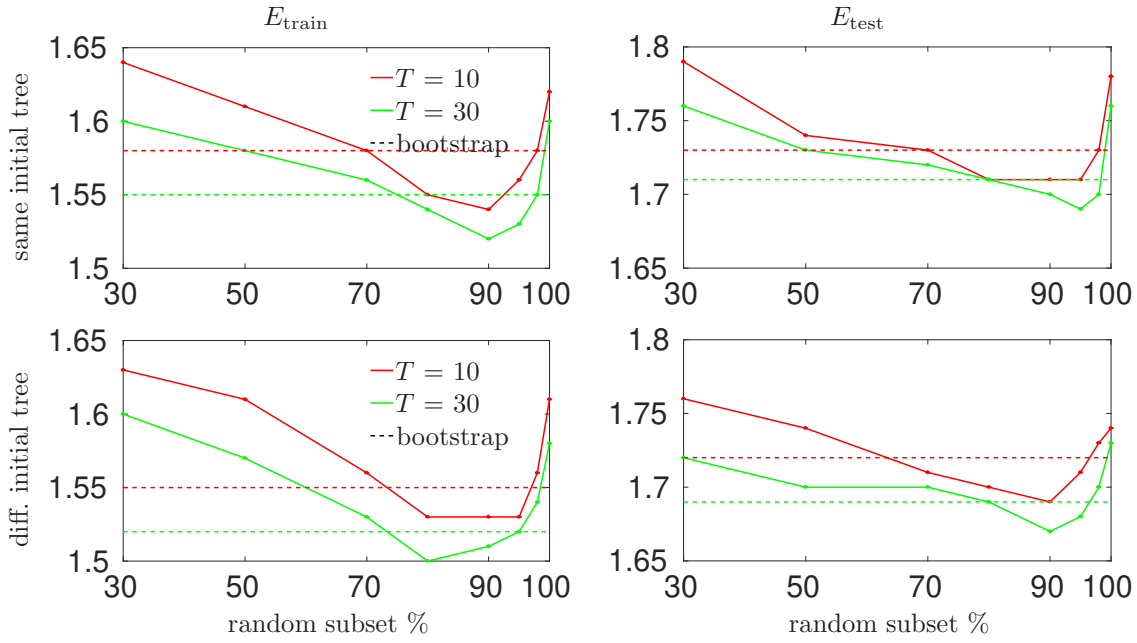

Figure 7: Like fig. 4 in the main paper but for the ailerons dataset.

	Size m of feature subset	$T = 10$ E_{train}	$T = 10$ E_{test}	$T = 20$ E_{train}	$T = 20$ E_{test}
	$m = D$ (all features)	1.70	1.75	1.59	1.63
Local	$m = D^{9/10}$	1.76	1.82	1.64	1.71
	$m = \lfloor D/3 \rfloor$	1.84	1.88	1.73	1.76
	$m = D^{8/10}$	1.97	2.04	1.88	1.94
	$m = D^{7/10}$	2.32	2.37	2.29	2.36
	$m = D^{6/10}$	2.97	3.07	2.93	3.01
Global	$m = D^{9/10}$	2.71	2.79	2.63	2.71
	$m = \lfloor D/3 \rfloor$	4.14	4.25	4.02	4.17
	$m = D^{8/10}$	6.61	6.73	6.52	6.61
	$m = D^{7/10}$	10.94	11.10	10.18	10.32
	$m = D^{6/10}$	15.73	15.81	14.45	14.67

Table 10: Like table 5 in the main paper but for the CT slice dataset. All trees are complete of depth 7.

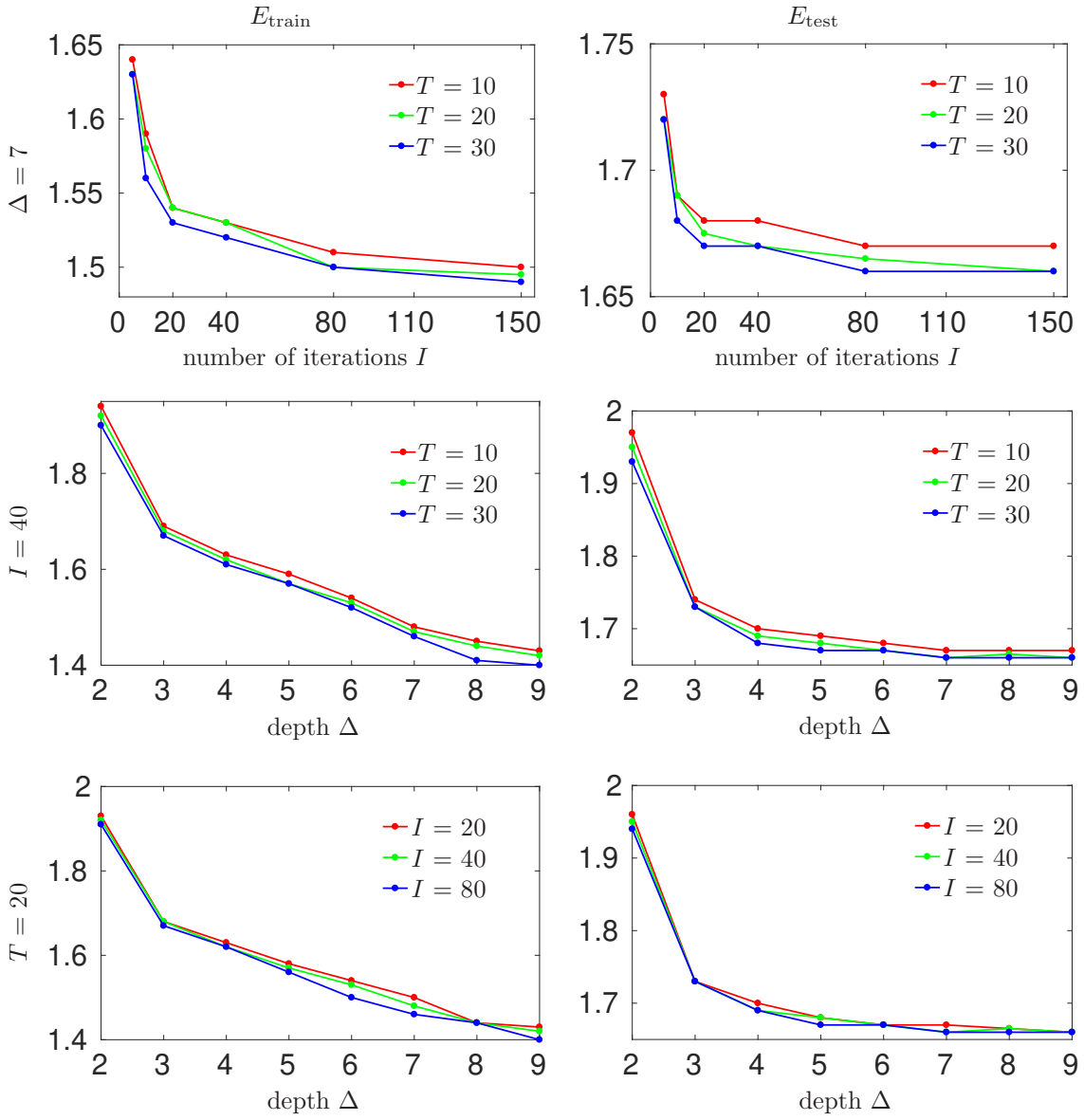

Figure 8: Like fig. 5 in the main paper but for the ailerons dataset.

4 Different ways to solve the weighted 0/1 loss binary classification problem in the decision node optimization

The optimization problem over the parameters $\boldsymbol{\theta}_i$ of a decision node i is as follows (from theorem 3.2 in the main paper):

$$\min_{\boldsymbol{\theta}_i} \overline{E}_i(\boldsymbol{\theta}_i) = \sum_{n \in \mathcal{R}_i} \overline{L}_{in}(\overline{y}_{in}, f_i(\mathbf{x}_n; \boldsymbol{\theta}_i)) + \alpha\, \phi_i(\boldsymbol{\theta}_i) \tag{4}$$

where the weighted 0/1 loss $\overline{L}_{in}(\overline{y}_{in}, \cdot)$: $\mathcal{C}_i \rightarrow \mathbb{R}^+ \cup \{0\}$ for instance $n \in \mathcal{R}_i$ is defined as $\overline{L}_{in}(\overline{y}_{in}, y) = l_{in}(y) - l_{in}(\overline{y}_{in})\ \forall y \in \mathcal{C}_i$, where $\overline{y}_{in} = \arg\min_{y \in \mathcal{C}_i} l_{in}(y)$ is the "best" child of i for n (or any $\overline{y}_{in} \in \arg\min_{y \in \mathcal{C}_i} l_{in}(y)$ in case of ties). Hence, the loss $\overline{L}_{in}$ in eq. (4) is either zero (if we predict the child correctly) or a value $\beta_{in} = l_{in}(y) - l_{in}(\overline{y}_{in}) \geq 0$ (if we predict it incorrectly). This latter value, or "weight", β_{in} depends on the instance n, hence the loss is a weighted 0/1 loss. We can write eq. (4) explicitly in terms of the weights like this:

$$\min_{\boldsymbol{\theta}_i} \overline{E}_i(\boldsymbol{\theta}_i) = \sum_{n \in \mathcal{R}_i} \beta_{in}\, L_0(\overline{y}_{in}, f_i(\mathbf{x}_n; \boldsymbol{\theta}_i)) + \alpha\, \phi_i(\boldsymbol{\theta}_i) \tag{6}$$

where L_0 is the regular 0/1 loss.

In section 3.3 "Solving the decision node optimization problem" of the main paper, we solve this approximately via a convex surrogate loss, such as the ℓ_1-penalized logistic regression:

$$\min_{\boldsymbol{\theta}_i} \sum_{n \in \mathcal{R}_i} \log\left(1 + e^{-z_{in}(\mathbf{w}_i^T \mathbf{x}_n + b_i)}\right) + \alpha \left\|\boldsymbol{\theta}_i\right\|_1$$

where we define the label z_{in} to be $+1$ (-1) if the right (left) child is the correct child, and the binary classifier is such that $\mathbf{w}_i^T \mathbf{x}_n + b_i \geq 0$ (< 0) corresponds to the right (left) child. However, this must be adapted to handle the weights $\{\beta_{in}\}_{n \in \mathcal{R}_i}$. We studied different ways to do this:

Unweighted 0/1 loss by binarization We simply binarize the losses of all instances to make them 1 if $\beta_{in} > 0$. This considers as equally important a mistake no matter what instance it happens in. Also, we discard instances if $\beta_{in} < \epsilon$ for a fixed value $\epsilon \geq 0$; this allows us to ignore instances for which going left or right makes a small difference.

Unweighted 0/1 loss by instance replication We approximate the weighted 0/1 loss problem with an unweighted one by replicating instances proportionally to their β_{in} values. That is, if we have N instances in the reduced set, we create a dataset of M instances such that the number of times instance n appears is (approximately) proportional to its β_{in} value. As $M \rightarrow \infty$, the unweighted problem on the replicate dataset becomes equivalent to the weighted problem on the original dataset.

Weighted surrogate loss We use the weights directly as multipliers in the logistic loss:

$$\min_{\boldsymbol{\theta}_i} \sum_{n \in \mathcal{R}_i} \beta_{in}\, \log\left(1 + e^{-z_{in}(\mathbf{w}_i^T \mathbf{x}_n + b_i)}\right) + \alpha \left\|\boldsymbol{\theta}_i\right\|_1.$$

Table 11 shows the results on two datasets. The best option, consistently, is to use a weighted surrogate loss, and this is the option we use in the main paper.

	Approximation type		E_{train}	E_{test}
ailerons ($E \times 10^{-4}$)	unweighted 0/1 loss by binarization	$\epsilon = 0$	1.67±0.02	1.77±0.03
		$\epsilon = 0.01$	1.66±0.03	1.77±0.03
		$\epsilon = 0.05$	1.65±0.02	1.76±0.03
		$\epsilon = 0.10$	1.65±0.03	1.76±0.04
		$\epsilon = 0.20$	1.64±0.03	1.78±0.02
	unweighted 0/1 loss by instance replication	$M = 2N$	1.68±0.04	1.81±0.03
		$M = 5N$	1.63±0.03	1.78±0.03
		$M = 10N$	1.62±0.02	1.78±0.04
	weighted surrogate loss		1.62±0.02	1.73±0.02
cpuact	unweighted 0/1 loss by binarization	$\epsilon = 0$	2.56±0.06	2.83±0.05
		$\epsilon = 0.01$	2.54±0.06	2.81±0.06
		$\epsilon = 0.05$	2.51±0.08	2.84±0.03
		$\epsilon = 0.10$	2.45±0.03	2.80±0.07
		$\epsilon = 0.20$	2.43±0.02	2.91±0.05
	unweighted 0/1 loss by instance replication	$M = 2N$	2.71±0.05	2.91±0.06
		$M = 5N$	2.54±0.05	2.81±0.07
		$M = 10N$	2.62±0.07	2.85±0.06
	weighted surrogate loss		2.47±0.07	2.71±0.04

Table 11: Comparison of different ways to solve the weighted 0/1 loss decision node optimization, for TAO trees with constant-label leaves, on the ailerons and cpuact datasets: unweighted 0/1 loss by binarization, unweighted 0/1 loss by instance replication, and weighted surrogate loss. We report the train/test RMSE (avg±stdev over 5 repeats). All trees use the same depth ($\Delta = 7$).

5 Experimental setup

5.1 Datasets

Table 12 summarizes the characteristics of the datasets used in our experiments. For CCS and airfoil, we randomly choose 2/3 samples for training and the rest for testing as in [12]. As for other benchmarks which do not have separate test split (abalone, cpuact, ailerons and CT slice), we split them into 60% training and 40% testing sets. We repeat this procedure 4 times and run each algorithm 5 times for each train/test split. The description of the datasets is as follows (all regression tasks have dense features unless otherwise stated):

CCS The task is to predict compressive strength of the concrete from material characteristics (age, amount of cement, water, etc). The dataset is taken from the UCI Machine Learning Repository [13].

airfoil This regression dataset is also available in the UCI Machine Learning Repository [13]. The features are various airfoil measurements (frequency, chord length, etc.) and the target variable is the scaled sound pressure level (in decibels).

cpuact Predict the portion of time that CPUs run in user mode given different system measures. We obtained it from the DELVE data collection[1].

abalone Predict the age of an abalone from physical measurements. The first attribute ("sex") is categorical and we encode it as one-hot. Available in the UCI Machine Learning Repository [13].

ailerons Aircraft control action prediction[2]. The attributes describe the status of the aircraft and the target is the command given to its ailerons.

CT slice The attributes are histogram features (in polar space) of the Computer Tomography (CT) slice. The task is to predict the relative location of the image on the axial axis (in the range [0 180]). Available in the UCI Machine Learning Repository [13].

SARCOS [22] Robot arm inverse dynamics problem[3]. The task is to map from a 21-dimensional input space (7 joint positions, 7 joint velocities, 7 joint accelerations) to the corresponding 7 joint torques.

YearPredictionMSD A subset of the Million Song Dataset [3]. The task is to predict the age of a song from several song statistics given as metadata (timbre average, timbre covariance). Obtained from the UCI Machine Learning Repository [13].

[1] http://www.cs.toronto.edu/~delve/data/comp-activ/desc.html
[2] https://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html
[3] http://www.gaussianprocess.org/gpml/data/

Dataset	N_{train}	N_{test}	D	K	output range
CCS	687	343	8	1	[4.78 82.6]
airfoil	1 002	501	5	1	[103.38 139.92]
abalone	2 506	1 671	8	1	[2 29]
cpuact	4 915	3 277	21	1	[−0.5 99.47]
ailerons	7 154	6 596	40	1	$[-3.5 \times 10^{-3} \ -2.51 \times 10^{-3}]$
CT slice	42 800	10 700	384	1	[1.73 97.49]
SARCOS	44 484	4 449	21	7	[−104.80 121.38]
YearPredictionMSD	463 715	51 630	90	1	[1922 2011]

Table 12: Specs of the datasets used in our experiments. N is sample size, D is input dimension (number of features) and K is output dimension. We also give the range of the output, for reference when evaluating the RMSE in the forest predictions.

5.2 Forest-based methods we compare with

Random Forests (RF) [4] uses an ensemble of independent trees, each trained on a bootstrap subset of the training data (bagging). Each tree is trained using CART but each node split can only use features from a random subset of size m (see below regarding the choice of m).
We use the Python implementation in scikit-learn [16]. We do not restrict the `max_depth` hyperparameter and allow each tree to grow fully, as is recommended for random forests [4].

Extremely randomized trees (ET) [10] is similar to random forests, but additional randomization is introduced in the cut-point choice.
We use the Python implementation in scikit-learn [16]. As in RF, we do not restrict the `max_depth` hyperparameter and allow each tree to grow fully, as is recommended for Random Forests [4].

AdaBoost [19]. This is one of the earliest boosting frameworks. It uses a set of "weak learners" (typically shallow trees) which are trained sequentially. At each boosting iteration, the training instances are reweighted so the new learner focuses mostly on those instances which have larger errors.
We use the Python implementation in scikit-learn [16], which implements the version of AdaBoost known as SAMME [24]. We tune the most important hyperparameters `max_depth`, `n_estimators` (in the tables, Δ and T, respectively) and `learning_rate` on a subset of the training data for each dataset separately.

Gradient boosting [9] is a generalization of AdaBoost as an approximate gradient optimization in function space of stagewise additive models.
We use the highly optimized XGBoost implementation [6]. We use the CPU version of the Python API provided by the authors. As in AdaBoost, we tune the most important hyperparameters `max_depth`, `n_estimators` (in the tables, Δ and T, respectively) and `learning_rate` on a subset of the training data for each dataset separately. All other parameters, such as the regularization parameters, booster, etc., are set to their default values.

We also compare with other recently proposed forest-based algorithms. For the rest of the methods below, we compare with their published results:

Alternating Regression Forests (ARF) [20]: this algorithm essentially trains a Random Forest using a combination of boosting and greedy tree growing.

Adaptive Neural Trees (ANT) [21]: this trains probabilistic decision trees (soft trees) which additionally transform the initial input feature space at each edge along the path. For a fair comparison, we only compare with the ANT version using linear decision nodes (including transformers) and linear leaves.

Globally Induced Forest (GIF) [2]: this algorithm starts with a predefined number of trees which are all decision stumps and grows only a subset of trees, those which show the best error reduction across the ensemble. This is repeated until a stopping criteria is met.

Robust Forest [12]: this algorithm trains a RF for regression task in a regular way but the prediction of the forest is obtained by solving an optimization problem instead of taking simple average.

Consistent Random Forest (cRF) [7]: this is a modified version of the conventional Random Forest where, when training each individual tree, two separate data points are used for the decision node optimization and a split threshold is calculated on a subset of those points.

Refined Random Forest (rRF) [17]: this takes as input a pretrained forest and, leaving the tree structure and decision nodes unchanged, globally optimizes over the parameters on the leaves, which results in an improved prediction error.

Note that, although Random Forests, AdaBoost and gradient boosting are considered to be robust to hyperparameter choice, sometimes they do require some tuning to do their best, depending on the dataset. We explored as best as we could their hyperparameters, often improving over reported results in the literature

(for the same dataset and method). In particular, we tried different choices of the number of trees and maximum depth (see the tables). For RF and ET we tried 3 possible options of choosing m: $m = \sqrt{D}$, $m = D$ and $m = \lfloor D/3 \rfloor$, and picked the best one.

We ran our experiments (for all methods) in a computer with an Intel Xeon CPU E5-2699 v3 @ 2.30GHz and 128 GB RAM. We did not use any GPUs.

TAO forests We use oblique decision trees (having a hyperplane function at each decision node) with constant leaves (TAO-c) or linear leaves (TAO-l). We take as initial tree a complete binary tree of given depth (Δ in the tables) with random parameters at each node (each node's weight vector has Gaussian (0,1) entries, and then we normalize the vector to unit length). Unless otherwise indicated, we train each TAO tree on a 90% random sample of the training data using 40 iterations and a small sparsity penalty of $\alpha = 0.01$. (It is possible to tune α individually for each dataset: decreasing α a bit reduces the error but increases the number of parameters; we found $\alpha = 0.01$ worked well enough with all datasets.) To set the depth Δ and number of trees T, we often followed the following simple procedure: we try first a large value of Δ and T, to get an estimate of best error, and then try to reduce both without hurting the error. More generally, one could use cross-validation over Δ and T. We report the mean error (training and test) and standard deviation over 5 independent runs.

We implemented TAO in Python 3.7.3 with process level parallel processing. We approximate the weighted 0/1 loss in the decision node optimization with a weighted surrogate loss (ℓ_1-regularized weighted logistic regression). For TAO-l, the leaf optimization involves an ℓ_1-regularized linear regression (Lasso). We implement the logistic regression using LIBLINEAR [8] and the linear regression using coordinate descent, both of which are available inside scikit-learn [16].

5.3 Forest size: number of parameters and FLOPS

For each method's forest, we report its total number of parameters and (estimated) FLOPS for inference:

Total number of parameters We count the parameters for each node of each tree (decision nodes and leaves). In a decision node we count the number of nonzero weights (and the bias), which is 2 for an axis-aligned tree and $D+1$ for an oblique tree (where D is the number of features). In a leaf, we count K for constant-label leaves and DK for a linear regressor.

Inference FLOPS We take the inference time (FLOPS) for one instance along one tree as the number of nonzero parameters it encounters in the root-leaf path it follows. We repeat this for each tree in the forest and average over all training instances.

For the forests created by some algorithms, we do not have access to the actual forest, in which case we report an upper bound and mark it with parentheses in the tables. This is computed as follows:

Total number of parameters For each tree of depth Δ (but not necessarily complete), the maximum number of decision nodes and leaves is $\max(N, 2^\Delta) - 1$ and $\max(N, 2^\Delta)$, respectively, where N is the number of training points. We then count the number of parameters as above and multiply it times the number of trees.

Inference FLOPS We assume each root-leaf path to have depth Δ.

References

[1] E. Agustsson, R. Timofte, S. Escalera, X. Baro, I. Guyon, and R. Rothe. Apparent and real age estimation in still images with deep residual regressors on APPA-REAL database. In *Proc. 12th IEEE Int. Conf. Automatic Face & Gesture Recognition (FG 2017)*, pages 87–94, 2017.

[2] J.-M. Begon, A. Joly, and P. Geurts. Globally induced forest: A prepruning compression scheme. In D. Precup and Y. W. Teh, editors, *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, pages 420–428, Sydney, Australia, Aug. 6–11 2017.

[3] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere. The Million Song Dataset. In *Proc. 12th Int. Society for Music Information Retrieval Conference (ISMIR 2011)*, pages 591–596, Miami, FL, Oct. 24–28 2011.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, Oct. 2001.

[5] M. Á. Carreira-Perpiñán. The Tree Alternating Optimization (TAO) algorithm: A new way to learn decision trees and tree-based models. arXiv, 2020.

[6] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proc. of the 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2016)*, pages 785–794, San Francisco, CA, Aug. 13–17 2016.

[7] M. Denil, D. Matheson, and N. de Freitas. Narrowing the gap: Random forests in theory and in practice. In E. P. Xing and T. Jebara, editors, *Proc. of the 31st Int. Conf. Machine Learning (ICML 2014)*, pages 665–673, Beijing, China, June 21–26 2014.

[8] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *J. Machine Learning Research*, 9:1871–1874, Aug. 2008.

[9] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29 (5):1189–1232, 2001.

[10] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine Learning*, 63(1):3–42, Apr. 2006.

[11] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, CA, May 7–9 2015.

[12] A. H. Li and A. Martin. Forest-type regression with general losses and robust forest. In D. Precup and Y. W. Teh, editors, *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, pages 2091–2100, Sydney, Australia, Aug. 6–11 2017.

[13] M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

[14] M. Mathias, R. Benenson, M. Pedersoli, and L. Van Gool. Face detection without bells and whistles. In *Proc. 13th European Conf. Computer Vision (ECCV'14)*, pages 720–735, Zürich, Switzerland, Sept. 6–12 2014.

[15] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NIPS Workshop on The Future of Gradient-Based Machine Learning Software (Autodiff)*, 2017.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and É. Duchesnay. Scikit-learn: Machine learning in Python. *J. Machine Learning Research*, 12:2825–2830, Oct. 2011. Available online at https://scikit-learn.org.

[17] S. Ren, X. Cao, Y. Wei, and J. Sun. Global refinement of random forest. In *Proc. of the 2015 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'15)*, pages 723–730, Boston, MA, June 7–12 2015.

[18] R. Rothe, R. Timofte, and L. V. Gool. Deep expectation of real and apparent age from a single image without facial landmarks. *Int. J. Computer Vision*, 126:144–157, 2016.

[19] R. E. Schapire and Y. Freund. *Boosting. Foundations and Algorithms.* Adaptive Computation and Machine Learning Series. MIT Press, 2012.

[20] S. Schulter, C. Leistner, P. Wohlhart, P. M. Roth, and H. Bischof. Alternating regression forests for object detection and pose estimation. In *Proc. 14th Int. Conf. Computer Vision (ICCV'13)*, pages 417–424, Sydney, Australia, Dec. 1–8 2013.

[21] R. Tanno, K. Arulkumaran, D. C. Alexander, A. Criminisi, and A. Nori. Adaptive neural trees. In K. Chaudhuri and R. Salakhutdinov, editors, *Proc. of the 36th Int. Conf. Machine Learning (ICML 2019)*, pages 6166–6175, Long Beach, CA, June 9–15 2019.

[22] S. Vijayakumar, A. D'Souza, T. Shibata, J. Conradt, and S. Schaal. Statistical learning for humanoid robots. *Autonomous Robots*, 12(1):55–69, Jan. 2002.

[23] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proc. of the 2017 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'17)*, pages 1492–1500, Honolulu, HI, July 21–26 2017.

[24] J. Zhu, H. Zou, S. Rosset, and T. Hastie. Multi-class AdaBoost. *Statistics and Its Interface*, 2(3): 349–360, 2009.