Smaller, More Accurate Regression Forests Using Tree Alternating Optimization

Arman Zharmagambetov [1] **Miguel Á. Carreira-Perpiñán** [1]

Abstract

Regression forests, based on ensemble approaches such as bagging or boosting, have long been recognized as the leading off-the-shelf method for regression. However, forests rely on a greedy top-down procedure such as CART to learn each tree. We extend a recent algorithm for learning classification trees, *Tree Alternating Optimization (TAO)*, to the regression case, and use it with bagging to construct regression forests of oblique trees, having hyperplane splits at the decision nodes. In a wide range of datasets, we show that the resulting forests exceed the accuracy of state-of-the-art algorithms such as random forests, AdaBoost or gradient boosting, often considerably, while yielding forests that have usually fewer and shallower trees and hence fewer parameters and faster inference overall. This result has an immense practical impact and advocates for the power of optimization in ensemble learning.

1. Introduction

We consider regression, the problem of learning a function that predicts a continuous scalar or vector from an input feature vector. Forests (ensembles of trees) are widely recognized as the leading off-the-shelf regression methods. In many datasets, we can expect a forest to achieve close to optimal performance compared to other methods with little hyperparameter tuning. This has made them very widely applied in data mining (Verikas et al., 2011), computer vision (Criminisi et al., 2012; Criminisi & Shotton, 2013; Viola & Jones, 2004) and other areas.

However, why forests work so well is not completely understood. They are constructed, at least in part, using heuristic techniques, without directly optimizing an objective function—unlike most other machine learning models, which define an objective over the model parameters and optimize it, usually with gradient-based methods. In particular, forests need a way to train individual trees, and in the most successful forests, such as random forests (Breiman, 2001) or gradient boosting (Friedman, 2001), this is done using the CART algorithm (Breiman et al., 1984) or a variation of it such as C4.5 (Quinlan, 1993). In these algorithms, one starts with a root node, splits it using a heuristic criterion and proceeds recursively with its children, top-down, stopping at some point using a variety of criteria. The algorithm is greedy in that after each split the node's decision function is fixed. This and the fact that the algorithm does not optimize an objective function directly over the tree parameters makes CART produce severely suboptimal trees (Hastie et al., 2009).

Given this state of affairs, it is remarkable that CART and C4.5 have remained the state-of-the-art for decades for training both single trees and forests. The reason is that tree optimization is a very difficult problem, NP-hard in most formulations (Hyafil & Rivest, 1976; Hancock et al., 1996), to which gradient methods do not apply. Thus, while many other tree optimization algorithms have been proposed, none has been able to do significantly better. Also, the vast majority of trees in practice, whether single or ensembled, are axis-aligned (where each split tests a single feature rather than a combination of features) and each leaf outputs a constant prediction.

We propose to improve regression forests by improving the optimization of each regression tree, and by allowing the use of more general types of trees. To do this, we build on a recently proposed algorithm for learning classification trees, *Tree Alternating Optimization (TAO)* (Carreira-Perpiñán & Tavallali, 2018; Carreira-Perpiñán, 2020). TAO optimizes a parametric tree of fixed structure, monotonically decreasing its 0/1 classification loss at each iteration. It applies to trees beyond axis-aligned splits (such as oblique splits) and constant-predictor leaves (such as linear), is scalable, and produces far more accurate classification trees than CART. We adapt TAO to the regression case

[1] Dept. of Computer Science & Engineering, University of California, Merced, USA. Correspondence to: Arman Zharmagambetov <azharmagambetov@ucmerced.edu>, Miguel Á. Carreira-Perpiñán <mcarreira-perpinan@ucmerced.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020. Copyright 2020 by the author(s).

and then use it in combination with bagging to learn forests of oblique trees. The resulting forests show remarkable performance: they beat state-of-the-art algorithms such as random forests, gradient boosting or AdaBoost in accuracy, consistently over all datasets and by a considerable margin, while usually having fewer parameters and faster inference.

2. Related Work

Regression forests is a well-established area and various books provide extensive reviews (Zhou, 2012; Kuncheva, 2014; Hastie et al., 2009). The prediction of the forest is usually the average of the individual tree predictions. Learning a regression forest is generally done based on a procedure for training individual trees and a procedure for ensembling different trees. We review both of these.

The most widespread way to learn individual trees is by a greedy top-down induction method such as CART (Breiman et al., 1984), C4.5 (Quinlan, 1993) and variations thereof. Splitting a node is achieved by approximately optimizing a variance-based criterion that is indirectly related to the loss of the overall tree. This criterion depends in a discrete way on which instances go to each child. Optimizing it is easy for axis-aligned (univariate) splits by enumerating all possible features, but it is difficult for oblique (hyperplane) splits. In practice with both single trees and forests, axis-aligned trees where each leaf predicts a constant output are the norm. Also, while for single trees the size of the tree must be carefully determined to avoid overfitting (by early stopping or post-pruning), in a forest trees are usually grown fully, until each leaf contains one (or a few) instances. Finally, some forest methods use random splits rather than a variance-based criterion, such as Extra-Trees (Geurts et al., 2006); while other forests use very simple trees, such as stumps (depth-1 trees), for which a specialized learning algorithm exists.

The main approaches for ensembling different trees are based on bagging (Breiman, 1996), where individual trees are trained independently on boostrap samples of the data; or on boosting (Schapire, 1990; Schapire & Freund, 2012), where individual trees are trained sequentially on the whole data but with adaptively weighted instances. Many variations of these exist. Random Forests (Breiman, 2001; Biau & Scornet, 2016) combine bagging with choosing random feature subsets at each node when considering candidate splits. Extremely randomized trees (Extra-Trees) (Geurts et al., 2006) combine bagging with random splits. AdaBoost (Freund & Schapire, 1997; Schapire & Freund, 2012) is probably the most practical of all boosting algorithms, although gradient boosting (Friedman, 2001) has attracted much attention in recent years, particularly with the availability of efficient implementations that can scale to large datasets (Chen & Guestrin, 2016; Ke et al., 2017). Other variations include random subspace forests (Ho, 1998) and rotation forests (Rodríguez et al., 2006). After several decades where regression forests have been actively researched, it is fair to say that the methods that at present are recognized as the state-of-the-art are Random Forests, AdaBoost and gradient boosting.

Other methods exist which combine the individual tree learning with the ensembling or which use more complex node models (Schulter et al., 2013; Tanno et al., 2019; Begon et al., 2017). The training algorithm in such methods is quite more complicated and it is not clear that it translates into more accurate or compact forests, although they may be preferable in particular applications. Most regression forests use the squared error to fit the leaves, but it is also possible to use robust losses (Li & Martin, 2017). A few works propose forests of oblique trees (Breiman, 2001; Frank & Kramer, 2004; Menze et al., 2000; Zhang et al., 2017), but they mostly focus on classification rather than regression and improve marginally over axis-aligned trees.

Finally, some techniques exist to take an existing forest and postprocess it. Pruning a forest by removing redundant trees can be done greedily with forward selection (Zhou, 2012). This can often reduce the size of a forest with little degradation of its accuracy. Also, it is possible to optimize jointly the constant predictors at the leaves of all trees (keeping the decision nodes and tree structure the same) and increase the accuracy a bit (Ren et al., 2015). These techniques are complementary to our work, as they can be applied to any existing regression forest with constant-predictor leaves.

3. Tree Alternating Optimization for Regression[1]

3.1. Definition of the Tree 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

[1] A more general version of the Tree Alternating Optimization (TAO) algorithm is given by Carreira-Perpiñán (2020).

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.

Our TAO algorithm for regression is based on 3 theorems. We give them next and explain them afterwards; the proof is given in the suppl. mat. and in Carreira-Perpiñán (2020). 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 3.1 (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.

This follows from the fact that the reduced sets of nodes i and j are disjoint, because the tree makes hard decisions.

Theorem 3.2 (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:*

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

where $\mathcal{R}_i$ is the reduced set of node i, and we define the function $l_{in}\colon\ \mathcal{C}_i \rightarrow \mathbb{R}$ 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 $\overline{L}_{in}(\overline{y}_{in}, \cdot)\colon\ \mathcal{C}_i \rightarrow \mathbb{R}^+ \cup \{0\}$ 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).

This follows from the fact that all a decision node can do with an instance is send it down its left or right child, and the ideal choice is the one that results in the best prediction downstream from that node.

Theorem 3.3 (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:*

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

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

Theorem 3.1 says that $E(\boldsymbol{\Theta})$ becomes an additively separable function of the non-descendant nodes' parameters given the parameters of all other nodes. The specific form of the resulting function for each node is given by the next theorems. For a decision node, the function $l_{in}(z) = L(\mathbf{y}_n, \mathbf{T}_z(\mathbf{x}_n; \boldsymbol{\Theta}_z))$ maps a child $z \in \mathcal{C}_i$ of node i to the value of the loss L incurred by instance $\mathbf{x}_n$ in the leaf that it reaches when propagated down z's subtree. The *reduced problem* of eq. (4) is a *weighted 0/1 loss binary classification problem* with binary *pseudolabels* $\{\overline{y}_{in}\}$, defined as the child of i that gives the best prediction for $\mathbf{x}_n$ under the current tree; see section 3.3. For a leaf, the reduced problem is simple: fit the leaf's predictor to the instances (and ground-truth labels) in its reduced set to minimize the original loss. For constant predictors, this gives the reduced set mean; for linear ones, we solve an ℓ_1-penalized linear regression (Lasso; Hastie et al., 2015).

The separability condition is similar to that of TAO for classification (Carreira-Perpiñán & Tavallali, 2018), but the reduced problems are different: for a decision node it is a weighted 0/1 binary classification, not unweighted; and for a leaf it is a regression rather than a classification.

3.2. TAO Algorithm for Regression Trees

Although the previous derivations appear complicated, the algorithm works in a simple way in the end (see algorithm 1): we repeatedly update nodes' parameters until convergence. We can update in parallel nodes that are not descendants of each other (from the separability condition). In this paper, we do this in breadth-first search (BFS) order, i.e., we update in parallel all nodes at the same depth; one pass over the entire tree defines a TAO iteration. Updating a node requires solving its reduced problem via a binary classifier (decision node) or regressor (leaf). After each iteration, the objective (1) decreases or stays unchanged. We stop iterating when eq. (1) changes little or we reach a set number of iterations.

Finally, we can remove dead subtrees from the tree, i.e., nodes receiving no training instances, as done in Carreira-Perpiñán & Tavallali (2018). This is particularly likely to happen with an ℓ_1 regularizer on the decision nodes or leaves, which encourages weights to become zero (a decision node with $\mathbf{w}_i = \mathbf{0}$ creates a dead subtree).

We emphasize that, unlike in traditional tree induction algorithms such as CART or C4.5, with TAO we do not grow a tree greedily. Instead, we assume a parametric tree with a given structure, just as when we train a neural net we choose an architecture and then optimize its parameters. That said, the final tree structure can be smaller than the initial one, similarly to pruning a deep net.[2]

3.3. Solving the Decision Node Optimization Problem

Theorem 3.2 shows that the reduced problem for a decision node is a weighted 0/1 loss binary classification problem on the node's reduced set instances. It is binary because the output of the decision function must be either the left or the right child. It is weighted because the loss of the best child is different for each instance. Optimizing the (unweighted) 0/1 loss over a hyperplane and other variants of it is an NP-hard problem (Hoffgen et al., 1995; Pitt & Valiant, 1988; Megiddo, 1988; Ben-David et al., 2003) (except for axis-aligned hyperplanes, which can be solved exactly and quickly by enumeration over the features). However, good approximate solutions can be obtained efficiently by using a convex surrogate loss. We use the logistic loss and an ℓ_1 regularizer, hence we train an ℓ_1-penalized logistic regression (for which well-developed code exists, such as LIBLINEAR; Fan et al., 2008). This must be adapted to handle the weights in the 0/1 loss, and we explored several approaches (such as resampling or repeating instances; see suppl. mat.). We found it best to use the weights directly as multipliers in the logistic loss.

Algorithm 1 TAO regression tree algorithm (BFS order)

```
input: training set; initial tree $\mathbf{T}(\cdot;\boldsymbol{\Theta})$ of depth $\Delta$
$\mathcal{N}_0,\dots,\mathcal{N}_\Delta \leftarrow$ nodes at depth $0,\dots,\Delta$, respectively
$\mathcal{R}_1 \leftarrow \{1,\dots,N\}$
repeat
  for $d = 0$ to $\Delta$ do
    parfor $i \in \mathcal{N}_d$ do
      if $i$ is a leaf then
        $\boldsymbol{\theta}_i \leftarrow$ train regressor $\mathbf{g}_i$ on reduced set $\mathcal{R}_i$
      else
        $\boldsymbol{\theta}_i \leftarrow$ train decision function $f_i$ on $\mathcal{R}_i$
        compute the reduced sets of each child of $i$
      end if
    end parfor
  end for
until stop
prune dead subtrees of $\mathbf{T}$
return $\mathbf{T}$
```

Note: to compute the reduced set of each of node i's children we pass each instance $\mathbf{x}_n$ in i's reduced set $\mathcal{R}_i$ through i's decision function f_i, and add $\mathbf{x}_n$ to the resulting child's reduced set.

It is possible that the approximate solution has a (slightly) higher value of objective (4) than the current parameters $\boldsymbol{\theta}_i$. This does occasionally happen, usually near convergence of TAO. We can always ensure decrease of the reduced problem objective, hence of the overall tree objective (1), by rejecting the update if it is worse and leaving that node unchanged. However, in practice we find it is better to accept the update always, since the increase in the objective is small and it evens out as one keeps iterating.

3.4. Computational Complexity

The computational complexity of training one TAO regression tree is as follows. For TAO-c (constant leaves), the complexity of one TAO iteration (pass over all nodes) is upper bounded by the tree depth times the cost of solving a logistic regression on the whole training set. This can be estimated as in Carreira-Perpiñán & Tavallali (2018) (who considered classification trees). Consider all the nodes i at a given depth. Each node solves a reduced problem on a subset $\mathcal{C}_i$ of the training set, of size N_i. The subsets $\mathcal{C}_i$ from these nodes are disjoint so their aggregated size is at most N. Each node solves a logistic regression on its subset $\mathcal{C}_i$ in time $\mathcal{O}(DN^\alpha)$ for $\alpha \geq 1$ (which is similar to the time it takes to solve an SVM, although exactly what this is depends on the SVM solver; Bottou & Lin, 2007, section 4.2). Hence, since $\sum_i N_i^\alpha \leq \left(\sum_i N_i\right)^\alpha \leq N^\alpha$ if $\alpha \geq 1$,

[2] Indeed, by fixing the tree structure and making it a parametric model, regularization via an ℓ_1 norm promotes sparsification and pruning in a similar way to a Lasso (Hastie et al., 2015) or to neural net pruning (Carreira-Perpiñán & Idelbayev, 2018).

solving all the logistic regressions at the same depth is at most as costly as solving a single logistic regression on all N points. The overhead of propagating points through the tree to determine the subsets is negligible compared to this.

For TAO-l (linear leaves), the cost is slightly larger because each leaf solves a linear regression rather than computing a majority vote (as in classification trees). However, this cost is again of the same order as solving for the decision nodes, so the overall computational complexity is the same.

4. Regression Forests of Bagged TAO Trees

There are many ways to ensemble individual learners, such as bagging, boosting, etc. In this paper we choose a simple one: we train each TAO tree independently on a random subset of M samples of the available training data (N instances). A particular, simple case of this is bagging, where the subset is a bootstrap sample ($M = N$ but sampled with replacement; Breiman, 1996). As we will see, the best accuracy is obtained by tuning M. Bagging may sometimes be preferable, as it strikes a good tradeoff between accuracy (often close to that of the best M) and simplicity (it does not need to tune M). We initialize each TAO tree from a complete tree of depth Δ and random node parameters (each node's weight vector has Gaussian (0,1) entries, and then we normalize the vector to unit length). The trees can be trained in parallel, just as with random forests. The forest prediction is the average of its trees' predictions.

Although our TAO regression algorithm works with axis-aligned trees, in this paper we only use oblique trees. These are more powerful, since they can better model correlations between features, and indeed the resulting forests achieve much higher accuracy. Regarding the leaf predictors, we show experiments using constant- and linear-predictor leaves. However, the latter are the clear winner in both accuracy and forest size.

We train each tree with an ℓ_1 regularizer but set its hyperparameter α to a small value (0.01). This has the effect of applying sparsity (both within the decision node and leaf weights, and in pruning the tree) only if it does not hurt the accuracy appreciably. So the only important hyperparameters of the forest are the depth Δ and number of trees T.

5. Experiments

The goal of this section is to provide convincing evidence that *our TAO regression forests consistently dominate many competing forest algorithms in accuracy, often by a large margin, while using fewer parameters and faster inference time*. This is true with practically no exceptions across a range of benchmarks of varying type, sample size and input or output dimensionality. Notably, the accuracy of *a single TAO regression tree* far beats that of a CART tree (in agreement with the comparison reported by Zharmagambetov et al. (2020) for classification and regression), and is often comparable to that of other forests, which demonstrates the better optimization done by TAO. We start with comparison results on benchmarks and MNIST (sections 5.2–5.3) as well as training time (section 5.4). Then, we analyze the impact of different diversity mechanisms and hyperparameters (section 5.5), which helps us to determine how best to construct a TAO regression forest. Throughout, TAO-c and TAO-l stand for TAO regression forests of oblique trees with constant- and linear-predictor leaves, respectively. The suppl. mat. has additional results.

5.1. Experiment Settings

We compare TAO with the state-of-the-art tree ensembling algorithms: Random Forests (RF) (Breiman, 2001), Extra-Trees (ET) (Geurts et al., 2006), AdaBoost (Freund & Schapire, 1997) (all using the Python scikit-learn implementation; Pedregosa et al., 2011); and gradient boosting (Friedman, 2001) (using the highly optimized XGBoost implementation; Chen & Guestrin, 2016). We explored as best as we could their hyperparameters, often improving over reported results in the literature (for the same dataset and method, e.g. for RF in several datasets in Schulter et al., 2013). In particular, we tried different choices of the number of trees T and maximum depth Δ. We do not restrict the `max_depth` hyperparameter for RF and ET, and allow each tree to grow fully, as is recommended for random forests (Breiman, 2001). But we do tune this hyperparameter for XGBoost and AdaBoost. We also compare with published results of some recent forest algorithms: Alternating Regression Forests (ARF) (Schulter et al., 2013), Adaptive Neural Trees (ANT) (Tanno et al., 2019), Globally Induced Forest (GIF) (Begon et al., 2017), Consistent Random Forest (cRF) (Denil et al., 2014), Refined Random Forest (rRF) (Ren et al., 2015) and Robust Forests (Li & Martin, 2017). Finally, we also give the result of training a single CART tree (Breiman et al., 1984) for reference. As for TAO, we train each tree on a 90% random sample of the training data using 40 iterations. We implemented TAO in Python. All reported errors are root mean squared error (RMSE) $E = \sqrt{\frac{1}{NK} \sum_{n=1}^{N} \|\mathbf{y}_n - \hat{\mathbf{y}}_n\|^2}$ unless otherwise specified, where N is sample size, K is output dimension, and $\mathbf{y}$ and $\hat{\mathbf{y}}$ are the ground truth and predicted vectors, respectively. See the suppl. mat. for detailed information on the datasets, comparison methods and hyperparameters.

5.2. Standard Machine Learning Benchmarks

Tables 1–2 report the results on several regression benchmarks of sample size N, input dimension D and output dimension K. The output dimension K is quite low, ranging from 1 to 7, as is common in many regression papers.

Table 1. Performance comparison of different regression forest methods, sorted by decreasing test error E_{test}. Datasets: abalone, ailerons, cpuact, CT slice; for each, we give (N, D, K) = sample size and input and output dimensionality. We report the test RMSE (avg±stdev over 5 repeats), number of parameters and inference FLOPS (numbers in parentheses are estimates), number of trees T and maximum depth of the forest Δ. TAO forests are in boldface (TAO-c: constant leaves; TAO-l: linear leaves).

	Forest	E_{test}	#pars.	FLOPS	T	Δ
abalone (4k,8,1)	CART	3.01±0.01	2 891	20	1	20
	XGBoost	2.22±0.00	31k	(1 089)	100	10
	XGBoost	2.20±0.00	220k	(9 349)	1k	10
	GIF	2.18	(50k)	–	10	–
	TAO-c	2.18±0.05	287	41	1	6
	AdaBoost	2.16±0.01	53k	(1k)	100	10
	AdaBoost	2.15±0.00	0.5M	(10k)	1k	10
	ET	2.14±0.00	443k	(3 011)	100	36
	RF	2.12±0.01	230k	(2 473)	100	29
	rRF	2.10±0.01	(100k)	(1 000)	100	10
	ARF	2.10±0.03	(100k)	(1 000)	100	10
	RF	2.10±0.00	2M	(25k)	1k	34
	TAO-c	2.08±0.01	9k	1 307	30	6
	TAO-l	2.07±0.01	303	40	1	5
	TAO-c	2.05±0.01	33k	1 718	30	8
	TAO-l	2.04±0.01	8k	1 204	30	5
ailerons ($E \times 10^{-4}$,14k,40,1)	CART	2.88±0.00	103	9	1	9
	RF	1.84±0.02	875k	(3 512)	100	45
	ET	1.84±0.00	1.4M	(4 068)	100	49
	ARF	1.78±0.01	(36k)	(750)	50	15
	AdaBoost	1.77±0.01	18k	(1 175)	100	15
	TAO-c	1.76±0.02	681	87	1	6
	rRF	1.75±0.02	(71k)	(1 000)	100	10
	RF	1.75±0.00	9M	(35k)	1k	47
	AdaBoost	1.75±0.00	200k	(12k)	1k	15
	XGBoost	1.74±0.00	2k	(300)	100	7
	TAO-l	1.74±0.01	447	93	1	5
	XGBoost	1.72±0.00	4k	(1 264)	1k	7
	TAO-c	1.67±0.04	21k	2 513	30	6
	TAO-l	1.66±0.04	27k	2 611	30	5
cpuact (8k,21,1)	CART	3.63±0.32	9 691	25	1	25
	TAO-c	2.71±0.04	498	51	1	6
	RF	2.62±0.04	0.6M	(2 842)	100	36
	ARF	2.62±0.01	(98k)	750	50	15
	AdaBoost	2.61±0.16	72k	(1k)	100	10
	RF	2.60±0.01	6M	(28k)	1k	37
	XGBoost	2.60±0.00	40k	(1 000)	100	10
	ET	2.58±0.03	1M	(3 733)	100	45
	TAO-l	2.58±0.02	246	41	1	5
	XGBoost	2.57±0.00	294k	(8 780)	1k	10
	AdaBoost	2.56±0.11	0.7M	(10k)	1k	10
	ET	2.49±0.03	10M	(38k)	1k	50
	TAO-c	2.39±0.05	24k	1 590	30	7
	TAO-l	2.35±0.01	8k	1 179	30	5
CT slice (54k, 384,1)	CART	2.71±0.06	85k	51	1	51
	TAO-c	1.54±0.05	7k	1 123	1	7
	AdaBoost	1.48±0.03	122k	(1 000)	100	10
	XGBoost	1.45±0.00	71k	(1 000)	100	10
	AdaBoost	1.31±0.01	1M	(10k)	1k	10
	XGBoost	1.18±0.00	465k	(10k)	1k	10
	TAO-l	1.16±0.02	5k	768	1	5
	ET	1.06±0.01	85M	(62k)	100	82
	RF	1.03±0.01	5M	(5 818)	100	71
	cRF	1.00	(17M)	–	1k	–
	RF	0.97±0.01	54M	(57k)	1k	78
	TAO-c	0.89±0.02	214k	31k	30	7
	TAO-l	0.71±0.02	165k	23k	30	5
	TAO-l	0.58±0.03	242k	25k	30	6

Table 2. Like table 1 but for YearPredictionMSD, SARCOS.

	Forest	E_{test}	#pars.	FLOPS	T	Δ
YearPredictMSD (515k,90,1)	CART	13.41±0.11	621k	49	1	49
	RF	9.31±0.00	40M	(5 237)	100	68
	ET	9.31±0.00	77M	(6 091)	100	73
	AdaBoost	9.25±0.01	2.5M	(1 500)	100	15
	RF	9.23±0.00	401M	(52k)	1k	73
	AdaBoost	9.21±0.03	24M	(15k)	1k	15
	TAO-c	9.11±0.05	7k	448	1	8
	TAO-l	9.08±0.03	2k	388	1	6
	XGBoost	9.04±0.00	103k	(1 000)	100	10
	XGBoost	9.01±0.00	1.1M	(10k)	1k	10
	cRF	8.90	(184M)	–	1000	–
	TAO-c	8.90±0.01	186k	13k	30	7
	TAO-l	8.87±0.01	73k	12k	30	6
	TAO-c	8.85±0.01	246k	14k	30	9
	TAO-l	8.83±0.01	148k	12k	30	7
SARCOS (49k,21,7)	CART	3.62±0.00	84k	23	1	21
	TAO-c	2.44±0.04	71k	202	1	14
	RF	1.56±0.01	5.1M	(2 997)	100	30
	RF	1.54±0.01	51M	(30k)	1k	30
	AdaBoost	1.40±0.01	1.2M	(7k)	700	10
	TAO-c	1.36±0.03	2M	5 796	30	14
	AdaBoost	1.33±0.03	12M	(70k)	7k	10
	TAO-c	1.30±0.03	6.5M	10k	50	15
	XGBoost	1.24±0.00	774k	(7k)	700	10
	ANT	1.18	104k	62k	1	–
	ANT	1.11	598k	361k	8	–
	XGBoost	1.10±0.00	4M	(70k)	7k	10
	TAO-l	1.04±0.02	18k	151	1	10
	TAO-l	0.85±0.02	694k	4 833	30	10

For ET, RF, AdaBoost and XGBoost (which we ran ourselves), we obtain similar results in terms of accuracy as those reported in previous works (e.g. Schulter et al., 2013, Ren et al., 2015, etc.). In general, XGBoost and particularly AdaBoost take much longer to train. They also generate forests with many more trees if the output is high-dimensional (K times more trees if there are K outputs). ETs and RFs are simplest to use in terms of hyperparameter choice and have extremely high training speed.

Let us look at TAO closely (boldfaced lines in the tables). Firstly, a single TAO tree ($T = 1$) already shows very good accuracy, while naturally being very small in size and inference. A single TAO-l tree beats all other forest methods for abalone and SARCOS and is comparable to the best in the other datasets. A single TAO-c tree has consistently lower accuracy than TAO-l, although it is still comparable to other methods in some datasets. This demonstrates two important things. First, and as expected, that oblique trees with linear-predictor leaves are a better model than axis-aligned trees (which are ill-suited for correlated, high-dimensional input features) or than oblique trees with constant-predictor leaves (which represent a piecewise constant regression function and is ill-suited for continuously-varying outputs). Second, that our TAO regression algorithm is able to find good optima of such trees.

More importantly, and the main focus of our paper, let us look at the TAO forests ($T > 1$). *TAO-l forests, followed by TAO-c forests, have the lowest test error in all datasets,* *often by a considerable margin*. The TAO-l error can be close to half the next best error (of much larger forests) in the CT slice and SARCOS datasets. Also, the standard deviation of the error shown in the tables is very small (even for single trees), which makes the algorithm robust.

In terms of model size, state-of-the-art forest methods are based on axis-aligned (univariate) trees, where each decision node thresholds a single feature, having constant-predictor leaves. Therefore, both decision nodes and leaves are lightweight: 2 parameters for decision nodes (feature index and threshold) and K for leaves (output vector). Whereas we use oblique trees in our TAO forests, where each decision node uses up to $D+1$ parameters (hyperplane weights and bias), and each leaf uses either K or up to $K(D+1)$ parameters (constant output or linear predictor). While TAO trees are potentially much heavier, the forest ends up being smaller and faster, for two reasons. First, each TAO tree is trained with an ℓ_1 penalty that (even with a small hyperparameter α) sparsifies the tree. This happens in terms of the nonzero weights in both the decision and linear leaf nodes, and in the number of nodes in the tree (since some nodes are pruned). The ratio of the number of parameters between a trained TAO tree and its initial (dense) tree is quite small, varying from 3% for MNIST, 22% for CT slice, 35% for cpuact and 83% for abalone. And second, the resulting forests have fewer trees (up to $T = 30$ in all datasets) that are shallower (up to $\Delta = 5$–10 for TAO-l). This contrasts with RFs or gradient boosting, which use many more trees (often over thousands) that are much deeper. For example, in the abalone and cpuact datasets, a 100-tree RF has 1 to 2 orders of magnitude more parameters than the best-accuracy TAO-l forest.

5.3. High-dimensional Regression: MNIST Digit Rotation

Most works on regression focus on problems where the output vector is of low dimension, very often 1. We explore a more challenging, high-dimensional output vector by mapping an input MNIST handwritten digit image (of dimension $D = 784 = 28 \times 28$ grayscale pixels) to another image or image patch using a synthetic, severely nonlinear transformation. We do two versions of this. In "full-image rotation" we apply a class-specific image rotation (e.g. 1s are rotated by 49°, 2s by -57°, etc.), so the output is an image of dimension $K = D$. In "patch selection and rotation", we apply a class-specific image rotation to an image patch of 8×8 of class-specific location, so the output is an image of dimension $K = 64$. Full details are in the suppl. mat.

Table 3. Like table 1 but for the MNIST regression tasks: full-image rotation (top) and patch selection and rotation (bottom).

	Forest	$E_{\text{test}} \times 10^{-2}$	#pars.	FLOPS	T	Δ
full-image (60k,784,784)	AdaBoost	>24 hours runtime			39k	25
	CART	23.08±0.12	120k	28	1	28
	TAO-c	21.10±0.37	17M	5 108	1	16
	RF	14.38±0.23	7.6M	(2 830)	100	39
	RF	14.08±0.25	68M	(28k)	1k	40
	ET	13.83±0.12	12M	(3 091)	100	35
	TAO-c	13.76±0.09	9M	42k	30	29
	ET	13.72±0.13	109M	(3 360)	1k	38
	XGBoost	10.35±0.00	180M	(613k)	39k	25
	TAO-l	9.63±0.17	288k	4 491	1	7
	TAO-l	6.59±0.11	7.7M	126k	30	7
patch (60k,784,64)	CART	28.51±0.11	119k	49	1	49
	TAO-c	21.17±0.02	1.2M	2 501	1	14
	RF	17.87±0.04	7.6M	(4 669)	100	59
	RF	17.18±0.03	71M	(42k)	1k	61
	ET	16.97±0.01	12M	(4 595)	100	57
	XGBoost	16.79±0.00	44M	(84k)	3.2k	25
	ET	16.69±0.01	114M	(41k)	1k	64
	AdaBoost	16.65±0.09	109M	(80k)	3.2k	25
	TAO-c	16.61±0.03	35M	70k	30	14
	TAO-l	16.13±0.05	65k	2 674	1	7
	TAO-l	9.91±0.03	2M	78k	30	7

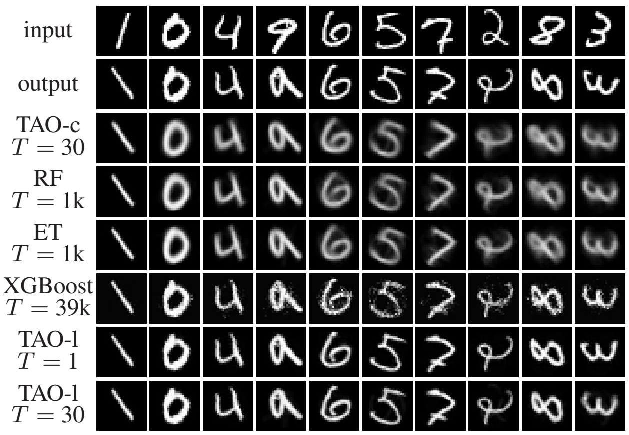

Figure 1. Selected MNIST test images, corresponding ground-truth output (class-dependent full-image rotation) and predicted output by different forest algorithms. You may want to zoom in.

Table 3 shows the results of different forest methods for both tasks, confirming our earlier results even more drastically. A single TAO-l regression tree already beats RFs and all other methods in test error (and forest size), while a TAO-l forest almost halves this error. Fig. 1 shows some input images and their ground-truth and predicted outputs, demonstrating the gain in RMSE is visually obvious too. TAO trees are also better at discarding features (input pixels) that are not useful for the prediction (see suppl. mat.).

It is possible to solve a regression with K output dimensions by concatenating K scalar regression models. This makes some methods reduce their error, but TAO-l is still the clear winner (see suppl. mat.).

5.4. Training Time

Table 4 gives representative runtimes for several datasets. For the larger datasets, TAO is a bit slower than XGBoost but much faster than AdaBoost. The suppl. mat. gives

Table 4. Training times in seconds (”) or minutes (’) for several of the datasets in tables 1–2, assuming T = 30 trees for TAO forests and T = 1k trees for the other methods. The training times for cpuact and abalone were very similar to those of ailerons.

	ailerons	CT slice	YearPredictionMSD
RF	1.4”	55”	81’
XGBoost	38”	303”	234’
AdaBoost	79”	1 976”	1 410’
TAO-c	297”	849”	316’
TAO-l	213”	691”	279’

more detailed training times for the MNIST tasks. Roughly speaking, the time increases as ET, RF (minutes) $\lll$ XGBoost (4 h) $\lesssim$ TAO (3–7h) $<$ AdaBoost ($>$ 24h). Note that, unlike XGBoost, our Python implementation is not optimized[3].

Overall, the runtime of TAO forests is reasonable and more than justified by the consistently low test error they achieve, which is sometimes surprisingly much lower (for MNIST and CT slice, the TAO forests’ error is about half the error of the closest competitor while using smaller forests).

5.5. Study of Different Diversity Mechanisms

The success of bagging-based forests is due to their ability to reduce variance by combining weakly correlated trees (Breiman, 2001; Friedman, 2001; Hastie et al., 2009), so making the trees dissimilar from each other is important. Next, we study systematically different mechanisms to diversify oblique regression forests trained with our TAO regression algorithm, as well as the forest hyperparameters. Suppl. mat. shows additional results (other datasets, etc.).

Different training samples One of the most common approaches of introducing diversity is to learn each tree on a different subset of the training set. Fig. 2 shows, for the cpuact dataset, the results of using a bootstrap sample (size $M = N$ sampled with replacement) and a random sample of size $M < N$ (where N is the total training set size). In general, bagging (bootstrap sample) performs well but the best accuracy occurs if using $M \approx 90\%$ random samples. Hence, bagging offers a simple, robust option, but somewhat better accuracy is obtained by tuning M. We expect this to be particularly true in large training sets.

Different initialization Another way of introducing diversity is to generate different random initial parameters (hyperplane weights and bias) for each tree, which will result in different local optima. Fig. 2, for the cpuact dataset, clearly shows that this produces lower test error than using the same initialization for all trees.

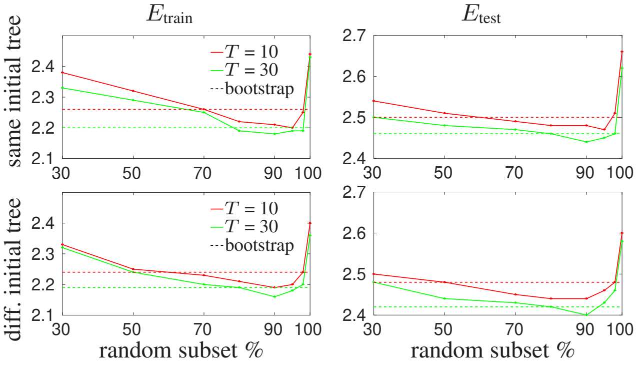

Figure 2. Diversity mechanism (cpuact dataset): training each of the T trees on a bootstrapped (dashed horizontal lines) or random data subset (solid lines, with size given by the X axis as a percentage of the total training set). All trees are complete of depth 6 and their initial parameters are random. The T trees in the forest use the same initial random tree (*top*) or each uses a different initial random tree (*bottom*).

Feature subsets Another classic diversity mechanism is to use a random subset of features at each node split (or at each tree). Random Forests benefit hugely from this; it is recommended to use $m = \lfloor D/3 \rfloor$ features in regression RFs, where D is the total number of features (Hastie et al., 2009, sec. 15.3). However, as table 5 clearly shows for cpuact, TAO forests achieve best accuracy in both training and test if using all D features. We suspect that having a good tree optimization (certainly much better than that of CART or C4.5) means using all features results in better individual trees, and this helps the forest more than the diversity that random feature subsets introduce.

Table 5. Diversity mechanism (cpuact dataset): training each of the T trees on a different, random feature subset of size m and where D is the total number of features. “Local” means each node picks m features at random, “Global” means each tree picks m features at random, the same for each node. All trees are complete of depth 6 and their initial parameters are random.

	Size m of feature subset	$T = 10$		$T = 20$	
		E_{train}	E_{test}	E_{train}	E_{test}
	$m = D$ (all features)	2.18	2.43	2.17	2.41
Local	$m = D^{9/10}$	2.23	2.50	2.22	2.48
	$m = D^{8/10}$	2.28	2.53	2.26	2.51
	$m = D^{7/10}$	2.43	2.67	2.40	2.62
	$m = \lfloor D/3 \rfloor$	2.56	2.79	2.52	2.68
	$m = D^{6/10}$	2.62	2.84	2.57	2.71
Global	$m = D^{9/10}$	2.58	2.68	2.41	2.59
	$m = D^{8/10}$	3.24	3.41	2.97	3.03
	$m = D^{7/10}$	4.01	4.11	3.16	3.24
	$m = \lfloor D/3 \rfloor$	5.17	5.29	3.64	3.70
	$m = D^{6/10}$	6.95	7.08	4.57	4.66

[3] We now have a C implementation which is several times faster than the Python code used in this paper.

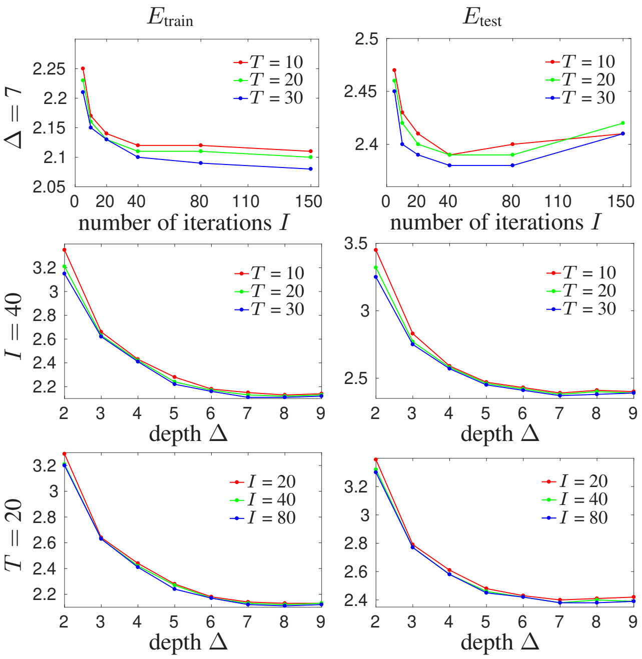

Figure 3. Training error (column 1) and test error (column 2) on the cpuact dataset for TAO forests as a function of 3 factors: tree depth Δ, number of TAO iterations I and number of trees T. Each row fixes one factor and varies the other two.

Forest hyperparameters: tree depth Δ, number of trees T and number of TAO iterations I Figure 3, for the cpuact dataset, explores several combinations of these hyperparameters. The results are quite sensible: increasing the model size (Δ or T) or the optimization amount (I) always results in lower training error, and in lower test error until a point where overfitting sets in. Overfitting is particularly noticeable if $I \geq 80$, although this is due to the small sample size of cpuact. This is also clear evidence that our TAO regression algorithm is indeed able to do a good optimization, and suggests that TAO forests make better use of the available parameters. RFs or boosting overfit only if increasing enormously the number of trees (Hastie et al., 2009, sec. 15.3.4), resulting in huge forests in practice.

Fig. 4 compares TAO, RF and XGBoost as a function of the number of trees T. It is interesting that RF and XGBoost have lower training error but higher test error than TAO. Indeed, for TAO the test error is only somewhat bigger than the training error, but for RF and XGBoost it is quite bigger, indicating an imbalance in their learning procedure.

Recommended hyperparameters Based on the previous study, we suggest the following practical construction of TAO regression forests. Each tree should always use all features and initialize parameters randomly; it can be trained on a bootstrap sample, or (with more effort but better accuracy) on a random subset of size $M < N$. Each tree should be trained for as many iterations as possible but avoiding overfitting. Most importantly, the forest size should be as big as possible (depth Δ, number of trees T) but also avoiding overfitting; practically, Δ and T could be determined by cross-validation. Also, we use a small ℓ_1 sparsity penalty $\alpha = 0.01$, which helps remove unnecessary features, weights and nodes (also reducing overfitting) while barely hurting the accuracy.

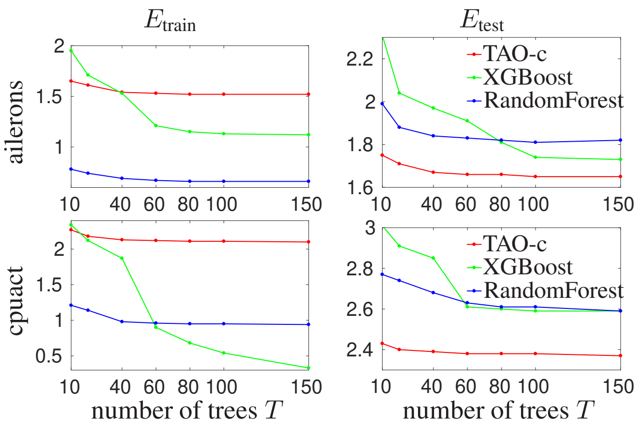

Figure 4. Training error (column 1) and test error (column 2) on the ailerons (row 1) and cpuact datasets (row 2) for TAO forests (40 iterations, tree depth 7), and XGBoost and Random Forests as a function of the number of trees T.

6. Conclusion

We have extended the Tree Alternating Optimization (TAO) algorithm to handle regression problems and used it with bootstrap (bagging) or random data samples to produce regression forests of oblique trees. In terms of accuracy, these forests outperform all competing algorithms we tested, notably random forests, AdaBoost and gradient boosting, in a range of datasets. The improvement is particularly drastic if using trees having linear leaves. In addition, these TAO forests are smaller in total number of parameters and in inference time. Their design in terms of hyperparameter tuning is as simple as with random forests or boosting: we simply need to choose a tree depth and number of trees as large as computationally possible, but without overfitting. Training time is not as lightning-fast as random forests but it is comparable to boosting, and the trees can be trained in parallel. This makes our TAO forests a model of immediate, widespread practical applicability and impact, and suggests it could replace random forests as the state-of-the-art in ensemble learning. We hope our work will encourage others to verify this in other datasets. Finally, the effectiveness of TAO as base learner is not limited to regression or bagging. Elsewhere, we report consistent improvements in accuracy with smaller forests in classification (Carreira-Perpiñán & Zharmagambetov, 2020) and with boosting (work under submission).

References

Begon, J.-M., Joly, A., and Geurts, P. Globally induced forest: A prepruning compression scheme. In Precup & Teh (2017), pp. 420–428.

Ben-David, S., Eiron, N., and Long, P. M. On the difficulty of approximately maximizing agreements. *J. Computer and System Sciences*, 66(3):496–514, May 2003.

Biau, G. and Scornet, E. A random forest guided tour. *TEST*, 25(2):197–227 (with comments, pp. 228–268), June 2016.

Bottou, L. and Lin, C.-J. Support vector machine solvers. In Bottou, L., Chapelle, O., DeCoste, D., and Weston, J. (eds.), *Large Scale Kernel Machines*, Neural Information Processing Series, pp. 1–28. MIT Press, 2007.

Breiman, L. Random forests. *Machine Learning*, 45(1): 5–32, October 2001.

Breiman, L. J. Bagging predictors. *Machine Learning*, 24 (2):123–140, August 1996.

Breiman, L. J., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees*. Wadsworth, Belmont, Calif., 1984.

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

Carreira-Perpiñán, M. Á. and Idelbayev, Y. "Learning-compression" algorithms for neural net pruning. In *Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18)*, pp. 8532–8541, Salt Lake City, UT, June 18–22 2018.

Carreira-Perpiñán, M. Á. and Tavallali, P. Alternating optimization of decision trees, with application to learning sparse oblique trees. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems (NEURIPS)*, volume 31, pp. 1211–1221. MIT Press, Cambridge, MA, 2018.

Carreira-Perpiñán, M. Á. and Zharmagambetov, A. Bagged TAO trees consistently improve over random forests and gradient boosting. In *Proc. of the 1st ACM-IMS Foundations of Data Science Conference (FODS 2020)*, Seattle, WA, October 18–20 2020.

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

Criminisi, A. and Shotton, J. *Decision Forests for Computer Vision and Medical Image Analysis*. Advances in Computer Vision and Pattern Recognition. Springer-Verlag, 2013.

Criminisi, A., Shotton, J., and Konukoglu, E. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends in Computer Graphics and Vision*, 7(2–3):81–227, 2012.

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

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

Frank, E. and Kramer, S. Ensembles of nested dichotomies for multi-class problems. In *Proc. of the 21st Int. Conf. Machine Learning (ICML'04)*, pp. 305–312, Banff, Canada, July 4–8 2004.

Freund, Y. and Schapire, R. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Computer and System Sciences*, 55(1):119–139, 1997.

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

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

Hancock, T., Jiang, T., Li, M., and Tromp, J. Lower bounds on learning decision lists and trees. *Information and Computation*, 126(2):114–122, May 1 1996.

Hastie, T., Tibshirani, R., and Wainwright, M. *Statistical Learning with Sparsity: The Lasso and Generalizations*. Monographs on Statistics and Applied Probability. Chapman & Hall/CRC, 2015.

Hastie, T. J., Tibshirani, R. J., and Friedman, J. H. *The Elements of Statistical Learning—Data Mining, Inference and Prediction*. Springer Series in Statistics. Springer-Verlag, second edition, 2009.

Ho, T. K. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 20(8):832–844, August 1998.

Hoffgen, K.-U., Simon, H. U., and Vanhorn, K. S. Robust trainability of single neurons. *J. Computer and System Sciences*, 50(1):114–125, February 1995.

Hyafil, L. and Rivest, R. L. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, May 1976.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. LightGBM: A highly efficient gradient boosting decision tree. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pp. 3146–3154. MIT Press, Cambridge, MA, 2017.

Kuncheva, L. I. *Combining Pattern Classifiers: Methods and Algorithms*. John Wiley & Sons, second edition, 2014.

Li, A. H. and Martin, A. Forest-type regression with general losses and robust forest. In Precup & Teh (2017), pp. 2091–2100.

Megiddo, N. On the complexity of polyhedral separability. *Discrete & Computational Geometry*, 3(4):325–337, December 1988.

Menze, B. H., Kelm, B. M., Splitthoff, D. N., Koethe, U., and Hamprecht, F. A. On oblique random forests. In de Mántaras, R. L. and Plaza, E. (eds.), *Proc. of the 11st European Conf. Machine Learning (ECML–00)*, pp. 453–469, Barcelona, Spain, May 31 – June 2 2000.

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

Pitt, L. and Valiant, L. G. Computational limitations on learning from examples. *Journal of the ACM*, 35(4):965–984, October 1988.

Precup, D. and Teh, Y. W. (eds.). *Proc. of the 34th Int. Conf. Machine Learning (ICML 2017)*, Sydney, Australia, August 6–11 2017.

Quinlan, J. R. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

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

Rodríguez, J. J., Kuncheva, L. I., and Alonso, C. J. Rotation forest: A new classifier ensemble method. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28 (10):1619–1630, October 2006.

Schapire, R. E. The strength of weak learnability. *Machine Learning*, 5(2):197–227, June 1990.

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

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

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

Verikas, A., Gelzinis, A., and Bacauskiene, M. Mining data with random forests: A survey and results of new tests. *Pattern Recognition*, 44(2):330–349, 2011.

Viola, P. and Jones, M. J. Robust real-time face detection. *Int. J. Computer Vision*, 57(2):137–154, May 2004.

Zhang, L., Varadarajan, J., Suganthan, P. N., Ahuja, N., and Moulin, P. Robust visual tracking using oblique random forests. In *Proc. of the 2017 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'17)*, pp. 5825–5834, Honolulu, HI, July 21–26 2017.

Zharmagambetov, A., Hada, S. S., Carreira-Perpiñán, M. Á., and Gabidolla, M. An experimental comparison of old and new decision tree algorithms. arXiv:1911.03054, March 20 2020.

Zhou, Z.-H. *Ensemble Methods: Foundations and Algorithms*. Chapman & Hall/CRC Machine Learning and Pattern Recognition Series. CRC Publishers, 2012.