FORESTPRUNE: Compact Depth-Pruned Tree Ensembles

Brian Liu
Operations Research Center
MIT

Rahul Mazumder
Sloan School of Management & Operations Research Center
MIT

Abstract

Tree ensembles are powerful models that achieve excellent predictive performances, but can grow to unwieldy sizes. These ensembles are often post-processed (pruned) to reduce memory footprint and improve interpretability. We present FORESTPRUNE, a novel optimization framework to post-process tree ensembles by pruning depth layers from individual trees. Since the number of nodes in a decision tree increases exponentially with tree depth, pruning deep trees drastically compactifies ensembles. We develop a specialized optimization algorithm to efficiently obtain high-quality solutions to problems under FORESTPRUNE. Our algorithm typically reaches good solutions in seconds for medium-size datasets and ensembles, with 10000s of rows and 100s of trees, resulting in significant speedups over existing approaches. Our experiments demonstrate that FORESTPRUNE produces parsimonious models that outperform models extracted by existing post-processing algorithms.

1 INTRODUCTION

Tree ensembles are popular in machine learning for their predictive accuracy and interpretability. These ensembles combine weak decision trees by either averaging independently grown trees in bagging (Breiman, 2001), or by adding trees grown sequentially in boosting (Friedman, 2001). The resulting model is more accurate and generalizes better than any single decision tree, but is much more complex.

The complexity of tree ensembles raises several issues. Tree ensembles can grow to massive sizes and require substantial memory to store. Computing the predictions of large ensembles is slow since each observation in a dataset must pass through each tree. Ensembles with deep trees are also difficult to interpret since the structures of deep trees are hard to visualize (Haddouchi and Berrado, 2019). Finally, complex tree ensembles can overfit, and tuning large ensembles is time-consuming. To remedy these issues, tree ensembles can be post-processed after training to more parsimonious forms. Post-processing a tree ensemble improves memory footprint, prediction speed, and interpretability without retraining the model (Sagi and Rokach, 2018).

We present FORESTPRUNE, a novel optimization framework for tree ensemble post-processing. Compared to traditional methods that post-process tree ensembles by selecting subsets of decision trees, FORESTPRUNE trims the depths of trees in an ensemble. By removing layers, decision trees can be reduced in depth or eliminated, and as a result, the framework can prune shallow and sparse subensembles that perform well. We develop a specialized block coordinate descent algorithm (Nutini et al., 2017) to obtain high-quality solutions to problems in FORESTPRUNE by minimizing regularized loss. Our specialized solver allows computation for instances that appear to be beyond the capabilities of off-the-shelf optimization solvers. Finally, we show in our experiments the advantages of trimming depth layers compared to removing trees across a range of scenarios, in terms of both improving prediction performance and interpretability. Our implementation of FORESTPRUNE is open-source and publicly available via GitHub [1].

1.1 Background and Related Work

We give a brief overview of tree-based methods for regression problems. Decision trees form the base learners used in tree ensembles. A regression tree of depth d partitions a dataset into at most 2^d non-overlapping partitions (leaf nodes). Within each leaf node, the prediction is the mean response of the observations in that partition. Decision trees are not necessarily balanced, but the number of nodes in a tree generally increases exponentially with tree depth. Bagging averages the predictions of many weakly correlated decision trees to reduce variance (Breiman, 2001). The method generally does not overfit with respect to the num-

Proceedings of the 26th International Conference on Artificial Intelligence and Statistics (AISTATS) 2023, Valencia, Spain. PMLR: Volume 206.

[1] https://github.com/brianliu12437/ForestPruneAISTATS2023

ber of features used, the number of trees grown, or the depth of each tree (Maclin and Opitz, 1997). Typically each tree is grown to full depth where each leaf node contains a single observation. Boosting sums the predictions of a sequence of decision trees, where each tree in the sequence is fit on the residuals of the prior ensemble. At each boosting iteration, the predictions of the current tree are dampened (i.e, shrunk) by the learning rate γ. This parameter controls the effect that each tree has on the ensemble as well as the similarity between trees adjacent in the boosting sequence.

We also provide an overview of existing work on post-processing tree ensembles to improve compactness. Friedman et al. (2003) introduces the importance sampled learning ensemble (ISLE) framework. This framework involves growing a tree ensemble using various sampling techniques and post-processing the ensemble in the following manner. Assign each tree t in the ensemble coefficient β_t. Incorporate an ℓ_1-penalty on the weights, $\sum_t |\beta_t|$, to encourage shrinkage and sparsity. The ensemble produced is compact and often performs comparably to the full model. ISLE regularizes the number of trees in the ensemble as a proxy for ensemble size, the framework is unable to trim individual trees in the ensemble. Friedman and Popescu (2008) extends this approach by proposing ℓ_1-regularization over the node space. While this approach is interesting, the number of nodes in a tree increases exponentially with tree depth, so the total number of nodes in a full depth ensemble is enormous. As a result, it is often infeasible to minimize ℓ_1-regularized loss over the node space. Pruning nodes also destroys the tree structures in the ensemble, which hurts model interpretability. Besides regularization, various selection heuristics can be used to decide which trees to remove from an ensemble (Lucchese et al., 2016; Martinez-Munoz et al., 2008). These heuristics select a compact subset of trees with respect to some target such as performance, diversity, or interpretability.

2 FORESTPRUNE FRAMEWORK

We detail FORESTPRUNE for regression ensembles and assume without loss of generality that all trees in the ensemble are grown to the same maximum depth. Our main goal is to introduce an optimization framework to prune depth layers from trees in a trained ensemble. Using so-called depth-difference matrices, we show that this framework can be expressed as a regularized least-squares criteria, where the combinatorial penalty controls how many layers to prune.

Notation Given dataset X with m rows and p columns, $X \in \mathbb{R}^{m\times p}$, and response $y \in \mathbb{R}^m$, tree T_i maps $T_i(X) : \mathbb{R}^{m\times p} \to \mathbb{R}^m$. Let the prediction of $T_i(X) = \hat{y}_i$. A tree ensemble is a collection of n trees $T_i,\ i \in [n]$ grown via bagging or boosting, and the prediction of the ensemble is given by $\hat{y} = \sum_{i=1}^{n} \gamma T_i(X)$. In the context of bagging, $\gamma = \frac{1}{n}$ and in the context of boosting γ is the learning rate. Let d denote the depth of trees in the ensemble.

2.1 Depth-Difference Matrix

For tree T_i, depth-difference matrix $D_i \in \mathbb{R}^{m\times d}$ encodes the decision path of each observation in X, with respect to the depth layers that the observations traverse. To initialize FORESTPRUNE compute D_i for each tree in the ensemble.

Algorithm 1 presents the procedure for constructing D_i. Consider a single decision tree T_i of depth d. For each observation $x_j \in X$, find the sequence of nodes in the tree traversed by x_j, $N_1 \to N_2 \to \ldots \to N_k$. For each node in this sequence, compute the mean of the observations found in the node, $\mu_1 \to \mu_2 \to \ldots \to \mu_k$. The last element of this sequence μ_k is the prediction for observation x_j, $T_i(x_j)$. Take the rolling difference of values in this sequence of means and store these differences in vector v_j where,

$$v_j = [\mu_1, \mu_2 - \mu_1, \mu_3 - \mu_2, \ldots, \mu_k - \mu_{k-1}].$$

Pad the tail end of v_j with zeros so that $v_j \in \mathbb{R}^d$. The vector v_j has the property that $\langle v_j, \mathbf{1}\rangle = T_i(x_j)$, i.e., the elements of v_j sum to the prediction of tree T_i for observation x_j. Repeat this procedure for $j \in [m]$ and store the results v_j as the rows of $D_i \in \mathbb{R}^{m\times d}$.

Algorithm 1: Computing Depth-Difference Matrices

Input: Tree T_i of depth d, $X \in \mathbb{R}^{m\times p}$, $y \in \mathbb{R}^m$

$D_i \leftarrow \{\}$

for $x_j \in X$ **do**

  Find sequence of nodes in T_i traversed by x_j, $N_1 \to N_2 \to \ldots \to N_k$.

  Compute the mean of y partitioned by each node, $\mu_1 \to \mu_2 \to \ldots \to \mu_k$.

  Difference the sequence of means and store in vector $v_j = [\mu_1, \mu_2 - \mu_1, \ldots, \mu_k - \mu_{k-1}]$.

  Pad the tail of v_j with zeros, $v_j \in \mathbb{R}^d$.

  Append v_j as a row to D_i.

end

Output: $D_i \in \mathbb{R}^{m\times d}$

Depth-difference matrices are efficient to compute since the sequence of nodes each observation traverses, along with the corresponding μ values of each node, are obtained as a byproduct of the tree training process. Differencing the sequence of μ values requires d operations, and since $d \ll m$, the cost of computing depth-difference matrices is linear in the number of training observations, $O(m)$.

Below, we introduce notation and functions for the vectors of binary decision variables, $z_i \in \{0,1\}^d,\ i \in [n]$, that will appear in our optimization formulation. These variables work in conjunction with depth-difference matrices to prune tree ensembles.

Pruning Depth Layers Given depth-difference matrix D_i, the predictions of the full tree for all observations, $T_i(X) = \hat{y}_i$, can be obtained by summing over all the columns of D_i, $\hat{y}_i = \text{colsum}(D_i)$. This can be equivalently expressed as $\hat{y}_i = D_i\mathbf{1}$, where $\mathbf{1}$ is a d-dimensional vector of all ones.

Consider the case where we want to prune the deepest layer of tree T_i. The predictions of this new tree, of depth $d-1$, can be obtained by summing over all but the last column of D_i. Let z_i be a vector in $\{0,1\}^d$ with $d-1$ ones followed by a single zero, $z_i = [1, 1, \ldots, 1, 0]$. The predictions of the pruned tree equal $D_i z_i$. To prune tree T_i to depth $d-k$, set $z_i \in \{0,1\}^d$ as a vector of $d-k$ ones followed by k zeros. Figure 1 visualizes this effect on a depth 4 decision tree. Note that setting $z_i = \mathbf{0}$ removes tree T_i from the ensemble.

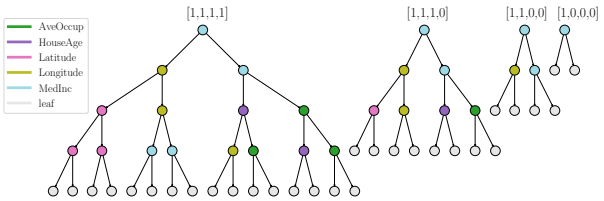


Figure 1: Pruning a decision tree with vector z_i.

Ensemble Pruning Given tree ensemble $T_i,\ i \in [n]$, compute depth-difference matrices $D_i,\ i \in [n]$ for each tree. The predictions of the original ensemble can be represented by $\hat{y} = \sum_{i=1}^{n} \gamma D_i z_i$, where $z_i = \mathbf{1}\ \forall\ i \in [n]$. To prune the ensemble, modify the z_i vectors. For example, setting $z_1 = [1, 0, \ldots, 0]$ prunes the first tree in the ensemble to depth 1 and setting $z_2 = \mathbf{0}$ removes the second tree entirely.

2.2 Ensemble Optimization Problem

Notationally, let $(z_i)_k$ represent the k-th element of vector z_i. FORESTPRUNE uses the following optimization problem to prune depth layers from an ensemble:

$$\min_{z_1,\ldots,z_n} \quad \frac{1}{m}\|y - \gamma \sum_{i}^{n} D_i z_i\|_2^2 + \frac{\alpha}{K}\sum_{i=1}^{n}\|W_i z_i\|_1 \tag{1a}$$

$$\text{s.t.} \quad (z_i)_k \in \{0,1\} \quad \forall\ i \in [n],\ k \in [d], \tag{1b}$$

$$(z_i)_{k_1} \geq (z_i)_{k_2} \quad \forall\ i \in [n],\ k_1 < k_2 \tag{1c}$$

$W_i \in \mathbb{R}^{d \times d}$ is a diagonal weight matrix assigned to each tree T_i, of the form $W_i = \text{diag}(w_{i,1}, \ldots, w_{i,d})$. These weights are prespecified, and we discuss weighting schemes in §2.3. A normalization constant K is computed from $W_i,\ i \in [n]$, and α is the regularization parameter.

Each decision variable $(z_i)_k$ is binary (1b) and represents whether to include the k-th layer of tree T_i in the processed ensemble. Each tree T_i has a corresponding decision vector $z_i \in \{0,1\}^d$ that contains d decision variables. The goal of FORESTPRUNE is to minimize regularized loss (1a) with respect to decision vectors $z_i,\ i \in [n]$. Constraint (1c) ensures that the pruned trees are contiguous since trees should not be able to skip depth layers. For example, a solution $z_i = [1, 0, 1, \ldots, 1, 0]$ is infeasible since a tree cannot skip depth layer 2 and proceed to depth layer 3. Trees can only be pruned from the bottom up.

Consider the regularized loss function in objective (1a). The first term of the function is least-squares loss and the second term is the regularization penalty. The parameter α controls regularization; larger values of α encourage shallower trees in the processed ensemble. Each entry of W_i, $w_{i,k}$ for $k \in [d]$, is nonnegative and represents the weight associated with including depth layer k of tree T_i into the processed ensemble. Since the entries of W_i and z_i are both nonnegative, the term $\|W_i z_i\|_1$ is equivalent to summing the elements of vector $W_i z_i$. Finally, K is a normalization constant set such that $K = \sum_{i=1}^{n}\sum_{k=1}^{d} w_{i,k}$—this ensures that α remains within a reasonable range which facilitates tuning the regularization parameter.

2.3 Weighting Schemes

We specify weight matrices $W_i,\ i \in [n]$ to produce post-processed ensembles with different parsimony properties. While various choices of W are possible, we discuss a couple of choices that we explored in detail. In **depth-weighting**, we assign $w_{i,k} = 1$ if the depth of decision tree T_i is less than or equal to k, otherwise we assign $w_{i,k} = 0$. We set normalization constant $K = nd$. This directly penalizes the total number of layers in the ensemble and produces ensembles with fewer trees. In **node-weighting**, we assign $w_{i,k}$ equal to the number of nodes in layer k of tree i. The normalization constant K is equal to the total number of nodes in the ensemble. This directly penalizes ensemble size and produces the most compact processed ensemble, with the shallowest trees. Figure 2 compares the effect of depth

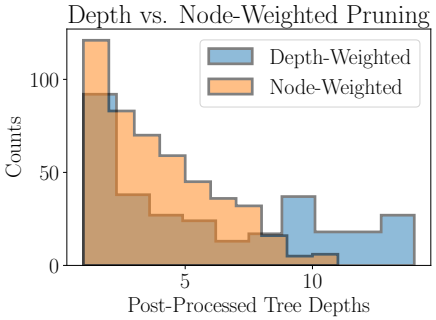


Figure 2: Node-weighting encourages shallow trees in the pruned ensemble.

vs. node-weighted FORESTPRUNE. Node-weighting produces shallower trees compared to depth-weighting. In addition, the total size of the node-weighted pruned ensemble is approximately three times smaller than the depth-weighted one, **7247** vs. **21588** nodes. However, the ensemble processed by node-weighted FORESTPRUNE contains more trees than the depth-weighted one, **95** vs. **62** trees. The test errors of the post-processed ensembles are comparable under both weighting schemes. Empirically, we observe that node-weighting encourages shallower trees and more compact ensembles, while depth-weighting produces an

ensemble with fewer trees.

2.4 Polishing Schemes

Solving Problem (1) yields solution vectors $z_i^*,\ i \in [n]$. These vectors, combined with depth-difference matrices $D_i,\ i \in [n]$ represent the pruned ensemble, with trees $T_i^*(x) = D_i z_i^*,\ i \in [n]$. We can adjust the weights of trees in this pruned ensemble to better fit the model to y. This especially helps for boosting, where trees are grown sequentially, since pruning a tree breaks the boosting sequence. We can also select a small subset pruned trees for further analysis by setting the weights of some trees to 0. This may be especially useful for bagging, where the pruned ensemble typically consists of many shallow trees. Reducing the number of trees improves the interpretability of the final model.

Motivated by the above discussion, our polishing scheme reweights the post-processed ensemble via the following optimization criterion:

$$\min_{\beta_1 \dots \beta_n} \quad \frac{1}{m}\|y - \gamma \sum_{i=1}^{n} \beta_i D_i z_i^*\|_2^2 + \alpha_2 \|\beta\|_\rho^\rho \qquad (2)$$

where coefficients $\beta_i,\ i \in [n]$ are the weights assigned to each tree, $\|\beta\|_\rho^\rho$ is the regularization penalty, and α_2 is the regularization hyperparameter. We set $\rho = 2$ for **ridge polishing** to reweight the trees. The ridge penalty ensures a unique estimator in β when $n > m$ and offers stability when the bases elements $D_i z_i^*$ are correlated; we observe empirically that small values of $\alpha_2 \approx 10^{-2}$ work well. We set $\rho = 0$ for **best subset polishing** and vary α_2 to select a small subset of trees for further analysis. To find good solutions for β, we use iterative hard thresholding (Blumensath and Davies, 2009) or mixed-integer programming solvers (Bertsimas et al., 2016). In §3.4, we discuss a variant of FORESTPRUNE combines polishing with pruning by jointly optimizing β and z.

2.5 Putting Together the Pieces

Algorithm 2: FORESTPRUNE Framework

Input: $T_i,\ i \in [n]$, $X \in \mathbb{R}^{m \times p}$, $y \in \mathbb{R}^m$, α
Compute $D_i,\ i \in [n]$.
Choose weighting scheme $W_i,\ i \in [n]$.
Solve Problem (1) for $z_i^*,\ i \in [n]$.
if *polish* = *True* **then**
  Solve Problem (2) for $\beta_i^*,\ i \in [n]$.
else
  $\beta^* = \mathbf{1}$.
end
Output: z^*, β^*

Algorithm 2 presents the overall framework of FORESTPRUNE. We recommend using both node-weighting and the polishing heuristic to compactify ensembles.

Ensemble optimization Problem (1) can be expressed as a mixed-integer optimization problem (Wolsey and Nemhauser, 1999), which can be computationally challenging for large problems (Bertsimas et al., 2016; Bertsimas and Van Parys, 2020; Hazimeh et al., 2021). Here we propose a novel algorithm based on block coordinate descent to obtain high quality solutions for Problem (1), that appear to work well for the problem-scales we study.

3 OPTIMIZATION ALGORITHM

Note that Problem (1) can be written as:

$$\min_{z_1 \dots z_n} \quad L(z_1 \dots z_n) + \sum_{i=1}^{n} g_i(z_i) \qquad (3a)$$

$$\text{s.t.} \quad z_i \in C_i \quad \forall\, i \in [n] \qquad (3b)$$

where $L(z_1 \dots z_n) = (1/m)\|y - \gamma \sum_i^n D_i z_i\|_2^2$ is a smooth function and $\sum_{i=1}^n g_i(z_i) = (\alpha/K) \sum_{i=1}^n \|W_i z_i\|_1$ is separable across blocks z_i's. Constraints (1b) and (1c) can be represented by constraint (3b), where C_i is the set of vectors in $\{0,1\}^d$ that satisfy condition $(z_i)_{k_1} \geq (z_i)_{k_2}\ \forall\, z_i \in C_i,\ k_1 < k_2$.

The non-smooth part of the objective in (3a) and constraint (3b) are both separable across z_i's. Motivated by the success of cyclic block coordinate descent methods (CBCD) for large-scale sparse regression problems (Wright, 2015; Hazimeh and Mazumder, 2020b), we apply CBCD methods to Problem (3). CBCD yields a sequence of decreasing objective values, and since objective (3a) is nonnegative the sequence will converge.

3.1 Cyclic Block Coordinate Descent (CBCD)

Consider a partition of the decision variables in Problem (3) into the blocks $z_i,\ i \in [n]$ with $z_i \in C_i$. CBCD works as follows. Initialize the algorithm by setting all blocks equal to the zero vector, $z_i = \mathbf{0}\ \forall\, i \in [n]$, and set index $\omega = 1$. Start with the first block of decision variables $z_\omega = z_1$ and minimize the objective with respect to z_ω while holding the other blocks $\{z_2 \dots z_n\}$ constant. Repeat this procedure, cycling through the blocks by incrementing ω, until the objective value converges.

Block Update Given fixed index ω, let set $\delta = \{1 \dots n\} \backslash \omega$. To update block z_ω, we solve the following problem:

$$\min_{z_\omega} \quad \frac{1}{m}\|\tilde{y}_\delta - \gamma D_\omega z_\omega\|_2^2 + \frac{\alpha}{K} \sum_{k=1}^{d} (W_\omega z_\omega)_k \qquad (4a)$$

$$\text{s.t.} \quad (z_\omega)_k \in \{0,1\} \quad \forall\, k \in [d], \qquad (4b)$$

$$(z_\omega)_{k_1} \geq (z_\omega)_{k_2} \quad \forall\, k_1 < k_2, \qquad (4c)$$

where $\tilde{y}_\delta = y - \gamma \sum_{i \in \delta} D_i z_i$. This is equivalent to solving Problem (1) or (3) with respect to z_ω while holding all the other blocks constant. As we show below Problem (4), despite its non-convexity, can be solved to optimality.

In Problem (4), z_ω is constrained to be a contiguous binary vector where the zeros occur after the ones. As a result, z_ω can only attain $d+1$ unique possible values. For example if $d=3$, the candidates for z_ω are $[1,1,1]$, $[1,1,0]$, $[1,0,0]$, and $[0,0,0]$. By enumerating through all possible candidates for z_ω, Problem (4) can be solved very quickly.

Each block update requires a single pass through $d+1$ candidates, where the objective is evaluated for each candidate. Evaluating the objective requires approximately $m \cdot d$ flops, so each block update requires approximately $m \cdot d^2$ flops. Typically the maximal depth d is below 20, so when $m >> d$ the per-iteration complexity of CBCD is linear in the number of training samples, $O(m)$.

Local Search Heuristic Due to the non-convexity of Problem (3), CBCD can get stuck in local suboptimal solutions. To improve solution quality, we employ a local combinatorial search procedure; a similar procedure was used in L0Learn in the context of sparse high-dimensional regression (Hazimeh et al., 2021).

After CBCD converges to a local solution $\hat{z}_i,\ i \in [n]$, partition the solution into two sets. Let S denote the set of indices for which $\hat{z}_i \neq \mathbf{0}$ and let S^c denote its complement. Select a random index $\xi \in S$ and set $z_\xi = \mathbf{0}$. Find the index $i^* \in S^c$ for which $D_i \hat{z}_{i^*}$ is the most correlated with the response y and set $z_{i^*} = \mathbf{1}$. Run CBCD with this new candidate solution till convergence and continue interlacing CBCD with local search steps, until local search no longer improves the objective value.

We observe that the local search heuristic discussed above is effective in guiding CBCD methods from suboptimal local solutions. A drawback of this procedure is that it can be expensive to find the best swap-coordinate $i^* \in S^c$ that would result in the largest improvement in the current objective. In fact, this would require computing the correlation coefficient between each block in S^c and y. We present a heuristic that we found to work quite well for our problem in the context of boosting: we always select the block in S^c with the smallest index, let $i^* = \min\{i \mid \hat{z}_i \in S^c\}$. In boosting ensembles, trees grown early in the boosting sequence are generally more correlated with the response. We can also extend this rule to bagging ensembles by first sorting the trees in the ensemble by training performance. The smallest index rule ensures that the local search heuristic always swaps in the earliest-grown tree, for negligible computational cost.

The complete CBCD algorithm with local search is presented in Algorithm 3.

Algorithm 3: FORESTPRUNE CBCD

Input: $D_i,\ W_i$ for $i \in [n]$, $y \in \mathbb{R}^m$, α
Initialize $z_i = \mathbf{0}\ \ \forall\, i \in [n], \omega = 1$
repeat
  repeat
    $\omega = \omega \mod n$
    Solve Problem (4) to update z_ω.
    $\omega = \omega + 1$
  until *converged*
  Local search.
until *objective no longer improves*
Output: $z_i,\ i \in [n]$

3.2 Regularization Paths

We use warm start continuation with Algorithm 3 to efficiently compute the entire regularization path of z_i's, across tuning parameter α. Start with a value of α sufficiently large such that $z_i^* = \mathbf{0},\ \ \forall\, i \in [n]$ and decrement α using the previous solution as a warm start until the full model is reached (Friedman et al., 2010). This provides a sequence of solutions with varying ensemble sizes that a practitioner can use to quickly select a suitable model.

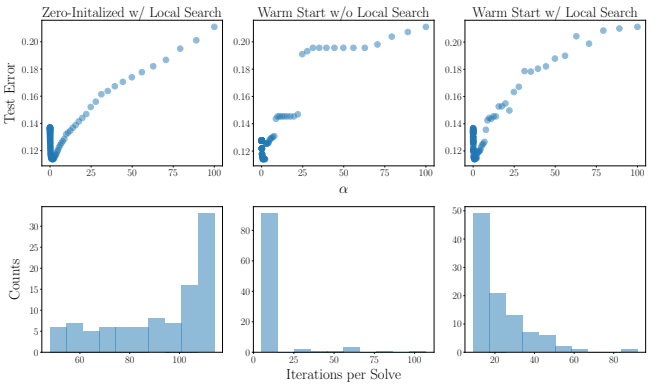


Figure 3: Warm start continuation with local search reduces the iterations (# of passes over all features) required to compute the regularization path while avoiding local minima.

When warm start continuation is used without local search, CBCD tends to get stuck at suboptimal local solutions. This can be seen in the middle plots of Figure 3. In this example, CBCD with warm start continuation is used to compute the entire regularization path of FORESTPRUNE. The scatter plot shows that the regularization path is nearly piecewise, and the histogram shows that CBCD often terminates immediately after initialization. Combining warm start continuation with local search resolves this issue, as seen in the rightmost plots of Figure 3. The regularization path for CBCD with warm starts and local search is very similar to the regularization path when CBCD is always zero-initialized (leftmost plots), however, our warm start procedure requires substantially fewer iterations, i.e., the # of passes over all features. As a result, this procedure makes α very efficient to tune

3.3 Computation Time

We now discuss the time required to compute good solutions to Problem (1) using our FORESTPRUNE CBCD algorithm. The computation time of CBCD depends on the number of

ForestPrune CBCD Single Solve			
Rows / Trees	**100**	**500**	**1000**
1145	0.1 ± 0.01	0.8 ± 0.10	1.5 ± 0.16
15437	0.2 ± 0.03	0.9 ± 0.10	2.2 ± 0.25
50164	3.0 ± 0.02	69.7 ± 2.60	131.0 ± 20.03
ForestPrune CBCD Regularization Path			
Rows / Trees	**100**	**500**	**1000**
1145	0.2 ± 0.01	1.5 ± 0.01	4.3 ± 0.10
15437	2.3 ± 0.08	11.6 ± 0.15	24.2 ± 0.45
50164	10.2 ± 0.10	106.3 ± 1.20	206.5 ± 1.23

Linear Relaxation (ECOS) Single Solve			
Rows / Trees	**100**	**500**	**1000**
1145	9.7 ± 0.20	829.8 ± 26.72	> 1800
15437	184.3 ± 0.60	> 1800	> 1800
50164	> 1800	> 1800	> 1800
Linear Relaxation (MOSEK) Single Solve			
Rows / Trees	**100**	**500**	**1000**
1145	7.0 ± 0.28	21.4 ± 0.27	59.8 ± 1.06
15437	140.5 ± 2.04	874.4 ± 59.93	> 1800
50164	> 1800	> 1800	> 1800

Table 1: Timing results in seconds. The red cells indicate that the method did not converge within 30 minutes.

observations in the data and the number of trees (blocks) in the ensemble. We fit tree ensembles with 100, 500, and 1000 trees of depth 6 on datasets with 1145, 15437, and 50164 rows. We set regularization parameter $\alpha = 1$ and use FORESTPRUNE CBCD to compute solutions to Problem (1). Also, we use CBCD with warm start continuation and local search to compute the entire regularization path.

We compare CBCD against this benchmark. We relax integrality constraint (1b) in Problem (1); the relaxed problem is a convex second order conic program. We use the open-source solver ECOS (Domahidi et al., 2013) and the proprietary industrial-grade solver MOSEK (Andersen and Andersen, 2000) to compute solutions to this relaxed problem. Table 1 shows the results of this experiment conducted on a personal laptop with a 2.80 GHz Intel Core i7 processor.

FORESTPRUNE CBCD computes solutions to Problem (1) orders of magnitude faster than the time it takes ECOS or MOSEK to solve the relaxed problem. Problem (1) is difficult to solve for larger instances: For an ensemble of 100 trees of depth 6 fit on a dataset with 50164 rows, the corresponding optimization problem has 600 decision variables and even after relaxing the integrality constraints, it takes an industrial-grade solver over 30 minutes to reach a solution. In contrast, our specialized CBCD algorithm can find a good solution in seconds. FORESTPRUNE CBCD can also efficiently compute the entire regularization path for α, for example, for Problem (1) with 1145 rows and 100 trees, it takes CBCD 0.2 seconds to compute the entire regularization path. MOSEK takes over 11 minutes to compute the same regularization path for the relaxed problem.

3.4 Joint Optimization for Pruning and Polishing

As mentioned in §2.4, we present a variant of FORESTPRUNE that combines polishing with pruning by jointly optimizing z and β. The corresponding optimization problem considers the following:

$$\begin{aligned} \min_{z,\,\beta} \quad & \frac{1}{m}\|y - \gamma \sum_{i}^{n} \beta_i D_i z_i\|_2^2 \\ & + \frac{\alpha}{K} \sum_{i=1}^{n} \|W_i z_i\|_1 + \alpha_2 \|\beta\|_\rho^\rho, \end{aligned} \tag{5}$$

and inherits the constraints from Problem (1). Variable β separates across blocks so Problem (5) is again block separable.

We can apply our CBCD algorithm to this problem with a slight modification to how we conduct block updates. Each block update now considers an optimization problem of the form:

$$\begin{aligned} \min_{z_\omega,\,\beta_\omega} \quad & \frac{1}{m}\|\tilde{y}_\delta - (\gamma D_\omega z_\omega)\beta_\omega\|_2^2 \\ & + \frac{\alpha}{K} \sum_{k=1}^{d} (W_\omega z_\omega)_k + \alpha_2 \|\beta_\omega\|_\rho^\rho, \end{aligned} \tag{6}$$

where $\tilde{y}_\delta = y - \gamma \sum_{i \in \delta} \beta_i D_i z_i$. This new block update problem also inherits the constraints from Problem (4), and, as discussed in §3.1, vector z_w has $d+1$ unique possible values due to these constraints. Therefore, vector $q_w = \gamma D_\omega z_\omega$ has $d+1$ unique possible values as well. For each unique value of q_ω, we solve this problem for scalar β_w:

$$\min_{z_\omega,\,\beta_\omega} \quad \frac{1}{m}\|\tilde{y}_\delta - \beta_\omega q_\omega\|_2^2 + \alpha_2 \|\beta_\omega\|_\rho^\rho. \tag{7}$$

For ridge polishing ($\rho = 2$) this univariate ridge problem has a closed-form solution for β_ω. For best subset polishing ($\rho = 0$) good solutions for β_ω can be obtained through hard thresholding (Hazimeh and Mazumder, 2020a). We return the z_ω, β_ω pair that yields the lowest objective value for Problem (6) to complete the block update. The other steps of our CBCD algorithm remain the same.

In practice, we observe that the ensembles post-processed by jointly optimizing β and z in FORESTPRUNE can outperform the ensembles extracted by pruning followed by polishing. However, this improved performance comes at a computational cost, since our CBCD algorithm requires more iterations to converge over both sets of variables. We present this joint optimization method as an additional flexibility in our FORESTPRUNE framework. Users can choose between joint optimization and pruning then polishing depending on their problem size and desired application.

4 EXPERIMENTS

We evaluate FORESTPRUNE against these competing ensemble post-processing algorithms.

Baseline A simple but useful baseline is to conduct fewer bagging/boosting iterations. For bagging, since trees are grown independently, we start with the full model and repeatedly remove randomly selected trees until a sparse model is obtained. For boosting, we start with the full boosting sequence and trim trees from the tail of the sequence.

Cost-Complexity Pruning Cost-complexity pruning (CCP) prunes individual decision trees by recursively removing

weak branches (Breiman, 2017). Each decision tree is pruned independently of the other trees in the ensemble. To post-process the entire tree ensemble, we set the cost-complexity sparsity parameter to be the same for all trees. By varying this sparsity parameter, we control the trade-off between the degree of individual tree pruning and ensemble performance.

LASSO Given a tree ensemble $T_i, i \in [n]$, LASSO ensemble pruning solves the following problem:

$$\min_{\beta_1 \dots \beta_n} \quad \frac{1}{m}\|y - \gamma \sum_{i}^{n} \beta_i T_i(X)\|_2^2 + \lambda \sum_{i=1}^{n} |\beta_i|, \quad (8)$$

where tree i has prediction $T_i(X)$ and coefficient β_i. The ℓ_1-penalty over the coefficients encourages sparsity.

Best Subset Tree Selection Best subset tree selection (BSTS) uses mixed integer quadratic programming (MIQP) to select the optimal subset of trees from an ensemble, subject to a size constraint. Let n_i be the number of nodes used by tree T_i and let ν be the maximum number of nodes to include in the pruned ensemble. The optimization problem can be expressed as follows,

$$\begin{aligned}
\min_{\beta_1 \dots \beta_n} \quad & \frac{1}{m}\|y - \gamma \sum_{i}^{n} \beta_i T_i(X)\|_2^2 && \text{(9a)}\\
\text{s.t.} \quad & (\beta_i,\ 1 - \zeta_i) \quad \text{SOS-1} \quad \forall\, i \in [n], && \text{(9b)}\\
& \sum_{i=1}^{n} n_i \zeta_i \leq \nu, && \text{(9c)}\\
& \zeta_i \in \{0, 1\} \quad \forall\, i \in [n], && \text{(9d)}
\end{aligned}$$

where constraint (9b) is a Type 1 Special Ordered Set (SOS-1) constraint (Bertsimas et al., 2016) that ensures that $\zeta_i = 1$ if tree T_i is assigned a nonzero weight. We implement BSTS in Gurobi (Gurobi Optimization, LLC, 2022) and warm start the solver using LASSO solutions. For smaller values of ν, we are able to solve the MIQP to optimality in minutes.

Discussion on the Choice of Competing Methods There is a large corpus of work on selecting good subsets of trees from tree ensembles. Notable examples include a forward step-wise selection algorithm proposed by Caruana et al. (2004), a ranking algorithm prescribed by Rokach et al. (2006), and an evolutionary optimization algorithm proposed by Qian et al. (2015). Most of these algorithms are heuristics to approximate the NP-hard problem of selecting the best subset of trees from an ensemble.

To simplify comparisons between FORESTPRUNE and this family of competing algorithms we formulate BSTS, a new stronger benchmark that selects the optimal subset of trees using MIQP. Recent advancements in MIQP techniques (Bertsimas et al., 2016; Bertsimas and Van Parys, 2020) have made this approach tractable for problems of the sizes that we are interested in. As such, we omit comparisons between FORESTPRUNE and various heuristics in favor of evaluating our algorithm directly against our stronger benchmark. We include LASSO pruning as a competing algorithm since both LASSO and FORESTPRUNE use warm start continuation to efficiently compute regularization paths (Friedman et al., 2007). We also compare FORESTPRUNE against cost-complexity pruning for pruning bagging ensembles, since both algorithms can prune deep trees trees individually.

4.1 Compact Bagging Ensembles

Procedure Bagging ensembles are generally robust to overfitting so pruning ensembles typically will not improve performance. To evaluate how well FORESTPRUNE performs at compactifying bagging ensembles, we fix a threshold for acceptable performance loss and find the smallest pruned ensemble within this threshold. Repeat this procedure for 25 regression datasets in the OpenML repository (Vanschoren et al., 2013) using 5-fold cross-validation; the full list of datasets can be found in the supplement. First, fit a bagging ensemble of 500 trees where each tree is grown to depth $d = 20$, and a subsample of $\sqrt{p}$ features is considered at each split. Next, fix a threshold for acceptable performance loss, $\phi \in \{0.01, 0.025, 0.05\}$, and compute the entire regularization path for FORESTPRUNE. We select α such that the validation loss of the pruned ensemble is within ϕ of the loss of the full model. Compare the reduction in ensemble size and the % difference in test error between the pruned ensemble and the original. Finally, use the 4 competing methods described above to find the best model no larger than the ensemble pruned by FORESTPRUNE, and compare the test performances of all methods.

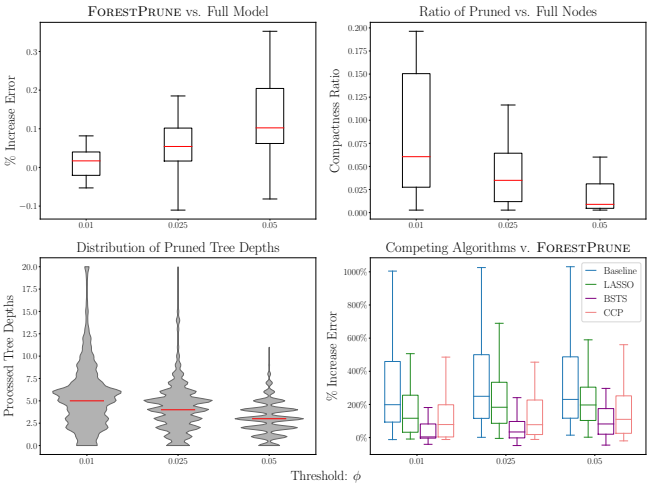


Figure 4: FORESTPRUNE on bagging ensembles. FORESTPRUNE can reduce ensemble size 20x for a ~5% increase in error.

Results Figure 4 presents the results of the compact bagging experiment. The distributions are obtained across all folds and datasets in the experiment. The top left plot shows

the % increase in test error between the model pruned by FORESTPRUNE and the full model; as the threshold for acceptable performance loss ϕ increases, the % increase in test error rises correspondingly. The top right plot in Figure 4 presents the compactness ratio, the ratio of the number of nodes in the pruned model over the number of nodes in the full model. The compactness ratio decreases as ϕ increases; if the threshold for acceptable performance loss is increased, the model can be made much smaller. The bottom left plot in Figure 4 presents the distribution of tree depths in the pruned ensemble. Originally all trees in the bagging ensemble are grown to depth $d = 20$. Node-weighted FORESTPRUNE produces much shallower trees, across all levels of ϕ the average tree depth in the pruned ensemble is around 5.

Across all levels of ϕ, FORESTPRUNE produces substantially smaller ensembles for modest increases in test error. For example, when $\phi = 0.01$, the ensembles post-processed by FORESTPRUNE perform nearly the same as the full ensembles, with only a 1-5% increase in test error, and are reduced 10-20 fold in size. The average depth of trees in the pruned ensemble is also reduced by a factor of 4. By varying the threshold ϕ, FORESTPRUNE can be tuned to balance performance and compactness.

FORESTPRUNE also outperforms the competing algorithms. The bottom right plot in Figure 4 shows the % increase in test error between the model produced by FORESTPRUNE and the models produced by the competing algorithms. The competing methods perform worse than FORESTPRUNE if the % increase in test error is positive. Across all values of ϕ, the distributions of the % increase in test error between the competing algorithms and FORESTPRUNE are almost entirely positive, with medians of $\{\mathbf{220}\%, \mathbf{250}\%, \mathbf{250}\%\}$ for the baseline, $\{\mathbf{130}\%, \mathbf{190}\%, \mathbf{220}\%\}$ for LASSO, $\{\mathbf{5}\%, \mathbf{40}\%, \mathbf{80}\%\}$ for BSTS, and $\{\mathbf{78}\%, \mathbf{78}\%, \mathbf{110}\%\}$ for CCP.

The baseline, LASSO, and BSTS competing methods are only capable of excluding or selecting trees, and the size of a single deep tree may be larger than the shallow ensemble produced by FORESTPRUNE. As a result, these post-processing methods are unable to produce a model similar in size to FORESTPRUNE with comparable test performance. CCP, on the other hand, prunes individual trees, but prunes each tree independently from the other trees in the ensemble. FORESTPRUNE outperforms CCP since FORESTPRUNE prunes trees with respect to the overall performance of the ensemble.

4.2 Compact Boosting Ensembles

Procedure To evaluate how well FORESTPRUNE compactifies and regularizes boosting ensembles, we evaluate the performance of the algorithm across the regularization path for α. On the same datasets and folds described in §4.1, we fit stochastic gradient boosting ensembles of 250 depth 5 trees with $\gamma = 0.1$, subsampling 25% of the training data for each tree. We post-process the ensembles and compare errors along the regularization paths of FORESTPRUNE and the LASSO and baseline competing methods.

Computing regularization paths for BSTS is infeasible since the procedure requires repeatedly solving MIQP problems of increasing difficulty. To evaluate BSTS against FORESTPRUNE, we fix an ensemble size budget and find the model selected by BSTS constrained by this budget. We use the regularization paths computed above to find the best models selected by FORESTPRUNE, LASSO, and baseline within this budget and compare the test performances of all models (size budget boosting experiment).

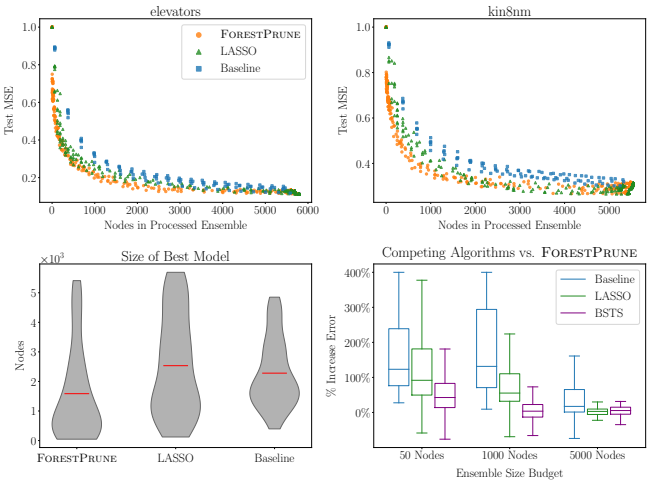


Figure 5: FORESTPRUNE on boosting ensembles.

Results The scatter plots in the top row of Figure 5 show visualizations of the experiment above. The horizontal axes show post-processed ensemble size and the vertical axes show test error; the spread in each scatter plot is due to the 5-fold CV. We observe that the best models pruned by each of the 3 methods have similar test errors, however, FORESTPRUNE produces more compact models compared to baseline and LASSO post-processing. This result is consistent across all datasets in the experiment. The bottom left plot in Figure 5 compares the distribution of the sizes of the best models pruned. On average the best model pruned by FORESTPRUNE is around 1000 nodes smaller than the best models pruned by the competing methods.

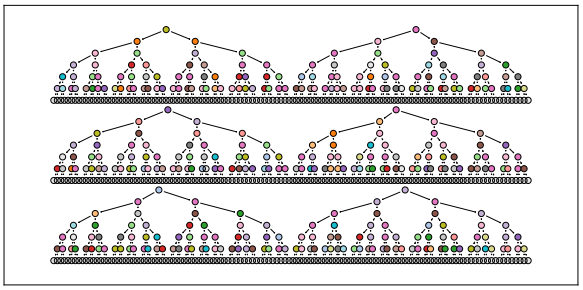
The bottom right plot in Figure 5 shows the results of our size budget boosting experiment. The vertical axis shows the % increase in test error between the competing methods and FORESTPRUNE, and the horizontal axis shows the maximum number of nodes allowed in the pruned ensemble. We observe that for pruning sparse ensembles, with a size budget of 50 nodes, FORESTPRUNE outperforms the competing algorithms. The sparse ensembles produced by the strongest competing method, BSTS, perform around **43%** worse than the ensembles pruned by FORESTPRUNE. Given such a tight node budget, the competing algorithms can only

select a few trees. FORESTPRUNE is more flexible and can trim and then select multiple shallow trees.

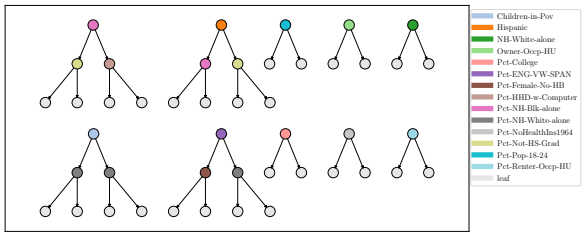
When the budget is increased to 1000 nodes, FORESTPRUNE still significantly outperforms LASSO and baseline post-processing but performs similarly to BSTS. Finally, for a budget of 5000 nodes, LASSO, BSTS, and FORESTPRUNE all perform comparably. This is expected since nearly the entire ensemble can be selected for this budget. We conclude that FORESTPRUNE is the best performing algorithm for pruning sparse models from boosted tree ensembles, and matches our strong BSTS benchmark for selecting larger subensembles.

Code for our experiments, along with a Python implementation of FORESTPRUNE can be found in our project GitHub repository.

4.3 Interpretability Case Study



(a) A sample of full depth trees in the original ensemble.



(b) FORESTPRUNE trims and selects 10 shallow trees.

Figure 6: FORESTPRUNE improves the interpretability of a random forest model used to predict census response rates.

We conclude with a case study to showcase how FORESTPRUNE prunes tree ensembles into interpretable models. Building off of the work by Ibrahim et al. (2021) we use the Census Planning Database (50,000 rows and 295 features) to predict decennial census response rates at the tract level. A modeling competition by the Census Bureau found that tree ensembles work well for this problem, but are uninterpretable and fail to produce actionable insights towards improving response rates (Erdman and Bates, 2017).

We start with a random forest of 500 depth 6 trees; the test RMSE of the model is **7.54%** and all 295 features are used. A sample of 6 out of the 500 trees is presented in Figure 6a. This model is difficult to interpret since the depth of each tree makes it impossible to track the relationships between the features. We use FORESTPRUNE with $\alpha = 15$ to prune this random forest and use best subset polishing (§2.4) to select 10 trees for further analysis. Figure 6b presents this sparse model which uses just 14 out of the 295 features. The pruned ensemble achieves a test RMSE of **8.47%**; for a small increase in test error, FORESTPRUNE extracts a much more interpretable model. The model post-processed by FORESTPRUNE performs much better than a single decision tree, which achieves a test RMSE of **10.6%**.

We can examine the structure of each tree in Figure 6b to understand the relationships between the features and the response. The single split, main effect trees with the features *Pct-College*, *Pct-Pop-18-24*, *Pct-Renter-Occup-HU*, and *Owner-Occup-HU*, reveal that census tracts with many non-permanent residents (renters and college students) have lower response rates. The depth 2 trees show pairwise feature interactions between ethnicity/race, education, and poverty levels and we can examine these trees to determine underserved communities have lower census response rates as well. FORESTPRUNE extracts a compact transparent model from the full tree ensemble with nominal performance loss.

5 CONCLUSION

In this work, we develop FORESTPRUNE, an optimization framework to prune depth layers from trees in an ensemble. We propose a CBCD method with local search to compute high-quality solutions to the optimization problems formulated under this framework. Our specialized optimization algorithm is computationally efficient; the per-iteration cost of our method is linear in the number of training samples and our main algorithm converges in a few seconds for medium-sized problems. The framework only contains a single hyperparameter to tune, the regularization parameter α, and we can leverage warm start continuation to rapidly compute the entire regularization path. Our results show that FORESTPRUNE can drastically reduce the size of bagging and boosting ensembles with nominal performance loss compared to competing algorithms. Finally, FORESTPRUNE can produce interpretable models by pruning ensembles shallow enough for practitioners to examine by hand, increasing the transparency of tree-based models.

6 ACKNOWLEDGMENTS

We thank the anonymous reviewers for their comments that helped us improve the paper. This research is funded in part by a grant from the Office of Naval Research (ONR-N00014-21-1-2841).

References

Erling D Andersen and Knud D Andersen. The mosek interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In *High performance optimization*, pages 197–232. Springer, 2000.

Dimitris Bertsimas and Bart Van Parys. Sparse high-dimensional regression: Exact scalable algorithms and phase transitions. *The Annals of Statistics*, 48(1):300–323, 2020.

Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *The annals of statistics*, 44(2):813–852, 2016.

Thomas Blumensath and Mike E Davies. Iterative hard thresholding for compressed sensing. *Applied and computational harmonic analysis*, 27(3):265–274, 2009.

Leo Breiman. Random forests. *Machine learning*, 45(1): 5–32, 2001.

Leo Breiman. *Classification and regression trees*. Routledge, 2017.

Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. Ensemble selection from libraries of models. In *Proceedings of the twenty-first international conference on Machine learning*, page 18, 2004.

Alexander Domahidi, Eric Chu, and Stephen Boyd. Ecos: An socp solver for embedded systems. In *2013 European Control Conference (ECC)*, pages 3071–3076. IEEE, 2013.

Chandra Erdman and Nancy Bates. The low response score (lrs) a metric to locate, predict, and manage hard-to-survey populations. *Public Opinion Quarterly*, 81(1): 144–156, 2017.

Jerome Friedman, Trevor Hastie, Holger Höfling, and Robert Tibshirani. Pathwise coordinate optimization. *The annals of applied statistics*, 1(2):302–332, 2007.

Jerome Friedman, Trevor Hastie, and Rob Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of statistical software*, 33(1):1, 2010.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

Jerome H Friedman and Bogdan E Popescu. Predictive learning via rule ensembles. *The annals of applied statistics*, pages 916–954, 2008.

Jerome H Friedman, Bogdan E Popescu, et al. Importance sampled learning ensembles. *Journal of Machine Learning Research*, 94305:1–32, 2003.

Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2022. URL `https://www.gurobi.com`.

Maissae Haddouchi and Abdelaziz Berrado. A survey of methods and tools used for interpreting random forest. In *2019 1st International Conference on Smart Systems and Data Science (ICSSD)*, pages 1–6. IEEE, 2019.

Hussein Hazimeh and Rahul Mazumder. Fast best subset selection: Coordinate descent and local combinatorial optimization algorithms. *Operations Research*, 68(5): 1517–1537, 2020a.

Hussein Hazimeh and Rahul Mazumder. Fast best subset selection: Coordinate descent and local combinatorial optimization algorithms. *Operations Research*, 68 (5):1517–1537, 2020b. doi: 10.1287/opre.2019.1919. URL `https://doi.org/10.1287/opre.2019.1919`.

Hussein Hazimeh, Rahul Mazumder, and Ali Saab. Sparse regression at scale: Branch-and-bound rooted in first-order optimization. *Mathematical Programming*, pages 1–42, 2021.

Shibal Ibrahim, Rahul Mazumder, Peter Radchenko, and Emanuel Ben-David. Predicting census survey response rates via interpretable nonparametric additive models with structured interactions. *arXiv preprint arXiv:2108.11328*, 2021.

Claudio Lucchese, Franco Maria Nardini, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Salvatore Trani. Post-learning optimization of tree ensembles for efficient ranking. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 949–952, 2016.

Richard Maclin and David Opitz. An empirical evaluation of bagging and boosting. *AAAI/IAAI*, 1997:546–551, 1997.

Gonzalo Martinez-Munoz, Daniel Hernández-Lobato, and Alberto Suárez. An analysis of ensemble pruning techniques based on ordered aggregation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):245–259, 2008.

Julie Nutini, Issam Laradji, and Mark Schmidt. Let's make block coordinate descent converge faster: Faster greedy rules, message-passing, active-set complexity, and superlinear convergence. *arXiv preprint arXiv:1712.08859*, 2017.

Chao Qian, Yang Yu, and Zhi-Hua Zhou. Pareto ensemble pruning. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

Lior Rokach, Oded Maimon, and Reuven Arbel. Selective voting—getting more for less in sensor fusion. *International Journal of Pattern Recognition and Artificial Intelligence*, 20(03):329–350, 2006.

O Sagi and L Rokach. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2018.

Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL `http://doi.acm.org/10.1145/2641190.2641198`.

Laurence A Wolsey and George L Nemhauser. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

Stephen J Wright. Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34, 2015.

FORESTPRUNE Supplementary Materials

A OpenML Dataset Descriptions

Dataset Name	Rows	Features
humandevel	130	2
triazines	186	61
tecator	240	125
autoMpg	398	8
no2	500	8
boston	506	14
stock	950	10
socmob	1156	6
Moneyball	1232	15
balloon	2001	2
space_ga	3107	7
abalone	4177	9
Mercedes_Benz_Greener_Manufacturing	4209	377
mtp	4450	203
wine_quality	6497	12
wind	6574	15
kin8nm	8192	9
cpu_small	8192	13
puma32H	8192	33
bank32nh	8192	33
pol	15000	49
elevators	16599	19
houses	20640	9
house_16H	22784	17
2dplanes	40768	11

Table 1: OpenML Datasets used in the experiments along with metadata.

B Code Implementation and Detailed Results

An implementation of FORESTPRUNE in Python, code for our experiments, and detailed experimental results can be found in this repository: `https://github.com/brianliu12437/ForestPruneAISTATS2023`.