# End-to-end Feature Selection Approach for Learning `Skinny Trees`

**Shibal Ibrahim**
MIT
Cambridge, MA, USA
shibal@mit.edu

**Kayhan Behdin**
MIT
Cambridge, MA, USA
behdink@mit.edu

**Rahul Mazumder**
MIT
Cambridge, MA, USA
rahulmaz@mit.edu

## Abstract

We propose a new optimization-based approach for feature selection in tree ensembles, an important problem in statistics and machine learning. Popular tree ensemble toolkits e.g., Gradient Boosted Trees and Random Forests support feature selection post-training based on feature importance scores, while very popular, they are known to have drawbacks. We propose `Skinny Trees`: an end-to-end toolkit for feature selection in tree ensembles where we train a tree ensemble while controlling the number of selected features. Our optimization-based approach learns an ensemble of differentiable trees, and simultaneously performs feature selection using a grouped $\ell_0$-regularizer. We use first-order methods for optimization and present convergence guarantees for our approach. We use a dense-to-sparse regularization scheduling scheme that can lead to more expressive and sparser tree ensembles. On 15 synthetic and real-world datasets, `Skinny Trees` can achieve $1.5\times -$ $620\times$ feature compression rates, leading up to $10\times$ faster inference over dense trees, without any loss in performance. `Skinny Trees` lead to superior feature selection than many existing toolkits e.g., in terms of AUC performance for 25% feature budget, `Skinny Trees` outperforms LightGBM by 10.2% (up to 37.7%), and Random Forests by 3% (up to 12.5%).

Proceedings of the 27[th] International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238. 

## 1 INTRODUCTION

Decision trees have been popular in various machine learning applications (Erdman and Bates, 2016; Chen and Guestrin, 2016) for their competitive performance, interpretability, robustness to outliers, and ease of tuning (Hastie et al., 2009). Many scalable toolkits for learning tree ensembles have been developed (Breiman, 2001; Chen and Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2018). While these toolkits are excellent for building tree ensembles, they do not allow for feature selection during the training process.

Feature selection is a fundamental problem in machine learning and statistics and has widespread usage across various real-world tasks (Wulfkuhle et al., 2003; Cai et al., 2018; Li et al., 2017). Popular tree ensemble toolkits only allow selecting informative features post-training based on feature importance scores, which are known to have drawbacks[1] in the context of feature selection (Strobl et al., 2007; Boulesteix et al., 2011; Zhou and Hooker, 2021). Recently, there has been some work on optimization-based approaches for feature selection in trees. For example, (Zharmagambetov and Carreira-Perpiñán, 2020) consider oblique decision trees (hyperplane splits at every node), and use $\ell_1$-penalization to encourage coefficient sparsity at every node of the tree. This achieves node-level feature selection and does not appear to be well-suited for tree-level or ensemble-level feature selection (See Sec. 7.1). Liu et al. (2021) proposed ControlBurn that considers a lasso based regularizer for feature selection on a pre-trained forest. This can be viewed as a two-stage procedure (unlike an end-to-end training procedure we propose here), where one performs feature selection after training a tree ensemble with all original features, While these methods serve as promising candidates for feature selection, these works highlight that identifying relevant features *while* learning compact trees remains an open challenge—an avenue we address in this work.

[1] They are found to hurt performance in (a) settings where number of samples are smaller than features and (b) settings with correlated features (see Sec. 6).

In many real world problems there are costs associated with features reflecting time, money, and other costs related to the procurement of data (Min et al., 2014; Zhang, 2010). In this context, one would like to collect a compact set of features to reduce experimental costs. Additionally, selecting a compact set of relevant features can lead to enhanced interpretability (Ribeiro et al., 2016), faster inference, decreased memory footprint, and even improved model generalization on unseen data (Chandrashekar and Sahin, 2014).

In this paper, we propose an end-to-end optimization framework for feature selection in tree ensembles where we jointly learn the trees and the relevant features in *one* shot. Our framework is based on differentiable (a.k.a. soft) tree ensembles (Jordan and Jacobs, 1994; Kontschieder et al., 2015; Hazimeh et al., 2020; Ibrahim et al., 2022) where tree ensembles are learnt based on differentiable programming. These works, however, do not address feature selection in trees which is our focus. We use a sparsity-inducing penalty (based on the group $\ell_0 - \ell_2$-penalty) to encourage feature selection. While group $\ell_0 - \ell_2$ penalty has been found to be useful in recent work on high-dimensional linear models (Hazimeh et al., 2022) and additive models (Hazimeh et al., 2022; Ibrahim et al., 2021), their adaptation to tree ensemble presents unique challenges. To obtain high-quality models with good generalization-and-sparsity tradeoffs, we need to pay special attention to dense-to-sparse training, which differs from sparse-to-dense training employed in linear/additive models above. We demonstrate that our end-to-end learning approach leads to better accuracy and feature sparsity tradeoffs.

**Contributions.** We propose a novel end-to-end optimization-based framework for feature selection in tree ensembles. We summarize our contributions in the paper as follows:

- We propose a joint optimization approach, where we simultaneously perform feature selection and tree ensemble learning. Our joint training approach is different from post-training feature selection in trees. Our approach learns (differentiable) tree ensembles with a budget on feature sparsity where the latter is achieved via a group $\ell_0$-based regularizer.
- Our algorithmic workhorse is based on proximal mini-batch gradient descent (GD). We also discuss the convergence properties of our approach in the context of a nonconvex and nonsmooth objective. When our first-order optimization methods are used with *dense-to-sparse* scheduling of regularization parameter, we obtain tree ensembles with better accuracy and feature-sparsity tradeoffs.
- We introduce a new toolkit: `Skinny Trees`. We consider 15 synthetic and real-world datasets, showing that `Skinny Trees`can lead to superior feature selection and test AUC compared to popular toolkits. In particular, for 25% feature budget, `Skinny Trees` outperforms LightGBM by 10.2% (up to 37.7%), XGBoost by 3.1% (up to 17.4%), and Random Forests by 3% (up to 12.5%) in test AUC.

## 2 RELATED WORK

**Trees.** A popular and effective method to construct a single decision tree is based on recursive greedy partitioning (e.g., CART) (Hastie et al., 2009). Popular methods to construct tree ensembles are based on bagging (Breiman, 2001), sequential methods like Boosting (Hastie et al., 2009), etc. These have led to various popular toolkits, e.g., Random Forests (Breiman, 2001), Gradient Boosted Trees (Chen and Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2018). Another line of work that is most related to our current approach uses differentiable (or smooth) approximations of indicator functions at the split nodes (Jordan and Jacobs, 1994). First-order methods (e.g, SGD) are used for end-to-end differentiable training of tree ensembles (Kontschieder et al., 2015; Hazimeh et al., 2020; Ibrahim et al., 2022). Joint training of soft tree ensembles often results in more compact representations (i.e., fewer trees) compared to boosting-based procedures (Hazimeh et al., 2020). Despite the success and usefulness of both these approaches, to our knowledge, there is no prior work that performs simultaneous training and feature selection—a void we seek to fill in this work. We next summarize some popular feature selection methods.

**Feature selection.** We review some prior work on feature selection as they relate to our work. We group them into three major categories:

1. Filter methods attempt to remove irrelevant features before model training. These methods perform feature screening based on statistical measures that quantify feature-specific relevance scores (Battiti, 1994; Peng et al., 2005; Estevez et al., 2009; Song et al., 2007, 2012; Chen et al., 2017). These scores consider the marginal effect of a feature over the joint effect of feature interactions.
2. Wrapper methods (Kohavi and John, 1997; Stein et al., 2005; Zhu et al., 2007; Reunanen, 2003; Allen, 2013; Onnia et al., 2001; Monirul Kabir et al., 2010; Roy et al., 2015) use the outcome of a model to determine the relevance of each feature. Some of these methods require recomputing the model for a subset of features and can be computationally expensive. This category also includes feature selection using feature importance scores of a pre-trained model. Many tree ensemble toolk-

its (Breiman, 2001; Chen and Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2018) produce feature-importance scores from a pre-trained ensemble. Lundberg and Lee (2017) propose SHAP values as a unified measure of feature importance. Sharma et al. (2023) uses SHAP values to select a subset of features that can be useful for secondary model performance characteristics e.g., fairness, robustness etc. Liu et al. (2021) propose ControlBurn, which formulates an optimization problem with a Lasso-type regularizer to perform feature selection on a pre-trained forest.

3. Embedded methods *simultaneously* learn the model and the relevant subset of relevant features. Notable among these methods include $\ell_0$-based procedures (Hazimeh and Mazumder, 2020; Hazimeh et al., 2022; Ibrahim et al., 2021, 2023a) and their variants based on lasso (Tibshirani, 1996; Ravikumar et al., 2009; Zhao and Liu, 2012) in the linear and additive model settings. Some distributed and stochastic greedy methods have also been explored for subset selection (see, for example, Khanna et al., 2017, and references therein). Embedded nonlinear feature selection methods have been explored for neural networks. For example, Chen et al. (2021) use an active-set style algorithm for cardinality-constrained feature selection. Other approaches include the use of (group) lasso type methods (Scardapane et al., 2017; Feng and Simon, 2017; Dinh and Ho, 2020; Lemhadri et al., 2021), or reparameterizations of $\ell_0$-penalty with stochastic gates (Louizos et al., 2018; Yamada et al., 2020).

We propose an embedded approach that simultaneously performs feature selection and tree ensemble learning. This joint training approach can be useful for compression, efficient inference, and/or generalization.

**Organization.** The rest of the paper is organized as follows. Section 3 summarizes relevant preliminaries. Section 4 presents a formulation for feature selection in soft tree ensembles. Section 5 discusses our optimization algorithm and its convergence properties. Later, we discuss a scheduling approach that can result in better accuracy and feature sparsity tradeoffs. Sections 6 and 7 perform experiments on a combination of 15 synthetic and real-world datasets to highlight the usefulness of our proposals.

## 3 PRELIMINARIES

We learn a mapping $\boldsymbol{f} : \mathbb{R}^p \to \mathbb{R}^c$, from input space $\mathcal{X} \subseteq \mathbb{R}^p$ to output space $\mathcal{Y} \subseteq \mathbb{R}^c$, where we parameterize function $\boldsymbol{f}$ with a soft tree ensemble. In a regression setting $c = 1$, while in multi-class classification setting $c = C$, where $C$ is the number of classes. Let $m$ be the number of trees in the ensemble and let $\boldsymbol{f}^j$ be the $j$th tree in the ensemble. We learn an additive model with the output being sum over outputs of all the trees: $\boldsymbol{f}(\boldsymbol{x}) = \sum_{j=1}^{m} \boldsymbol{f}^j(\boldsymbol{x})$ for an input feature-vector $\boldsymbol{x} \in \mathbb{R}^p$. A summary of the notation can be found in Table S1 in Supplement.

Compared to classical trees, soft trees allow for much more flexibility in catering to different loss functions (Ibrahim et al., 2022), sparse routing in Mixture of Experts (Ibrahim et al., 2023b) etc. A soft tree is a differentiable variant of a classical decision tree, so learning can be done using gradient-based methods. It was proposed by Jordan and Jacobs (1994), and further developed by Kontschieder et al. (2015); Hazimeh et al. (2020); Ibrahim et al. (2022) for end-to-end optimization. Soft trees typically perform soft routing, i.e., a sample is fractionally routed to all leaves; but can be modified to do hard routing in the spirit of conditional computing (Hazimeh et al., 2020; Ibrahim et al., 2023b). Soft trees are based on hyperplane splits, where the routing decisions rely on a linear combination of the features. Particularly, each internal node is associated with a trainable weight vector that defines the associated hyperplane. Formulations (Ibrahim et al., 2022) for training soft tree ensembles more efficiently have been proposed, which exploit tensor structure of a tree ensemble. A summary of soft trees and tensor formulation is in Supplement Sec. S2.

## 4 PROBLEM FORMULATION

Feature selection plays a ubiquitous role in modern statistical regression, especially when the number of predictors is large relative to the number of observations. We describe the problem of global feature selection. We assume a data-generating model $p(\mathbf{x}; \mathbf{y})$ over a $p$-dimensional space, where $\mathbf{x} \in \mathbb{R}^p$ is the covariate and $\mathbf{y}$ is the response. The goal is to find the best function $\boldsymbol{f}(\mathbf{x})$ for predicting $\mathbf{y}$ by minimizing:

$$\min_{\boldsymbol{f} \in \mathcal{F}, \mathcal{Q}} \quad \mathbb{E}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x}_{\mathcal{Q}}))] \tag{1}$$

where $\mathcal{Q} \subseteq \{1, 2, \cdots, p\}$ is an unknown (learnable) subset of features of size at most $K$, $\boldsymbol{f}$ is a learnable non-parametric function from the function class $\mathcal{F}$, and $L : \mathbb{R}^p \times \mathbb{R}^c \to \mathbb{R}$ is a loss function. The principal difficulty in solving (1) lies in the joint optimization of $(f, \mathcal{Q})$—the number of subsets $\mathcal{Q}$ grows exponentially with $p$. In addition, the family of functions $\mathcal{F}$ needs to be sufficiently flexible (here, $\mathcal{F}$ is the class of soft tree ensembles with fixed ensemble size $m$ and depth $d$).

In the context of tree ensembles, our goal for global feature selection is to select a small subset of features across *all* the trees in the ensemble. More specifically,

we consider the framework with response $\boldsymbol{y}$ and prediction $\boldsymbol{f}(\boldsymbol{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})$, where the function $\boldsymbol{f}$ is parameterized by learnable hyperplane parameters $\boldsymbol{\mathcal{W}} \in \mathbb{R}^{p,m,|\mathcal{I}|}$ across all the split nodes and learnable leaf parameters $\boldsymbol{\mathcal{O}}$ across all trees. Note that $m$ is the number of trees in the ensemble and $|\boldsymbol{\mathcal{I}}|$ represents the number of (split) nodes in each tree. This parameterization of soft trees points to a group structure in $\boldsymbol{\mathcal{W}}$, where the whole slice $\boldsymbol{\mathcal{W}}_{k,:,:}$ in the tensor formulation has to be zero to maintain feature sparsity both across all split nodes in each tree and across the trees in the ensemble — see Fig. 1. This is a natural feature-wise non-overlapping group structure and allows adaptation of the grouped selection problem in linear models (Bertsimas et al., 2016; Hazimeh et al., 2022) to soft tree ensembles.

**Mixed Integer Problem (MIP) Formulation.** Let us consider the tensor $\boldsymbol{\mathcal{W}}$. Using binary variables to model feature selection we obtain a regularized loss function:

$$\min_{\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}},\boldsymbol{z}} \hat{\mathbb{E}}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})] + \lambda_0 \sum_{k\in[p]} z_k, \quad (2)$$
$$\text{s.t. } ||\boldsymbol{\mathcal{W}}_{k,:,:}||(1 - z_k) = 0, z_k \in \{0, 1\} \quad k \in [p],$$

where, the binary variable $z_k$ controls whether the $k$-th feature is on or off via the constraint $||\boldsymbol{\mathcal{W}}_{k,:,:}||(1 - z_k) = 0$. $\hat{\mathbb{E}}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}))] := (1/N) \sum_{n\in[N]} L(\boldsymbol{y}_n, \boldsymbol{f}(\boldsymbol{x}_n; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}))$ is the empirical loss; and $\lambda_0$ is regularization strength. Note MIP formulations (Bertsimas and Dunn, 2019) can also be setup with classical trees under feature selection, but they would be difficult to scale beyond small problems.

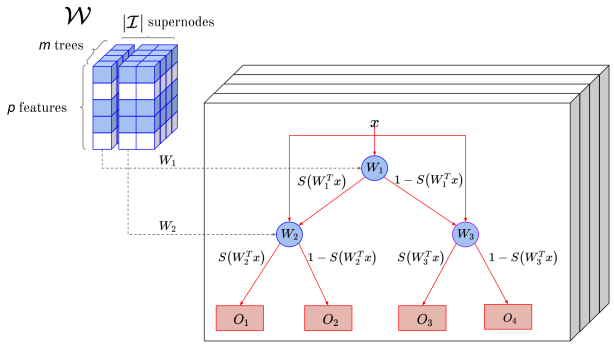


Figure 1: *Illustration of* ***Skinny Trees****. Each horizontal slice $\boldsymbol{\mathcal{W}}_{k,:,:}$ depicts a single feature. White slices indicate features filtered out by the ensemble while training. Each vertical slice (along the depth of the page), $\boldsymbol{\mathcal{W}}_{:,:,j} = \boldsymbol{W}_j$ corresponds to parameters in $j$-th (splitting) supernode (blue circles) in the ensemble, eventually producing the routing decisions. The red squares depict leaf nodes. $S(\cdot)$ denotes an activation function, which can be Sigmoid (Jordan and Jacobs, 1994) or Smooth-Step (Hazimeh et al., 2020).*

**Unconstrained formulation of Problem (2).** For computation we consider a penalized version of (2) involving variables $(\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})$ with the grouped $\ell_0$ (pseudo) norm encouraging feature sparsity. We perform end-to-end training via first-order methods (see Sec. 5 for details). It has been observed in the linear model setting that a vanilla (group) $\ell_0$ penalty may result in overfitting (Hazimeh et al., 2022). A possible way to ameliorate this problem is to include an additional ridge regularization for shrinkage (Mazumder et al., 2023; Hazimeh et al., 2022; Ibrahim et al., 2021). We consider the following group $\ell_0 - \ell_2$ regularized problem:

$$\min_{\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}}} \hat{\mathbb{E}}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}))] \quad (3)$$
$$+ \lambda_0 \sum\nolimits_{k\in[p]} 1[\boldsymbol{\mathcal{W}}_{k,:,:} \neq \mathbf{0}] + (\lambda_2/m|\mathcal{I}|)||\boldsymbol{\mathcal{W}}||_2^2$$

where $1[\cdot]$ is the indicator function, $\lambda_0 \geq 0$ controls the number of features selected, and $\lambda_2 \geq 0$ controls the amount of shrinkage of each group. We normalize $\lambda_2$ by the product $m|\boldsymbol{\mathcal{I}}|$ for convenience in hyperparameter tuning.

## 5 END-TO-END OPTIMIZATION APPROACH

We propose a fast approximate algorithm to obtain high-quality solutions to Problem (3). We use a proximal (mini-batch) gradient-based algorithm (Lan, 2012) that involves two operations. A vanilla mini-batch GD step is applied to all model parameters followed by a proximal operator applied to the hyperplane parameters $\boldsymbol{\mathcal{W}}$. This sequence of operations on top of backpropagation makes the procedure simple to implement in popular ML frameworks e.g. Tensorflow (Abadi et al., 2015), and contributes to overall efficiency.

### 5.1 Proximal mini-batch gradient descent

We first present the proximal mini-batch GD algorithm for solving Problem (3) in Algorithm 1. We also discuss computation of the *Prox* operator in line 7 of Algorithm 1. *Prox* finds the global minimum of the optimization problem:

$$\boldsymbol{\mathcal{W}}^{(t)} = \text{argmin}_{\boldsymbol{\mathcal{W}}} \ (1/2\eta)||\boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}^{(t)}||_2^2$$
$$+ \lambda_0 \sum\nolimits_{k\in[p]} 1[\boldsymbol{\mathcal{W}}_{k,:,:} \neq 0] \quad (4)$$

where $\boldsymbol{\mathcal{Z}}^{(t)} = \boldsymbol{\mathcal{W}}^{(t-1)} - \eta \nabla_{\boldsymbol{\mathcal{W}}} h$. Problem 4 decomposes across features and a solution for the $k$-th feature can be found by a hard-thresholding operator given by:

$$H_{\eta\lambda_0}(\boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:}) = \boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:} \odot 1\left[||\boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:}|| \geq \sqrt{2\eta\lambda_0}\right]. \quad (5)$$

We use feature-wise separability for faster computation. The cost of *Prox* is of the order $\mathcal{O}(v)$, where $v$ is the total number of hyperplane parameters being updated (i.e. $v = pm|\mathcal{I}|$). This cost is negligible compared to the computation of the gradients with respect to the same parameters. We implement the optimizer in standard deep learning APIs.

To our knowledge, our proposed approach (and algorithm) for group $\ell_0$-based nonlinear feature selection in soft tree ensembles is novel. Note that Chen et al. (2021) considers group $\ell_0$ based cardinality constrained formulation for feature selection in neural networks. However, their active-set style approach is very different from our iterative-hard-thresholding based approach. Their toolkit also appears to be up to 900× slower than our toolkit. In the context of neural network pruning, modifications of iterative-hard-thresholding based approaches (Jin et al., 2016; Peste et al., 2021) have appeared for individual weight pruning in neural networks which is different from feature-selection.

### 5.2 Convergence Analysis of Algorithm 1

In this section, we analyze the convergence properties of Algorithm 1. For simplicity, we assume that the outcomes are scalar, i.e. $c = 1$ and consider the least squares loss (see Assumption **(A2)** below). We also analyze the full-batch algorithm, i.e., we assume batch-size $= N$. Extending our result to vector outputs and a mini-batch algorithm would require appropriate changes, and is omitted here. Before stating our formal results, we discuss our assumptions on the model.

**(A1)** *(Activation function, $S(\cdot)$).* $S(\cdot)$ is a differentiable piece-wise polynomial function,

$$S(x) = \begin{cases} 1 & \text{if} \quad x > \theta \\ p(x) & \text{if} \quad -\theta \le x \le \theta \\ 0 & \text{if} \quad x < -\theta \end{cases}$$

for some $\theta > 0$ and a polynomial function $p(x)$. We assume the derivative of $S(x)$ is continuous.

**(A2)** *(Loss function, $L$).* We use the least squares loss $L(x, y) = (x - y)^2/2$.

**(A3)** *(Solutions).* There exists a numerical constant $B > 0$ only depending on the data, such that $\|\boldsymbol{\mathcal{O}}\|_2 \le B$ for all iterations in Algorithm 1.

Assumption **(A1)** encompasses a general class of activation functions, and includes the smooth-step function (Hazimeh et al., 2020) used in soft trees. The least squares loss in Assumption **(A2)** is a standard loss for regression problems. We consider the assumption on the boundedness of leaf weights in Assumption **(A3)** to be weak as the data is bounded[2].

Theorem 1 states our main result in this section:

**Theorem 1.** *Let $\lambda_2 > 0$ and suppose Assumptions **(A1)**, **(A2)** and **(A3)** hold. Then:*

1. *There is a sufficiently small $\eta > 0$ for which Algorithm 1 (using full-batch) is a descent algorithm with non-increasing objective values.*
2. *The sequence of hyperplane parameters $\boldsymbol{\mathcal{W}}$ generated from Algorithm 1 is bounded.*
3. *The sequence of parameters $\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}$ generated from Algorithm 1 converges if $\eta > 0$ is chosen as in Part 1.*

Theorem 1 shows that Algorithm 1 is a convergent (descent) method for a suitably selected learning rate. The proof of Theorem 1 is presented in Supplement S3. Here, we note that Problem (3) is non-convex and non-smooth. Moreover, the activation function and therefore $\hat{\mathbb{E}}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}))]$ are not twice differentiable everywhere. These lead to technical challenges in proof, as discussed in the Supplement.

### 5.3 Dense-to-Sparse Learning (DSL)

Prior work in feature selection (Lemhadri et al., 2021) recommend an interesting multi-stage approach: Train a dense model completely and then learn a progessively sparser model. At each sparsity level, the model is trained till convergence. This approach appears to effectively leverage favorable generalization properties

**Algorithm 1** Proximal Mini-batch Gradient Descent for Optimizing (3).

**Input:** Data: $X, Y$;
**Input:** Hyperparameters: $\lambda_0, \lambda_2$, epochs, batch-size, learning rate ($\eta$);
Initialize ensemble with $m$ trees of depth $d$ ($|\mathcal{I}| = 2^d - 1$): $\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}$
**for** epoch $= 1, 2, \ldots,$ epochs **do**
  **for** *batch* $= 1, \ldots, N/$batch-size **do**
    Randomly sample a batch: $\boldsymbol{X}_{batch}, \boldsymbol{Y}_{batch}$
    Compute gradient of loss $h$ w.r.t. $\boldsymbol{\mathcal{O}}$, $\mathbf{W}$.
    Update leaves: $\boldsymbol{\mathcal{O}} \leftarrow \boldsymbol{\mathcal{O}} - \eta \nabla_{\boldsymbol{\mathcal{O}}} h$
    Update hyperplanes: $\boldsymbol{\mathcal{W}} \leftarrow Prox(\boldsymbol{\mathcal{W}} - \eta \nabla_{\boldsymbol{\mathcal{W}}} h, \eta, \lambda_0))$
  **end for**
**end for**
where $h = \hat{\mathbb{E}}[L(\boldsymbol{Y}_{batch}, \boldsymbol{F}_{batch}] + (\lambda_2/m|\mathcal{I}|)||\boldsymbol{\mathcal{W}}||_2^2$

[2] Alternatively, one can add additional projection steps for leaf weights in Algorithm 1, ensuring the boundedness property, and remove Assumption **(A3)**

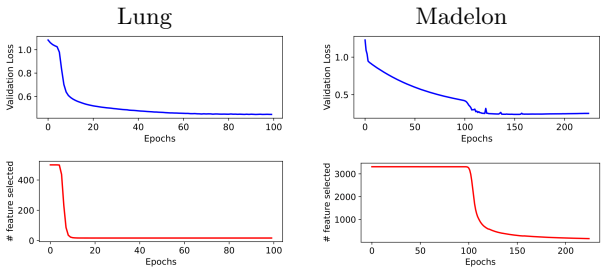


Figure 2: *Trajectory of validation loss and feature sparsity during training with dense-to-sparse learning.*

of the dense solution and preserves them after drifting into sparse local minima (Lemhadri et al., 2021)—in particular, this seems to perform better than sparsely training from scratch. However, this approach can be expensive as it requires learning a dense model completely before starting the sparse-training process—the training runtime is likely to increase for higher sparsity settings (i.e., fewer number of features).

To reduce the computational cost of the above approach, we propose single-stage approach based on dense-to-sparse learning (DSL). To this end, we anneal the sparsity-inducing penalty $\lambda_0$ from small to large values ($0 \to \gamma$) during the course of training. We use an exponential annealing scheduler of the form: $\lambda_0 = \gamma(1 - \exp(-s * t))$, where $\gamma$ is the largest value of regularization penalty (corresponds to a fully sparse soft tree), $s$ controls the rate of increase of the regularization penalty and $t$ denotes the iteration step.

We show the trajectory of the validation loss and the number of features selected during training with dense-to-sparse learning in Figure 2. We empirically observed this scheduler to result in better out-of-sample accuracy and feature-sparsity tradeoffs (see Figure 4 in Sec. 7.5).

## 6 SYNTHETIC EXPERIMENTS

We first evaluate our proposed method using data with correlated features. In real-world, high-dimensional datasets, features are often correlated. Such correlations pose challenges for feature selection. Existing tree ensemble toolkits, e.g., XGBoost and Random Forests, based on feature importance scores, may produce misleading results (Zhou and Hooker, 2021)—any of the correlated features can work as a splitting variable, and the feature importance scores can get distributed (and hence deflated) among the correlated features. Below, we consider a setting with correlated features and demonstrate the strong performance of `Skinny Trees` in terms of true support recovery on synthetic data.

Table 1: *Test MSE, feature sparsity and support recovery metrics (F1-score) for a linear setting with correlated design matrix.* ***Skinny Trees*** *outperforms feature-importance-based methods across all metrics.*

| $\sigma$ | $p$ | $N$ | Model | Test MSE | #features | F1-score |
|---|---|---|---|---|---|---|
| 0.7 | 512 | 100 | RF | 6.49 ± 0.19 | 79 ± 17 | 0.21 ± 0.02 |
| | | | XGBoost | 8.65 ± 0.27 | 32 ± 9 | 0.18 ± 0.03 |
| | | | Skinny Trees | **0.65** ± 0.12 | **12** ± 1 | **0.86** ± 0.04 |
| | | 200 | RF | 4.90 ± 0.15 | 40 ± 8 | 0.35 ± 0.03 |
| | | | XGBoost | 5.97 ± 0.12 | 110 ± 21 | 0.18 ± 0.02 |
| | | | Skinny Trees | **0.34** ± 0.00 | **11** ± 1 | **0.89** ± 0.03 |
| | | 1000 | RF | 2.97 ± 0.03 | 11 ± 1 | 0.84 ± 0.02 |
| | | | XGBoost | 1.81 ± 0.02 | 24 ± 1 | 0.50 ± 0.02 |
| | | | Skinny Trees | **0.26** ± 0.00 | **8** ± 0 | **1.00** ± 0.00 |
| 0.5 | 256 | 100 | RF | 6.24 ± 0.13 | 42 ± 11 | 0.35 ± 0.03 |
| | | | XGBoost | 7.93 ± 0.27 | 35 ± 10 | 0.25 ± 0.02 |
| | | | Skinny Trees | **0.45** ± 0.06 | **10** ± 1 | **0.89** ± 0.03 |
| | | 200 | RF | 4.40 ± 0.13 | 18 ± 3 | 0.61 ± 0.04 |
| | | | XGBoost | 5.61 ± 0.13 | 67 ± 12 | 0.26 ± 0.02 |
| | | | Skinny Trees | **0.31** ± 0.00 | **12** ± 1 | **0.87** ± 0.04 |
| | | 1000 | RF | 2.90 ± 0.02 | 9 ± 0 | 0.94 ± 0.01 |
| | | | XGBoost | 1.45 ± 0.01 | 10 ± 0 | 0.91 ± 0.01 |
| | | | Skinny Trees | **0.26** ± 0.00 | **8** ± 0 | **1.00** ± 0.00 |

We evaluate our approach in a setting where the underlying data comes from a sparse linear model. We generate the data matrix, $\boldsymbol{X} \in \mathbb{R}^{N \times p}$ with samples drawn from a multivariate normal distribution $\mathcal{N}(0, \boldsymbol{\Sigma})$ where entries of the covariance matrix $\boldsymbol{\Sigma}$ are given by $\Sigma_{ij} = \sigma^{|i-j|}$. We construct the response variable $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^* + \epsilon$ where $\epsilon_i, i \in [N]$ are drawn independently from $\mathcal{N}(0, 0.5)$. The locations of nonzero entries of $\boldsymbol{\beta}^*$ are equi-spaced in $[p]$, with each nonzero entry one, and $\|\boldsymbol{\beta}^*\|_0 = 8$. We experiment with a range of training set sizes $N \in \{100, 200, 1000\}$, correlation strengths $\sigma \in \{0.5, 0.7\}$, and number of total features $p \in \{256, 512\}$. We evaluate the final performance averaged across 25 runs in terms of (i) test MSE, (ii) number of features selected and (iii) support recovery (computed via the F1-score between the true and recovered support). More details are in Supplement Sec. S4.

`Skinny Trees` significantly outperforms both Random Forests and XGBoost in all three measures across various settings. With `Skinny Trees`, we observe a 5-15 fold improvement in MSE performance and $9\% - 65\%$ improvement in the support recovery metric (F1-score). Table 1 shows that even if the features are correlated, `Skinny Trees` successfully recovers the true support with high probability. We also visualize this in Figure 3. Indices corresponding to those in the true support are depicted in red. This confirms the usefulness of our end-to-end feature selection approach.

## 7 REAL DATA EXPERIMENTS

We study the performance of `Skinny Trees` on real-world datasets and compare against popular competing methods. We make the following comparisons:

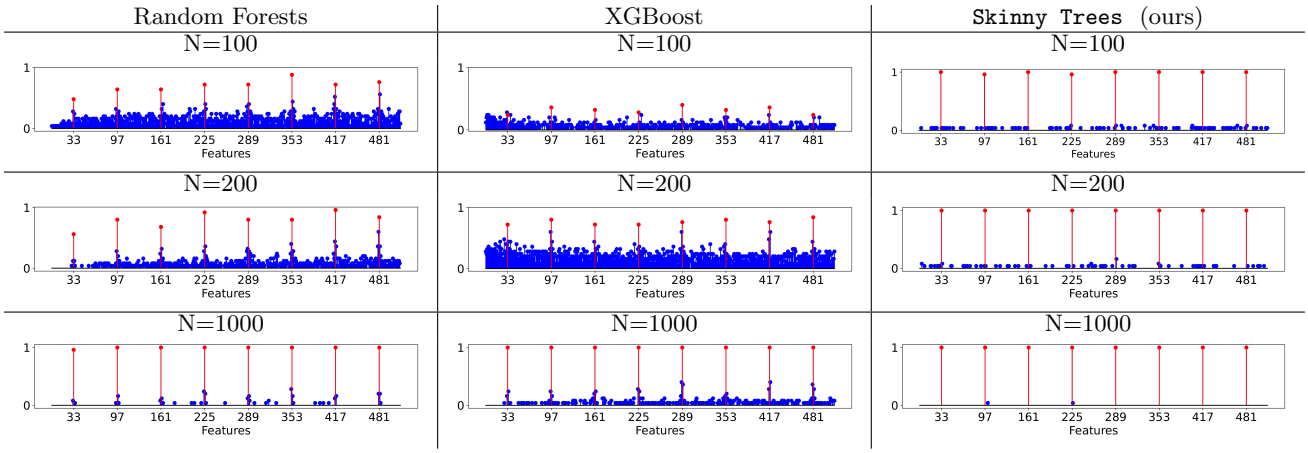


Figure 3: *Features selected by Random Forests, XGBoost and* ***Skinny Trees*** *for different sample sizes*

(i) Single `Skinny Tree` vs other single tree baseline approaches with a limit on number of features, (ii) `Skinny Trees` vs dense soft trees, (iii) `Skinny Trees` vs wrapper-based feature selection tree toolkits, (iv) `Skinny Trees` vs neural network based embedded feature selection toolkits, (v) Ablation study for dense-to-sparse learning for feature selection.

**Implementation.** `Skinny Trees` are implemented in TensorFlow Keras. Our code for `Skinny Trees` is available at
https://github.com/mazumder-lab/SkinnyTrees.

**Datasets.** We use 14 open-source classification datasets (binary and multiclass) from various domains with a range of number of features: $20 - 100000$. Dataset details are in Table S3 in Supplement.

**Tuning, Toolkits, and Details.** For all the experiments, we tune the hyperparameters using Optuna (Akiba et al., 2019) with random search. The number of selected features affects the AUC. Therefore, to treat all the methods in a fair manner, we tune the hyperparameter that controls the sparsity level using Optuna which optimizes the AUC across different $K$'s (budget on number of selected features) e.g., $0.25p$ or $0.50p$ on a held-out validation set. Details are in the Supplement.

### 7.1 Studying a single tree

We first study feature selection for a single tree on 4 classification tasks. We study the performance of `Skinny Tree` (a single soft tree with group $\ell_0 - \ell_2$ regularization).

**Competing Methods.** We compare against:

1. Decision tree with hyperplane splits (TAO (Carreira-Perpinan and Tavallali, 2018)) using $\ell_1$ regularization for node-level feature selection.
2. Soft Tree with a Group Lasso (Yuan and Lin, 2006; Scardapane et al., 2017) regularization given by $\frac{\lambda_1}{\sqrt{m|\mathcal{I}|}} \sum_{k\in[p]} \|\boldsymbol{W}_{k,:,:}\|_2$.

Table 2: *Test AUC for TAO with $\ell_1$ regularization, single soft tree with Group Lasso and* ***Skinny Tree*** *(a single soft tree with Group $\ell_0 - \ell_2$).*

| | Classical Tree TAO | Soft Tree w/ Group Lasso | `Skinny Tree` |
|---|---|---|---|
| Churn | 58.36 | 76.23 | **89.35**±0.15 |
| Satimage | 58.53 | 83.89 | **88.66**±0.05 |
| Texture | 58.90 | 93.83 | **98.42**±0.01 |
| Mice-protein | 57.13 | 87.88 | **99.19**±0.00 |

**Results.** The numbers for classical-tree based TAO with $\ell_1$ regularization and soft tree with Group Lasso regularization are shown in Table 2 for 50% sparsity budget. Results for `Skinny Tree` are also shown. We see a huge gain in test AUC performance across all 4 datasets with `Skinny Tree` in comparison with TAO and group lasso variant of a soft tree. This confirms

Table 3: Test AUC for `Skinny Trees` vs *dense* Soft Trees. We also report feature compression.

| | Dense Trees | `Skinny Trees` | Compression |
|---|---|---|---|
| Churn | 91.15±0.09 | **93.20**±0.08 | 1.8× |
| Gisette | **99.81**±0.003 | **99.81**±0.002 | 1.5× |
| Arcene | 89.57±0.11 | **90.80**±0.30 | 2× |
| Dorothea | 90.67±0.03 | **92.15**±0.25 | 2.7× |
| Madelon | 65.32±0.15 | **95.44**±0.05 | 26× |
| Smk | **84.10**±0.16 | 79.29±0.22 | 253× |
| Cll | 81.70±0.82 | **92.86**±0.31 | 189× |
| Gli | 88.65±0.90 | **99.80**±0.07 | 619× |
| Lung | 99.40±0.09 | **99.80**±0.03 | 253× |
| Tox | 99.19±0.04 | **99.74**±0.02 | 189× |

Table 4: *Test AUC (%) performance of* ***Skinny Trees*** *and feature-importance-based toolkits for* ***trees*** *for* 25% *feature budget* ($K = 0.25p$). *Bold and italics indicates best and runner-up models respectively.*

| Case | Dataset | Random Forests | XGBoost | LightGBM | CatBoost | Skinny Trees |
|---|---|---|---|---|---|---|
| $N < p$ | Lung | 93.80±0.28 | 86.38±0.48 | 80.83±1.87 | *94.72*±0.56 | **99.80**±0.03 |
| | Tox | 94.52±0.14 | *97.10*±0.09 | 95.94±0.54 | 95.95±0.14 | **99.74**±0.02 |
| | Arcene | 74.80±0.36 | 76.36±0.16 | *76.92*±0.36 | 76.64±0.22 | **80.80**±0.30 |
| | Cll | 94.08±0.27 | *94.21*±0.18 | 55.17±1.14 | **94.41**±0.26 | 92.86±0.31 |
| | Smk | 77.78±0.20 | 76.88±0.40 | 67.29±0.91 | *78.44*±0.41 | **79.29**±0.22 |
| | Gli | 87.35±1.08 | 82.37±1.47 | 71.28±2.05 | *91.31*±0.73 | **99.80**±0.07 |
| | Dorothea | *89.71*±0.12 | 89.09±0.09 | 88.14±0.18 | 88.50±0.27 | **90.87**±0.02 |
| $N > p$ | Churn | 83.79±0.24 | *88.68*±0.06 | 86.33±0.08 | 83.73±0.06 | **91.38**±0.08 |
| | Satimage | 97.62±0.005 | **98.23**±0.01 | 94.00±0.05 | 95.11±0.05 | *98.05*±0.01 |
| | Texture | 99.60±0.003 | *99.94*±0.001 | 96.14±0.03 | 94.90±0.07 | **99.97**±0.002 |
| | Mice-protein | 99.30±0.01 | **99.77**±0.01 | 89.59±0.22 | 95.03±0.07 | *99.59*±0.02 |
| | Isolet | 99.17±0.002 | 99.86±0.002 | 97.62±0.003 | *99.89*±0.001 | **99.94**±0.01 |
| | Madelon | 94.11±0.02 | *94.65*±0.01 | 86.46±0.08 | **96.41**±0.01 | 94.14±0.09 |
| | Gisette | 98.99±0.004 | *99.64*±0.004 | 98.09±0.50 | 99.57±0.01 | **99.81**±0.002 |
| | Average | 91.75 | 91.65 | 84.56 | 91.76 | **94.72** |

that in the context of feature selection at the ensemble level, a node-level $\ell_1$ penalty is sub-optimal. Similarly, it also suggests that joint selection and shrinkage using Group Lasso can be less useful than Group $\ell_0 - \ell_2$.

### 7.2 Skinny Trees vs Dense Soft Trees

In this section, we compare our sparse trees with dense soft trees. For dense soft trees, we use FASTEL (Ibrahim et al., 2022) (an efficient state-of-the-art toolkit for training soft tree ensembles). We present test AUC performances in Table 3. Skinny Trees matches or outperforms dense soft trees in 10 datasets. Notably, we observe a 30% gain in test AUC on Madelon dataset with Skinny Trees. We also observe 11% improvements in test AUC on Cll and Gli datasets. Additionally, sparse trees achieve $1.3\times - 620\times$ feature compression on 10 datasets. Note that in soft trees, feature compression has a direct impact on model compression—this has reduced storage requirements and results in faster inference. We observed up to $10\times$ faster inference times for Skinny Trees compared to dense soft trees for compression rates of $1.5\times - 620\times$.

### 7.3 Skinny Trees vs Classical Trees

We compare Skinny Trees against wrapper methods for feature selection as available from ensembles of classical trees (e.g., Random Forests, XGBoost, LightGBM, and CatBoost) on real-world datasets. For Skinny Trees, we use the combined dense-to-sparse scheduler. The tuning protocol and hyperparameters for all methods are reported in the Supplement Sec. S5.3. The results are in Table 4. Skinny Trees leads on 10 datasets. In contrast, other methods lead on 2 datasets. In terms of test AUC, Skinny Trees outperforms LightGBM by 10.2% (up to 37.7%), XGBoost by 3.1% (upto 17.4%), Random Forests by 3% (up to 12.5%) and CatBoost by 3% (up to 8.5%). Overall, Skinny Trees provides a strong alternative to existing wrapper-based methods.

Additional comparison with ControlBurn (Liu et al., 2021) is included in Supplement Sec. S5.6. Skinny Trees also outperforms ControlBurn, achieving 2% (up to 6%) improvement in AUC.

### 7.4 Skinny Trees vs Neural Networks

In this paper, we pursue embedded feature selection methods for *tree ensembles*. However, for completeness, we compare Skinny Trees against some state-of-the-art embedded feature selection methods from neural networks, namely LassoNet (Lemhadri et al., 2021), AlgNet (Dinh and Ho, 2020) and DFS (Chen et al., 2021). Details are in Supplement Sec. S5.4.

**Results.** We report AUC performance for 25% feature budget in Table 5. Skinny Trees leads across many datasets. In terms of test AUC, Skinny Trees outperforms LassoNet by 4.3% (up to 24%), AlgNet by 25.9% (up to 49%), and DFS by 2.6% (up to 11.4%).

### 7.5 Dense-to-Sparse Learning

We perform an ablation study in which we compare the predictive performance achieved with dense-to-sparse learning (DSL) over a range of feature selection budgets. Tuning details are in the Supplement Sec. S5.5. The results are reported in Figure 4. Interestingly, we improve in test AUC across a range of feature selection budgets with dense-to-sparse learning over fixed regularization tuning.

Table 5: *Test AUC (%) performance of* ***Skinny Trees*** *and embedded feature selection methods from* ***neural networks*** *(LassoNet, AlgNet, DFS) for* 25% *feature budget.*

| Case | Dataset | LassoNet | AlgNet | DFS | Skinny Trees |
|---|---|---|---|---|---|
| $N < p$ | Lung | 99.56±0.02 | 56.72±1.34 | *98.05*±0.36 | **99.80**±0.03 |
| | Tox | 99.63±0.03 | 51.01±0.70 | *99.13*±0.24 | **99.74**±0.02 |
| | Arcene | 66.26±0.29 | 51.00±1.77 | 69.37±0.59 | **80.80**±0.30 |
| | Cll | **95.04**±0.24 | 64.96±2.54 | 92.85±0.32 | *92.86*±0.31 |
| | Smk | **85.56**±0.31 | 53.92±2.16 | *79.62*±0.29 | 79.29±0.22 |
| | Gli | *97.78*±0.56 | 61.37±4.27 | 92.25±0.64 | **99.80**±0.07 |
| | Dorothea | out of mem. | 81.74±0.91 | *85.18** | **90.87**±0.02 |
| $N > p$ | Churn | 67.34±1.58 | 70.10±0.91 | *85.70*±0.52 | **91.38**±0.08 |
| | Satimage | 94.73±0.19 | 95.30±0.20 | *97.39*±0.04 | **98.05**±0.01 |
| | Texture | 98.02±0.40 | 76.24±1.94 | *99.63*±0.04 | **99.97**±0.002 |
| | Mice-protein | 94.90±0.26 | 89.07±0.59 | *99.04*±0.03 | **99.59**±0.02 |
| | Isolet | 99.64±0.01 | 70.21±2.92 | *99.92*±0.00 | **99.94**±0.01 |
| | Madelon | 81.15±2.53 | 68.55±1.42 | *92.73*±0.45 | **94.14**±0.09 |
| | Gisette | 99.81±0.002 | 73.49±1.54 | *99.72** | **99.81**±0.002 |
| | Average | 90.43** | 68.83 | 92.18 | **94.72** |

*DFS is very time-consuming to run, we report the test AUC for best trial (based on validation AUC) during tuning on Gisette and Dorothea.

**Adjusted Average: $\frac{90.72}{(94.72*14-90.87)/13} * 94.72 = 90.43$.

### 7.6 Discussion on training times.

`Skinny Trees` is very competitive in terms of training times in comparison to existing toolkits. We compared timings on a single Tesla V100 GPU. For example, on dorothea dataset, `Skinny Trees` trained in under 3 minutes for optimal hyperparameter setting. XGBoost took 10 minutes. In contrast, DFS took 45 hours.

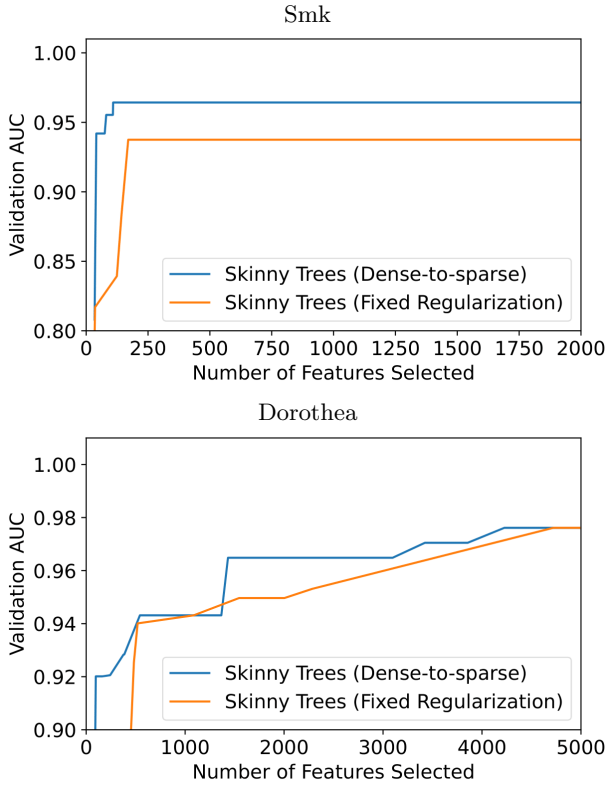


Figure 4: Performance without/with Dense-to-sparse learning for different feature selection budgets.

## 8 CONCLUSION

We introduce an end-to-end optimization approach for joint feature selection and tree ensemble learning. Our approach is based on differentiable trees with group $\ell_0 - \ell_2$ regularization. We use a simple but effective proximal mini-batch gradient descent algorithm and present convergence guarantees. We propose a dense-to-sparse regularization scheduling approach that can lead to better feature-sparsity-vs-accuracy tradeoffs. We demonstrate on various datasets that our toolkit `Skinny Trees` can improve feature selection over several state-of-the-art wrapper-based feature selection methods in trees and embedded feature selection methods in neural networks.

## 9 Acknowledgments

This research was supported in part, by grants from the Office of Naval Research (N000142112841), and Liberty Mutual Insurance. The authors acknowledge the MIT SuperCloud (Reuther et al., 2018) and Lincoln Laboratory for providing HPC resources that have contributed to the research reported within this paper.

## References


Martín Abadi, Ashish Agarwal, Paul Barham, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, et al. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Genevera I. Allen. Automatic feature selection via weighted kernels and regularization. *Journal of Computational and Graphical Statistics*, 22(2):284–299, 04 2013.

Hedy Attouch, Jérôme Bolte, and Benar Fux Svaiter. Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized gauss–seidel methods. *Mathematical Programming*, 137(1-2):91–129, 2013.

R. Battiti. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks*, 5(4):537–550, 1994. doi: 10.1109/72.298224.

D. Bertsimas and J. Dunn. *Machine Learning Under a Modern Optimization Lens*. Dynamic Ideas LLC, 2019.

Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2):813–852, April 2016. doi: 10.1214/15-aos1388.

Jérôme Bolte, Aris Daniilidis, and Adrian Lewis. The łojasiewicz inequality for nonsmooth subanalytic functions with applications to subgradient dynamical systems. *SIAM Journal on Optimization*, 17(4):1205–1223, 2007.

A.-L. Boulesteix, A. Bender, J. Lorenzo Bermejo, and C. Strobl. Random forest gini importance favours SNPs with large minor allele frequency: impact, sources and recommendations. *Briefings in Bioinformatics*, 13(3):292–304, September 2011. doi: 10.1093/bib/bbr053. URL https://doi.org/10.1093/bib/bbr053.

L. Breiman, J. Friedman, C.J. Stone, and R.A. Olshen. *Classification and Regression Trees*. Taylor & Francis, 1984. ISBN 9780412048418.

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

Jie Cai, Jiawei Luo, Shulin Wang, and Sheng Yang. Feature selection in machine learning: A new perspective. *Neurocomputing*, 300:70–79, 2018.

Miguel A. Carreira-Perpinan and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Girish Chandrashekar and Ferat Sahin. A survey on feature selection methods. *Computers & Electrical Engineering*, 40(1):16–28, 2014. ISSN 0045-7906. doi: https://doi.org/10.1016/j.compeleceng.2013.11.024. 40th-year commemorative issue.

Jianbo Chen, Mitchell Stern, Martin J Wainwright, and Michael I Jordan. Kernel feature selection via conditional covariance minimization. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322.

Yao Chen, Qingyi Gao, Faming Liang, et al. Nonlinear variable selection via deep neural networks. *Journal of Computational and Graphical Statistics*, 30(2):484–492, 2021.

V. Dinh and L. S. T. Ho. Consistent feature selection for analytic deep neural networks. In *NIPS'20*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

L. P. D. van den Dries. *SEMIALGEBRAIC SETS*, page 31–42. London Mathematical Society Lecture Note Series. Cambridge University Press, 1998. doi: 10.1017/CBO9780511525919.006.

Chandra Erdman and Nancy Bates. The low response score (LRS). *Public Opinion Quarterly*, 81(1):144–156, December 2016.

Pablo A. Estevez, Michel Tesmer, Claudio A. Perez, and Jacek M. Zurada. Normalized mutual information feature selection. *IEEE Transactions on Neural Networks*, 20(2):189–201, 2009. doi: 10.1109/TNN.2008.2005601.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, jun 2008. ISSN 1532-4435.

Jean Feng and Noah Simon. Sparse-input neural networks for high-dimensional nonparametric regression and classification. *arXiv: Methodology*, 2017.

Nicholas Frosst and Geoffrey Hinton. Distilling a neural network into a soft decision tree. *ArXiv*, abs/1711.09784, 2017.

Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Result analysis of the nips 2003 feature selection challenge. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004.

T. J. Hastie, R. J. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2 edition, 2009.

H. Hazimeh, N. Ponomareva, P. Mol, et al. The tree ensemble layer: Differentiability meets conditional computation. In Hal Daumé III and Aarti Singh, editors, *ICML'20*, volume 119 of *Proceedings of Machine Learning Research*, pages 4138–4148. PMLR, 13–18 Jul 2020.

Hussein Hazimeh and Rahul Mazumder. Fast best subset selection: Coordinate descent and local combinatorial optimization algorithms. *Oper. Res.*, 68(5): 1517–1537, September 2020. ISSN 0030-364X. doi: 10.1287/opre.2019.1919.

Hussein Hazimeh, Rahul Mazumder, and Peter Radchenko. Grouped variable selection with discrete optimization: Computational and statistical perspectives. *arXiv preprint arXiv:2104.07084 (Annals of Statistics, to appear)*, 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Thomas M. Hehn, Julian F. P. Kooij, and Fred A. Hamprecht. End-to-end learning of decision trees and forests. *International Journal of Computer Vision*, 128:997–1011, 2019.

Shibal Ibrahim, Rahul Mazumder, Peter Radchenko, and Emanuel Ben-David. Predicting census survey response rates with parsimonious additive models and structured interactions. *arXiv*, abs/2108.11328, 2021.

Shibal Ibrahim, Hussein Hazimeh, and Rahul Mazumder. Flexible modeling and multitask learning using differentiable tree ensembles. In *KDD'22*, KDD '22, page 666–675, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393850. doi: 10.1145/3534678.3539412.

Shibal Ibrahim, Gabriel Afriat, Kayhan Behdin, and Rahul Mazumder. GRAND-SLAMIN' interpretable additive modeling with structural constraints. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023a. URL https://openreview.net/forum?id=F5DYsAc7Rt.

Shibal Ibrahim, Wenyu Chen, Hussein Hazimeh, Natalia Ponomareva, Zhe Zhao, and Rahul Mazumder. Comet: Learning cardinality constrained mixture of experts with trees and local search. KDD '23, page 832–844, New York, NY, USA, 2023b. Association for Computing Machinery. ISBN 9798400701030. doi: 10.1145/3580305.3599278.

Xiaojie Jin, Xiaotong Yuan, Jiashi Feng, and Shuicheng Yan. Training skinny deep neural networks with iterative hard thresholding methods, 2016.

Michael I. Jordan and Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Comput.*, 6(2):181–214, mar 1994. ISSN 0899-7667.

Guolin Ke, Qi Meng, Thomas Finley, et al. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, et al., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Rajiv Khanna, Ethan Elenberg, Alex Dimakis, Sahand Negahban, and Joydeep Ghosh. Scalable Greedy Feature Selection via Weak Submodularity. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1560–1568. PMLR, 20–22 Apr 2017. URL https://proceedings.mlr.press/v54/khanna17b.html.

Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273–324, 1997.

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, et al. Deep neural decision forests. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1467–1475, 2015.

Krzysztof Kurdyka. On gradients of functions definable in o-minimal structures. *Annales de l'institut Fourier*, 48(3):769–783, 1998. URL http://eudml.org/doc/75302.

Guanghui Lan. An optimal method for stochastic composite optimization. *Mathematical Programming*, 133(1):365–397, 2012.

Ismael Lemhadri, Feng Ruan, and Rob Tibshirani. Lassonet: Neural networks with feature sparsity. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 10–18. PMLR, 13–15 Apr 2021.

Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu. Feature selection: A data perspective.

*ACM Comput. Surv.*, 50(6), dec 2017. ISSN 0360-0300. doi: 10.1145/3136625.

Brian Liu, Miaolan Xie, and Madeleine Udell. Controlburn: Feature selection by sparse forests. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '21, page 1045–1054, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383325. doi: 10.1145/3447548.3467387.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l0 regularization. In *International Conference on Learning Representations*, 2018.

Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

Rahul Mazumder, Peter Radchenko, and Antoine Dedieu. Subset selection with shrinkage: Sparse linear modeling when the snr is low. *Operations Research*, 71(1):129–147, 2023.

Fan Min, Qinghua Hu, and William Zhu. Feature selection with test cost constraint. *International Journal of Approximate Reasoning*, 55(1, Part 2):167–179, 2014. ISSN 0888-613X. doi: https://doi.org/10.1016/j.ijar.2013.04.003. Special issue on Decision-Theoretic Rough Sets.

Md. Monirul Kabir, Md. Monirul Islam, and Kazuyuki Murase. A new wrapper feature selection approach using neural network. *Neurocomputing*, 73(16): 3273–3283, 2010.

Sreerama K. Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *J. Artif. Int. Res.*, 2(1):1–32, aug 1994. ISSN 1076-9757.

Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1): 36, 2017.

V. Onnia, M. Tico, and J. Saarinen. Feature selection method using neural network. In *Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205)*, volume 1, pages 513–516 vol.1, 2001. doi: 10.1109/ICIP.2001.959066.

Hanchuan Peng, Fuhui Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005. doi: 10.1109/TPAMI.2005.159.

Alexandra Peste, Eugenia Iofinova, Adrian Vladu, and Dan Alistarh. AC/DC: Alternating compressed/decompressed training of deep neural networks. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=T3_AJr9-R5g.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, et al. Catboost: Unbiased boosting with categorical features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 6639–6649, Red Hook, NY, USA, 2018. Curran Associates Inc.

J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1558602380.

P. Ravikumar, J. Lafferty, H. Liu, et al. Sparse additive models. *Journal of the Royal Statistical Society, B.*, 71:1009–1030, 2009.

Juha Reunanen. Overfitting in making comparisons between variable selection methods. *J. Mach. Learn. Res.*, 3(null):1371–1382, mar 2003. ISSN 1532-4435.

Albert Reuther, Jeremy Kepner, Chansup Byun, Siddharth Samsi, William Arcand, David Bestor, Bill Bergeron, Vijay Gadepally, Michael Houle, Matthew Hubbell, Michael Jones, Anna Klein, Lauren Milechin, Julia Mullen, Andrew Prout, Antonio Rosa, Charles Yee, and Peter Michaleas. Interactive supercomputing on 40,000 cores for machine learning and data analysis. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–6. IEEE, 2018.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?": Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1135–1144, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939778.

Debaditya Roy, K. Sri Rama Murty, and C. Krishna Mohan. Feature selection using deep neural networks. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6, 2015. doi: 10.1109/IJCNN.2015.7280626.

Simone Scardapane, Danilo Comminiello, Amir Hussain, and Aurelio Uncini. Group sparse regularization for deep neural networks. *Neurocomputing*, 241: 81–89, 2017.

Shubham Sharma, Sanghamitra Dutta, Emanuele Albini, Freddy Lecue, Daniele Magazzeni, and Manuela Veloso. Refresh: Responsible and efficient feature reselection guided by shap values. In

*Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '23, page 443–453, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702310. doi: 10.1145/3600211.3604706. URL https://doi.org/10.1145/3600211.3604706.

Le Song, Alex Smola, Arthur Gretton, Karsten M. Borgwardt, and Justin Bedo. Supervised feature selection via dependence estimation. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 823–830, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595937933. doi: 10.1145/1273496.1273600.

Le Song, Alex Smola, Arthur Gretton, Justin Bedo, and Karsten Borgwardt. Feature selection via dependence maximization, 2012.

Gary Stein, Bing Chen, Annie S. Wu, and Kien A. Hua. Decision tree classifier for network intrusion detection with ga-based feature selection. In *Proceedings of the 43rd Annual Southeast Regional Conference - Volume 2*, ACM-SE 43, page 136–141, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930590. doi: 10.1145/1167253.1167288.

Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, 8(1):25, 2007.

Ryutaro Tanno, Kai Arulkumaran, Daniel C. Alexander, Antonio Criminisi, and Aditya V. Nori. Adaptive neural trees. *ArXiv*, abs/1807.06699, 2019.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 2022/05/03 1996.

Julia D. Wulfkuhle, Lance A. Liotta, and Emanuel F. Petricoin. Proteomic applications for the early detection of cancer. *Nature Reviews Cancer*, 3(4):267–275, 2003.

Yutaro Yamada, Ofir Lindenbaum, Sahand Negahban, and Yuval Kluger. Feature selection using stochastic gates. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 10648–10659. PMLR, 13–18 Jul 2020.

M. Yuan and Y. Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society, Series B*, 68:49–67, 2006.

Shichao Zhang. Cost-sensitive classification with respect to waiting cost. *Know.-Based Syst.*, 23(5):369–378, jul 2010. ISSN 0950-7051. doi: 10.1016/j.knosys.2010.01.008. URL https://doi.org/10.1016/j.knosys.2010.01.008.

Tuo Zhao and Han Liu. Sparse additive machine. In Neil D. Lawrence and Mark Girolami, editors, *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 1435–1443, La Palma, Canary Islands, 21–23 Apr 2012. PMLR.

Arman Zharmagambetov, Suryabhan Singh Hada, and Miguel Á. Carreira-Perpiñán. An experimental comparison of old and new decision tree algorithms. *CoRR*, abs/1911.03054, 2019.

Arman Serikuly Zharmagambetov and Miguel Á. Carreira-Perpiñán. Smaller, more accurate regression forests using tree alternating optimization. In *ICML*, 2020.

Zhengze Zhou and Giles Hooker. Unbiased measurement of feature importance in tree-based methods. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15:1 – 21, 2021.

Zexuan Zhu, Yew-Soon Ong, and Manoranjan Dash. Wrapper–filter feature selection algorithm using a memetic framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(1):70–76, 2007. doi: 10.1109/TSMCB.2006.883267.

## Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Code is available at https://github.com/mazumder-lab/SkinnyTrees]

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]
   (b) Complete proofs of all theoretical results. [Yes]
   (c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Code is available at https://github.com/mazumder-lab/SkinnyTrees]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator, if your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

# SUPPLEMENTARY MATERIAL

## S1 NOTATIONS AND ACRONYMS

**Notation** Table S1 lists the notation used throughout the paper.

Table S1: List of notation used.

| Notation | Space or Type | Explanation |
|---|---|---|
| $[n]$ | Set | The set of integers $\{1, 2, ...., n\}$. |
| $\mathbf{1}_m$ | $\mathbb{R}^m$ | Vector with all coordinates equal to 1. |
| $\boldsymbol{U}$ | $\mathbb{R}^{m,n}$ | Matrix with elements $((U_{ij}))$ |
| $\boldsymbol{u} \cdot \boldsymbol{v}$ | $\mathbb{R}$ | A dot product between two vectors $\boldsymbol{u}, \boldsymbol{v}$. |
| $\boldsymbol{U} \cdot \boldsymbol{v}$ | $\mathbb{R}^n$ | A dot product between a matrix $\boldsymbol{U} \in R^{m,n}$ and a vector $\boldsymbol{v} \in \mathbb{R}^m$ is denoted as $\boldsymbol{U} \cdot \boldsymbol{v} = \boldsymbol{U}^T v \in \mathbb{R}^n$. |
| $\mathcal{X}$ | $\mathbb{R}^p$ | Input feature space. |
| $\mathcal{Y}$ | $\mathbb{R}^c$ | Output (label) space. |
| $m$ | $\mathbb{Z}_{>0}$ | Number of trees in `Skinny Trees`. |
| $\boldsymbol{f}(\boldsymbol{x})$ | Function | The output of `Skinny Trees`, a function that takes an input sample and returns a logit which corresponds to the sum of all the trees in the ensemble. Formally, $\boldsymbol{f} : \mathcal{X} \to \mathbb{R}^c$. |
| $\boldsymbol{f}^j(\boldsymbol{x})$ | Function | A single perfect binary tree which takes an input sample and returns a logit, i.e., $\boldsymbol{f}^j : \mathcal{X} \to \mathbb{R}^c$. |
| $d$ | $\mathbb{Z}_{>0}$ | The depth of tree $\boldsymbol{f}^j$. |
| $\mathcal{I}^j$ | Set | The set of internal (split) nodes in $\boldsymbol{f}^j$. |
| $\mathcal{I}$ | Set | The set of internal (split) supernodes in $\boldsymbol{f}$. |
| $\mathcal{L}^j$ | Set | The set of leaf nodes in $\boldsymbol{f}^j$. |
| $\mathcal{A}(i)$ | Set | The set of ancestors of node $i$. |
| $\{x \to i\}$ | Event | The event that sample $x \in \mathbb{R}^p$ reaches node $i$. |
| $\boldsymbol{w}_i$ | $\mathbb{R}^p$ | Weight vector of internal node $i$ (trainable). Defines the hyperplane split used in sample routing. |
| $\boldsymbol{W}_i$ | $\mathbb{R}^{p,m}$ | Matrix of all weights in the internal supernode $i$ of the ensemble in the tensor-formulation. |
| $\boldsymbol{\mathcal{W}}$ | $\mathbb{R}^{p,m,\lvert\mathcal{I}\rvert}$ | Tensor of all weights across all internal supernodes in the ensemble. |
| $\boldsymbol{\mathcal{W}}_{k,:,:}$ | $\mathbb{R}^{m,\lvert\mathcal{I}\rvert}$ | Matrix of all weights for $k$-th feature/covariate across all internal supernodes in the ensemble. |
| $S$ | Function | Activation function $\mathbb{R} \to [0, 1]$ |
| $S(\boldsymbol{w}_i \cdot \boldsymbol{x})$ | $[0, 1]$ | Probability (proportion) that internal node $i$ routes $x$ to the left. |
| $S'(v)$ | Function | The derivative of $S(v)$ |
| $[l \swarrow i]$ | Event | The event that leaf $l$ belongs to the left subtree of node $i \in \mathcal{I}$. |
| $[l \searrow i]$ | Event | The event that leaf $l$ belongs to the right subtree of node $i \in \mathcal{I}$. |
| $\boldsymbol{o}_l$ | $\mathbb{R}^c$ | Leaf $l$'s weight vector (trainable). |
| $\boldsymbol{O}_l$ | $\mathbb{R}^{m,c}$ | Matrix of weights in superleaf $l$. |
| $\boldsymbol{\mathcal{O}}$ | $\mathbb{R}^{m,c,\lvert\mathcal{L}\rvert}$ | Tensor of weights across the superleaves in the ensemble. |
| $L$ | Function | Loss function for training (e.g., cross-entropy). |
| $\mathcal{Q}$ | Set | Unknown (learnable) subset of features of size at most $K$. |
| $z_k$ | $\{0, 1\}$ | Binary variable controls whether $k$-th feature is on or off in Problem (2). |
| $\boldsymbol{z}$ | $\{0, 1\}^p$ | Binary vector controlling which features are on or off in Problem (2). |
| $\lambda_0$ | $\mathbb{R}_{\geq 0}$ | Non-negative $\ell_0$ regularization parameter controlling the number of features selected in Problem (3) |
| $\lambda_2$ | $\mathbb{R}_{\geq 0}$ | Non-negative $\ell_2$ regularization parameter controlling the shrinkage in Problem (3) |
| $\lambda_1$ | $\mathbb{R}_{\geq 0}$ | Non-negative $\ell_0$ regularization parameter controlling the number of features selected and shrinkage in Problem (S19) |
| $\gamma$ | $\mathbb{R}_{\geq 0}$ | Non-negative scaling parameter for the exponential ramp-up of $\ell_0$-penalty in dense-to-sparse learning. |
| $s$ | $\mathbb{R}_{\geq 0}$ | Non-negative temperature parameter for controlling the ramping rate of the $\ell_0$-penalty in dense-to-sparse learning. |
| $\eta$ | $\mathbb{R}_{\geq 0}$ | Learning rate parameter for proximal mini-batch gradient descent. |

**Acronyms** Table S2 lists the acronyms used throughout the paper.

Table S2: List of Acronyms used.

| Terms | Acronyms |
|---|---|
| Gradient Descent | GD |
| Dense-to-sparse learning | DSL |

## S2 BACKGROUND: DIFFERENTIABLE (A.K.A. SOFT) DECISION TREES

A soft tree is a variant of a classical decision tree that performs soft routing, i.e., a sample is fractionally routed to all leaves. It was proposed by Jordan and Jacobs (1994), and further developed in (Kontschieder et al., 2015; Hazimeh et al., 2020) for end-to-end optimization. Soft routing makes soft trees differentiable, so learning can be done using gradient-based methods.

Let us fix some $j \in [m]$ and consider a single tree $\boldsymbol{f}^j$, which takes an input sample and returns an output vector (logit), i.e., $\boldsymbol{f}^j : X \in \mathbb{R}^p \rightarrow \mathbb{R}^c$. Moreover, we assume that $\boldsymbol{f}^j$ is a perfect binary tree with depth $d$. Let $\mathcal{I}^j$ and $\mathcal{L}^j$ denote the sets of internal (split) nodes and the leaves of the tree, respectively. For any node $i \in \mathcal{I}^j \cup \mathcal{L}^j$, we define $A^j(i)$ as its set of ancestors and use the notation $\boldsymbol{x} \rightarrow i$ for the event that a sample $\boldsymbol{x} \in \mathbb{R}^p$ reaches $i$. See Table S1 for detailed notation summary.

**Routing** Following existing work (Kontschieder et al., 2015; Hehn et al., 2019; Hazimeh et al., 2020), we present routing in soft trees with a probabilistic model. Although the sample routing is formulated with a probabilistic model, the final prediction of the tree $\boldsymbol{f}$ is a deterministic function as it assumes an expectation over the leaf predictions. According to this probabilistic model, internal (split) nodes in a soft tree perform soft routing, where a sample is routed left and right with different probabilities. Classical decision trees are modeled with either axis-aligned splits (Breiman et al., 1984; Quinlan, 1993) or hyperplane (a.k.a. oblique) splits (Murthy et al., 1994). Soft trees are based on hyperplane splits, where the routing decisions rely on a linear combination of the features. Particularly, each internal node $i \in \mathcal{I}^j$ is associated with a trainable weight vector $\boldsymbol{w}_i^j \in \mathbb{R}^p$ that defines the node's hyperplane split.

Given a sample $\boldsymbol{x} \in \mathbb{R}^p$, the probability that internal node $i$ routes $\boldsymbol{x}$ to the left is defined by $S(\boldsymbol{w}_i^j \cdot \boldsymbol{x})$. Now we discuss how to model the probability that $\boldsymbol{x}$ reaches a certain leaf $l$. Let $[l \swarrow i]$ (resp. $[i \searrow l]$) denote the event that leaf $l$ belongs to the left (resp. right) subtree of node $i \in \mathcal{I}^j$. The probability that $\boldsymbol{x}$ reaches $l$ is given by $P^j(\{x \rightarrow l\}) = \prod_{i \in A(l)} r_{i,l}^j(\boldsymbol{x})$, where $r_{i,l}^j(\boldsymbol{x})$ is the probability of node $i$ routing $\boldsymbol{x}$ towards the subtree containing leaf $l$, i.e., $r_{i,l}^j(x) := S(\boldsymbol{w}_i^j \cdot \boldsymbol{x})^{1[l \swarrow i]} \odot (1 - S(\boldsymbol{w}_i^j \cdot \boldsymbol{x}))^{1[i \searrow l]}$. Let $S : R \rightarrow [0, 1]$ be an activation function. Popular choices for $S$ include logistic function (Jordan and Jacobs, 1994; Kontschieder et al., 2015; Frosst and Hinton, 2017; Tanno et al., 2019; Hehn et al., 2019) and Smooth-step function (for hard routing) (Hazimeh et al., 2020). Next, we define how the root-to-leaf probabilities can be used to make the final prediction of the tree.

**Prediction** As with classical decision trees, we assume that each leaf stores a learnable weight vector $\boldsymbol{o}_l^j \in R^c$. For a sample $\boldsymbol{x} \in \mathbb{R}^p$, prediction of the tree is defined as an expectation over the leaf outputs, i.e., $\boldsymbol{f}^j(\boldsymbol{x}) = \sum_{l \in \mathcal{L}^j} P^j(\{\boldsymbol{x} \rightarrow l\}) \boldsymbol{o}_l^j$.

### S2.1 Tree Ensemble Tensor Formulation

Ibrahim et al. (2022) proposed a tensor formulation for modeling tree ensembles more efficiently, which can lead to faster training times than classical formulations (Kontschieder et al., 2015; Hazimeh et al., 2020): $\sim 10\times$ on CPUs and $\sim 20\times$ on GPUs. We use a similar tensor formulation, as we discuss below. The internal nodes in the trees across the ensemble are jointly modeled as a "supernode". In particular, an internal node $i \in \mathcal{I}^j$ at depth $d$ in all trees can be condensed together into a supernode $i \in \mathcal{I}$. Let $\boldsymbol{W}_i \in \mathbb{R}^{p,m}$ be learnable weight matrix, where each $j$-th column of the weight matrix contains the learnable weight vector $\boldsymbol{w}_i^j$ of the original $j$-th tree in the ensemble. Similarly, the leaves in the trees across the ensemble are jointly modeled as a superleaf. Let $\boldsymbol{O}_l \in \mathbb{R}^{m,c}$ be the learnable weight matrix to store the leaf nodes, where each $j$-th row contains the learnable weight vector $\boldsymbol{o}_l^j$ in the original $j$-th tree in the ensemble. The prediction of the tree ensemble is $\boldsymbol{f}(\boldsymbol{x}) = (\sum_{l \in \mathcal{L}} \boldsymbol{O}_l \odot \prod_{i \in A(l)} \boldsymbol{R}_{i,l}) \cdot \boldsymbol{1}_m$, where $\odot$ denotes the element-wise product, $\boldsymbol{R}_{i,l} = S(\boldsymbol{W}_i \cdot \boldsymbol{x})^{1[l \swarrow i]} \odot (1 - S(\boldsymbol{W}_i \cdot \boldsymbol{x}))^{1[i \searrow l]} \in \mathbb{R}^{m,1}$ and the activation

function $S$ is applied element-wise. $\mathbf{1}_m \in \mathbb{R}^m$ is a vector of ones that combines the predictions of trees in the ensemble. We denote all the hyperplane parameters across all the supernodes of the tree ensemble as a tensor $\boldsymbol{\mathcal{W}} \in R^{p,m,|\mathcal{I}|}$ and the parameters across all the superleaves as a tensor $\boldsymbol{\mathcal{O}} \in R^{m,c,|\mathcal{L}|}$.

## S3 PROOF OF THEOREM 1

**Overview and Preliminaries.** Let us denote the training loss corresponding to the sample $n$ as

$$\Phi_n(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{W}}) = L(y_n, \boldsymbol{f}(\mathbf{x}_n; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})). \tag{S1}$$

We also use the notation $\mathbb{X}$ to denote the set of all decision variables in the model, $\mathbb{X} := (\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})$. For $j \in [m]$ and $i \in \mathcal{I}^j$ and $t \in [p]$, we let $w^j_{i,t}$ be the $t$-th coordinate of $\boldsymbol{w}^j_i$.

In this proof, we follow the general steps outlined below:

1. First, we show that $\Phi_n$ is $\mathcal{M}$-smooth for some $\mathcal{M} > 0$ only depending on the data and the constants appearing in Assumptions **(A1)**, **(A2)** and **(A3)**. That is, there exists $\mathcal{M} > 0$ such that for any two $\mathbb{X}_1 = (\boldsymbol{\mathcal{W}}_1, \boldsymbol{\mathcal{O}}_1), \mathbb{X}_2 = (\boldsymbol{\mathcal{W}}_2, \boldsymbol{\mathcal{O}}_2)$ with $\|\boldsymbol{\mathcal{O}}_1\|_2, \|\boldsymbol{\mathcal{O}}_2\|_2 \leq B$, we have
$$\|\nabla\Phi_n(\boldsymbol{\mathcal{O}}_1, \boldsymbol{\mathcal{W}}_1) - \nabla\Phi_n(\boldsymbol{\mathcal{O}}_2, \boldsymbol{\mathcal{W}}_2)\|_2 \leq \mathcal{M}\|\mathbb{X}_1 - \mathbb{X}_2\|_2. \tag{S2}$$
This will prove the descent property of the algorithm.
2. Next, we show that as long as $\lambda_2 > 0$, the sequence of solutions generated by the algorithm is bounded.
3. Finally, we show that $\Phi_n$ is semi-algebraic (Dries, 1998; Attouch et al., 2013, Chapter 2) and therefore satisfies the Kurdyka–Łojasiewicz (KL) property (Bolte et al., 2007; Kurdyka, 1998). This will complete the proof of convergence.

Before continuing with the proof, we derive some results that will be useful. For notational convenience, we drop the sample index $n$ as our results will be true for all samples.

First, for $j \in [m]$, $i \in \mathcal{I}^j$ and $l \in \mathcal{L}^j$

$$\begin{aligned}\frac{\partial f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})}{\partial \boldsymbol{w}^j_i} &= \sum_{l \in \mathcal{L}^j} o^j_l \frac{\partial P^j(\{\mathbf{x} \to l\})}{\partial \boldsymbol{w}^j_i} \\ \frac{\partial f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})}{\partial o^j_l} &= P^j(\{\mathbf{x} \to l\}).\end{aligned} \tag{S3}$$

Thus,

$$\begin{aligned}\frac{\partial \Phi}{\partial \boldsymbol{w}^j_i} &= L'(y, f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})) \frac{\partial f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})}{\partial \boldsymbol{w}^j_i} \\ &= L'(y, f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})) \sum_{l \in \mathcal{L}^j} o^j_l \frac{\partial P^j(\{\mathbf{x} \to l\})}{\partial \boldsymbol{w}^j_i}.\end{aligned} \tag{S4}$$

If $i \notin \mathcal{A}(l)$, then $\frac{\partial P^j(\{\mathbf{x} \to l\})}{\partial \boldsymbol{w}^j_i} = \mathbf{0}$. Otherwise,

$$\frac{\partial P^j(\{\mathbf{x} \to l\})}{\partial \boldsymbol{w}^j_i} = \prod_{\substack{k \in \mathcal{A}(l) \\ k \neq i}} r^j_{k,l}(\mathbf{x}) \frac{\partial r^j_{i,l}(\mathbf{x})}{\partial \boldsymbol{w}^j_i}. \tag{S5}$$

Moreover, by the definition of $r^j_{i,l}$, we have

$$\frac{\partial r^j_{i,l}(\mathbf{x})}{\partial \boldsymbol{w}^j_i} = \begin{cases} S'(\boldsymbol{w}^j_i \cdot \mathbf{x})\mathbf{x} & \text{if} \quad l \swarrow i \\ -S'(\boldsymbol{w}^j_i \cdot \mathbf{x})\mathbf{x} & \text{if} \quad l \searrow i. \end{cases} \tag{S6}$$

In addition, for $j \in [m]$ and $l \in \mathcal{L}^j$,

$$\begin{aligned}\frac{\partial \Phi}{\partial o_l^j} &= L'(y, f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})) \frac{\partial f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})}{\partial o_l^j} \\ &= L'(y, f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})) P^j(\{\mathbf{x} \to l\}).\end{aligned} \tag{S7}$$

We define

$$\begin{aligned}\phi_{i,t}^j(\mathbb{X}) &= \frac{\partial \Phi}{\partial w_{i,t}^j}, \\ \phi_l^j(\mathbb{X}) &= \frac{\partial \Phi}{\partial o_l^j}.\end{aligned} \tag{S8}$$

Next, we state a few technical lemma that will be useful in our proof.

**Lemma 1.** *Define $E_{i,+}^j, E_{i,-}^j \subseteq \mathbb{R}^{p,m,|\mathcal{I}|} \times \mathbb{R}^{m,|\mathcal{L}|}$ for $i \in \mathcal{I}^j, j \in [m]$:*

$$E_{i,+}^j = \{\boldsymbol{w}_i^j \cdot \mathbf{x} = \theta\}, \quad E_{i,-}^j = \{\boldsymbol{w}_i^j \cdot \mathbf{x} = -\theta\} \tag{S9}$$

*and*

$$\mathcal{D} = \left(\left[\mathbb{R}^{p,m,|\mathcal{I}|} \times \mathbb{R}^{m,|\mathcal{L}|}\right] \bigcap \{\|\boldsymbol{\mathcal{O}}\|_2 \le B\}\right) \setminus \bigcup_{\substack{j \in [m] \\ i \in \mathcal{I}^j}} \left\{E_{i,+}^j \cup E_{i,-}^j\right\}$$

*where $B$ is defined in Assumption **(A3)**. If $\mathbb{X} \in \mathcal{D}$, then $\Phi(\mathbb{X})$ is infinitely differentiable. Moreover, $\mathcal{D}$ can be partitioned into finitely many subsets.*

*Proof.* Note that $\Phi(\mathbb{X})$ may not have infinitely many derivatives only if $S(\boldsymbol{w}_i^j \cdot \mathbf{x})$ is not smooth for some $j \in [m], i \in \mathcal{I}^j$. By the construction of $S(\cdot)$ from Assumption **(A1)**, the activation $S(x)$ function is not infinitely differentiable only for $x = \pm\theta$. As a result, if $\mathbb{X} \in \mathcal{D}$, all activation functions are smooth and therefore $\Phi(\mathbb{X})$ is infinitely differentiable.
Moreover, note that each $E_{i,\pm}^j$ is an affine space with codimension 1. Therefore, each $E_{i,\pm}^j$ partitions the space (excluding $E_{i,\pm}^j$) into two subsets $\{\boldsymbol{w}_i^j \cdot \mathbf{x} > \pm\theta\}, \{\boldsymbol{w}_i^j \cdot \mathbf{x} < \pm\theta\}$. Therefore, all $E_{i,\pm}^j$ can partition $\mathcal{D}$ into finitely many subsets, as there are finitely many of sets $E_{i,\pm}^j$. □

**Lemma 2.** *Suppose $\mathbb{X} \in \mathcal{D}$. Under the assumption of Theorem 1, there exists a numerical constant $C > 0$, only depending on the data and the constants appearing in the assumptions of the theorem, such that the following functions and their gradients are bounded by $C$:*

$$\begin{aligned}&r_{i,l}^j(\mathbf{x}), \ j \in [m], l \in \mathcal{L}^j, i \in \mathcal{A}(l) \\ &\frac{\partial r_{i,j}^j(\mathbf{x})}{\partial w_{i,t}^j}, \ j \in [m], l \in \mathcal{L}^j, i \in \mathcal{A}(l), t \in [p] \\ &P^j(\{\mathbf{x} \to l\}), \ j \in [m], l \in \mathcal{L}^j.\end{aligned} \tag{S10}$$

*Proof.* First, note that $r_{i,l}^j(\mathbf{x}) \in [0,1]$ and therefore $r_{i,l}^j(\mathbf{x})$ is uniformly bounded. Moreover, from (S6)

$$\frac{\partial r_{i,l}^j(\mathbf{x})}{\partial \boldsymbol{w}_i^j} = \begin{cases} S'(\boldsymbol{w}_i^j \cdot \mathbf{x})\mathbf{x} & \text{if} \quad l \swarrow i \\ -S'(\boldsymbol{w}_i^j \cdot \mathbf{x})\mathbf{x} & \text{if} \quad l \searrow i \end{cases} \tag{S11}$$

with other partial derivatives of $r_{i,l}^j(\mathbf{x})$ being zero. As a result, $r_{i,l}^j(\mathbf{x})$ has uniformly bounded derivative, where the bound only depends on Assumption **(A1)** and the data. This is true as $S'$ is bounded by the assumption. This also shows that $\frac{\partial r_{i,l}^j(\mathbf{x})}{\partial w_{i,t}^j}$ is uniformly bounded. Next,

$$\frac{\partial^2 r_{i,l}^j(\mathbf{x})}{\partial (\boldsymbol{w}_i^j)^2} = \begin{cases} S''(\boldsymbol{w}_i^j \cdot \mathbf{x})\mathbf{x}\mathbf{x}^T & \text{if} \quad l \swarrow i \\ -S''(\boldsymbol{w}_i^j \cdot \mathbf{x})\mathbf{x}\mathbf{x} & \text{if} \quad l \searrow i \end{cases} \tag{S12}$$

which is similarly uniformly bounded by Assumption **(A1)**. Therefore, $\frac{\partial r_{i,j}^j(\mathbf{x})}{\partial w_{i,t}^j}$ has a uniformly bounded gradient. Finally, $P^j(\{\mathbf{x} \to l\}) \in [0,1]$ and by equation S5, the gradient of $P^j(\{\mathbf{x} \to l\})$ is the product of bounded functions, and therefore bounded. □

**Lemma 3.** *Suppose $\mathbb{X} \in \mathcal{D}$. Under the assumption of Theorem 1, there exists a numerical constant $C > 0$, only depending on the data and the constants appearing in the assumptions of the theorem, such that the following functions and their gradients are bounded by $C$:*

$$\begin{aligned} &f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}) \\ &L'(y, f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})) \end{aligned} \tag{S13}$$

*Proof.* First,

$$\begin{aligned} |f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})| &= \left| \sum_{j=1}^{m} \sum_{l \in \mathcal{L}^j} o_l^j P^j(\{\mathbf{x} \to l\}) \right| \\ &\leq \sum_{j=1}^{m} \sum_{l \in \mathcal{L}^j} |o_l^j| \\ &\leq m|\mathcal{L}^1|B. \end{aligned} \tag{S14}$$

Moreover, from equation S3 and Lemma 2, each coordinate of the gradient of $f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})$ is finite summation and product of uniformly bounded functions, which is bounded.
Next, as $f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})$ is bounded, $L'(y, f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}))$ is bounded by Assumption **(A2)**. Moreover,

$$\nabla L'(y, f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}) = L''(y, f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})) \nabla f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})$$

which is bounded as $L''$ is bounded by Assumption **(A2)** and $\nabla f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})$ is bounded as we showed above. □

**Lemma 4.** *There exists a numerical constant $M > 0$, only depending on the data and constants introduced in Assumptions **(A1)**, **(A2)** and **(A3)**, such that if for $\mathbb{X}_1, \mathbb{X}_2$ and $\alpha \in (0,1)$, $\alpha\mathbb{X}_1 + (1-\alpha)\mathbb{X}_2 \in \mathcal{D}$, then one has*

$$|\phi_{i,t}^j(\mathbb{X}_1) - \phi_{i,t}^j(\mathbb{X}_2)| \leq M\|\mathbb{X}_1 - \mathbb{X}_2\|_2, \ t \in [p], j \in [m], i \in \mathcal{I}^j$$

*where $\phi_{i,t}^j$ is defined in equation S8.*

*Proof.* Note that by the definition of $\phi_{i,t}^j$,

$$\phi_{i,t}^j(\mathbb{X}) = L'(y, f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})) \frac{\partial f(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})}{\partial w_{i,t}^j}$$

which is the product of two bounded functions with bounded derivatives by Lemma 3. As a result, there exists a constant $M > 0$ such that

$$\left\| \frac{\partial \phi_{i,t}^j(\mathbb{X})}{\partial \mathbb{X}} \right\|_2 \leq M. \tag{S15}$$

For $\alpha \in [0,1]$, let $\hat{\phi}_{i,t}^j(\alpha) = \phi_{i,t}^j((1-\alpha)\mathbb{X}_1 + \alpha\mathbb{X}_2)$. Then, $\hat{\phi}_{i,t}^j(\alpha)$ is differentiable for $\alpha \in (0,1)$, $\hat{\phi}_{i,t}^j(0) = \phi_{i,t}^j(\mathbb{X}_1)$ and $\hat{\phi}_{i,t}^j(1) = \phi_{i,t}^j(\mathbb{X}_2)$. Moreover, by the chain rule,

$$\begin{aligned} \frac{d\hat{\phi}_{i,t}^j(\alpha)}{d\alpha} &= \left\langle \frac{\partial \hat{\phi}_{i,t}^j(\alpha)}{\partial \left[(1-\alpha)\mathbb{X}_1 + \alpha\mathbb{X}_2\right]}, \frac{\partial \left[(1-\alpha)\mathbb{X}_1 + \alpha\mathbb{X}_2\right]}{\partial \alpha} \right\rangle \\ &= \left\langle (\mathbb{X}_2 - \mathbb{X}_1), \frac{\partial \phi_{i,t}^j(\alpha)}{\partial \left[(1-\alpha)\mathbb{X}_1 + \alpha\mathbb{X}_2\right]} \right\rangle \end{aligned}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product. As a result, by the fundamental theorem of calculus,

$$\begin{aligned}\left|\hat{\phi}^j_{i,j}(\alpha) - \hat{\phi}^j_{i,j}(0)\right| &= \left|\int_0^\alpha \frac{d\hat{\phi}^j_{i,t}(u)}{du} du\right| \\ &\leq \|\mathbb{X}_2 - \mathbb{X}_1\|_2 \int_0^\alpha \underbrace{\left\| \frac{\partial \hat{\phi}^j_{i,t}(u)}{\partial\left[(1-u)\mathbb{X}_1 + u\mathbb{X}_2\right]} \right\|_2}_{\leq M \text{ by equation S15}} du \\ &\leq \alpha M \|\mathbb{X}_2 - \mathbb{X}_1\|_2.\end{aligned}$$

In particular, by setting $\alpha = 1$ the proof is complete. □

**Lemma 5.** *Take any two $\mathbb{X}_1, \mathbb{X}_2$ such that $\|\boldsymbol{\mathcal{O}}_1\|_2, \|\boldsymbol{\mathcal{O}}_2\|_2 \leq B$. Suppose for all $j \in [m], i \in \mathcal{I}^j$, $\{\mathbb{X}_1, \mathbb{X}_2\} \not\subseteq E^j_{i,+}$ or $\{\mathbb{X}_1, \mathbb{X}_2\} \not\subseteq E^j_{i,-}$. Then*

$$|\phi^j_{i,t}(\mathbb{X}_1) - \phi^j_{i,t}(\mathbb{X}_2)| \leq M\|\mathbb{X}_1 - \mathbb{X}_2\|_2 \;\; t \in [p], j \in [m], i \in \mathcal{I}^j$$

*where $\phi^j_{i,t}$ is defined in equation S8 and $M$ is defined in Lemma 4.*

*Proof.* By Lemma 1, the sets $E^j_{i,\pm}$ are affine and therefore if both $\mathbb{X}_1, \mathbb{X}_2$ do not belong to one of these sets, the segment connecting $\mathbb{X}_1, \mathbb{X}_2$ will intersect each set $E^j_{i,\pm}$ at most once. Thus, there exist $K \geq 1$ and $\tilde{\mathbb{X}}_0, \cdots, \tilde{\mathbb{X}}_K$ such that $\tilde{\mathbb{X}}_0 = \mathbb{X}_1, \tilde{\mathbb{X}}_K = \mathbb{X}_2$, and $\tilde{\mathbb{X}}_0, \cdots, \tilde{\mathbb{X}}_K$ lie in the segment connecting $\mathbb{X}_1, \mathbb{X}_2$. Moreover, we have that

$$\{(1-\alpha)\tilde{\mathbb{X}}_k + \alpha\tilde{\mathbb{X}}_{k+1} : \alpha \in (0,1)\} \subseteq \mathcal{D} \;\; \forall k \in [K-1].$$

By triangle inequality, we have

$$\begin{aligned}|\phi^j_{i,t}(\mathbb{X}_1) - \phi^j_{i,t}(\mathbb{X}_2)| &= \left|\sum_{k=0}^{K-1} \{\phi^j_{i,t}(\tilde{\mathbb{X}}_k) - \phi^j_{i,t}(\tilde{\mathbb{X}}_{k+1})\}\right| \\ &\leq \sum_{k=0}^{K-1} \left|\phi^j_{i,t}(\tilde{\mathbb{X}}_k) - \phi^j_{i,t}(\tilde{\mathbb{X}}_{k+1})\right| \\ &\leq M \sum_{k=0}^{K-1} \|\tilde{\mathbb{X}}_k - \tilde{\mathbb{X}}_{k+1}\|_2 \\ &= M\|\mathbb{X}_1 - \mathbb{X}_2\|_2\end{aligned}$$

where the last inequality is by Lemma 4 and the last equality is by the fact that $\tilde{\mathbb{X}}_0, \cdots, \tilde{\mathbb{X}}_K$ lie in the segment connecting $\mathbb{X}_1, \mathbb{X}_2$. □

**Lemma 6.** *There exists a numerical constant $\mathcal{M} > 0$, only depending on the data and constants introduced in Assumptions **(A1)**, **(A2)** and **(A3)**, such that $\Phi(\boldsymbol{\mathcal{O}}, \boldsymbol{\mathcal{W}})$ is $\mathcal{M}$-smooth for $\|\boldsymbol{\mathcal{O}}\|_2 \leq B$.*

*Proof.* First, note that

$$\|\nabla\Phi(\boldsymbol{\mathcal{O}}_1, \boldsymbol{\mathcal{W}}_1) - \nabla\Phi(\boldsymbol{\mathcal{O}}_2, \boldsymbol{\mathcal{W}}_2)\|_2 \leq \sum_{j=1}^m \sum_{l \in \mathcal{L}^j} |\phi^j_l(\mathbb{X}_1) - \phi^j_l(\mathbb{X}_2)| + \sum_{j=1}^m \sum_{i \in \mathcal{I}^j} \sum_{t=1}^p |\phi^j_{i,t}(\mathbb{X}_1) - \phi^j_{i,t}(\mathbb{X}_2)|. \tag{S16}$$

Take $\mathbb{X}_1, \mathbb{X}_2$ with $\|\boldsymbol{\mathcal{O}}_1\|_2, \|\boldsymbol{\mathcal{O}}_2\|_2 \leq B$. If these two solutions simultaneously do not belong to some $E^j_{i,\pm}$, by Lemma 5 we have

$$|\phi^j_{i,t}(\mathbb{X}_1) - \phi^j_{i,t}(\mathbb{X}_2)| \leq M\|\mathbb{X}_1 - \mathbb{X}_2\|_2.$$

A similar result follows for $\phi^j_l$, hence by equation S16 we achieve

$$\|\nabla\Phi(\boldsymbol{\mathcal{O}}_1, \boldsymbol{\mathcal{W}}_1) - \nabla\Phi(\boldsymbol{\mathcal{O}}_2, \boldsymbol{\mathcal{W}}_2)\|_2 \leq m(|\mathcal{L}^1| + p|\mathcal{I}^1|)M\|\mathbb{X}_1 - \mathbb{X}_2\|_2$$

which completes the proof. Suppose there exists $i_0, j_0$ such that $\mathbb{X}_1, \mathbb{X}_2 \in E^{j_0}_{i_0,+}$. Then,

$$\left((1-\alpha)(\boldsymbol{w}_1)^{j_0}_{i_0}\right)\cdot \mathbf{x} + \left(\alpha(\boldsymbol{w}_2)^{j_0}_{i_0}\right)\cdot \mathbf{x} = \theta$$

therefore $S\left(\left((1-\alpha)(\boldsymbol{w}_1)^{j_0}_{i_0}\right)\cdot \mathbf{x} + \left(\alpha(\boldsymbol{w}_2)^{j_0}_{i_0}\right)\cdot \mathbf{x}\right) = 1$ by Assumption **(A1)**. As a result, $S(\boldsymbol{w}^{j_0}_{i_0}\cdot \mathbf{x}) = 1$ for the whole segment connecting $\mathbb{X}_1, \mathbb{X}_2$. Let $\tilde{\mathbb{X}}_1, \tilde{\mathbb{X}}_2$ be such that all their weights are the same as $\mathbb{X}_1, \mathbb{X}_2$, except that

$$(\tilde{\boldsymbol{w}}_1)^{j_0}_{i_0} = 2(\boldsymbol{w}_1)^{j_0}_{i_0}, (\tilde{\boldsymbol{w}}_2)^{j_0}_{i_0} = 2(\boldsymbol{w}_2)^{j_0}_{i_0}.$$

As a result,

$$\left((1-\alpha)(\tilde{\boldsymbol{w}}_1)^{j_0}_{i_0}\right)\cdot \mathbf{x} + \left(\alpha(\tilde{\boldsymbol{w}}_2)^{j_0}_{i_0}\right)\cdot \mathbf{x} = 2\theta$$

so $S\left(\left((1-\alpha)(\tilde{\boldsymbol{w}}_1)^{j_0}_{i_0}\right)\cdot \mathbf{x} + \left(\alpha(\tilde{\boldsymbol{w}}_2)^{j_0}_{i_0}\right)\cdot \mathbf{x}\right) = 1$. Therefore, one has

$$f(\mathbf{x}; (1-\alpha)\tilde{\mathbb{X}}_1 + \alpha\tilde{\mathbb{X}}_2) = f(\mathbf{x}; (1-\alpha)\mathbb{X}_1 + \alpha\mathbb{X}_2),\ \alpha \in [0,1],$$
$$\frac{\partial f(\mathbf{x}; (1-\alpha)\tilde{\mathbb{X}}_1 + \alpha\tilde{\mathbb{X}}_2)}{\partial \boldsymbol{w}^j_i} = \frac{\partial f(\mathbf{x}; (1-\alpha)\mathbb{X}_1 + \alpha\mathbb{X}_2)}{\partial \boldsymbol{w}^j_i},\ \alpha \in [0,1], j \in [m], i \in \mathcal{I}^j$$

as for all $i, j$,

$$S\left(\left((1-\alpha)(\tilde{\boldsymbol{w}}_1)^j_i\right)\cdot \mathbf{x} + \left(\alpha(\tilde{\boldsymbol{w}}_2)^j_i\right)\cdot \mathbf{x}\right) = S\left(\left((1-\alpha)(\boldsymbol{w}_1)^j_i\right)\cdot \mathbf{x} + \left(\alpha(\boldsymbol{w}_2)^j_i\right)\cdot \mathbf{x}\right).$$

As a result, for all $i, j, t$,

$$\phi^j_{i,t}(\tilde{\mathbb{X}}_1) = \phi^j_{i,t}(\mathbb{X}_1), \phi^j_{i,t}(\tilde{\mathbb{X}}_2) = \phi^j_{i,t}(\mathbb{X}_2)$$

However, $\tilde{\mathbb{X}}_1, \tilde{\mathbb{X}}_2 \notin E^{j_0}_{i_0,+}$. In words, the new points effectively replace the problematic coefficient $\boldsymbol{w}^{j_0}_{i_0}$ in the model coefficients. Repeat this process until all such coefficients are removed and therefore $\tilde{\mathbb{X}}_1, \tilde{\mathbb{X}}_2$ fit into the assumptions of Lemma 5. This completes the proof. □

**Lemma 7.** *Under Assumption **(A1)**, the activation function $S(x)$ is semi-algebraic.*

*Proof.* Consider the epigraph of the activation function

$$\mathcal{G}(S) = \{(x, y) : y \geq S(x)\}.$$

Let

$$A_1 = \{y \geq 0, x \leq -\theta\}$$
$$A_2 = \{y \geq 1, x \geq \theta\}$$
$$A_3 = \{y \geq p(x), -\theta \leq x \leq \theta\}.$$

Then,

$$\mathcal{G}(S) = A_1 \cup A_2 \cup A_3$$

showing $\mathcal{G}(S)$ and consequently, $S$ are semi-algebraic. □

**Lemma 8.** *The function $1[\boldsymbol{\mathcal{W}}_{k,:,:} \neq \mathbf{0}]$ is semi-algebraic.*

*Proof.* Consider the epigraph:

$$G = \mathcal{G}(1[\boldsymbol{\mathcal{W}}_{k,:,:} \neq \mathbf{0}]) = \{(y, \boldsymbol{\mathcal{W}}) : y \geq 1[\boldsymbol{\mathcal{W}}_{k,:,:} \neq \mathbf{0}]\}.$$

Let $A_1 = \{y \geq 1\}$ and $A_2 = \{\|\boldsymbol{\mathcal{W}}_{k,:,:}\|^2_2 = 0, y \geq 0\}$. Then, $G = A_1 \cup A_2$ showing $1[\boldsymbol{\mathcal{W}}_{k,:,:} \neq \mathbf{0}]$ is semi-algebraic. □

*Proof of Theorem 1.* **Part 1)** Note that by Assumption **(A3)**, Algorithm 1 can be equivalently run on the problem

$$\min_{\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}}} \ \hat{\mathbb{E}}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x};\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}}))] + \lambda_0 \sum\nolimits_{k\in[p]} 1[\boldsymbol{\mathcal{W}}_{k,:,:} \neq \mathbf{0}] + (\lambda_2/m|\mathcal{I}|)\|\boldsymbol{\mathcal{W}}\|_2^2 \qquad \text{s.t.} \qquad \|\boldsymbol{\mathcal{O}}\|_2 \leq B \tag{S17}$$

and result in the same sequence of solutions. Let

$$\begin{aligned} h(\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}}) &= \hat{\mathbb{E}}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x};\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}}))] + (\lambda_2/m|\mathcal{I}|)\|\boldsymbol{\mathcal{W}}\|_2^2 \\ g(\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}}) &= \lambda_0 \sum\nolimits_{k\in[p]} 1[\boldsymbol{\mathcal{W}}_{k,:,:} \neq \mathbf{0}] + \chi(\|\boldsymbol{\mathcal{O}}\|_2 \leq B) \end{aligned} \tag{S18}$$

where $\chi(\|\boldsymbol{\mathcal{O}}\|_2 \leq B) = 0$ if $\|\boldsymbol{\mathcal{O}}\|_2 \leq B$ and $\chi(\|\boldsymbol{\mathcal{O}}\|_2 \leq B) = \infty$ otherwise. Then, Problem S17 can be written as minimizing $g + h$. By Lemma 6, the function $h$ is $(\mathcal{M} + 2\lambda_2/m|\mathcal{I}|)$-smooth. Hence, if $\eta < 1/(\mathcal{M} + 2\lambda_2/m|\mathcal{I}|)$, by Lemma 3.1 of Attouch et al. (2013) (and calculations leading to (52) of Attouch et al. (2013)) the descent property is proved.

**Part 2)** Let $c_0$ be the objective value for the initial solution to Algorithm 1. That is,

$$\frac{1}{N}\sum_{n=1}^{N} L(y_n, \boldsymbol{f}(\mathbf{x}_n;\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}}))] + \lambda_0 \sum\nolimits_{k\in[p]} 1[\boldsymbol{\mathcal{W}}_{k,:,:} \neq \mathbf{0}] + (\lambda_2/m|\mathcal{I}|)\|\boldsymbol{\mathcal{W}}\|_2^2 = c_0.$$

As the sequence of the objectives is non-increasing by the first part of the theorem, at each iteration we have

$$\begin{aligned} &(\lambda_2/m|\mathcal{I}|)\|\boldsymbol{\mathcal{W}}\|_2^2 \\ \leq &\frac{1}{N}\sum_{n=1}^{N} L(y_n, \boldsymbol{f}(\mathbf{x}_n;\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}}))] + \lambda_0 \sum\nolimits_{k\in[p]} 1[\boldsymbol{\mathcal{W}}_{k,:,:} \neq \mathbf{0}] + (\lambda_2/m|\mathcal{I}|)\|\boldsymbol{\mathcal{W}}\|_2^2 \\ \leq &c_0 \end{aligned}$$

completing the proof as $\lambda_2 > 0$.

**Part 3)** Note that the function $f(\mathbf{x};\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}})$ is a composition, finite sum and finite product of semi-algebraic functions by Lemma 7, therefore it is semi-algebraic. As $L$ is polynomial, $L(f(\mathbf{x};\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}}))$ is semi-algebraic. As a result, by Lemma 8 the functions $g, h$ defined in equation S18 are semi-algebraic and hence, possess the KL property. Therefore, $g, h$ satisfy the required conditions of Theorem 5.1 of Attouch et al. (2013), proving the convergence. □

## S4 APPENDIX FOR SECTION 6

**Data Design** We generate the data matrix, $\boldsymbol{X} \in \mathbb{R}^{N\times p}$ with samples randomly drawn from a multivariate normal distribution $\mathcal{N}(0, \boldsymbol{\Sigma})$ with a correlated design matrix $\boldsymbol{\Sigma}$ whose values are defined by $\Sigma_{ij} = \sigma^{|i-j|}$. We construct the response variable $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta}^* + \epsilon$ where the values of the noise $\epsilon_i, i = [N]$ are drawn independently from $\mathcal{N}(0, 0.5)$. We use a known sparsity $\boldsymbol{\beta}^*$ with equi-spaced nonzeros, where $\|\boldsymbol{\beta}^*\|_0 = 8$.

**Simulation Procedure** We experiment with a range of training set sizes $N \in \{100, 200, 1000\}$, correlation strengths $\sigma \in \{0.5, 0.7\}$, and number of total features $p \in \{256, 512\}$. For each setting, we run 25 simulations with randomly generated samples. We evaluate on $10,000$ test samples drawn from the data generating model described above. Out of $N$ samples, we allocate 80% for training and 20% for validation for model selection. For each simulation, we perform 500 tuning trials (hyperparameters are given below) and select the model with the smallest validation mean squared error (MSE). We evaluate the final performance in terms of (i) test MSE, (ii) number of features selected and (iii) support recovery via f1-score between the true support and the recovered feature set. We compute averages and standard errors across the 25 simulations to report final results.

**`Skinny Trees` with DSL**

- Number of trees: Discrete uniform with range $[1, 50]$,
- Depths: Discrete uniform with range $[1, 5]$,
- Batch sizes: $16b$ with b uniform over the range $\{1, 2, \ldots, 8\}$,

- Number of Epochs: Discrete uniform with range $[5, 500]$,
- $\lambda_0$: exponentially ramped up $0 \to 100$ with temperature distributed as Log uniform in the range $[10^{-4}, 0.1]$ for group $\ell_0 - \ell_2$ with DSL,
- $\lambda_2$: Log uniform over the range $[10^{-2}, 10^2]$ for group $\ell_0 - \ell_2$,
- Learning rates, $lr$: Log uniform over the range $[10^{-3}, 10^{-1}]$.

**XGBoost, Random Forests**

- Number of trees: Discrete uniform with range $[1, 100]$ for XGBoost and Random Forests,
- Depths: Discrete uniform with range $[1, 10]$ for XGBoost and Random Forests,
- Learning rates: Discrete uniform with range $[10^{-4}, 1.0]$ for XGBoost,
- Feature importance threshold: Log uniform over the range $[10^{-7}, 10^{-1}]$ for XGBoost and Random Forests.
- Subsample: Uniform over the range $[0.5, 1.0]$.

# S5 APPENDEX FOR SECTION 7

## S5.1 Datasets, Computing Setup and Tuning Setup

**Datasets** We consider 5 classification datasets from the Penn Machine Learning Benchmarks (PMLB) repository (Olson et al., 2017). These are Churn, Satimage, Mice-protein, Isolet and Texture; Mice-protein and Isolet were used in prior feature selection literature Lemhadri et al. (2021). We used fetch_openml in Sklearn to download the full datasets. We used 4 datasets from NIPS2003 feature selection challenge (Guyon et al., 2004). These are Arcene, Madelon, Gisette, Dorothea. We used 5 datasets (Smk, Cll, Gli, Lung, Tox) from the feature selection datasets given in this repo[1]. The datasets contain continuous, categorical and binary features. The metadata was used to identify the type of features and categorical features were one-hot encoded. For first non-NIPS2003 datasets, we randomly split each of the dataset into 64% training, 16% validation and 20% testing sets. For the datasets from NIPS2003, we split the training set into 80% training and 20% validation and treated the original validation set as the test set. The labels for original test set were unavailable, hence we discarded the original test set. The continuous features in all datasets were z-normalized based on training set statistics. A summary of the 14 datasets considered is in Table S3.

Table S3: Classification datasets

| Dataset | N | p | C |
|---|---|---|---|
| Churn | 5,000 | 20 | 2 |
| Satimage | 6,435 | 36 | 6 |
| Texture | 5,500 | 40 | 11 |
| Mice-protein | 1,080 | 77 | 8 |
| Isolet | 7,797 | 617 | 26 |
| Madelon | 2,600 | 500 | 2 |
| Lung | 203 | 3,312 | 5 |
| Gisette | 7,000 | 5,000 | 2 |
| Tox | 171 | 5,748 | 4 |
| Arcene | 200 | 10,000 | 2 |
| CLL | 111 | 11,340 | 3 |
| SMK | 187 | 19,993 | 2 |
| GLI | 85 | 22,283 | 2 |
| Dorothea | 1,150 | 100,000 | 2 |

**Computing Setup** We used a cluster running Ubuntu 7.5.0 and equipped with Intel Xeon Platinum 8260 CPUs and Nvidia Volta V100 GPUs. For all experiments of Sec. 6 and 7, each job involving `Skinny Trees`, Random Forests, XGBoost, LightGBM, CatBoost and neural networks were restricted to 4 cores and 64GB of RAM. Jobs involving `Skinny Trees` and neural networks on larger datasets (Gisette, Dorothea) were run on Tesla V100 GPUs.

**Tuning** The tuning was done in parallel over the competing models and datasets. The number of selected features affects the AUC. Therefore, to treat all the methods in a fair manner, we tune the hyperparameter that controls the sparsity level using Optuna (Akiba et al., 2019) which optimizes the overall AUC across different Ks. A list of all the tuning parameters and their distributions is given for every experiment below.

## S5.2 Appendix for Sec. 7.1

**Classical tree: TAO Implementation** Given that the authors of TAO do not open-source their implementation, we have written our own implementation of the TAO algorithm proposed by Carreira-Perpinan and Tavallali (2018), following the procedure outlined in Sec. 3.1 of Zharmagambetov et al. (2019). For TAO, we use oblique (i.e. hyperplane splits) decision trees with constant leaves. We take as an initial tree a complete binary tree with random parameters at each node. We perform TAO updates until maximum number of iterations are reached (i.e. there is no other stopping criterion). TAO uses an $\ell_1$-regularized logistic regression to solve the decision

[1] https://jundongl.github.io/scikit-feature/datasets.html

node optimization (using LIBLINEAR Fan et al. (2008)) where $\lambda \geq 0$ parameter (controlling node-level sparsity of the tree) is used as a regularization parameter ($C = 1/\lambda$). We tune depth in the range $[1, 10]$, $\lambda \in [10^{-5}, 10^{5}]$ and number of maximum iterations in the range $[20, 100]$. We perform 500 tuning trials. We find the optimal trial that satisfies 50% sparsity budget.

**Soft tree with group lasso** We compare group $\ell_0$-based method with a competitive benchmark: the convex group lasso regulurizer, popularly used in high-dimensional statistics literature (Yuan and Lin, 2006). We consider group-lasso regularization in the context of soft trees: $(\lambda_1/\sqrt{m|\mathcal{I}|})\sum_{k\in[p]} \|\boldsymbol{\mathcal{W}}_{k,:,:}\|_2$. Although group lasso has not been used in soft trees, it has been used for feature selection in neural networks (Scardapane et al., 2017). However, their proposal to use GD on a group $\ell_1$ regularized objective does not lead to feature selection as highlighted by (Lemhadri et al., 2021). For a fairer comparison, we implement our own proximal mini-batch GD method for group lasso in the context of soft trees, which actually leads to feature selection.

We consider the following optimization problem with group-lasso regularization in the context of soft trees:

$$\min_{\boldsymbol{\mathcal{W}},\boldsymbol{\mathcal{O}}} \ \hat{\mathbb{E}}[L(\mathbf{y}, \boldsymbol{f}(\mathbf{x}; \boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}})] + (\lambda_1/\sqrt{m|\mathcal{I}|})\sum\nolimits_{k\in[p]} \|\boldsymbol{\mathcal{W}}_{k,:,:}\|_2 \,. \tag{S19}$$

where $\lambda_1$ is a non-negative regularization parameter that controls both shrinkage and sparsity. We solve it with the algorithm presented in Alg. 2.

**Algorithm 2** Proximal Mini-batch Gradient Descent for Optimizing (S19).

**Input:** Data: $X, Y$; Hyperparameters: $\lambda_1$, epochs, batch-size ($b$), learning rate ($\eta$);
1: Initialize ensemble with $m$ trees of depth $d$ ($|\mathcal{I}| = 2^d - 1$): $\boldsymbol{\mathcal{W}}, \boldsymbol{\mathcal{O}}$
2: **for** epoch $= 1, 2, \ldots,$ epochs **do**
3:  **for** *batch* $= 1, \ldots, N/b$ **do**
4:  Randomly sample a batch: $\boldsymbol{X}_{batch}, \boldsymbol{Y}_{batch}$
5:  Compute gradient of loss $g$ w.r.t. $\boldsymbol{\mathcal{O}}$, $\mathbf{W}$, where $g = \hat{\mathbb{E}}[L(\boldsymbol{Y}_{batch}, \boldsymbol{F}_{batch}]$.
6:  Update leaves: $\boldsymbol{\mathcal{O}} \leftarrow \boldsymbol{\mathcal{O}} - \eta \nabla_{\boldsymbol{\mathcal{O}}} g$
7:  Update hyperplanes: $\boldsymbol{\mathcal{W}} \leftarrow$ *S-Prox*$(\boldsymbol{\mathcal{W}} - \eta \nabla_{\boldsymbol{\mathcal{W}}} g, \eta, \lambda_1))$
8:  **end for**
9: **end for**

*S-Prox* in Algorithm 2 finds the global minimum of an optimization problem of the form:

$$\boldsymbol{\mathcal{W}}^{(t)} = \text{argmin}_{\boldsymbol{\mathcal{W}}} \frac{1}{2\eta} \left\| \boldsymbol{\mathcal{W}} - \boldsymbol{\mathcal{Z}}^{(t)} \right\|_2^2 + \frac{\lambda_1}{\sqrt{m|\mathcal{I}|}} \sum_{k=1}^{p} \|\boldsymbol{\mathcal{W}}_{k,:,:}\|_2 \tag{S20}$$

where $\boldsymbol{\mathcal{Z}}^{(t)} = \boldsymbol{\mathcal{W}}^{(t-1)} - \eta \nabla_{\boldsymbol{\mathcal{W}}} g$ and $g = \hat{\mathbb{E}}[L(\boldsymbol{Y}_{batch}, \boldsymbol{F}_{batch}]$. This leads to a feature-wise soft-thresholding operator given by:

$$S_{\frac{\eta\lambda_1}{\sqrt{m|\mathcal{I}|}}}(\boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:}) = \begin{cases} \boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:} - \frac{\eta\lambda_1}{\sqrt{m|\mathcal{I}|}} \frac{\boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:}}{\left\|\boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:}\right\|} & \text{if } \left\|\boldsymbol{\mathcal{Z}}^{(t)}_{k,:,:}\right\| \geq \frac{\eta\lambda_1}{\sqrt{m|\mathcal{I}|}} \\ 0 & \text{otherwise} \end{cases} \tag{S21}$$

In these experiments, we used a single tree and mini-batch PGD *without* any dense-to-sparse scheduler for both models i.e, (i) Soft tree with group lasso (ii) `Skinny Tree`. We tune the key hyperparameters, which are given below.

- Batch sizes: $64 * b$ with $b$ uniform over the range $\{1, 2, \ldots, 64\}$,
- Learning rates for mini-batch PGD: Log uniform over the range $[10^{-2}, 10]$,
- Number of Epochs: Discrete uniform with range $[5, 1000]$,
- $\lambda_0$: Log uniform over the range $[10^{-3}, 10]$ for group $\ell_0 - \ell_2$ for `Skinny Tree`,
- $\lambda_2$: Log uniform over the range $[10^{-2}, 10^{2}]$ for group $\ell_0 - \ell_2$ for `Skinny Tree`,
- $\lambda_1$: Log uniform over the range $[10^{-3}, 10]$ for group lasso.

### S5.3 Appendix for Sec. 7.2 and 7.3

We describe the tuning grid for these experiments below.

`Skinny Trees`

- Number of trees: Discrete uniform with range $[1, 100]$,
- Depths: Discrete uniform with range $[1, 5]$,
- Batch sizes: Uniform over the set $\{16, 64, 256, 1024\}$,
- Number of Epochs: Discrete uniform over the range $[5, 1000]$ (Note tuning over epochs is important to achieve some trials with desired sparsity for dense-to-sparse learning, we do not use any validation loss early stopping as that is less robust during averages across runs/seeds in terms of feature support.),
- $\lambda_0$: exponentially ramped up $0 \to 1$ with temperature distributed as Log uniform in the range $[10^{-4}, 1]$ for group $\ell_0 - \ell_2$ with DSL,
- $\lambda_2$: Log uniform over the range $[10^{-3}, 1]$ for group $\ell_0 - \ell_2$,
- Learning rate ($lr$) for minibatch PGD: Log uniform over the range $[10^{-2}, 10]$.

**XGBoost, LightGBM, CatBoost, Random Forests**

- Number of trees: Discrete uniform with range $[1, 300]$ for XGBoost, LightGBM, CatBoost and Random Forests,
- Depths: Discrete uniform with range $[1, 10]$ for XGBoost, LightGBM, CatBoost and Random Forests,
- Learning rates: Discrete uniform with range $[10^{-4}, 1.0]$ for XGBoost, LightGBM and CatBoost,
- Feature importance threshold: Log uniform over the range $[10^{-7}, 10^{-1}]$ for XGBoost, LightGBM, CatBoost and Random Forests,
- Subsample: Uniform over the range $[0.5, 0.9]$ for XGBoost, LightGBM, CatBoost and Random Forests.
- Bagging Frequency: Uniform over the set $\{1, \cdots, 7\}$ for LightGBM.

**FASTEL (Dense soft trees)**

- Number of trees: Discrete uniform with range $[1, 100]$,
- Depths: Discrete uniform with range $[1, 5]$,
- Batch sizes: Uniform over the set $\{16, 64, 256, 1024\}$,
- Number of Epochs: Discrete uniform over the range $[5, 1000]$,
- Learning rates ($lr$) for minibatch SGD: Log uniform over the range $[10^{-2}, 10]$.

### S5.4 Appendix for Sec. 7.4

In this paper, we pursue embedded feature selection methods for tree ensembles. However, for completeness, we also compare `Skinny Trees` against some state-of-the-art embedded feature selection methods from neural networks.

**Competing Methods** We compare against the following baselines.

1. LassoNet (Lemhadri et al., 2021) is based on a ResNet-like (He et al., 2016) architecture with residual connections. Feature selection is done using hierarchical group lasso.
2. AlgNet (Dinh and Ho, 2020) is based on adaptive lasso and uses proximal full-batch gradient descent for feature selection in a tanh-activated feedforward network.
3. DFS (Chen et al., 2021) solves cardinality constrained feature selection problem with an active-set style method.

We describe the tuning details for these neural network models below. We perform 2000 tuning trials for each method. For DFS, we capped number of trials completed in total 200 GPU hours. DFS is really slow for even medium sized datasets (in terms of feature dimensions).

**LassoNet (Lemhadri et al., 2021)**

- ResNet architecture with 2-layered relu-activated feedforward network with (linear) skip connections.
- Number of hidden units: Discrete uniform in the set $\{\frac{p}{3}, \frac{2}{3}p, p, \frac{4}{3}p\}$,
- Batch sizes: Discrete uniform in the set $\{64, 128, 256, 512\}$,
- $\lambda$: Log uniform over the range $[1, 10000]$.
- Tuning protocol: 1000 for dense training stage and 1000 for each successive sparse training stages with early stopping with a patience of 25. We consider 100 sequential stages (100 values for $\lambda$).

**AlgNet (Dinh and Ho, 2020)**

- Tanh-activated 4-layered feedforward neural network with number of hidden units chosen uniformly from a discrete set $\{\frac{p}{3}, \frac{2}{3}p, p, \frac{4}{3}p\}$,
- Learning rates ($lr$) for proximal GD: Log uniform over the range $[0.01, 1]$.
- $\lambda$: Log uniform over the range $[0.1, 1000]$.
- Number of Epochs: Discrete uniform with range $[5, 2000]$,

**DFS (Chen et al., 2021)**

- ReLU-activated 4-layered feedforward neural network with number of hidden units chosen uniformly from a discrete set $\{\frac{p}{3}, \frac{2}{3}p, p, \frac{4}{3}p\}$,
- Learning rates ($lr$) for Adam: Log uniform over the range $[0.001, 0.1]$.
- Weight decay: 0.0025 for feature selection layer and 0.005 for remaining layers.
- $k$: Number of features selected in the range $[1, 0.5p]$.
- Number of Epochs: 500 with early stopping.

### S5.5 Appendix for Sec. 7.5

We perform 2000 tuning trials for each method.

`Skinny Trees`

- Number of trees: Discrete uniform with range $[1, 100]$,
- Depths: Discrete uniform with range $[1, 5]$,
- Batch sizes: Uniform over the set $\{16, 64, 256, 1024\}$,
- Number of Epochs: Discrete uniform with range $[5, 1000]$,
- $\lambda_0$ (without Dense-to-sparse learning): Log uniform in the range $[1, 10^4]$ for group $\ell_0 - \ell_2$,
- $\lambda_0$ (with Dense-to-sparse learning): exponentially ramped up $0 \to 1$ with temperature $s$ distributed as Log uniform in the range $[10^{-4}, 1]$ for group $\ell_0 - \ell_2$ with DSL,
- $\lambda_2$: Log uniform over the range $[10^{-3}, 1]$ for group $\ell_0 - \ell_2$,
- Learning rates ($lr$): Log uniform over the range $[10^{-2}, 10]$.

### S5.6 Comparison with ControlBurn

We also compare against ControlBurn (Liu et al., 2021) on binary classification tasks. ControlBurn does not support multiclass classification. ControlBurn is another post-training feature selection method, which formulates an optimization problem with group-lasso regularizer to perform feature selection on a pre-trained forest. It relies on a commercial solver to optimize their formulation. We report the results for 25% feature selection budget in Table S4. We can observe that `Skinny Trees` outperforms ControlBurn across many datasets, achieving 2% (up to 6%) improvement in AUC.

Table S4: Comparison of test AUC (%) performance of `Skinny Trees` against ControlBurn for 25% feature selection budget on binary classification tasks.

| Dataset | ControlBurn | `Skinny Trees` |
|---|---|---|
| Churn | 85.66±0.25 | **91.38**±0.08 |
| Madelon | 94.23±0.08 | 94.14±0.09 |
| Gisette | 99.36±0.01 | **99.81**±0.002 |
| Arcene | 77.89±0.32 | **80.80**±0.30 |
| Smk | **79.94**±0.32 | 79.29±0.22 |
| Gli | 98.62±0.16 | **99.80**±0.07 |
| Dorothea | 84.85±0.28 | **90.87**±0.02 |
| Average | 88.65 | **90.87** |

We note the hyperparameter tuning for ControlBurn below:

- Method: bagboost,
- Depths: Discrete uniform with range $[1, 10]$,
- $\alpha$: Log uniform over the range $[10^{-7}, 1.0]$.

## S6 MEASURING STATISTICAL SIGNIFICANCE

We follow the following procedure to test the significance of all models. For all models, we tune over hyperparameters for 2000 trials. We select the optimal trial (within the desired feature sparsity budget) based on

validation set. Next, we train each model for 100 repetitions with the optimal hyperparameters and report the mean results on test set along with the standard errors.