LEARNING BINARY TREES VIA SPARSE RELAXATION

Anonymous authors
Paper under double-blind review

ABSTRACT

One of the most classical problems in machine learning is how to learn binary trees that split data into useful partitions. From classification/regression via decision trees to hierarchical clustering, binary trees are useful because they (a) are often easy to visualize; (b) make computationally-efficient predictions; and (c) allow for flexible partitioning. Because of this there has been extensive research on how to learn such trees that generally fall into one of three categories: 1. greedy node-by-node optimization; 2. probabilistic relaxations for differentiability; 3. mixed-integer programs (MIP). Each of these have downsides: greedy can myopically choose poor splits, probabilistic relaxations do not have principled ways to prune trees, MIP methods can be slow on large problems and may not generalize. In this work we derive a novel sparse relaxation for binary tree learning. By deriving a new MIP and sparsely relaxing it, our approach is able to learn tree splits and tree pruning using argmin differentiation. We demonstrate how our approach is easily visualizable and is competitive with current tree-based approaches in classification/regression and hierarchical clustering.

1 INTRODUCTION

Learning discrete structures from unstructured data is extremely useful for a wide variety of real-world problems (Gilmer et al., 2017; Kool et al., 2018; Yang et al., 2018). One of the most computationally-efficient, easily-visualizable discrete structures that are able to represent complex functions are *binary trees*. For this reason, there has been a massive research effort on how to learn such binary trees since the early days of machine learning (Breiman et al., 1984; İrsoy et al., 2012; Bertsimas & Dunn, 2017; Popov et al., 2019). Learning binary trees has historically been done in one of three ways. The first is via *greedy optimization*, which includes popular decision-tree methods such as classification and regression trees (CART) (Breiman et al., 1984) and ID3 trees (Quinlan, 1986), among many others. These methods optimize a splitting criterion for each tree node, based on the data routed to it. The second set of approaches are based on *probabilistic relaxations*. The idea is to optimize all splitting parameters at once via gradient-based methods, by relaxing hard branching decisions into branching probabilities. The third approach optimizes trees using *mixed-integer programming* (MIP). This jointly optimizes all continuous and discrete parameters to find globally-optimal trees.[1]

Each of these approaches have clear shortcomings. First, greedy optimization is clearly suboptimal: tree splitting criteria are even intentionally crafted to be different than the global tree loss, as the global loss does not encourage tree growth (Breiman et al., 1984). Second, probabilistic relaxations: (a) are rarely sparse, so inputs probabilistically contribute to branches they would never visit if splits are mapped to hard decisions; (b) they do not have principled ways to prune trees, as the distribution over pruned trees is often intractable. Third, MIP approaches, while optimal, are only computationally tractable on training datasets with thousands of inputs (Bertsimas & Dunn, 2017), and do not have well-understood out-of-sample generalization guarantees.

In this paper we present a new approach to binary tree learning based on sparse relaxation and argmin differentiation. Our main insight is that by quadratically relaxing an MIP that learns the discrete parameters of the tree (input traversal and node pruning), we can differentiate through it to simultaneously learn the continuous parameters of splitting decisions. This allows us to leverage the superior generalization capabilities of stochastic gradient optimization to learn splits, and gives a principled approach to learning tree pruning. Further, we can derive customized algorithms to

[1] Here we focus on learning single trees instead of tree ensembles; our work easily extends to ensembles.

compute the forward and backward passes through this program that are much more efficient than generic approaches (Amos & Kolter, 2017). We demonstrate that (a) in classification/regression our method, which learns a single tree and a classifier on top of it, is competitive with greedy, probabilistic, MIP-based tree methods, and even powerful ensemble methods; (b) in hierarchical clustering we match or improve upon the state-of-the-art.

2 Related Work

The paradigm of *binary tree learning* has the goal of finding a tree that iteratively splits data into meaningful, informative subgroups, guided by some criterion. Binary tree learning appears in a wide variety of problem settings across machine learning. We briefly review work in two learning settings where latent tree learning plays a key role: 1. *Classification/Regression*; and 2. *Hierarchical clustering*. Due to the generality of our setup (tree learning with arbitrary split functions, pruning, and downstream objective), our approach can be used to learn trees in any of these settings. Finally, we detail how parts of our algorithm are inspired by recent work in isotonic regression.

Classification/Regression. Decision trees for classification and regression have a storied history, with early popular methods that include classification and regression trees (CART; Breiman et al., 1984), ID3 (Quinlan, 1986), and C4.5 (Quinlan, 1993). While powerful, these methods are greedy: they sequentially identify 'best' splits as those which optimize a split-specific score (often different from the global objective). As such, learned trees are likely sub-optimal for the classification/regression task at hand. To address this, one approach is to probabilistically relax the discrete splitting decisions of the tree (İrsoy et al., 2012; Yang et al., 2018). This allows the (relaxed) tree to be optimized *w.r.t.* the overall objective using gradient-based techniques, with known generalization benefits (Hardt et al., 2016; Hoffer et al., 2017). Variations on this approach aim at learning tree ensembles termed 'decision forests' (Kontschieder et al., 2015; Lay et al., 2018; Popov et al., 2019). The downside of the probabilistic relaxation approach is that there is no principled way to prune these trees as inputs pass through all nodes of the tree with some probability. A recent line of work has explored mixed-integer program (MIP) formulations for learning decision trees. Motivated by the billion factor speed-up in MIP in the last 25 years, Bertsimas & Dunn (2017) phrased the objective of CART as an MIP that could be solved exactly. This resulted in a line of recent follow-up works (Günlük et al., 2018; Aghaei et al., 2019; Verwer & Zhang, 2019; Hu et al., 2019; Aghaei et al., 2020; Lin et al., 2020). Even given this speed-up these methods are only practical on datasets with at most thousands of inputs (Bertsimas & Dunn, 2017). Further, the out-of-sample generalizability of these approaches is not well-studied, unlike stochastic gradient descent learning.

Hierarchical clustering. Compared to standard flat clustering, hierarchical clustering provides a structured organization of unlabeled data in the form of a tree. To learn such a clustering the vast majority of methods are greedy and work in one of two ways: 1. *Agglomerative*: a 'bottom-up' approach that starts each input in its own cluster and iteratively merges clusters; and 2. *Divisive*: a 'top-down' approach that starts with one cluster and recusively splits clusters (Zhang et al., 1997; Widyantoro et al., 2002; Krishnamurthy et al., 2012; Dasgupta, 2016; Kobren et al., 2017; Moseley & Wang, 2017). These methods suffer from similar issues as do greedy approaches to tree learning for classification/regression: they may be sub-optimal for optimizing the overall tree. Further they are often computationally-expensive due to their sequential nature. Inspired by approaches for classification/regression, recent work has designed probabilistic relaxations for learning hierarchical clusterings via gradient-based methods (Monath et al., 2019).

Our work takes inspiration from both the MIP-based and gradient-based approaches. Specifically, we frame learning the discrete tree parameters as an MIP, which we sparsely relax to allow continuous parameters to be optimized by argmin differentiation methods.

Argmin differentiation. Solving an optimization problem as a differentiable module within a parent problem tackled with gradient-based optimization methods is known as argmin differentiation, a particular instance of bi-level optimization (Gould et al., 2016). This situation arises in as diverse scenarios as hyperparameter optimization (Pedregosa, 2016), meta-learning (Rajeswaran et al., 2019), or structured prediction (Stoyanov et al., 2011; Domke, 2013; Niculae et al., 2018). General algorithms for quadratic Amos & Kolter (2017) and disciplined convex programming (Section 7,

Amos, 2019; Agrawal et al., 2019a;b) have been given, as well as expressions for more specific cases like isotonic regression (Djolonga & Krause, 2017). By taking advantage of the very specific structure of the decision tree induction problem, we obtain a direct, efficient algorithm.

Isotonic regression. Also called monotonic regression, isotonic regression (Barlow et al., 1972) constrains the regression function to be non-decreasing/non-increasing. This is useful if one has prior knowledge of such monotonicity (*e.g.*, the mean temperature of the climate is non-decreasing). A classic algorithm is pooling-adjacent-violators (PAV), which optimizes the pooling of adjacent points that violate the monotonicity constraint (Barlow et al., 1972). This initial algorithm has been generalized and incorporated into convex programming frameworks (see Mair et al. (2009) for an excellent summary of the history of isotonic regression and its extensions). Our work builds off of the generalized PAV (GPAV) algorithm of Yu & Xing (2016).

3 METHOD

Given inputs $\{\mathbf{x}_i\}_{i=1}^n$, our goal is to learn a latent binary decision tree $\mathcal{T}$ with maximum depth D. This tree sends each input $\mathbf{x}$ through branching nodes to a specific leaf node in the tree. Specifically, all branching nodes $\mathcal{T}_B \subset \mathcal{T}$ split an input $\mathbf{x}$ by forcing it to go to its left child if $s_\theta(\mathbf{x}) < 0$, and right otherwise. There are three key parts of the tree that need to be identified: 1. The *continuous* parameters $\theta_t \in \mathbb{R}^d$ that describe how s_{θ_t} splits inputs at every node t; 2. The *discrete* paths $\mathbf{z}$ made by each input $\mathbf{x}$ through the tree; 3. The *discrete* choice a_t of whether a node t should be active or pruned. We describe how to learn each of these below.

3.1 TREE-TRAVERSAL & PRUNING PROGRAMS

Imagine the splitting functions of the tree s_{θ_t} are fixed. Given this, the following integer linear program (ILP) describes how inputs $\mathbf{x}$ traverse the tree. The solution $z_{it} \in \{0, 1\}$ indicates if $\mathbf{x}_i$ reaches node $t \in \mathcal{T}$ (for notational simplicity let $\mathbf{z}_i \in \{0, 1\}^{|\mathcal{T}|}$ be the vectorized indicator for $\mathbf{x}_i$),

$$\begin{aligned} \max_{\mathbf{z}_1,\ldots,\mathbf{z}_n} \quad & \sum_{i=1}^n \mathbf{z}_i^\top \mathbf{q}_i \\ \text{s.t. } \forall i \in [n],\ & q_{it} = \min\{r_{it}, l_{it}\} \\ & r_{it} = \min\{s_{\theta_{t'}}(\mathbf{x}_i),\ \ \forall t' \in A_R(t),\} \\ & l_{it} = \min\{-s_{\theta_{t'}}(\mathbf{x}_i),\ \ \forall t' \in A_L(t)\} \\ & z_{it} \in \{0, 1\}. \end{aligned} \tag{1}$$

Here $A_L(t)$ is the set of ancestors of node t whose left child must be followed to get to t, and similarly for $A_R(t)$. The quantities q_{it} (where $\mathbf{q}_i \in \mathbb{R}^{|\mathcal{T}|}$ is the tree-vectorized version of q_{it}) describe the 'reward' of sending $\mathbf{x}_i$ to node t. This is based on how well the splitting functions leading to t are satisfied (q_{it} is positive if all splitting functions are satisfied and negative otherwise).

Notice that the solution is unique so long as $s_{\theta_t}(\mathbf{x}_i) \neq 0$ for all $t \in \mathcal{T}, i \in \{1, \ldots, n\}$ (*i.e.*, $s_{\theta_t}(\mathbf{x}_i) = 0$ means there is no preference to split $\mathbf{x}_i$ left or right). Further note that **the integer constraint on z_{it} can be relaxed to an interval constraint $z_{it} \in [0, 1]$ without loss of generality**. This is because if $s_{\theta_t}(\mathbf{x}_i) \neq 0$ then $z_t = 0$ if and only if $q_t < 0$ and $z_t = 1$ when $q_t > 0$ (and $q_t \neq 0$).

While the above program works for any fixed tree, we would like to be able to also learn the structure of the tree itself. Let $\eta_t \in \mathbb{R}$ be our preference for pruning/keeping node t (larger η_t indicates node t should be kept). To encourage this pruning preference while ensuring connectivity, we introduce an additional optimization variable $a_t \in \{0, 1\}$, indicating if node $t \in \mathcal{T}$ is active (if 1) or pruned (if 0). We may now adapt eq. 1 into the following pruning-aware mixed integer program (MIP):

$$\begin{aligned} \max_{\mathbf{z}_1,\ldots,\mathbf{z}_n,\mathbf{a}} \quad & \sum_{i=1}^n \mathbf{z}_i^\top \mathbf{q}_i + \boldsymbol{\eta}^\top \mathbf{a} \\ \text{s.t. } \forall i \in [n],\ & a_t \leq a_{p(t)}, \quad \forall t \in \mathcal{T} \setminus \{\text{root}\} \\ & z_{it} \leq a_t \\ & z_{it} \in [0, 1], a_t \in \{0, 1\}. \end{aligned} \tag{2}$$

Here we have removed the first three constraints in eq. 1 as they are a deterministic computation independent of $\mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{a}$. Further $p(t)$ indicates the parent of node t. The added constraint $a_t \leq a_{p(t)}$ ensures that child nodes t are pruned if parent nodes $p(t)$ are pruned. While the new constraint $z_{it} \leq a_t$ ensures that no point $\mathbf{x}_i$ can reach node t if node t is pruned.

3.2 Learning Tree Parameters

A natural approach to learn splitting parameters θ_t would be to do so in the MIP itself, as in the optimal tree literature. However, this would severely restrict allowed splitting functions as even linear splitting functions can only practically run on at most thousands of training inputs (Bertsimas & Dunn, 2017). Instead, we propose to learn s_{θ_t} via gradient descent. To do so, we must be able to compute the gradients $\frac{\partial \mathbf{z}}{\partial \boldsymbol{\eta}}$ and $\frac{\partial \mathbf{z}}{\partial \mathbf{q}}$. However, the solutions of eq. 2 are discontinuous and piecewise-constant.

To solve this, we relax the integer constraint on $\mathbf{a}$ to the interval $[0, 1]$ and add quadratic regularization $1/2 \sum_i \|\mathbf{z}_i\|_2^2 + 1/2 \|\mathbf{a}\|_2^2$. Rearranging and negating the objective yields

$$\begin{aligned} \mathrm{T}_{\boldsymbol{\eta}}(\mathbf{q}_1, \ldots, \mathbf{q}_n) = \underset{\mathbf{z}_1,\ldots,\mathbf{z}_n,\mathbf{a}}{\arg\min} \quad & 1/2 \sum_{i=1}^{n} \|\mathbf{z}_i - \mathbf{q}_i\|^2 + 1/2 \|\boldsymbol{\eta} - \mathbf{a}\|^2 \\ \text{s.t. } \forall i \in [n], \quad & a_t \leq a_{p(t)}, \quad \forall t \in \mathcal{T} \setminus \{\text{root}\} \\ & z_{it} \leq a_t \\ & z_{it} \in [0, 1], a_t \in [0, 1]. \end{aligned} \tag{3}$$

The regularization makes the objective strongly convex, so $\mathrm{T}_{\boldsymbol{\eta}}$ is continuous and almost everywhere differentiable (Amos & Kolter, 2017). Rather than relying on generic solvers, by using the tree structure of the constraints, we derive an efficient specialized algorithm. The main insight, shown below, reframes pruned binary tree learning as isotonic optimization.

Proposition 1. *Let* $\mathcal{C} = \left\{\mathbf{a} \in \mathbb{R}^{|\mathcal{T}|} : a_t \leq a_{p(t)} \text{ for all } t \in \mathcal{T} \setminus \{\text{root}\}\right\}$. *Consider*

$$\mathbf{a}^\star = \underset{\mathbf{a} \in \mathcal{C} \cap [0,1]^{|\mathcal{T}|}}{\arg\min} \sum_{t \in \mathcal{T}} \left(1/2 (a_t - \eta_t)^2 + \sum_{i : a_t \leq q_{it}} 1/2 (a_t - q_{it})^2 \right). \tag{4}$$

Define[2] $[z^\star]_{it} = \mathrm{Proj}_{[0, a_t^\star]}(q_{it})$. *Then,* $\mathrm{T}_{\boldsymbol{\eta}}(\mathbf{q}_1, \ldots, \mathbf{q}_n) = \mathbf{z}_1^\star, \ldots, \mathbf{z}_n^\star, \mathbf{a}^\star$.

Proof. The constraints and objective function of eq. 3 are separable, so we may push the minimization *w.r.t.* $\mathbf{z}$ inside the objective, resulting in:

$$\underset{\mathbf{a} \in \mathcal{C} \cap [0,1]^{|\mathcal{T}|}}{\arg\min} \quad 1/2 \|\boldsymbol{\eta} - \mathbf{a}\|^2 + \sum_{t \in \mathcal{T}} \sum_{i=1}^{n} \min_{0 \leq z_{it} \leq a_t} 1/2 (z_{it} - q_{it})^2 . \tag{5}$$

Each of the inner nested minimizations, $\min_{0 \leq z_{it} \leq a_t} 1/2 (z_{it} - q_{it})^2$ is a one-dimensional projection onto box constraints, with solution $z_{it}^\star = \mathrm{Proj}_{[0, a_t]}(q_{it})$. We may use this result to eliminate $\mathbf{z}$ from the objective, noting that

$$1/2 (z_{it}^\star - q_{it})^2 = \begin{cases} 1/2 \, q_{it}^2, & q_{it} < 0 \\ 0, & 0 \leq q_{it} < a_t \\ 1/2 (a_t - q_{it})^2, & q_{it} \geq a_t \end{cases} \tag{6}$$

the first two conditions are constants *w.r.t.* a_t Thus, the objective functions eq. (3) and eq. (4) differ by a constant. As their constraints are also the same, they have equivalent minimizers $\mathbf{a}^\star$. □

Efficiently inducing trees as isotonic optimization. From Proposition 1, notice that eq. 4 is an instance of tree-structured isotonic optimization: the objective decomposes over *nodes*, and the inequality constraints correspond to edges in a rooted tree:

$$\underset{\mathbf{a} \in \mathcal{C}}{\arg\min} \sum_{t \in \mathcal{T}} g_t(a_t) , \quad \text{where} \quad g_t(a_t) = 1/2 (a_t - \eta_t)^2 + \sum_{i : a_t \leq q_{it}} 1/2 (a_t - q_{it})^2 + \iota_{[0,1]}(a_t). \tag{7}$$

[2] Here $\mathrm{Proj}_{\mathcal{S}}(x)$ is the projection of x onto set $\mathcal{S}$. If $\mathcal{S}$ are box constraints, projection amounts to clipping.

Algorithm 1 Latent decision tree induction via isotonic optimization.

initial partition $\mathcal{G} \leftarrow \{\{1\}, \{2\}, \cdots\} \subset 2^{\mathcal{T}}$
for all $G \in \mathcal{G}$ **do**
  $d_G \leftarrow \arg\min_a \sum_{t \in G} g_t(a)$ ▷ Proposition 2
while exists t such that $a_t > a_{p(t)}$ **do**
  $t_{\text{max}} \leftarrow \arg\max_t \{a_t : a_t > a_{p(t)}\}$
  merge $G \leftarrow G \cup G'$ where $t \in G$ and $p(t) \in G'$.
  update $a_G \leftarrow \arg\min_a \sum_{t \in G} g_t(a)$ ▷ Proposition 2 again

where $\iota_{[0,1]}(a_t) = \infty$ if $a_t \notin [0, 1]$ and 0 otherwise. This problem can be solved by a generalized *pool adjacent violators* (PAV) algorithm: Obtain a tentative solution by ignoring the constraints, then iteratively remove violating edges $a_t > a_{p(t)}$ by *pooling together* the nodes at the end points. At the optimum, the nodes are organized into a partition $\mathcal{G} \subset 2^{\mathcal{T}}$, such that if two nodes t, t' are in the same group $G \in \mathcal{G}$, then $a_t = a'_t := a_G$.

When the inequality constraints are the edges of a rooted tree, as is the case here, the PAV algorithm finds the optimal solution in at most $|\mathcal{T}|$ steps, where each involves updating the a_G value for a newly-pooled group by solving a one-dimensional subproblem of the form (Yu & Xing, 2016)[3]

$$a_G = \arg\min_{a \in \mathbb{R}} \sum_{t \in G} g_t(a)\,, \tag{8}$$

resulting in Algorithm 1. It remains to show how to solve eq. 8. The next result, proved in Appendix A.1, gives an exact and efficient solution, with an algorithm that requires finding the nodes with highest q_{it} (*i.e.*, the nodes where $\mathbf{x}_i$ is most highly encouraged to traverse).

Proposition 2. *The solution to the one-dimensional problem in eq. (8) for any G is given by*

$$\arg\min_{a \in \mathbb{R}} \sum_{t \in G} g_t(a) = \text{Proj}_{[0,1]}\left(a(k^\star)\right) \quad \textit{where} \quad a(k) := \frac{\sum_{t \in G} \eta_t + \sum_{(i,t) \in S(k^\star)} q_{it}}{|G| + k^\star}, \tag{9}$$

$S(k) = \{j^{(1)}, \ldots, j^{(k)}\}$ is the set of indices $j = (i, t) \in \{1, \ldots, n\} \times G$ into the k highest values of $\mathbf{q}$, i.e., $q_{j^{(1)}} \geq q_{j^{(2)}} \geq \ldots \geq q_{j^{(m)}}$, and $k^\star$ is the smallest k satisfying $a(k) > q_{j^{(k+1)}}$.

Figure 2 shows the speed of our specialized algorithm compared to a leading generic optimizer (details in Appendix A.2).

Backward pass and efficient implementation details. Algorithm 1 is a sequence of differentiable operations that can be implemented *as is* in automatic differentiation frameworks. However, because of the prominent loops and indexing operations, we opt for a low-level implementation as a `C++` extension. Since the $\mathbf{q}$ values are constant *w.r.t.* $\mathbf{a}$, we only need to sort them once as preprocessing. For the backward pass, rather than relying on automatic differentiation, we make two remarks about the form of $\mathbf{a}$. Firstly, its elements are organized in groups, *i.e.*, $a_t = a'_t = a_G$ for $\{t, t'\} \subset G$. Secondly, the value a_G inside each group depends only on the optimal support set $S^\star_G := S(k^\star)$ as defined for each subproblem by Proposition 2. Therefore, in the forward pass, we must store only the node-to-group mappings and the sets $S^\star_G$. Then, if G is the group of node t,

$$\frac{\partial a^\star_t}{\partial \eta_{t'}} = \begin{cases} \frac{1}{|G|+k^\star}, & 0 < a^\star_t < 1 \text{ and } t' \in G, \\ 0, & \text{otherwise.} \end{cases} \quad \frac{\partial a^\star_t}{\partial q_{it'}} = \begin{cases} \frac{1}{|G|+k^\star}, & 0 < a^\star_t < 1, \text{ and } (i, t') \in S^\star_G, \\ 0, & \text{otherwise.} \end{cases}.$$

3.3 The Overall Objective

We are now able to describe the overall optimization procedure that simultaneously learns tree parameters (a) input traversals $\mathbf{z}_1, \ldots, \mathbf{z}_n$; (b) tree pruning $\mathbf{a}$; and (c) split parameters θ_t. Given this

[3]Compared to Yu & Xing (2016), our tree inequalities are in the opposite direction. This is equivalent to a sign flip of parameter $\mathbf{a}$, *i.e.*, to selecting the *maximum* violator rather than the minimum one at each iteration.

tree, we will additionally learn a function $f_\phi(\mathbf{z}, \mathbf{x})$ to minimize an arbitrary loss $\ell(\cdot)$ as follows,

$$\begin{aligned} \min_{\boldsymbol{\theta}, \boldsymbol{\eta}, \boldsymbol{\phi}} \quad & \sum_{i=1}^{n} \ell\big(f_\phi(\mathbf{x}_i, \mathbf{z}_i)\big) + \lambda \|\boldsymbol{\eta}\|_\Omega \\ \text{where} \quad & \mathbf{z}_1, \ldots, \mathbf{z}_n, \mathbf{a} := \mathrm{T}_{\boldsymbol{\eta}}\big(q_{\boldsymbol{\theta}}(\mathbf{x}_1), \ldots, q_{\boldsymbol{\theta}}(\mathbf{x}_n)\big) \,. \end{aligned} \tag{10}$$

Recall $\mathrm{T}_{\boldsymbol{\eta}}(\cdot)$ is the solution to the tree program (eq. 3), so we may update the parameters $\boldsymbol{\eta}$ and $\boldsymbol{\theta}$ by back-propagation. Here we define $q_\theta(\mathbf{x}_i) := \mathbf{q}_i$ to make explicit the dependence of $\mathbf{q}_i$ on θ. The regularization $\| \cdot \|_\Omega$ can be any norm (in practice we find $\Omega = \infty$ to perform the best.)

4 EXPERIMENTS

In this section we showcase our method on both: (a) *Classification/Regression* for tabular data, where tree-based models have been demonstrated to have superior performance over MLPs (Popov et al., 2019); and (b) *Hierarchical clustering* on unsupervised data. Our experiments demonstrate that our method leads to predictors that are competitive with state-of-the-art tree-based approaches. Further we visualize the trees learned by our method, and how sparsity is easily adjusted by tuning the regularization parameter λ.

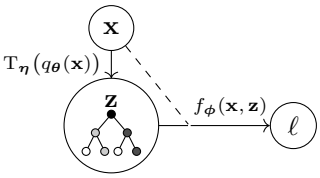


Figure 1: Model overview.

Architecture details. We use a linear function or a multi-layer perceptron (L fully-connected layers with Elu activation (Clevert et al., 2015) and dropout) for $f_\phi(\cdot)$ and choose between linear or linear followed by Elu splitting functions $s_\theta(\cdot)$ (we limit the search for simplicity, there are no restrictions except differentiability).

4.1 SUPERVISED LEARNING ON TABULAR DATASETS

Our first set of experiments is on supervised learning with heterogeneous tabular datasets, where we consider both regression and binary classification tasks. We minimize the Mean Squared Error (MSE) on regression datasets and the Binary Cross-Entropy (BCE) on classification datasets. We compare our results with tree-based architectures, which either train a single or an ensemble of decision trees. Namely, we compare to the greedy classification and regression trees (CART) (Breiman et al., 1984) and to the optimal decision tree learner (OPTREE; Bertsimas & Dunn, 2017). We also consider two baselines with probabilistic routing: deep neural decision trees (DNDT; Yang et al., 2018) and deep neural decision forests (Kontschieder et al., 2015) configured to use an ensemble size of 1 (NDF-1). As for the ensemble baselines, we compare to NODE (Popov et al., 2019), the state-of-the-art method for training a forest of differentiable oblivious decision trees on tabular data, and to XGBoost (Chen & Guestrin, 2016), a scalable tree boosting method. We carry out the experiments on the following datasets: (Dua & Graff, 2017) **Regression.** *Year* (Bertin-Mahieux et al., 2011): Temporal regression task constructed from the Million Song Dataset; *Microsoft* (Qin & Liu, 2013): Regression approach to the MSLR-Web10k Query–URL relevance prediction for learning to rank; *Yahoo* (Chapelle & Chang, 2011): Regression approach to the C14 learning-to-rank challenge. **Binary classification.** *Click*: Link click prediction based on the KDD Cup 2012 dataset, encoded and subsampled following Popov et al. (2019); *Higgs* (Baldi et al., 2014): prediction of Higgs boson–producing events.

For all tasks, we follow the preprocessing and task setup from (Popov et al., 2019). All datasets come with training/test splits. We make use of 20% of the training set as validation set for selecting the best model over the epochs and for tuning the hyperparameters. We tune the hyperparameters for all methods but for OPTREE, as its long training time makes hyper-parameter tuning unfeasible. Details are provided in the appendices. Finally, we optimize eq. (10) and all neural network methods (DNDT, NDF, and NODE) using the Quasi-Hyperbolic Adam (Ma & Yarats, 2018) stochastic gradient descent method, with default parameters and batch size equal to 512. Table 1 reports the obtained results.[4] Unsurprisingly, ensemble methods outperfom single-tree ones on all datasets, although at the cost of being harder to visualize/interpret. Our method has the advantage of (a) generalizing to any task; (b) outperforming or coming in second for all single-tree methods; (c) approaching the performance of ensemble-based methods.

[4]DNDT, OPTREE and NDF handle only classification tasks.

Table 1: Results on tabular datasets. We report average and standard deviations over 4 runs of MSE for regression datasets: *Year*, *Microsoft* and *Yahoo*, and error rate for classification datasets: *Click* and *Higgs*. Best result, and those within a standard deviation from it, for each family of algorithms (single tree or ensemble) are in **bold**. Dashes '-' indicate that the method cannot be run (i.e., a method is designed only for classification). Experiments are run on a machine with 16 CPUs and 63GB of RAM, with a training time limit of 3 days. We denote methods that exceed this memory and training time as OOM and OOT, respectively.

	method	YEAR	MICROSOFT	YAHOO	CLICK	HIGGS
Single Tree	CART	97.40 ± 0.37	$0.6914 \pm$ 3e-3	$0.6190 \pm$ 3e-3	$0.3780 \pm$ 6e-3	$0.3220 \pm$ 1e-3
	DNDT	-	-	-	$0.4866 \pm$ 1e-2	OOM
	OPTREE	-	-	-	$0.4100 \pm$ 4e-2	OOT
	NDF-1	-	-	-	**$0.3344 \pm$ 5e-4**	**$0.2644 \pm$ 8e-4**
	Ours	**77.52 ± 0.50**	**$0.5772 \pm$ 12e-4**	**$0.5937 \pm$ 6e-4**	$0.3383 \pm$ 15e-4	$0.2995 \pm$ 16e-4
Ens.	NODE	**76.21 ± 0.12**	$0.5570 \pm$ 2e-4	$0.5692 \pm$ 2e-4	**$0.3312 \pm$ 2e-3**	**$0.2101 \pm$ 5e-4**
	XGBoost	78.53 ± 0.09	**$0.5544 \pm$ 1e-4**	**$0.5420 \pm$ 4e-4**	**$0.3310 \pm$ 2e-3**	$0.2328 \pm$ 3e-4

Table 2: Results for hierarchical clustering. We report average and standard deviations of *dendrogram purity* over four runs. Best results and all within a standard deviation from it are in bold.

METHOD	GLASS	COVTYPE
Ours	**0.494 ± 0.025**	**0.467 ± 0.006**
gHHC	0.463 ± 0.002	0.444 ± 0.005
HKMeans	**0.508 ± 0.008**	0.440 ± 0.001
BIRCH	0.429 ± 0.013	0.440 ± 0.002

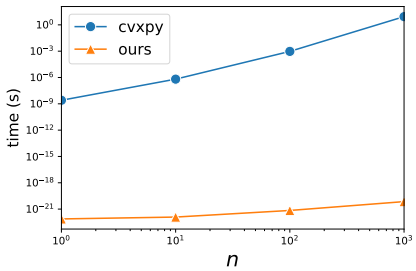


Figure 2: Comparison of Algorithm 1's running time (ours) with the running time of the cvxpy (Diamond & Boyd, 2016; Agrawal et al., 2018) with OSQP solver (Stellato et al., 2020) for tree depth $D = 3$, varying n.

4.2 Self-Supervised Hierarchical Clustering

To show the versatility of our method, we carry out a second set of experiments on hierarchical clustering tasks. Inspired by the recent success of self-supervised learning approaches (Lan et al., 2019; He et al., 2020), we learn a tree for hierarchical clustering in a self-supervised way. Specifically, we regress a subset of input features from the remaining features, minimizing the MSE. This allows us to use eq. (10) to learn a hierarchy (tree). To evaluate the quality of the learned trees, we compute its dendrogram purity (DP; Monath et al., 2019). DP measures the ability of the learned tree to separate points from different classes, and corresponds to the expected purity of the least common ancestors of points of the same class.

We experiment on the following datasets: *Glass* (Dua & Graff, 2017): glass identification for forensics, and *Covtype* (Blackard & Dean, 1999; Dua & Graff, 2017): cartographic variables for forest cover type identification. For *Glass*, we regress features 'Refractive Index' and 'Sodium,' and for *Covtype* the horizontal and vertical 'Distance To Hydrology.' We split the datasets into training/validation/test sets, with sizes 60%/20%/20%. Here we only consider linear f_ϕ. As before, we optimize Problem 10 using the Quasi-Hyperbolic Adam algorithm, with batch size equal to 512 for *Covtype* and 8 for *Glass*, and tune the hyper-parameters using the validation set.

As baselines, we consider: BIRCH (Zhang et al., 1996) and Hierarchical KMeans (HKMeans), the standard methods for performing clustering on large datasets; and the recently proposed gradient-based Hyperbolic Hierarchical Clustering (gHHC) (Monath et al., 2019) designed to construct trees in hyperbolic space. Table 2 reports the dendrogram purity scores for all methods. Our method matches or outperforms all methods, even though not specifically tailored to hierarchical clustering.

Tree Pruning The hyper-parameter λ in eq. (10) controls how aggressively the tree is pruned, hence the amount of tree splits that are actually used to make decisions. This is a fundamental feature

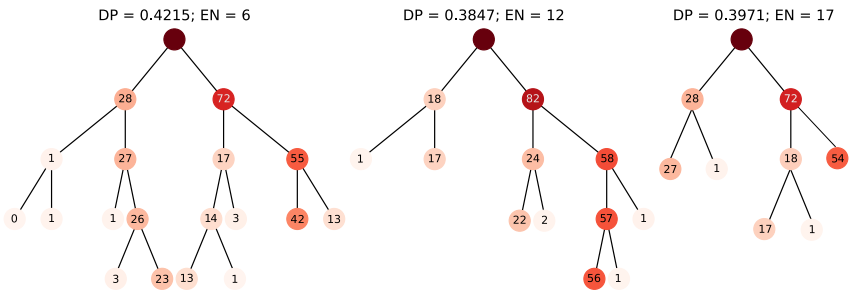
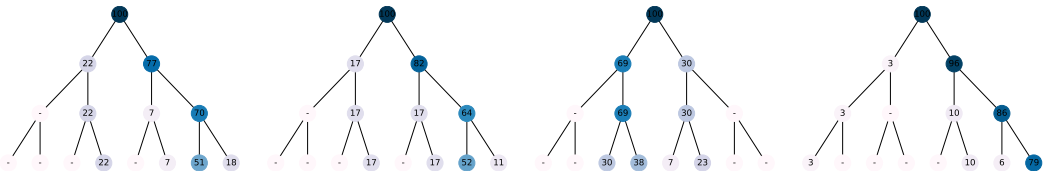


Figure 3: *Glass* tree routing distribution, in rounded percent of dataset, for λ left-to-right in $\{0, 0.001, 0.1\}$. The larger λ, the more nodes are pruned. We report dendrogram purity (DP) and total empty nodes (EN).



(a) Window-Float-Build (b) Window-Float-Vehicle (c) NonW-Containers (d) NonW-Headlamps

Figure 4: Class routing distributions on *Glass*, with distributions normalized over each depth level. The decision tree was trained with optimal hyper-parameters, (depth $D = 6$), but we plot nodes up to level $D = 4$ for visualization ease.

of our framework as it allows to smoothly trim the portions of the tree that are not necessary for the downstream task, resulting in lower computing and memory demands at inference. In Figure 3, we study the effects of pruning on the tree learned on *Glass* with a depth fixed to $D=3$. We report how inputs are distributed over the learned tree for different values of λ. We notice that the number of empty nodes, *i.e.*, nodes that are not traversed by any data point, increases together with λ up to a certain value in order not to significantly deteriorate results (as measured by dendrogram purity).

Class Routing In order to gain insights on the latent structure learned by our method, we study how points are routed through the tree, depending on their class. The *Glass* dataset is particularly interesting to analyze as its classes come with an intrinsic hierarchy, *e.g.*, with superclasses *Window* and *NonWindow*. Figure 4 reports the class routes for four classes. As the trees are constructed without supervision, we do not expect the structure to exactly reflect the class partition and hierarchy. Still, we observe that points from the same class or super-class to traverse the tree in a similar way. Indeed, trees for class *Build* 4(a) and class *Vehicle* 4(b), which both belong to *Window* super-class, share similar paths, unlike the classes Containers 4(c) and Headlamps 4(d).

5 Discussion

In this work we have presented a new optimization approach to learn trees for a variety of machine learning tasks. Our method works by sparsely relaxing a MIP for tree traversal and pruning, to enable simultaneous optimization of these parameters, alongside splitting parameters and downstream functions via argmin differentiation. Our approach nears or improves upon recent work in both supervised learning and hierarchical clustering. We believe there are many exciting avenues for future work. One particularly interesting direction would be to unify recent advances in tight relaxations of nearest neighbor classifiers with this approach to learn efficient neighbor querying structures such as ball trees. Another idea is to adapt this method to learn instance-specific trees such as parse-trees.

References

Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning optimal and fair decision trees for non-discriminative decision-making. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1418–1426, 2019.

Sina Aghaei, Andres Gomez, and Phebe Vayanos. Learning optimal classification trees: Strong max-flow formulations. *arXiv preprint arXiv:2002.09142*, 2020.

Akshay Agrawal, Robin Verschueren, Steven Diamond, and Stephen Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J. Zico Kolter. Differentiable convex optimization layers. In *Proc. of NeurIPS*, 2019a.

Akshay Agrawal, Shane Barratt, Stephen Boyd, Enzo Busseti, and Walaa M Moursi. Differentiating through a cone program. *Journal of Applied and Numerical Optimization*, 2019(2), 2019b. ISSN 25625527. doi: 10.23952/jano.1.2019.2.02. URL `http://jano.biemdas.com/archives/931`.

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

Brandon Amos. *Differentiable Optimization-Based Modeling for Machine Learning*. PhD thesis, Carnegie Mellon University, May 2019.

Brandon Amos and J Zico Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proc. of ICML*, 2017.

Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, 5(1):1–9, 2014.

RE Barlow, DJ Bartholomev, JM Brenner, and HD Brunk. Statistical inference under order restrictions: The theory and application of isotonic regression, 1972.

Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proc. of ISMIR*, 2011.

Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

Jock A Blackard and Denis J Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In *Proceedings of the learning to rank challenge*, pp. 1–24, 2011.

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, pp. 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939785. URL `http://doi.acm.org/10.1145/2939672.2939785`.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Sanjoy Dasgupta. A cost function for similarity-based hierarchical clustering. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pp. 118–127, 2016.

Steven Diamond and Stephen Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

Josip Djolonga and Andreas Krause. Differentiable learning of submodular models. In *Proc. of NeurIPS*, 2017.

Justin Domke. Learning graphical model parameters with approximate marginal inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(10):2454–2467, 2013.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272, 2017.

Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. *preprint arXiv:1607.05447*, 2016. URL http://arxiv.org/abs/1607.05447.

Oktay Günlük, Jayant Kalagnanam, Matt Menickelly, and Katya Scheinberg. Optimal decision trees for categorical data via integer programming. *arXiv preprint arXiv:1612.03225*, 2018.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020. URL http://www.gurobi.com.

Moritz Hardt, Ben Recht, and Yoram Singer. Train faster, generalize better: Stability of stochastic gradient descent. In *International Conference on Machine Learning*, pp. 1225–1234, 2016.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9729–9738, 2020.

Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pp. 1731–1741, 2017.

Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. In *NeurIPS*, pp. 7267–7275, 2019.

Ozan İrsoy, Olcay Taner Yıldız, and Ethem Alpaydın. Soft decision trees. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pp. 1819–1822. IEEE, 2012.

Ari Kobren, Nicholas Monath, Akshay Krishnamurthy, and Andrew McCallum. A hierarchical algorithm for extreme clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 255–264, 2017.

Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, pp. 1467–1475, 2015.

Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

Akshay Krishnamurthy, Sivaraman Balakrishnan, Min Xu, and Aarti Singh. Efficient active algorithms for hierarchical clustering. In *ICML*, 2012.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*, 2019.

Nathan Lay, Adam P Harrison, Sharon Schreiber, Gitesh Dawer, and Adrian Barbu. Random hinge forest for differentiable learning. *arXiv preprint arXiv:1802.03882*, 2018.

Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. Generalized optimal sparse decision trees. In *ICML*, 2020.

Jerry Ma and Denis Yarats. Quasi-hyperbolic momentum and adam for deep learning. *arXiv preprint arXiv:1810.06801*, 2018.

Patrick Mair, Kurt Hornik, and Jan de Leeuw. Isotone optimization in r: pool-adjacent-violators algorithm (pava) and active set methods. *Journal of statistical software*, 32(5):1–24, 2009.

Nicholas Monath, Manzil Zaheer, Daniel Silva, Andrew McCallum, and Amr Ahmed. Gradient-based hierarchical clustering using continuous representations of trees in hyperbolic space. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 714–722, 2019.

Benjamin Moseley and Joshua Wang. Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search. In *Advances in Neural Information Processing Systems*, pp. 3094–3103, 2017.

Vlad Niculae, André FT Martins, Mathieu Blondel, and Claire Cardie. SparseMAP: Differentiable sparse structured inference. In *Proc. of ICML*, 2018.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *Proc. of ICML*, 2016.

Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. *arXiv preprint arXiv:1909.06312*, 2019.

Tao Qin and Tie-Yan Liu. Introducing LETOR 4.0 datasets. *CoRR*, abs/1306.2597, 2013. URL `http://arxiv.org/abs/1306.2597`.

J Ross Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

J Ross Quinlan. C4. 5: Programs for machine learning. 1993.

Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. In *Proc. of NeurIPS*, 2019.

Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. OSQP: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 2020. doi: 10.1007/s12532-020-00179-2. URL `https://doi.org/10.1007/s12532-020-00179-2`.

Veselin Stoyanov, Alexander Ropson, and Jason Eisner. Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Proc. of AISTATS*, 2011.

Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1625–1632, 2019.

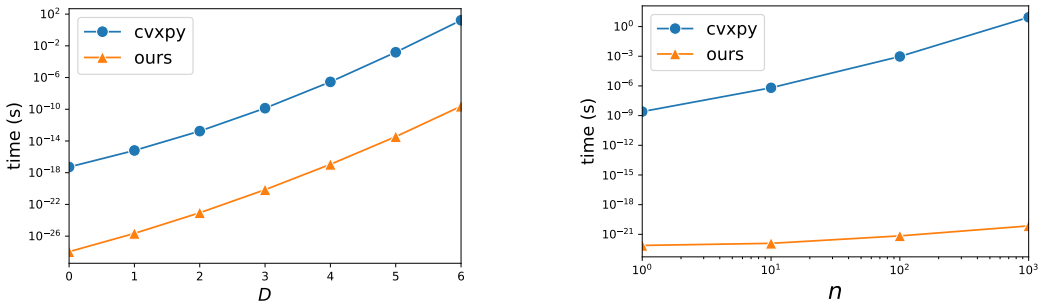
Dwi H Widyantoro, Thomas R Ioerger, and John Yen. An incremental approach to building a cluster hierarchy. In *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pp. 705–708. IEEE, 2002.

Yongxin Yang, Irene Garcia Morillo, and Timothy M Hospedales. Deep neural decision trees. *arXiv preprint arXiv:1806.06988*, 2018.

Yao-Liang Yu and Eric P Xing. Exact algorithms for isotonic regression and related. In *Journal of Physics: Conference Series*, volume 699, pp. 1–9, 2016.

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. *ACM sigmod record*, 25(2):103–114, 1996.

Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.



(a) Fixed number of points $n = 100$

(b) Fixed tree depth $D = 3$

Figure 5: Comparison of Algorithm 1's running time (ours) with the running time of **Cvxpy** with OSQP solver. n takes values in a range that covers common choices of batch size. Time and n axis are represented in logarithmic scale.

A Appendix

A.1 Proof of Proposition 2

Let $G \subset \mathcal{T}$ be a subset of (pooled) node indices. We seek to solve

$$\arg\min_{a \in \mathbb{R}} \sum_{t \in G} g_t(a) \;:=\; \arg\min_{a \in [0,1]} \sum_{t \in G} {}^1\!/_2 (a - \eta_t)^2 + \sum_{i : a \leq q_{it}} {}^1\!/_2 (a - q_{it})^2 \tag{11}$$

Note that the final summation implicitly depends on the unknown a. But, regardless of the value of a, if $q_{it} \leq q_{i't'}$ and q_{it} is included in the sum, then $q_{i't'}$ must also be included by transitivity. We may therefore characterize the solution $a^\star$ via the number of active inequality constraints $k^\star = \big|\{(i,t) : a^\star \leq q_{i,t}\}\big|$. We do not know $a^\star$, but it is trivial to find by testing all possible values of k. For each k, we may find the set $S(k)$ defined in the proposition. For a given k, the candidate objective is

$$J_k(a) = \sum_{t \in G} {}^1\!/_2 (a - \eta_t)^2 + \sum_{(i,t) \in S(k)} {}^1\!/_2 (a - q_{it})^2 \tag{12}$$

and the corresponding $a(k)$ minimizing it can be found by setting the gradient to zero:

$$J_k'(a) = \sum_{t \in G} (a - \eta_t) + \sum_{(i,t) \in S(k)} (a - q_{i,t}) := 0 \quad \Longleftrightarrow \quad a(k) = \frac{\sum_{t \in G} \eta_t + \sum_{(i,t) \in S(k)} q_{it}}{|G| + k}. \tag{13}$$

Since $|S(k)| = k$ and each increase in k adds a non-zero term to the objective, we must have $J_1\big(a(1)\big) \leq J_1\big(a(2)\big) \leq J_2\big(a(2)\big) \leq \ldots$, so we must take k to be as small as possible, while also ensuring the condition $|\{(i,t) : a(k) \leq q_{it}\}| = k$, or, equivalently, that $a(k) > q_{j^{([k+1])}}$. The box constraints may be integrated at this point via clipping, yielding $a^\star = \mathrm{Proj}_{[0,1]}\big(a(k^\star)\big)$.

A.2 Benchmarking solver times

We study the running time of Algorithm 1 depending on the number of data points n and the chosen tree depth D. We compare its solving time with the time needed by the popular convex optimization framework **Cvxpy** (Diamond & Boyd, 2016; Agrawal et al., 2018) to solve Problem (3) with OSQP solver (Stellato et al., 2020)[5]. We report the solving times in Figure 5, where we vary one parameter n or D at a time and fix the other. The algorithm that we specifically designed to solve Problem (3) is indeed several magnitude faster than the considered generic solver.

[5]We ran experiments with the commercial solver GUROBI (Gurobi Optimization, 2020) but didn't find significant differences with using OSQP.

A.3 HYPER-PARAMETER TUNING DETAILS

We tune the hyper-parameters of all methods with Optuna Bayesian optimizer (Akiba et al., 2019), fixing the number of trials to 100. For all back-propagation-base methods, we fix the learning rate for solving eq. (10) to 0.001 and use a scheduler reducing this parameter of a factor of 10 every 2 epochs where the validation loss has not improved.

Experiments on tabular datasets The other hyper-parameters for these experiments are chosen as follows:

- **Ours**: we tune D in $\{2, \dots, 6\}$, λ in $[1\text{e-}3, 1\text{e+}3]$ with log-uniform draws, the number of layers of the MLP L in $\{2, 5\}$ and its dropout probability uniformly in $[0, 0.5]$, and the choice of activation for the splitting functions as linear or Elu;
- **Optree**: we fix the tree depth $D = 6$;
- **CART**: we tune D in $\{2, \dots, 6\}$, *feature rate* uniformly in $[0.5, 1]$, *minimal impurity decrease* in $[0, 1]$, α log-uniformly in $[1\text{e-}16, 1\text{e+}2]$ and *splitter* chosen between best or random;
- **Node** and **XGBoost**: all results are reported from Popov et al. (2019), where they used the same experimental set-up;
- **DNDT**: we tune the softmax temperature uniformly between $[0, 1]$ and the number of feature cuts in $\{1, 2\}$;
- **NDF**: we tune D in $\{2, \dots, 6\}$ and fix the feature rate to 1.

Experiments on hierarchical clustering For this set of experiments, we make us of a linear predictor and of linear splitting functions without activation. The other hyper-parameters of our method are chosen as follows: we tune D in $\{2, \dots, 6\}$, λ in $[1\text{e-}3, 1\text{e+}3]$ with log-uniform draws. The results of the baselines are reported from Monath et al. (2019).