# Optimal Sparse Regression Trees

**Rui Zhang[1*], Rui Xin[1*], Margo Seltzer[2], Cynthia Rudin[1]**

[1] Duke University
[2] University of British Columbia
{r.zhang, rui.xin926, cynthia.rudin}@duke.edu, {mseltzer}@cs.ubc.ca

**Abstract**

Regression trees are one of the oldest forms of AI models, and their predictions can be made without a calculator, which makes them broadly useful, particularly for high-stakes applications. Within the large literature on regression trees, there has been little effort towards full provable optimization, mainly due to the computational hardness of the problem. This work proposes a dynamic-programming-with-bounds approach to the construction of provably-optimal sparse regression trees. We leverage a novel lower bound based on an optimal solution to the k-Means clustering algorithm on one dimensional data. We are often able to find optimal sparse trees in seconds, even for challenging datasets that involve large numbers of samples and highly-correlated features.

## 1 Introduction

*Regression trees* are one of the oldest and most popular forms of machine learning model, dating back to the 1963 AID algorithm of Morgan and Sonquist (1963). Since then, there has been a vast amount of work on regression trees, the overwhelming majority of which involves greedy tree induction and greedy pruning (Breiman et al. 1984; Quinlan 1993; Payne and Meisel 1977; Loh 2002). In these approaches, trees are grown from the top down, with greedy splitting at each branch node, and greedy pruning afterwards. These techniques are easy and fast, but their trees have no notion of global optimality. Greedily-grown trees can be much larger than necessary, sacrificing interpretability, and their performance suffers when compared to other machine learning approaches. Thus, questions remain – is it possible to create optimal regression trees? Would they be competitive with other machine learning algorithms if they were fully optimized? Certainly there would be many uses for sparse interpretable regression trees if we could create them with accuracy comparable to that of other machine learning approaches.

While the quest for fully-optimal decision trees began in the mid-90's with the work of Bennett and Blue (1996), fully optimal decision tree learning was rarely attempted over the last three decades, owing to the computational hardness of the problem. Works that did attempt it (Dobkin et al. 1997; Farhangfar, Greiner, and Zinkevich 2008; Narodytska et al. 2018; Janota and Morgado 2020; Shati, Cohen, and McIlraith 2021; Hu et al. 2020; Avellaneda 2020) had strong constraints, such as shallow depth or perfect classification accuracy. For classification (rather than regression), scientists have had recent success in producing fully optimal trees (Günlük et al. 2021; Blanquero et al. 2020; Hu, Rudin, and Seltzer 2019; Verwer and Zhang 2019; Angelino et al. 2017; Lin et al. 2020; McTavish et al. 2022; Farhangfar, Greiner, and Zinkevich 2008; Nijssen and Fromont 2007, 2010; Aghaei, Gomez, and Vayanos 2020; Verhaeghe et al. 2019; Nijssen, Schaus et al. 2020; Nijssen and Fromont 2010; Demirović et al. 2022) using mathematical programming or dynamic programming. However, building sparse optimal classification trees is a much easier problem, since the 0-1 loss has natural discrete lower bounds, and binary integer programming can be used; this is not true of regression, which uses (real-valued) mean squared error as its loss function.

Let us discuss the few works that do address challenges resembling optimal regression trees. The works of Blanquero et al. (2022) and Bertsimas, Dunn, and Wang (2021) do not construct traditional sparse trees with constant predictions in the leaves; their leaf nodes contain linear or polynomial classifiers, thus the formula for producing predictions is quite complex. The former (Blanquero et al. 2022) uses $\ell_\infty + \ell_1$ regularization for the linear models within the nodes and the latter (Bertsimas, Dunn, and Wang 2021) uses $\ell_2$ regularization for polynomial models in the leaves. Neither of these regularize the number of leaves. The evtree algorithm (Grubinger, Zeileis, and Pfeiffer 2014) claims to construct globally optimal trees, but since it is purely an evolutionary method (no bounds are used to reduce the search space), there is no guarantee of reaching optimality, and one never knows whether optimality has already been reached. Dunn (2018) and Verwer and Zhang (2017) provide mathematical programming formulas for optimal regression trees, but no open source code is available; regardless, mathematical programming solvers are generally slow. Interpretable AI (2022) provides proprietary software that requires a license, but it is not possible to ascertain whether it uses local search or mathematical programming; we suspect it uses local search heuristics, despite the claim of optimal solutions. In other words, as far as we know, there is no other prior

[*] These authors contributed equally.

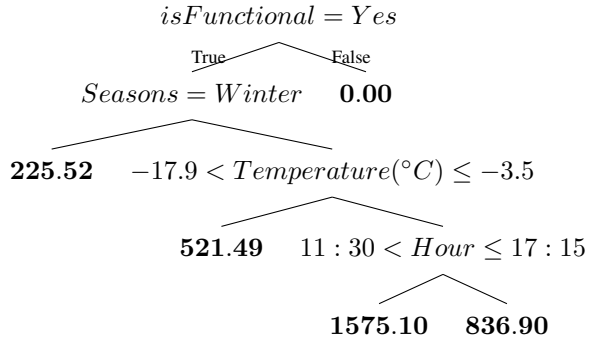



Figure 1: Optimal regression tree for *seoul bike* dataset with $\lambda = 0.05$, *max depth* $= 5$. This dataset predicts the number of bikes rented in an hour. It is binarized by splitting each feature into four categories.

peer-reviewed work that directly produces *sparse, provably-optimal regression trees* with publicly available code.

Our goal is to design optimal sparse regression trees in the classical sense, with a small number of leaves, a single condition at each split, and a constant prediction in each leaf. This makes the predictions easy to understand and compute, even for people who cannot understand equations. Given a trained tree, one can print it on an index card and compute a prediction without adding or multiplying any numbers, which makes these models easy to troubleshoot and use – even in high-stakes settings. An example tree for the **seoul bike** dataset (VE and Cho 2020; Sathishkumar, Park, and Cho 2020; Dua and Graff 2017) constructed by our method is shown in Figure 1.

Our formulation is a dynamic-programming-with-bounds approach, where the search space is either reduced or searched methodically. Such approaches have been highly successful for classification trees (Angelino et al. 2017; Lin et al. 2020; Nijssen, Schaus et al. 2020) but have not been previously used for regression trees. An important novel element of our formulation is a lower bound that we call the "k-Means equivalent points lower bound." To reduce the search space, we need as tight a bound as possible on the objective. Our bound makes use of the observation that any high-quality decision tree of $C$ leaves will perform as bad or worse than the performance of fully-optimal C-Means clustering on the labels alone (without any features). We discuss this in Section 3.

Our main results are: (1) The first algorithm with publicly available code for optimal sparse regression trees in the classical sense, with a *proof of optimality*. We call this algorithm Optimal Sparse Regression Trees (OSRT). (2) A substantial speedup over evtree, owing to our analytical bounds that reduce the search space. Evtree globally optimizes models, but does not provide a proof of optimality as OSRT does.

## 2 Notation and Objective

We denote the training dataset $(\mathbf{X}, \mathbf{y})$ as $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i \in \{0,1\}^M$ is a binary feature vector and $y_i \in \mathbb{R}$ is a target variable. (Real-valued features in the raw dataset can be transformed into binary features in many different ways, e.g., splitting the domain of the feature into equal-sized buckets, splitting between every two realized values of the variable in the training set, using splits from a reference model as in McTavish et al. (2022); we use the first technique.)

We denote $\mathcal{L}(t, \mathbf{X}, \mathbf{y}) := \frac{1}{N}\sum_{i=1}^N (y_i - \hat{y}_i)^2$ as the loss of tree $t$ on the training dataset, where $\hat{y}_i$ is the prediction of $\mathbf{x}_i$ by tree $t$, i.e., we use mean squared error (MSE) as the loss function. We define the objective function of tree $t$, $R(t, \mathbf{X}, \mathbf{y})$ as a combination of **tree loss** and **penalty on complexity**:

$$\mathcal{L}(t, \mathbf{X}, \mathbf{y}) + \lambda \cdot \text{complexity}(t)$$

where the complexity penalty is $H_t$, the number of leaves in tree $t$:

$$R(t, \mathbf{X}, \mathbf{y}) := \mathcal{L}(t, \mathbf{X}, \mathbf{y}) + \lambda H_t. \quad (1)$$

Computationally, it is easier when a depth constraint is added:

$$\mathcal{L}(t, \mathbf{X}, \mathbf{y}) + \lambda \cdot \text{complexity}(t), \text{ s.t. depth}(t) \leq d. \quad (2)$$

Adding a depth constraint dramatically reduces the search space, but it can lead to suboptimal values of the objective if the depth constraint is smaller than the depth of the optimal solution. Unlike all previous approaches, our algorithm can find provably-optimal trees that globally minimize Equation (1) without a depth constraint.

## 3 Bounds

Following Hu, Rudin, and Seltzer (2019); Lin et al. (2020), we represent a tree as a set of leaves. Trees with identical leaves, regardless of different internal branching nodes, are considered equivalent. This representation allows us to save memory and avoid duplicate computation during tree construction.

Our algorithm, like that of Lin et al. (2020) for classification, is a dynamic-programming-with-bounds algorithm. This algorithm searches the whole space of trees systematically from smaller to larger trees. If the algorithm determines (through the use of bounds) that the current partial tree it is constructing can never be extended to form an optimal full tree, it stops exploring that part of the search space. Thus, the tighter the bounds, the more the algorithm reduces the search space and the more quickly it converges to the optimal solution. Thus, we present a series of tight bounds that reduce computation by reducing the search space.

We start with notation. A tree $t$ is represented as a set of $H_t$ distinct leaves: $t = \{l_1, l_2, \ldots, l_{H_t}\}$. It can also be written as:

$$t = (t_{\text{fix}}, \delta_{\text{fix}}, t_{\text{split}}, \delta_{\text{split}}, K, H_t)$$

where $t_{\text{fix}} = \{l_1, l_2, \ldots, l_K\}$ are a set of $K$ **fixed leaves** that are not allowed to be further split in this part of the search space, $\delta_{\text{fix}} = \{\hat{y}_{l_1}, \hat{y}_{l_2}, \ldots, \hat{y}_{l_K}\} \in \mathbb{R}^K$ are predicted targets for the fixed leaves, $t_{\text{split}} = \{l_{K+1}, l_{K+2}, \ldots, l_{H_t}\}$ are $H_t - K$ **splitting leaves** that can be further split in this part of the search space, and their predicted targets are $\delta_{\text{split}} = \{\hat{y}_{l_{K+1}}, \hat{y}_{l_{K+2}}, \ldots, \hat{y}_{H_t}\} \in \mathbb{R}^{H_t - K}$.

We generate new trees by splitting different subsets of splitting leaves in tree $t$. We define $t' = (t'_{\text{fix}}, \delta'_{\text{fix}}, t'_{\text{split}}, \delta'_{\text{split}}, K', H_{t'})$ as a **child tree** of $t$ *if and only if* $t'_{\text{fix}}$ is a superset of $t_{\text{fix}}$, and $t'_{\text{split}}$ is generated through splitting a subset of $t_{\text{split}}$. We denote $\sigma(t)$ as the set of all child trees of $t$.

The following bounds start out analogous to those of Lin et al. (2020) for classification and diverge entirely when we get to the new k-Means Lower Bound.

### 3.1 Lower Bounds

The loss of a tree has contributions from its two parts: fixed leaves and splitting leaves. Since the fixed leaves cannot be further split in this part of the search space, their contribution provides a lower bound for tree $t$ and all of its child trees. Define the objective lower bound of tree $t$ as

$$R(t, \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t,$$

where $\mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y})$ is the sum of losses for fixed leaves:

$$\mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \cdot \mathbf{1}_{\text{cap}(t_{\text{fix}}, \mathbf{x}_i)} \tag{3}$$

$\mathbf{1}_{\text{cap}(t_{\text{fix}}, \mathbf{x}_i)}$ is 1 when one of the leaves in $t_{\text{fix}}$ captures $\mathbf{x}_i$, 0 otherwise. ($t_{\text{fix}}$ captures $\mathbf{x}_i$ when $\mathbf{x}_i$ falls into one of the fixed leaves of $t$.) If splitting leaves have 0 loss, then the tree's loss is equal to the lower bound.

We denote the current best objective we have seen so far as $R^c$. If the objective lower bound of $t$ is worse than $R^c$, i.e., $\mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t > R^c$, then $t$ cannot be an optimal tree, nor can any of its children, and the search space can be pruned. To show this, we need the following bound, stating that the child trees of $t$ all obey the same lower bound from the fixed leaves. *Note that all proofs are in the appendix.*

**Theorem 3.1.** *(Hierarchical Objective Lower Bound). Any tree $t' = (t'_{\text{fix}}, \delta'_{\text{fix}}, t'_{\text{split}}, \delta'_{\text{split}}, K', H_{t'}) \in \sigma(t)$ in the child tree set of tree $t = (t_{\text{fix}}, \delta_{\text{fix}}, t_{\text{split}}, \delta_{\text{split}}, K, H_t)$ obeys:*

$$R(t', \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t.$$

That is, the objective lower bound of the parent tree holds for all its child trees. This bound ensures that we do not further explore child trees if the parent tree can be pruned via the lower bound.

The next bound removes all of a tree's child trees from the search space, even if the tree itself could not be eliminated by the previous bound.

**Theorem 3.2.** *(Objective Lower Bound with One-step Lookahead). Let $t = (t_{\text{fix}}, \delta_{\text{fix}}, t_{\text{split}}, \delta_{\text{split}}, K, H_t)$ be a tree with $H_t$ leaves. If $\mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t + \lambda > R^c$, even if its objective lower bound obeys $\mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t \leq R^c$, then for any child tree $t' \in \sigma(t)$, $R(t', \mathbf{X}, \mathbf{y}) > R^c$.*

That is, even if a parent tree cannot be pruned via its objective lower bound, if $\mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t + \lambda > R^c$, all of its child trees are sub-optimal and can be pruned (and never explored).

### 3.2 Equivalent Points

Before making the lower bound of the objective tighter, let us introduce equivalent points. We define **equivalent points** as samples with *identical features* but possibly different target values. It is impossible to partition these samples into different leaves in any tree; a leaf that captures a set of equivalent points that have different targets can **never** achieve zero loss. Our bound exploits this fact.

Let $u$ be a set of equivalent points where samples have exactly the same feature vector $\mathbf{x}$, such that $\forall j_1, j_2, ...j_{|u|} \in u$:

$$\mathbf{x}_{j_1} = \mathbf{x}_{j_2} = \cdots = \mathbf{x}_{j_{|u|}}.$$

We define the **equivalence loss** $\mathcal{E}_u$ as the sum of squares error for set $u$ when the estimate of the leaf is the best possible, namely the mean of targets for points in $u$. Define $\bar{y}_u = \frac{1}{|u|} \sum_{(\mathbf{x}_i, y_i) \in u} y_i$:

$$\mathcal{E}_u = \frac{1}{N} \sum_{(\mathbf{x}_i, y_i) \in u} (y_i - \bar{y}_u)^2. \tag{4}$$

**Theorem 3.3.** *(Equivalent Points Lower Bound). Let $t = (t_{\text{fix}}, \delta_{\text{fix}}, t_{\text{split}}, \delta_{\text{split}}, K, H_t)$ be a tree with $K$ fixed leaves and $H_t - K$ splitting leaves. For any child tree $t' = (t'_{\text{fix}}, \delta'_{\text{fix}}, t'_{\text{split}}, \delta'_{\text{split}}, K', H_{t'}) \in \sigma(t)$:*

$$R(t', \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t + \sum_{u \in U} \mathcal{E}_u \cdot \mathbf{1}_{cap(t_{\text{split}}, u)}, \tag{5}$$

*where $U$ is the set of equivalent points sets in training dataset $(\mathbf{X}, \mathbf{y})$ and $\mathbf{1}_{cap(t_{\text{split}}, u)}$ is 1 when $t_{\text{split}}$ captures set $u$, 0 otherwise.*

*Combining with the idea of Theorem 3.2, we have*:

$$R(t', \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t + \lambda + \sum_{u \in U} \mathcal{E}_u \cdot \mathbf{1}_{\text{cap}(t_{\text{split}}, u)}. \tag{6}$$

The bound we introduce next, one of the main novel elements of the paper, is much tighter than the Equivalent Points Lower Bound.

### 3.3 k-Means Lower Bound

Let us consider the points within each leaf of a regression tree. The smallest possible losses within a leaf are achieved when the label values within the leaf are all similar to each other. If we know we will construct a tree with $C$ leaves and we could rearrange the points into any of the leaves, how would we arrange them to minimize loss? The best loss we could possibly achieve would come from grouping points with the most similar targets together in the same leaf. This procedure is equivalent to computing an optimal clustering of the targets (in 1 dimension) that minimizes the sum of squared errors between each point and the position of its cluster center (the mean of the cluster). The solution to this clustering problem gives the lowest loss we can possibly achieve for any regression tree with $C$ leaves. We can use this as a lower bound on the loss for $t_{\text{split}}$ by setting $C$ equal to the $H_t - K$ number of unsplittable leaves. There exists a deterministic algorithm that takes linear time for computing

the optimal k-Means loss on one dimensional data, which takes advantage the fact that the number line is totally ordered (Song and Zhong 2020).

**Definition 3.1.** *(k-Means Problem for 1D targets) Given a set of $N'$ 1D points $\mathbf{y}'$ and a number of clusters $C$, the goal is to assign points into $C$ clusters so that the sum of squared Euclidean distances between each point and its cluster mean is minimized. Define k-Means$(C, \mathbf{y}')$ to be the optimal objective of the k-Means algorithm for clustering 1D points $\mathbf{y}'$ of size $N'$ into $C$ clusters ($C \geq 1$):*

$$\textit{k-Means}(C, \mathbf{y}') := \min_{z,A} \sum_{i=1}^{N'} (y'_i - z_{A(y'_i)})^2. \tag{7}$$

*$A(y'_i)$ is a function that specifies the cluster assignment of $y'_i$ among $c_1, c_2, \ldots, c_C$, and $z_c$ is the centroid of cluster $c$, which is the mean of the points assigned to that cluster.*

$$z_c := \frac{\sum_{A(y'_i)=c} y'_i}{\sum_{A(y'_i)=c} \mathbf{1}}. \tag{8}$$

We note here that for an assignment, $A$, of points to a tree's $C$ leaves, choosing the mean $z_c$ as the predicted label in each leaf $c$ yields the following for the k-Means objective, which is optimized over $z$ for a fixed $A$:

$$\textit{k-Means-obj}(C, \mathbf{y}', A) := \min_{z} \sum_{i=1}^{N'} (y'_i - z_{A(y'_i)})^2. \tag{9}$$

That is, minimizing the regression loss (sum of squares to the mean target in each leaf) also yields the k-Means' choice of cluster center as the mean of the targets for points belonging to a leaf. Clearly k-Means-obj$(C, \mathbf{y}', A) \geq$ k-Means$(C, \mathbf{y}')$ since the latter is minimized over the assignment of points to clusters without regard to the tree structure at all. This logic is used in our lower bound.

**Theorem 3.4.** *(k-Means Lower Bound). Consider tree $t = (t_{\text{fix}}, \delta_{\text{fix}}, t_{\text{split}}, \delta_{\text{split}}, K, H_t)$. and any child tree $t' = (t'_{\text{fix}}, \delta'_{\text{fix}}, t'_{\text{split}}, \delta'_{\text{split}}, K', H_{t'}) \in \sigma(t)$. Let $(\mathbf{X}_{t_{\text{split}}}, \mathbf{y}_{t_{\text{split}}})$ be samples captured by the splitting leaves $t_{\text{split}}$. Then,*

$$\begin{aligned} R(t', \mathbf{X}, \mathbf{y}) \geq & \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda K \\ & + \min_{C} \left( \frac{1}{N} \textit{k-Means}(C, \mathbf{y}_{t_{\text{split}}}) + \lambda C \right). \end{aligned}$$

## 3.4 k-Means Equivalent Points Lower Bound

We can make the bound from the last section even tighter. In fact, in the k-Means lower bound above, we ignored information inherent to the regression tree problem, because we ignored all of the features $\mathbf{X}$. We can achieve a tighter bound if we leverage our knowledge of $\mathbf{X}$ to again consider equivalent points. Specifically, all points with the same features must be assigned to the same leaf. We first present the definition of a modified k-Means problem and then state our theorem.

**Definition 3.2.** *(Constrained k-Means Problem for 1D targets) Given a set of $N'$ 1D target points $\mathbf{y}'$ with feature vector $\mathbf{X}'$ and number of clusters $C$, the goal is to assign points into $C$ clusters so that the sum of squared Euclidean distances between each point and its cluster mean is minimized, under the constraint that all points with the same feature vector $\mathbf{x}'$ must be assigned to one cluster.*

$$\begin{aligned} & \textit{Constrained\_k-Means}(C, \mathbf{X}', \mathbf{y}') \\ = \quad & \min_{z,A} \sum_{i=1}^{N'} (y'_i - z_{A(y'_i)})^2 \\ \textit{s.t.} \quad & \textit{if } \mathbf{x}_i = \mathbf{x}_{i'}, \textit{ then } A(y'_i) = A(y'_{i'}). \end{aligned} \tag{10}$$

Adding this constraint makes the k-Means Lower Bound tighter.

**Theorem 3.5.** *(k-Means Equivalent Points Lower Bound). Consider tree $t = (t_{\text{fix}}, \delta_{\text{fix}}, t_{\text{split}}, \delta_{\text{split}}, K, H_t)$. and any child tree $t' = (t'_{\text{fix}}, \delta'_{\text{fix}}, t'_{\text{split}}, \delta'_{\text{split}}, K', H_{t'}) \in \sigma(t)$. Let $(\mathbf{X}_{t_{\text{split}}}, \mathbf{y}_{t_{\text{split}}})$ be samples captured by the splitting leaves $t_{\text{split}}$. Then,*

$$\begin{aligned} & R(t', \mathbf{X}, y) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda K + \\ & \min_C \left( \tfrac{1}{N} \textit{Constrained\_k-Means}(C, \mathbf{X}_{\text{split}}, \mathbf{y}_{t_{\text{split}}}) + \lambda C \right) \end{aligned} \tag{11}$$

*where Constrained_k-Means is defined in Equation 10.*

## 3.5 Computing k-Means Equivalent Points Bound

We now define a weighted version of the k-Means problem, where each sample point is associated with a weight. We derive these weights later as sizes of the equivalent sets.

**Definition 3.3.** *(Weighted k-Means Problem) Given a set of $N'$ 1D points $\mathbf{y}'$ with weights $\mathbf{w} \in \mathbb{R}^{N'}$ and number of clusters $C$, the goal is to assign points into $C$ clusters so that the weighted sum of squared Euclidean distances between each point and its cluster centroid is minimized. Define Weighted_k-Means$(C, \mathbf{y}', \mathbf{w})$ as the optimal objective of the k-Means algorithm clustering 1D points $\mathbf{y}'$ of size $N'$ into $C$ clusters ($C \geq 1$):*

$$\begin{aligned} & \textit{Weighted\_k-Means}(C, \mathbf{y}', \mathbf{w}) \\ & = \min_{z,A} \sum_{i=1}^{N'} w_i \cdot (y'_i - z_{A(y'_i)})^2. \end{aligned} \tag{12}$$

*$A(y'_i)$ is a function that specifies the cluster assignment of $y'_i$ among $c_1, c_2, \ldots, c_C$, and $z_c$ is the centroid of cluster $c$, which is the weighted mean of the points assigned to that cluster. The weighted mean for cluster $c_j$ is:*

$$z_{c_j} = \frac{\sum_{A(y'_i)=c_j} w_i \cdot y'_i}{\sum_{A(y'_i)=c_j} w_i} \tag{13}$$

*which is similar to the one defined by Song and Zhong (2020).*

Song and Zhong (2020) present an efficient $O(kN)$ solution to this weighted k-Means problem, where $k$ is the number of clusters and $N$ the number of data samples. We leverage this algorithm for the k-Means Equivalent Points

Lower Bound (Theorem 3.5). In the following theorem, we show that solving this weighted k-Means problem is equivalent to solving a constrained k-Means problem on a modified dataset.

**Theorem 3.6.** *(Constrained k-Means with Equivalent Points is equivalent to weighted k-Means) Recall in Definition 3.2, we have $N'$ 1D target points $\mathbf{y}'$ with features $\mathbf{X}'$and number of clusters $C$. We also have a constraint that all points in any equivalent set $u$ must be assigned to the same leaf. Define a modified dataset $(\mathbf{X}_{mod}, \mathbf{y}_{mod}, \mathbf{w}_{mod})$, where all points of equivalent set $u$ in the original dataset $(\mathbf{X}', \mathbf{y}')$ are represented by a single point $(\mathbf{x}_u, y_u, w_u)$, where $\mathbf{x}_u$ is the same as the feature vector of equivalent set $u$,*

$$y_u = \frac{1}{|u|} \sum_{(\mathbf{x}'_i, y'_i) \in u} y'_i, \tag{14}$$

*and the weight is the size of the equivalent set $u$*

$$w_u = |u|. \tag{15}$$

*An optimal clustering of the modified dataset will directly provide an optimal clustering of the original dataset with the equivalent points constraint from Equation 10. (All points from the original dataset contributing to a weighted point in the modified dataset will be assigned to its cluster.)*

That is, solving the Weighted k-Means problem produces the same solution(s) as solving the Constrained k-Means problem. Thus, solving the weighted k-Means problem on the modified dataset provides the same result as solving the constrained k-Means on the original dataset.

In Equation 11, we observe that computing the k-Means Equivalent Points Lower bound requires that we find the minimum of Constrained_k-Means across all possible $C$. One can easily see that it is sufficient to iterate $C$ from 1 to $|\mathbf{y}_{t_{\text{split}}}|$, where every data point is in its own cluster. However, this would be costly when dealing with large datasets. The following theorem, as proved in Aggarwal, Schieber, and Tokuyama (1994), shows that the loss improved from adding more clusters decreases as the number of clusters increases. It means we do not need to generate k-Means solutions for all $C$ up to the size of the subproblem, *we can stop as soon as the objective improvement from adding new clusters becomes less than the regularization $\lambda$.*

**Theorem 3.7.** *(Convexity of Weighted_k-Means in number of clusters, from Aggarwal, Schieber, and Tokuyama 1994) Recall Weighted_k-Means$(C, \mathbf{y}', \mathbf{w})$ from Definition 3.3 for number of clusters $C$, 1D points $\mathbf{y}'$, and weights $\mathbf{w}$. We have*

$$\begin{aligned} & \textit{Weighted\_k-Means}(C-1, \mathbf{y}', \mathbf{w}) \\ + \quad & \textit{Weighted\_k-Means}(C+1, \mathbf{y}', \mathbf{w}) \\ \geq \quad & 2 \times \textit{Weighted\_k-Means}(C, \mathbf{y}', \mathbf{w}). \end{aligned} \tag{16}$$

Other bounds that help reduce the search space (e.g. Leaf Bounds, Splitting Bounds, Permutation Bound, Subset Bound) can be found in Appendix B.

## 4 Algorithm

We implemented OSRT based on the GOSDT (Lin et al. 2020) framework, which uses a *dynamic-programming-with-bounds* formulation. Each *subproblem* in this formulation is identified by a support set $s = \{s_1, s_2, \ldots, s_N\}$, where $s_i$ is a boolean value indicating whether point $i$ is in the support set $s$. Each leaf and branching node corresponds to a subproblem, recording which samples traverse through that node (or leaf). GOSDT records and updates lower and upper bounds of the objective for each subproblem and stores them in a *dependency graph*. The dependency graph summarizes the relationship among subproblems. In dynamic programming formulations, finding tight bounds is crucial in reducing the runtime of the algorithm, because that is the key to eliminating large portions of search space. Our k-Means-based bounds for regression are tight and substantially reduce time-to-optimality, as we show in Section 6.4 and Appendix J.1. Like GOSDT, our method finds the optimal trees when the lower and upper bounds of the objective converge. Algorithm 1 below is a subroutine of OSRT.

**Compute_Lower_Bound (Algorithm 1)**: This algorithm implements the k-Means Equivalent Points Lower Bound as defined in Theorem 3.5. We leveraged a k-Means solver from Song and Zhong (2020), which is a dynamic programming formulation that fills in a $C$ by $N$ matrix, where $C$ represents the number of clusters and $N$ corresponds to the number of samples. We do not assume a maximum value for $C$ and instead grow the table one row at a time, using the *fill_kmeans_dp* function from their implementation. Each point $(a, b)$ in the table represents the optimal k-Means loss using $a$ clusters and the first $b$ datapoints.

**Line 1-3**: Compute equivalent target set by grouping equivalent points together, and gather all of their labels. **Lines 4-5**: Compute weight **w** and value **v** that defines the k-Means problem. **Lines 6-8**: Initialize current loss, *loss*, previous loss, *loss'*, number of clusters used, *nClusters*, and dynamic programming table, *dp_table*. **Lines 9-17**: Solve weighted k-Means problem by adding clusters one at a time. **Line 11**: Retrieve loss using *nClusters* clusters from the last entry of the last filled row of dynamic programming table. **Lines 12-14**: Terminate algorithm if we can no longer benefit from adding more clusters as the reduction of loss by adding one cluster is monotonically decreasing. See Theorem 3.7. **Line 18**: Compute constant correction term, *correction*, that restores weighted k-Means to constrained k-Means problem (see Theorem 3.6).

## 5 Comparison of Regression Tree Optimization Methods

*Unlike other methods, OSRT can optimize regression trees without a hard depth constraint and support mean absolute error (L1 loss).* Table 1 summarizes the comparison of different regression tree optimization methods. Blue cells are comparative advantages, and red cells are comparative disadvantages.

## 6 Experiments

We ran experiments on 12 datasets; the details are described in Appendix C.1. Our evaluation answers the following:

1. Are trees generated by existing regression tree optimization methods truly optimal? How well do optimal sparse regression trees generalize? How far from optimal are greedy-approach models? (§6.1)

| | OSRT | IAI | Evtree | GUIDE | CART | ORT | DTIP |
|---|---|---|---|---|---|---|---|
| Guarantee optimality | Yes | No | No | No | No | Yes | Yes |
| Optimization strategy | DPB | Local Search | Evolutionary | Greedy search | Greedy Search | MIO | MIO |
| Can optimize without depth constraint | Yes | No | No | Yes | Yes | No | No |
| Support (weighted) least absolute deviation | Yes | No | No | No | Yes | Unknown | Unknown |
| Implementation available | Yes | Yes (Executable Only) | Yes | Yes (Executable Only) | Yes | No | No |

Table 1: Comparison of OSRT, IAI (Interpretable AI 2022), Evtree (Grubinger, Zeileis, and Pfeiffer 2014), GUIDE (Loh 2002), CART (Breiman et al. 1984), ORT (Dunn 2018) and DTIP (Verwer and Zhang 2017). Executables for IAI and GUIDE are available, but their source code is not. DPB is dynamic programming with bounds, MIO is mixed integer optimization.

Algorithm 1: compute_lower_bound($dataset, sub, \lambda$) $\rightarrow$ *lower_bound*

*// For a subproblem sub and regularization $\lambda$, compute its Equivalent k-Means Lower Bound*

Let $U$ = the set of unique samples $\mathbf{x}_i \in$ *sub*
*// For each unique sample in $U$, create a set of all targets $y$*
*// corresponding to copies of that sample in sub*
*// (equivalent point sets)*
$E = \emptyset$
$\forall \mathbf{x}_i \in U, E_{\mathbf{x}_i}$.*append*$(\{y_j\} \mid \forall \mathbf{x}_j \in$ *sub*$, y_j$ *if* $\mathbf{x}_i = \mathbf{x}_j)$
*// For each unique sample in $U$, compute the number of*
*// identical samples to it (producing the vector $\mathbf{w}$) and the*
*// average of all targets (producing the vector $\boldsymbol{v}$)*
$\mathbf{w} \leftarrow \{|E_{\mathbf{x}}| \mid E_{\mathbf{x}} \in E\}$ where $E_{\mathbf{x}} \subset \mathbf{y}$ is a set of targets for one unique $\mathbf{x} \in U$.
$\mathbf{v} \leftarrow \{\overline{E_{\mathbf{x}}} \mid E_{\mathbf{x}} \in E\}$ where $\overline{E_{\mathbf{x}}}$ denotes *average* of $E_{\mathbf{x}}$
*loss*, *loss*$' \leftarrow$ inf
*nClusters* $\leftarrow 1$
*// We initialize the dynamic programming table with*
*// no rows, but one column for each element of U*
*dp_table* $\leftarrow [][|E|]$
**while** *true* **do**
  *// Fill in the $(nClusters-1)^{th}$ row of dp_table*
  *dp_table* $\leftarrow$ *fill_kmeans_dp*$(nClusters - 1, w, v,$ *dp_table*$)$
  *loss* $\leftarrow$ *dp_table*$[(nClusters - 1, |E| - 1)]$
  **if** *loss*$' -$ *loss* $\leq \lambda$ **then**
  *// Adding this cluster does not reduce loss enough*
  *// to justify addition of another cluster*
    **break**
  **end**
  *nClusters* $\leftarrow$ *nClusters* $+ 1$
  *loss*$' \leftarrow$ *loss*
**end**
*// Correct from weighted k-Means to constrained k-Means*
*correction* $\leftarrow \sum_{\mathbf{x}_j \in sub} y_j^2 - \sum_{\mathbf{x}_i \in U} w_i v_i^2$
**return** *loss*$' + \lambda \times$ *nClusters* $+$ *correction*

2. Does each method yield consistently high-quality results? (§6.2)
3. How fast does OSRT converge, given that it guarantees optimality? (§6.3)
4. How much do our novel bounds contribute to the performance of OSRT? (§6.4)
5. What do optimal regression trees look like? (§6.5)

## 6.1 Optimality and Generalization

We compare trees produced by CART (Breiman et al. 1984), GUIDE (Loh 2002), IAI (Interpretable AI 2022), Evtree (Grubinger, Zeileis, and Pfeiffer 2014) and OSRT, trained on various datasets. For each method, we swept a range of hyperparameters to illustrate the relationship between loss and sparsity (IAI, Evtree, and OSRT all penalize the number of leaves). Optimization experiments in Appendix D and cross-validation experiments in Appendix H, along with a demonstration of these results in Figure 2 show: (1) trees produced by other methods are usually *sub-optimal* even if they claim optimality (they do not *prove* optimality), and *only our method can consistently find the optimal trees*, which are the most efficient frontiers that optimize the trade-off between loss and sparsity, (2) OSRT has the best generalization performance among methods, and (3) *we can now quantify how far from optimal other methods are*.

## 6.2 Controllability

Unlike IAI and Evtree, our method does not rely on random seeds. *The results returned by OSRT are consistently high quality, while those of IAI and Evtree are not.* Figure 3 shows the stochasticity of various methods. Trees produced by IAI and Evtree have large variance in complexity and accuracy if we do not fix the random seed. High variance of loss and sparsity can result in inaccuracy and overfitting. Details and results of this experiment can be found in Appendix F.

## 6.3 Speed and Scalability

Our method is one of the *fastest* regression tree optimization methods and the *only one* that also guarantees optimality. Figure 4 shows that OSRT performs well in run time, and Figure 5 shows its outstanding scalability when tackling a large dataset with over 2 million samples. As the number of sample increases, Evtree slows down more than other methods and cannot converge within a 30-minute time limit when

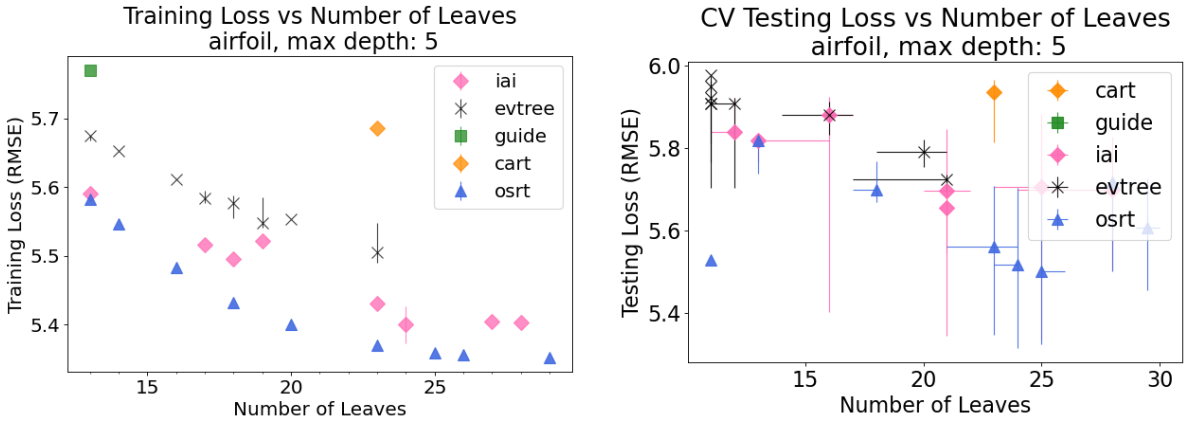



Figure 2: Training and testing loss achieved by IAI, Evtree, GUIDE, CART, OSRT on dataset *airfoil*, $d = 5$.

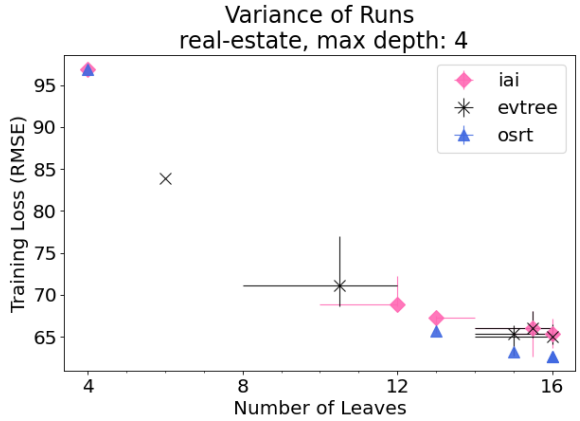



Figure 3: Variance (horizontal and vertical lines) of trees generated by IAI, Evtree, OSRT using 10 different random seeds on dataset *real-estate*.

the sample size exceeds 50,000. More results are shown in Appendices G and I.

### 6.4 Value of k-Means Lower Bound

The squared error used in regression tasks tends to make the equivalent points lower bound loose, preventing us from pruning more of the search space. The novel k-Means lower bound allows us to *aggressively prune the search space*, and Figure 6 shows that for the airfoil data set, the k-Means lower bound converged in less than one-fourth the time it took the equivalent points bound to converge. More results can be found in Appendix J.1.

### 6.5 Optimal Trees

Figure 7 presents two optimal trees generated by OSRT on dataset *servo*, with and without a depth constraint respectively, using the same regularization parameter. It shows that imposing a depth constraint sacrifices the global optimality of Equation 1. More results regarding the ablation study of depth limit can be found in Appendix J.2, and Appendix L

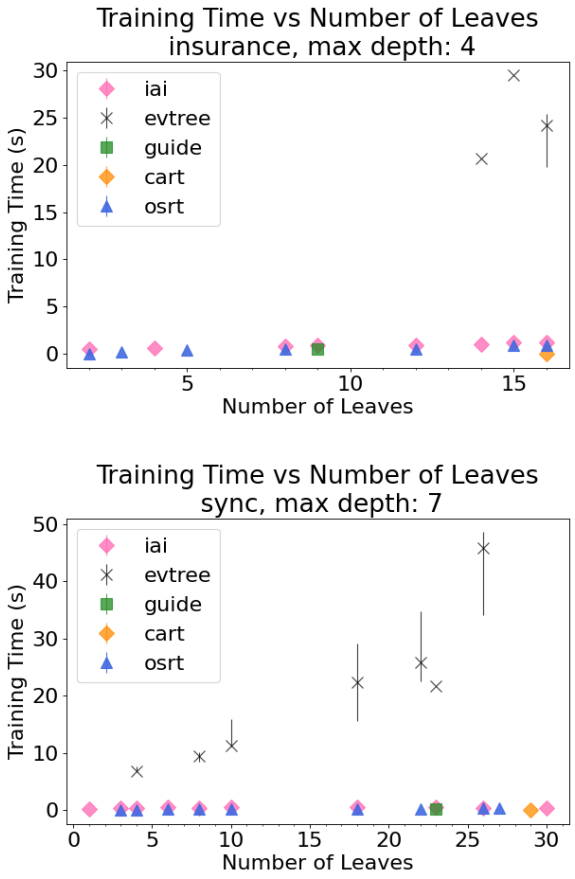



Figure 4: Training time of trees generated by CART, GUIDE, IAI, Evtree, OSRT.

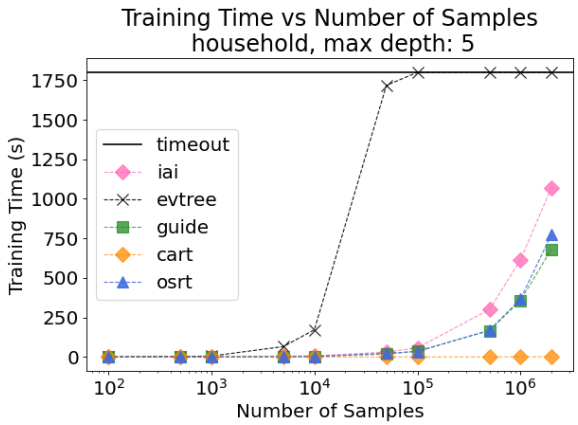



Figure 5: Training time of CART, GUIDE, IAI, Evtree and OSRT as a function of sample size on dataset *household*, $d = 5, \lambda = 0.035$. (30-minutes time limit; Evtree timed out when sample size is beyond 50,000)

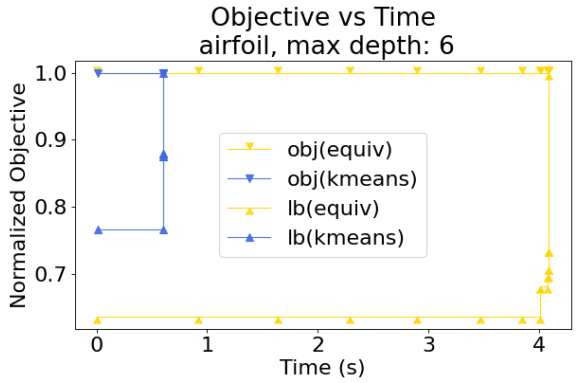



Figure 6: The time saved by k-Means lower bound (blue) over equivalent points bound (yellow), using $\lambda = 0.005$. The optimal solution is found when the lower bound equals objective. The k-Means bound converges in under a second.

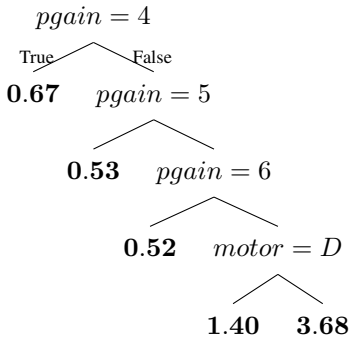
compares optimal trees generated by OSRT and sub-optimal trees generated by other methods.

## 7 Conclusion

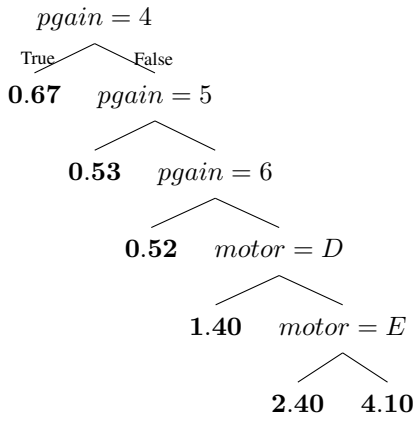
We provide the first method to find provably-optimal regression trees within a reasonable time. Our method quickly and consistently finds an optimal sparse model that tends to generalize well. Our method also scales well even for large datasets. OSRT provides a naturally human-interpretable option for solving regression problems in contrast to other, uninterpretable methods such as ridge regression, support vector regression, ensemble methods and neural networks.

## Code Availability

The implementation of OSRT is available at https://github.com/ruizhang1996/optimal-sparse-regression-tree-public. Our experiment code is available at https://github.com/ruizhang1996/regression-tree-benchmark.




(a) (Max depth 4) Optimal tree with 5 leaves, $R^2 = 69.63\%$.




(b) (No depth limit) Optimal tree with 6 leaves, $R^2 = 75\%$.

Figure 7: Optimal trees generated by OSRT on dataset *servo* with (a) depth limit 4 and (b) no depth limit. Tree (b) has only one more leaf but explains 5% more training data variance than Tree (a).

## Acknowledgments

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), the National Institute on Drug Abuse (NIDA) under grant DA054994, and the National Science Foundation (NSF) under grant IIS-2130250.

## References

Aggarwal, A.; Schieber, B.; and Tokuyama, T. 1994. Finding a minimum-weightk-link path in graphs with the concave monge property and applications. *Discrete & Computational Geometry*, 12(3): 263–280.

Aghaei, S.; Gomez, A.; and Vayanos, P. 2020. Learning Optimal Classification Trees: Strong Max-Flow Formulations. *arXiv e-print arXiv:2002.09142*.

Angelino, E.; Larus-Stone, N.; Alabi, D.; Seltzer, M.; and Rudin, C. 2017. Learning certifiably optimal rule lists for categorical data. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

Avellaneda, F. 2020. Efficient inference of optimal decision trees. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, 3195–3202.

Bennett, K. P.; and Blue, J. A. 1996. Optimal decision trees. *Rensselaer Polytechnic Institute Math Report*, 214: 24.

Bertsimas, D.; Dunn, J.; and Wang, Y. 2021. Near-optimal Nonlinear Regression Trees. *Operations Research Letters*, 49(2): 201–206.

Blanquero, R.; Carrizosa, E.; Molero-Río, C.; and Morales, D. R. 2020. Sparsity in optimal randomized classification trees. *European Journal of Operational Research*, 284(1): 255–272.

Blanquero, R.; Carrizosa, E.; Molero-Río, C.; and Morales, D. R. 2022. On sparse optimal regression trees. *European Journal of Operational Research*, 299(3): 1045–1054.

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

Demirović, E.; Lukina, A.; Hebrard, E.; Chan, J.; Bailey, J.; Leckie, C.; Ramamohanarao, K.; and Stuckey, P. J. 2022. MurTree: Optimal Decision Trees via Dynamic Programming and Search. *Journal of Machine Learning Research*, 23(26): 1–47.

Dobkin, D.; Fulton, T.; Gunopulos, D.; Kasif, S.; and Salzberg, S. 1997. Induction of shallow decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Dua, D.; and Graff, C. 2017. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml. Accessed: 2022-04-01.

Dunn, J. 2018. *Optimal Trees for Prediction and Prescription*. Ph.D. thesis, Massachusetts Institute of Technology.

Farhangfar, A.; Greiner, R.; and Zinkevich, M. 2008. A Fast Way to Produce Optimal Fixed-Depth Decision Trees. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*.

Grubinger, T.; Zeileis, A.; and Pfeiffer, K.-P. 2014. evtree: Evolutionary learning of globally optimal classification and regression trees in R. *Journal of Statistical Software*, 61: 1–29.

Günlük, O.; Kalagnanam, J.; Li, M.; Menickelly, M.; and Scheinberg, K. 2021. Optimal decision trees for categorical data via integer programming. *Journal of Global Optimization*, 1–28.

Hu, H.; Siala, M.; Hebrard, E.; and Huguet, M.-J. 2020. Learning optimal decision trees with maxsat and its integration in adaboost. In *IJCAI-PRICAI 2020, 29th International Joint Conference on Artificial Intelligence and the 17th Pacific Rim International Conference on Artificial Intelligence*.

Hu, X.; Rudin, C.; and Seltzer, M. 2019. Optimal Sparse Decision Trees. In *Proceedings of Conference on Neural Information Processing Systems (NeurIPS)*.

Interpretable AI, L. 2022. Interpretable AI Documentation. https://www.interpretable.ai. Accessed: 2022-04-01.

Janota, M.; and Morgado, A. 2020. Sat-based encodings for optimal decision trees with explicit paths. In *International Conference on Theory and Applications of Satisfiability Testing*, 501–518. Springer.

Lin, J.; Zhong, C.; Hu, D.; Rudin, C.; and Seltzer, M. 2020. Generalized and scalable optimal sparse decision trees. In *Proceedings of International Conference on Machine Learning (ICML)*, 6150–6160.

Loh, W.-Y. 2002. Regression tress with unbiased variable selection and interaction detection. *Statistica Sinica*, 361–386.

McTavish, H.; Zhong, C.; Achermann, R.; Karimalis, I.; Chen, J.; Rudin, C.; and Seltzer, M. 2022. Fast Sparse Decision Tree Optimization via Reference Ensembles. In *Proceedings of AAAI Conference on Artificial Intelligence*.

Morgan, J. N.; and Sonquist, J. A. 1963. Problems in the analysis of survey data, and a proposal. *J. Amer. Statist. Assoc.*, 58: 415–434.

Narodytska, N.; Ignatiev, A.; Pereira, F.; and Marques-Silva, J. 2018. Learning Optimal Decision Trees with SAT. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI)*, 1362–1368.

Nijssen, S.; and Fromont, E. 2007. Mining optimal decision trees from itemset lattices. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 530–539. ACM.

Nijssen, S.; and Fromont, E. 2010. Optimal constraint-based decision tree induction from itemset lattices. *Data Mining and Knowledge Discovery*, 21(1): 9–51.

Nijssen, S.; Schaus, P.; et al. 2020. Learning Optimal Decision Trees Using Caching Branch-and-Bound Search. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

Payne, H. J.; and Meisel, W. S. 1977. An algorithm for constructing optimal binary decision trees. *IEEE Transactions on Computers*, C-26(9): 905–916.

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

Sathishkumar, V.; Park, J.; and Cho, Y. 2020. Using data mining techniques for bike sharing demand prediction in metropolitan city. *Computer Communications*, 153: 353–366.

Shati, P.; Cohen, E.; and McIlraith, S. 2021. SAT-based approach for learning optimal decision trees with non-binary features. In *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

Song, M.; and Zhong, H. 2020. Efficient weighted univariate clustering maps outstanding dysregulated genomic zones in human cancers. *Bioinformatics*, 36(20): 5027–5036.

VE, S.; and Cho, Y. 2020. A rule-based model for Seoul Bike sharing demand prediction using weather data. *European Journal of Remote Sensing*, 53(sup1): 166–183.

Verhaeghe, H.; Nijssen, S.; Pesant, G.; Quimper, C.-G.; and Schaus, P. 2019. Learning optimal decision trees using constraint programming. In *The 25th International Conference on Principles and Practice of Constraint Programming (CP2019)*.

Verwer, S.; and Zhang, Y. 2017. Learning decision trees with flexible constraints and objectives using integer optimization. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 94–103. Springer.

Verwer, S.; and Zhang, Y. 2019. Learning optimal classification trees using a binary linear program formulation. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.