

Optimal Sparse Regression Trees

Rui Zhang^{1*}, Rui Xin^{1*}, Margo Seltzer², Cynthia Rudin¹

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 ℓ_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.

Copyright © 2023, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

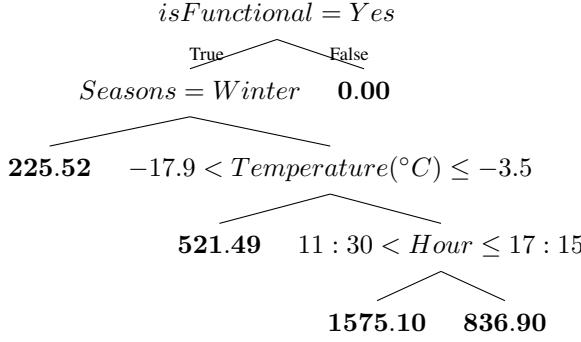


Figure 1: Optimal regression tree for *seoul bike* dataset with $\lambda = 0.05$, $\text{max depth} = 5$. This dataset predicts the number of bikes rented in an hour. It is binarized by splitting each feature into four categories.

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:

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

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: $t = \{l_1, l_2, \dots, l_{H_t}\}$. It can also be written as:

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

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

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'_{fix} is a superset of t_{fix} , and t'_{split} is generated through splitting a subset of t_{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:

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

$\mathbf{1}_{\text{cap}(t_{\text{fix}}, \mathbf{x}_i)}$ is 1 when one of the leaves in t_{fix} captures \mathbf{x}_i , 0 otherwise. (t_{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 $\forall j_1, j_2, \dots, j_{|u|} \in u$:

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

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 $\bar{y}_u = \frac{1}{|u|} \sum_{(\mathbf{x}_i, y_i) \in u} y_i$:

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

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 $t' = (t'_{\text{fix}}, \delta'_{\text{fix}}, t'_{\text{split}}, \delta'_{\text{split}}, K', H_{t'}) \in \sigma(t)$:

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

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

Combining with the idea of Theorem 3.2, we have:

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

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_{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 $k\text{-Means}(C, \mathbf{y}')$ to be the optimal objective of the k-Means algorithm for clustering 1D points \mathbf{y}' of size N' into C clusters ($C \geq 1$):

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

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

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

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 :

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

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 $k\text{-Means-obj}(C, \mathbf{y}', A) \geq k\text{-Means}(C, \mathbf{y}')$ 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_{split} . Then,

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

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.

Constrained k-Means($C, \mathbf{X}', \mathbf{y}'$)

$$= \min_{z, A} \sum_{i=1}^{N'} (y'_i - z_{A(y'_i)})^2 \quad (10)$$

s.t. if $\mathbf{x}_i = \mathbf{x}_{i'}$, then $A(y'_i) = A(y'_{i'})$.

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_{split} . Then,

$$R(t', \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda K + \min_C \left(\frac{1}{N} \text{Constrained k-Means}(C, \mathbf{X}_{t_{\text{split}}}, \mathbf{y}_{t_{\text{split}}}) + \lambda C \right) \quad (11)$$

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 $\text{Weighted k-Means}(C, \mathbf{y}', \mathbf{w})$ as the optimal objective of the k-Means algorithm clustering 1D points \mathbf{y}' of size N' into C clusters ($C \geq 1$):

$$\text{Weighted k-Means}(C, \mathbf{y}', \mathbf{w}) = \min_{z, A} \sum_{i=1}^{N'} w_i \cdot (y'_i - z_{A(y'_i)})^2. \quad (12)$$

$A(y'_i)$ is a function that specifies the cluster assignment of y'_i among c_1, c_2, \dots, 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:

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

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 $(\mathbf{X}_{\text{mod}}, \mathbf{y}_{\text{mod}}, \mathbf{w}_{\text{mod}})$, 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 ,*

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

and the weight is the size of the equivalent set u

$$w_u = |u|. \quad (15)$$

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 λ .

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*

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

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 $s = \{s_1, s_2, \dots, s_N\}$,

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 \mathbf{w} and value \mathbf{v} that defines the k-Means problem. **Lines 6-8:** Initialize current loss, loss , previous loss, loss' , number of clusters used, $n\text{Clusters}$, 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 $n\text{Clusters}$ 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, λ)`
 $\rightarrow \text{lower_bound}$
// For a subproblem sub and regularization λ , compute its Equivalent k-Means Lower Bound

1: Let U = the set of unique samples $\mathbf{x}_i \in \text{sub}$
// For each unique sample in U , create a set of all targets y corresponding to copies of that sample in sub
// (equivalent point sets)
2: $E = \emptyset$
3: $\forall \mathbf{x}_i \in U, E_{\mathbf{x}_i}.\text{append}(\{y_j\} \mid \forall \mathbf{x}_j \in \text{sub}, y_j \text{ if } \mathbf{x}_i = \mathbf{x}_j)$
// For each unique sample in U , compute the number of identical samples to it (producing the vector w) and the average of all targets (producing the vector v)
4: $w \leftarrow \{|E_{\mathbf{x}}| \mid E_{\mathbf{x}} \in E\}$ where $E_{\mathbf{x}} \subset y$ is a set of targets for one unique $\mathbf{x} \in U$.
5: $v \leftarrow \{\overline{E_{\mathbf{x}}} \mid E_{\mathbf{x}} \in E\}$ where $\overline{E_{\mathbf{x}}}$ denotes average of $E_{\mathbf{x}}$
6: $loss, loss' \leftarrow \text{inf}$
7: $nClusters \leftarrow 1$
// We initialize the dynamic programming table with no rows, but one column for each element of U
8: $dp_table \leftarrow \emptyset[|E|]$
9: **while** true **do**
 // Fill in the $(nClusters - 1)^{\text{th}}$ row of dp_table
10: $dp_table \leftarrow \text{fill_kmeans_dp}(nClusters - 1, w, v, dp_table)$
11: $loss \leftarrow dp_table[(nClusters - 1, |E| - 1)]$
12: **if** $loss' - loss \leq \lambda$ **then**
 // Adding this cluster does not reduce loss enough to justify addition of another cluster
13: **break**
14: **end**
15: $nClusters \leftarrow nClusters + 1$
16: $loss' \leftarrow loss$
17: **end**
// Correct from weighted k-Means to constrained k-Means
18: $correction \leftarrow \sum_{\mathbf{x}_j \in \text{sub}} y_j^2 - \sum_{\mathbf{x}_i \in U} w_i v_i^2$
19: **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

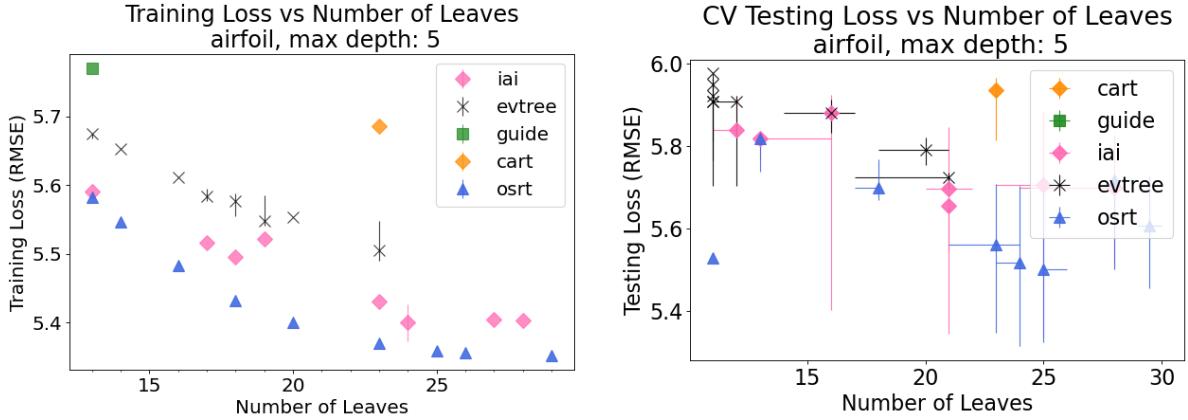


Figure 2: Training and testing loss achieved by IAI, Evtree, GUIDE, CART, OSRT on dataset *airfoil*, $d = 5$.

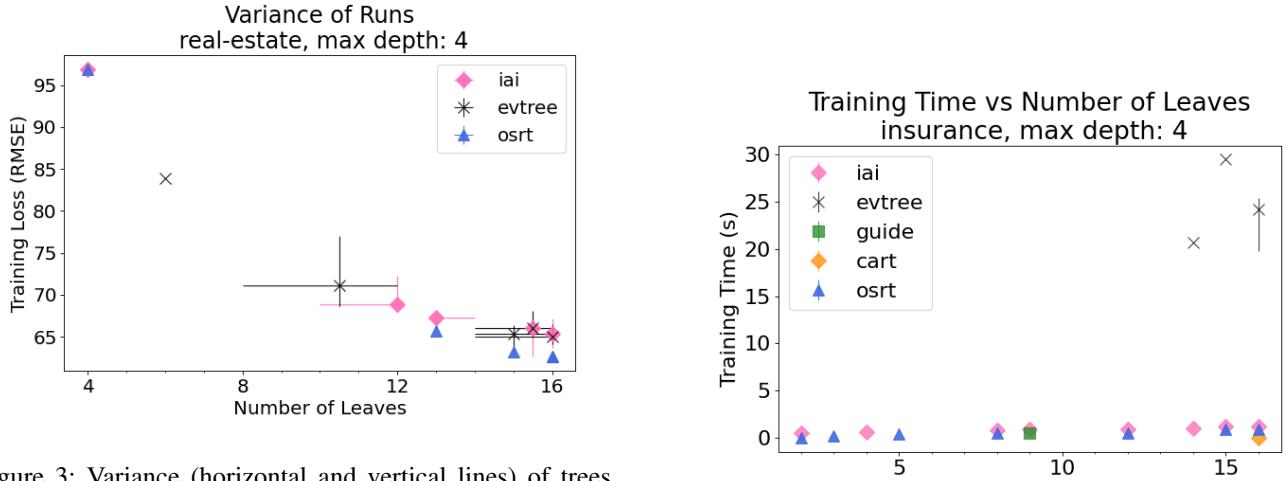


Figure 3: Variance (horizontal and vertical lines) of trees generated by IAI, Evtree, OSRT using 10 different random seeds on dataset *real-estate*.

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

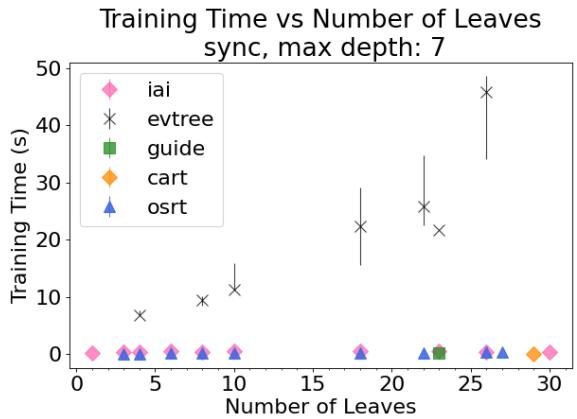


Figure 4: Training time of trees generated by CART, GUIDE, IAI, Evtree, OSRT.

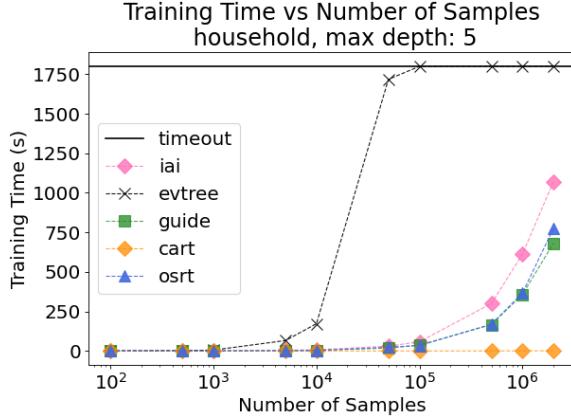


Figure 5: Training time of CART, GUIDE, IAI, Evtree and OSRT as a function of sample size on dataset *household*, $d = 5$, $\lambda = 0.035$. (30-minutes time limit; Evtree timed out when sample size is beyond 50,000)

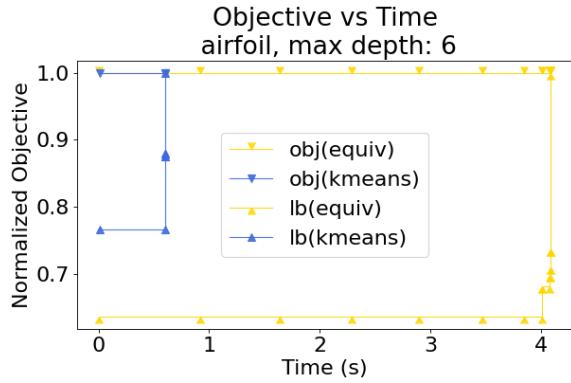


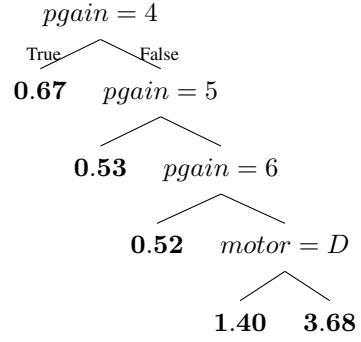
Figure 6: The time saved by k-Means lower bound (blue) over equivalent points bound (yellow), using $\lambda = 0.005$. The optimal solution is found when the lower bound equals objective. The k-Means bound converges in under a second. compares optimal trees generated by OSRT and sub-optimal trees generated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

7 Conclusion

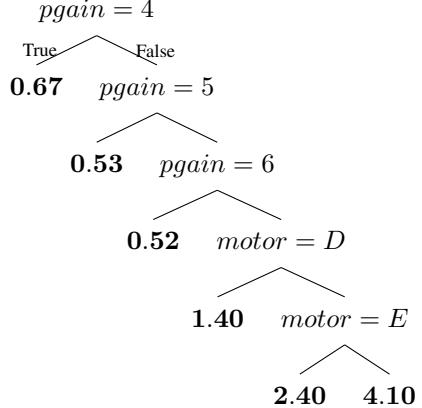
We provide the first method to find provably-optimal regression trees within a reasonable time. Our method quickly and consistently finds an optimal sparse model that tends to generalize well. Our method also scales well even for large datasets. OSRT provides a naturally human-interpretable option for 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-weight k-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.
- Chambers, J.; Cleveland, W.; Kleiner, B.; and Tukey, P. 1983. Graphical methods for data analysis. Wadsworth Int'l. Group, Belmont, CA.
- Choi, M. 2018. Kaggle insurance data. <https://www.kaggle.com/datasets/mirichoi0218/insurance>.
- 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 trees 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 (CP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

ence 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)*.

A Theorems and Proofs

A.1 Proof of Theorem 3.1

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

Proof. As we know, $K' \geq K, H_{t'} > H_t$, since t' is a child tree of t . The objective lower bound (which holds for all trees) of t' is:

$$R(t', \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t'_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_{t'}. \quad (17)$$

Since $\mathcal{L}(t'_{\text{fix}}, \mathbf{X}, \mathbf{y}) = \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{fix}} \setminus t_{\text{fix}}, \mathbf{X}, \mathbf{y})$ and the loss of $K' - K$ fixed leaves in t is nonnegative, i.e., $\mathcal{L}(t'_{\text{fix}} \setminus t_{\text{fix}}, \mathbf{X}, \mathbf{y}) \geq 0$, we have:

$$\mathcal{L}(t'_{\text{fix}}, \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) \quad (18)$$

therefore:

$$R(t', \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t. \quad (19)$$

□

A.2 Proof of Theorem 3.2

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 $\mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t \leq R^c$, 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).

Proof. From the objective lower bound of t' defined in Equation 17, and Equation 18, we have:

$$R(t', \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_{t'}. \quad (20)$$

Because the child tree has at least one more leaf than the parent tree, $H_{t'} \geq H_t + 1$, we have:

$$R(t', \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t + \lambda. \quad (21)$$

Thus, if $\mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t + \lambda > R^c$, then $R(t', \mathbf{X}, \mathbf{y}) > R^c$ for child trees t' , which means all the child trees can be pruned. □

A.3 Proof of Theorem 3.3

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 $t' = (t'_{\text{fix}}, \delta'_{\text{fix}}, t'_{\text{split}}, \delta'_{\text{split}}, K', H_{t'}) \in \sigma(t)$:

$$R(t', \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t + \sum_{u=1}^U \mathcal{E}_u \cdot \mathbf{1}_{\text{cap}(t_{\text{split}}, u)}, \quad (22)$$

where $\mathbf{1}_{\text{cap}(t_{\text{split}}, u)}$ is 1 when t_{split} captures set u , 0 otherwise. Combining with the idea of Theorem 3.2, we have:

$$R(t', \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda H_t + \lambda + \sum_{u=1}^U \mathcal{E}_u \cdot \mathbf{1}_{\text{cap}(t_{\text{split}}, u)}. \quad (23)$$

Proof.

$$\begin{aligned} R(t', \mathbf{X}, \mathbf{y}) &= \mathcal{L}(t', \mathbf{X}, \mathbf{y}) + \lambda H_{t'} \\ &= \mathcal{L}(t'_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{split}}, \mathbf{X}, \mathbf{y}) + \lambda H_{t'} \\ &= \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{fix}} \setminus t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{split}}, \mathbf{X}, \mathbf{y}) + \lambda H_{t'}. \end{aligned} \quad (24)$$

Since samples captured by t_{split} are captured either by $t'_{\text{fix}} \setminus t_{\text{fix}}$ or t'_{split} , and equivalence loss cannot be eliminated in any tree, we have:

$$\mathcal{L}(t'_{\text{fix}} \setminus t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{split}}, \mathbf{X}, \mathbf{y}) \geq \sum_{u=1}^U \mathcal{E}_u \cdot \mathbf{1}_{\text{cap}(t_{\text{split}}, u)}. \quad (25)$$

Equality is achieved when each leaf in $(t'_{\text{fix}} \setminus t_{\text{fix}}) \cup t'_{\text{split}}$ captures exactly one set of equivalent points and no other points. Substituting Equation 25 into Equation 24, we have:

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

Because $H_{t'} \geq H_t + 1$:

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

□

A.4 Proof of Theorem 3.4

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_{split} . Then,

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

where $\text{k-Means}(C, \mathbf{y}')$ is the optimal objective of the k-Means algorithm clustering 1D points \mathbf{y}' of size N' into C clusters ($C \geq 1$).

Proof. From Equation 24 we know that for any child tree t' :

$$R(t', \mathbf{X}, \mathbf{y}) = \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{fix}} \setminus t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{split}}, \mathbf{X}, \mathbf{y}) + \lambda H_{t'}.$$

Rearranging:

$$R(t', \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda K + \mathcal{L}(t'_{\text{fix}} \setminus t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{split}}, \mathbf{X}, \mathbf{y}) + \lambda(H_{t'} - K). \quad (26)$$

Here, $\mathcal{L}(t'_{\text{fix}} \setminus t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{split}}, \mathbf{X}, \mathbf{y})$ can be viewed as the squared loss of one way to assign samples $(\mathbf{X}_{t_{\text{split}}}, \mathbf{y}_{t_{\text{split}}})$ to $(H_{t'} - K)$ clusters, and the objective of this assignment is the sum of squared Euclidean distances between every point in $(\mathbf{X}_{t_{\text{split}}}, \mathbf{y}_{t_{\text{split}}})$ and its cluster mean, because in regression trees, we predict using the mean of the targets in each leaf.

We now follow the logic above the statement of the theorem, where the optimal k-Means assignment (which considers labels only) can achieve a better objective than any other assignment of points to leaves (like that of our current tree). By definition, we have:

$$\mathcal{L}(t'_{\text{fix}} \setminus t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{split}}, \mathbf{X}, \mathbf{y}) \geq \frac{1}{N} \text{k-Means}(H_{t'} - K, \mathbf{y}_{t_{\text{split}}}). \quad (27)$$

Adding $\lambda(H_{t'} - K)$ to each side, we have

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

Because

$$\begin{aligned} &\frac{1}{N} \text{k-Means}(H_{t'} - K, \mathbf{y}_{t_{\text{split}}}) + \lambda(H_{t'} - K) \\ &\geq \min_C \left(\frac{1}{N} \text{k-Means}(C, \mathbf{y}_{t_{\text{split}}}) + \lambda C \right), \end{aligned}$$

we have:

$$\begin{aligned} &\mathcal{L}(t'_{\text{fix}} \setminus t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{split}}, \mathbf{X}, \mathbf{y}) + \lambda(H_{t'} - K) \\ &\geq \min_C \left(\frac{1}{N} \text{k-Means}(C, \mathbf{y}_{t_{\text{split}}}) + \lambda C \right). \end{aligned} \quad (28)$$

Substituting Equation 28 into Equation 26, we have:

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

□

A.5 Proof of Theorem 3.5

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_{split} . Then,

$$R(t', \mathbf{X}, \mathbf{y}) \geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda K + \min_C \left(\frac{1}{N} \text{Constrained_k-Means}(C, \mathbf{X}_{\text{split}}, \mathbf{y}_{t_{\text{split}}}) + \lambda C \right)$$

where Constrained_k-Means is defined in Equation 10.

Proof. The proof is very similar to the k-Means lower bound (Theorem 3.4). From Equation 26, we have:

$$\begin{aligned} R(t', \mathbf{X}, \mathbf{y}) &\geq \mathcal{L}(t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \lambda K \\ &\quad + \mathcal{L}(t'_{\text{fix}} \setminus t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{split}}, \mathbf{X}, \mathbf{y}) \\ &\quad + \lambda(H_{t'} - K). \end{aligned}$$

View $\mathcal{L}(t'_{\text{fix}} \setminus t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{split}}, \mathbf{X}, \mathbf{y})$ as the objective of one way to assign samples $(\mathbf{X}_{t_{\text{split}}}, \mathbf{y}_{t_{\text{split}}})$ into $(H_{t'} - K)$ clusters, under the constraint that *equivalent points must be assigned to the same cluster*. This gives:

$$\begin{aligned} &\mathcal{L}(t'_{\text{fix}} \setminus t_{\text{fix}}, \mathbf{X}, \mathbf{y}) + \mathcal{L}(t'_{\text{split}}, \mathbf{X}, \mathbf{y}) \\ &\geq \frac{1}{N} \text{Constrained_k-Means}(H_{t'} - K, \mathbf{X}_{t_{\text{split}}}, \mathbf{y}_{t_{\text{split}}}). \end{aligned} \quad (29)$$

The rest would be the same as the proof for Theorem 3.4, with k-Means($H_{t'} - K, \mathbf{y}_{t_{\text{split}}}$) replaced by Constrained_k-Means($H_{t'} - K, \mathbf{X}_{t_{\text{split}}}, \mathbf{y}_{t_{\text{split}}}$). \square

A.6 Proof of Theorem 3.6

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 feature \mathbf{X}' and number of clusters C . We also have a constraint that for all points in any equivalent set u , they must be assigned to the same leaf. Define a modified dataset $(\mathbf{X}_{\text{mod}}, \mathbf{y}_{\text{mod}}, \mathbf{w}_{\text{mod}})$, where all points of equivalent set u in the original dataset $(\mathbf{X}', \mathbf{y}')$ are represented as a single point (\mathbf{x}_u, y_u, w_u) , where \mathbf{x}_u is the same as \mathbf{x}'_u the feature vector of equivalent set u ,

$$y_u = \frac{1}{|u|} \sum_{(\mathbf{x}'_i, y'_i) \in u} y'_i, \quad (30)$$

and the weight is the size of the equivalent set u

$$w_u = |u|, \quad (31)$$

then an optimal clustering of the modified dataset will 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 the same cluster.) That is, solving the Weighted k-Means problem would result in the same solution(s) as solving the Constrained k-Means problem.

Proof. Let U be the set of equivalent sets in the original dataset $(\mathbf{X}', \mathbf{y}')$, namely our modified dataset has $|U|$ points, (\mathbf{x}_u, y_u, w_u) where $u = 1, 2, \dots, |U|$. By the definition of the optimal clustering of the modified dataset, we have

$$\operatorname{argmin}_A \sum_{u \in U} w_u (y_u - z_{A(y_u)})^2$$

rewritten as:

$$\begin{aligned} &\operatorname{argmin}_A \sum_{k=1}^C \sum_{A(y_u)=c_k} w_u (y_u - z_{c_k})^2 \\ &= \operatorname{argmin}_A \sum_{k=1}^C \sum_{A(y_u)=c_k} w_u y_u^2 - 2w_u y_u z_{c_k} + w_u z_{c_k}^2 \\ &= \operatorname{argmin}_A \sum_{u \in U} w_u y_u^2 + \sum_{k=1}^C \sum_{A(y_u)=c_k} -2w_u y_u z_{c_k} + w_u z_{c_k}^2. \end{aligned} \quad (32)$$

$\sum_{u \in U} w_u y_u^2$ can be removed because it is a constant, and according to Equation 13, we have

$$\sum_{A(y_u)=c_k} w_u y_u = z_{c_k} \sum_{A(y_u)=c_k} w_u.$$

For cluster c_k , we also have

$$\sum_{A(y_u)=c_k} w_u z_{c_k}^2 = z_{c_k}^2 \sum_{A(y_u)=c_k} w_u.$$

Substituting them into the equation above, we continue:

$$\text{Equation 32} = \operatorname{argmin}_A \sum_{k=1}^C -z_{c_k}^2 \sum_{A(y_u)=c_k} w_u.$$

Substituting Equation 13 into it:

$$= \operatorname{argmin}_A \sum_{k=1}^C - \left(\frac{\sum_{A(y_u)=c_k} w_u y_u}{\sum_{A(y_u)=c_k} w_u} \right)^2 \sum_{A(y_u)=c_k} w_u. \quad (33)$$

Recall that w_u is the size of the equivalent set u in the original dataset $(\mathbf{X}', \mathbf{y}')$. We can switch the index of summation back to point i instead of equivalent points sets u ,

$$z_{c_k} = \frac{\sum_{A(y_u)=c_k} w_u y_u}{\sum_{A(y_u)=c_k} w_u} = \frac{\sum_{A(y'_i)=c_k} y'_i}{\sum_{A(y'_i)=c_k} \mathbf{1}}.$$

Then, we continue:

$$\begin{aligned} \text{Equation 33} &= \operatorname{argmin}_A \sum_{k=1}^C - \left(\frac{\sum_{A(y'_i)=c_k} y'_i}{\sum_{A(y'_i)=c_k} \mathbf{1}} \right)^2 \sum_{A(y'_i)=c_k} \mathbf{1} \\ &= \operatorname{argmin}_A \sum_{k=1}^C -z_{c_k}^2 \sum_{A(y'_i)=c_k} \mathbf{1} \\ &= \operatorname{argmin}_A \sum_{k=1}^C \left(-2z_{c_k}^2 \sum_{A(y'_i)=c_k} \mathbf{1} + z_{c_k}^2 \sum_{A(y'_i)=c_k} \mathbf{1} \right) \\ &= \operatorname{argmin}_A \sum_{k=1}^C \sum_{A(y'_i)=c_k} -2z_{c_k} y'_i + z_{c_k}^2. \end{aligned}$$

Adding a constant $\sum_{i=1}^{N'} y'_i^2$ that does not affect the argmin , we have:

$$\begin{aligned} &= \operatorname{argmin}_A \sum_{i=1}^{N'} y'_i^2 + \sum_{k=1}^C \sum_{A(y'_i)=c_k} -2z_{c_k} y'_i + z_{c_k}^2 \\ &= \operatorname{argmin}_A \sum_{k=1}^C \sum_{A(y'_i)=c_k} y'_i^2 - 2z_{c_k} y'_i + z_{c_k}^2 \\ &= \operatorname{argmin}_A (y'_i - z_{A(c_k)})^2. \end{aligned}$$

which is equivalent to the k-Means problem on the original dataset under the constraint that the equivalent points must be in the same cluster. \square

Thus, solving the weighted k-Means problem on the modified dataset provides the same results as solving the constrained k-Means on the original dataset.

A.7 Proof of Theorem 3.7

Theorem 3.7 (Convexity of Weighted k -Means Objective in Number of Clusters) Recall $\text{Weighted_k-Means}(C, \mathbf{y}', \mathbf{w})$ from Definition 3.3 for number of clusters C , 1D points \mathbf{y}' , and weights \mathbf{w} . Then, we have

$$\text{Weighted_k-Means}(C - 1, \mathbf{y}', \mathbf{w}) + \text{Weighted_k-Means}(C + 1, \mathbf{y}', \mathbf{w}) \geq 2 \times \text{Weighted_k-Means}(C, \mathbf{y}', \mathbf{w}). \quad (34)$$

Proof.

$$\begin{aligned} & \text{Weighted_k-Means}(C - 1, \mathbf{y}', \mathbf{w}) - \text{Weighted_k-Means}(C, \mathbf{y}', \mathbf{w}) \\ & \geq \text{Weighted_k-Means}(C, \mathbf{y}', \mathbf{w}) - \text{Weighted_k-Means}(C + 1, \mathbf{y}', \mathbf{w}). \end{aligned} \quad (35)$$

Equation 35 is proved in Aggarwal, Schieber, and Tokuyama (1994). They show in Application V that a $\text{Weighted_k-Means}(C, \mathbf{y}', \mathbf{w})$ problem can get reduced to a “minimum C -link path in a concave Monge DAG with $|\mathbf{y}'| + 1$ nodes”. Equation 35 is adapted directly from Corollary 7, rearranging it we have Equation 34. \square

B Theorems directly adapted from GOSDT

Theorem B.1. (Hierarchical Objective Lower Bound for Sub-trees). Let R^c be the current best objective so far. Let t be a tree such that the root node is split by a feature, where two sub-trees t_{left}, t_{right} are generated with H_{left} leaves for t_{left} and H_{right} leaves for t_{right} . The data captured by the left tree is $(\mathbf{X}_{left}, \mathbf{y}_{left})$ and the data captured by the right tree is $(\mathbf{X}_{right}, \mathbf{y}_{right})$. Then, the objective lower bounds of the left sub-tree and right sub-tree are $b(t_{left}, \mathbf{X}_{left}, \mathbf{y}_{left})$ and $b(t_{right}, \mathbf{X}_{right}, \mathbf{y}_{right})$, which obey $R(t_{left}, \mathbf{X}_{left}, \mathbf{y}_{left}) \geq b(t_{left}, \mathbf{X}_{left}, \mathbf{y}_{left})$, and $R(t_{right}, \mathbf{X}_{right}, \mathbf{y}_{right}) \geq b(t_{right}, \mathbf{X}_{right}, \mathbf{y}_{right})$. If $b(t_{left}, \mathbf{X}_{left}, \mathbf{y}_{left}) > R^c$ or $b(t_{right}, \mathbf{X}_{right}, \mathbf{y}_{right}) > R^c$ or $b(t_{left}, \mathbf{X}_{left}, \mathbf{y}_{left}) + b(t_{right}, \mathbf{X}_{right}, \mathbf{y}_{right}) > R^c$, then t is not an optimal tree, and none of its child trees are optimal.

Proof. This bound adapts directly from GOSDT (Lin et al. 2020), where the proof can be found. \square

This bound can be applied to any tree, even if the tree is partially constructed. In a partially constructed tree t , if one of its subtrees has objective worse than current best objective R^c , we can prune tree t and all of its child trees without constructing the other subtree.

Leaf Bounds

The following upper bounds on the number of leaves permit us to prune trees whose leaves exceed these upper bounds.

Theorem B.2. (Upper Bound on the Number of Leaves). Let H_t be the number of leaves of tree t and let R^c be the current best objective. For any optimal tree t^* with H_{t^*} leaves, it is true that:

$$H_{t^*} \leq \min\{\lfloor R^c/\lambda \rfloor, 2^M\}, \quad (36)$$

where M is the number of features.

Proof. This bound adapts directly from OSDT (Hu, Rudin, and Seltzer 2019), where the proof can be found. \square

Theorem B.3. (Parent-specific upper bound on the number of leaves). Let $t = (t_{fix}, \delta_{fix}, t_{split}, \delta_{split}, K, H_t)$ be a tree with child tree $t' = (t'_{fix}, \delta'_{fix}, t'_{split}, \delta'_{split}, K', H_{t'}) \in \sigma(t)$ with $H_{t'}$ leaves is a possibly optimal tree. Then:

$$H_{t'} \leq \min \left\{ H_t + \left\lfloor \frac{R_c - \mathcal{L}(t_{fix}, \mathbf{X}, \mathbf{y}) - \lambda H_t}{\lambda} \right\rfloor, 2^M \right\}. \quad (37)$$

Proof. This bound adapts directly from OSDT (Hu, Rudin, and Seltzer 2019), where the proof can be found. \square

Splitting Bounds

When constructing a new child tree $t' = (t'_{fix}, \delta'_{fix}, t'_{split}, \delta'_{split}, K', H_{t'})$, t'_{split} needs to be determined. Splitting bounds help determine which leaves in t' cannot be further split and which leaves must be further split.

Theorem B.4. (Incremental Progress Bound to Determine Splitting). For any optimal tree t^* , any parent node of its leaves must have loss at least $\geq \lambda$ when considered as a leaf.

Proof. Let $t^* = \{l_1, l_2, \dots, l_i, l_{i+1}, \dots, l_{H_{t^*}}\}$ be an optimal tree with H_{t^*} leaves. $t' = \{l_1, l_2, \dots, l_{i-1}, l_{i+2}, \dots, l_{H_{t^*}, l_j}\}$ is a tree created by deleting a pair of leaves l_i and l_{i+1} in t^* and adding their parent node l_j .

$$\begin{aligned} R(t', \mathbf{X}, \mathbf{y}) - R(t^*, \mathbf{X}, \mathbf{y}) &= \mathcal{L}(l_j, \mathbf{X}, \mathbf{y}) + \lambda(H_{t^*} - 1) - \mathcal{L}(l_i, \mathbf{X}, \mathbf{y}) - \mathcal{L}(l_{i+1}, \mathbf{X}, \mathbf{y}) - \lambda H_{t^*} \\ &\leq \mathcal{L}(l_j, \mathbf{X}, \mathbf{y}) - \lambda \end{aligned}$$

where $\mathcal{L}(l, \mathbf{X}, \mathbf{y}) \geq 0$ is loss of leaf l . And because t^* is an optimal tree, $R(t', \mathbf{X}, \mathbf{y}) - R(t^*, \mathbf{X}, \mathbf{y}) \geq 0$, and we have:

$$\mathcal{L}(l_j, \mathbf{X}, \mathbf{y}) - \lambda \geq 0.$$

□

Theorem B.5. (*Lower Bound on Incremental Progress*). Consider any optimal tree $t^* = \{l_1, l_2, \dots, l_i, l_{i+1}, \dots, l_{H_{t^*}}\}$ with H_{t^*} leaves. Let $t' = \{l_1, l_2, \dots, l_{i-1}, l_{i+2}, \dots, l_{H_{t^*}, l_j}\}$ be a tree created by deleting a pair of leaves l_i and l_{i+1} in t^* and adding their parent node l_j . The reduction in loss obeys:

$$\mathcal{L}(l_j, \mathbf{X}, \mathbf{y}) - \mathcal{L}(l_i, \mathbf{X}, \mathbf{y}) - \mathcal{L}(l_{i+1}, \mathbf{X}, \mathbf{y}) \geq \lambda.$$

Proof.

$$\begin{aligned} R(t', \mathbf{X}, \mathbf{y}) - R(t^*, \mathbf{X}, \mathbf{y}) &= \mathcal{L}(l_j, \mathbf{X}, \mathbf{y}) + \lambda(H_{t^*} - 1) - \mathcal{L}(l_i, \mathbf{X}, \mathbf{y}) - \mathcal{L}(l_{i+1}, \mathbf{X}, \mathbf{y}) - \lambda H_{t^*} \\ &= \mathcal{L}(l_j, \mathbf{X}, \mathbf{y}) - \mathcal{L}(l_i, \mathbf{X}, \mathbf{y}) - \mathcal{L}(l_{i+1}, \mathbf{X}, \mathbf{y}) - \lambda. \end{aligned}$$

Since t^* is an optimal tree, we have $R(t', \mathbf{X}, \mathbf{y}) - R(t^*, \mathbf{X}, \mathbf{y}) \geq 0$, and thus:

$$\mathcal{L}(l_j, \mathbf{X}, \mathbf{y}) - \mathcal{L}(l_i, \mathbf{X}, \mathbf{y}) - \mathcal{L}(l_{i+1}, \mathbf{X}, \mathbf{y}) \geq \lambda.$$

□

When constructing new trees, if a leaf has loss less than λ (it fails to meet Theorem B.4), then it cannot be further split. If a pair of leaves in that tree reduce loss from their parent node by less than λ (they fail to meet Theorem B.5), then at least one of this pair of leaves must be further split to search for optimal trees.

Permutation Bound

Theorem B.6. (*Leaf Permutation Bound*). Let π be any permutation of $\{1 \dots H_t\}$. Let $t = \{l_1, l_2, \dots, l_{H_t}\}$, $T = \{l_{\pi(1)}, l_{\pi(2)}, \dots, l_{\pi(H_t)}\}$, that is, the leaves in T are a permutation of the leaves in t . The objective lower bounds of t and T are the same and their child trees correspond to permutations of each other.

Proof. This bound adapts directly from OSDT (Hu, Rudin, and Seltzer 2019), where the proof can be found. □

This bound avoids duplicate computation of trees with leaf permutation.

Subset Bound

Theorem B.7. Let t and T to be two trees with the same root node, where t uses feature f_1 to split the root node and T uses feature f_2 to split the root node. Let t_1, t_2 be subtrees of t under its root node, and $(\mathbf{X}_{t_1}, \mathbf{y}_{t_1}), (\mathbf{X}_{t_2}, \mathbf{y}_{t_2})$ be samples captured by t_1 and t_2 . Similarly, let T_1, T_2 be subtrees of T under its root node, and $(\mathbf{X}_{T_1}, \mathbf{y}_{T_1}), (\mathbf{X}_{T_2}, \mathbf{y}_{T_2})$ be samples captured by T_1 and T_2 . Suppose t_1, t_2 are optimal trees for $(\mathbf{X}_{t_1}, \mathbf{y}_{t_1}), (\mathbf{X}_{t_2}, \mathbf{y}_{t_2})$ respectively, and T_1, T_2 are optimal trees for $(\mathbf{X}_{T_1}, \mathbf{y}_{T_1}), (\mathbf{X}_{T_2}, \mathbf{y}_{T_2})$ respectively. If $R(t_1, \mathbf{X}_{t_1}, \mathbf{y}_{t_1}) \leq R(T_1, \mathbf{X}_{T_1}, \mathbf{y}_{T_1})$ and $(\mathbf{X}_{t_2}, \mathbf{y}_{t_2}) \subset (\mathbf{X}_{T_2}, \mathbf{y}_{T_2})$, then $R(t, \mathbf{X}, \mathbf{y}) \leq R(T, \mathbf{X}, \mathbf{y})$.

Proof. This bound adapts directly from GOSDT (Lin et al. 2020), where the proof can be found. □

Similar to Theorem B.1, this bound ensures that we can safely prune a partially constructed tree without harming optimality. It checks whether subtree t_1 has a better objective than T_1 , despite handling more data.

C Experiment Details

In the following subsections, we provide details on the data sets, pre-processing, and experimental setup used in §6.

C.1 Datasets

We use twelve regression datasets. Ten of them are from the UCI Machine Learning Repository (Dua and Graff 2017), including **Airfoil Self-Noise**, **Auction Verification**, **Optical Interconnection Network**, **Real Estate Valuation**, **Seoul Bike Sharing Demand**, **Servo**, **Synchronous Machine**, **Yacht Hydrodynamics**, **Energy efficiency**, and **Individual Household Electric Power Consumption**. **Air quality** comes from Chambers et al. (1983) and **Medical Cost Personal** is from Choi (2018). We predict the scaled sound pressure level for Airfoil Self-Noise, the runtime of verification procedure for the Auction Verification dataset, the channel utilization for the Optical Interconnection Network dataset, the house price of unit area for the Real Estate Valuation dataset, the rented bike count for the Seoul Bike Sharing Demand dataset, the rise time of a servomechanism for Servo, the excitation current of a synchronous machine for the Synchronous Machine dataset, the residuary resistance per unit weight of displacement for Yacht Hydrodynamics, and the mean ozone in parts per billion from 1300 to 1500 hours at Roosevelt Island for Air Quality, the individual medical costs billed by health insurance for Medical Cost Personal, the heating load and cooling load for Energy Efficiency, and the global active power for Individual Household Electric Power Consumption.

C.2 Preprocessing

First, we removed all observations with missing values. Second, since we performed hundreds of experiments, each of which required substantial computation time and each needed to be solved to provable optimality, in all cases below where we transformed a continuous feature into binary features, we discretized the feature into equal-width partitions and used one-hot encoding. We preprocessed datasets as follows:

Airfoil Self-Noise (airfoil): We discretized each of the features *frequency*, *angle of attack*, *suction side displacement thickness* into 4 categories.

Air Quality (airquality): We discretized each of features *solar R*, *temp*, *wind*, *day* into 4 categories.

Auction Verification (auction): We discretized each continuous variable by using a binary feature to encode a threshold between each pair of adjacent values; the threshold is set equal to the average of the two surrounding values. The classification label is treated as a categorical feature.

Optical Interconnection Network (optical): We discretized each of *processor utilization*, *channel waiting time*, *input waiting time*, *network response time* into 4 categories.

Real Estate Valuation (real-estate): We discretized each continuous feature into 4 categories.

Seoul Bike Sharing Demand (seoul-bike): We discretized each continuous feature into 4 categories.

Servo (servo): We directly use this dataset that only contains categorical features.

Synchronous Machine (sync): We discretized each feature into 4 categories.

Yacht Hydrodynamics (yacht): We discretized each of *Beam-draught ratio*, *Froude number* into 4 categories.

Medical Cost Personal (insurance): We discretized each of *age*, *bmi* into 4 categories.

Energy efficiency (enb-heat, enb-cool): We discretized each of *X1 Relative Compactness*, *X2 Surface Area* into 4 categories. **enb-heat** predicts heating load and **enb-cool** predicts cooling load.

Individual Household Electric Power Consumption (household): We transformed the *Date* feature into *Month*, *Time* into *Hour*. Then we discretized each of *Month*, *Hour*, *Global_reactive power*, *Voltage*, *Global_intensity* into 4 categories.

Table 2 summarizes all datasets after preprocessing.

Dataset	Samples	Orig. Features	Encoded Binary Features	Prediction Target
airfoil	1503	5	17	scaled sound pressure level
airquality	111	6	17	ozone
auction	2043	8	48	verification time
optical	640	9	29	channel utilization
real-estate	414	6	18	house price of unit area
seoul-bike	8760	12	32	rented bike count
servo	167	4	15	class
sync	557	4	12	“If” (current excitation of synchronous machine)
yacht	308	6	35	residuary resistance per unit weight of displacement
insurance	1338	6	16	charges
enb-heat	768	8	27	Y1 heating load
enb-cool	768	8	27	Y2 cooling load
household	2,049,280	5	15	global active power

Table 2: Datasets Summary

C.3 Experimental Platform

We ran all experiments on a TensorEX TS2-673917-DPN Intel Xeon Gold 6226 Processor, 2.7Ghz (768GB RAM 48 cores). We set a 5-minute time limit for objective optimality (Section D) and cross-validation experiments (Section H) and 30-minute

time limit for running time and scalability experiments(Section G, I, J.1 and K). The memory limit is 200GB. All algorithms ran single-threaded.

C.4 Software Packages

InterpretableAI (IAI):

We used OptimalTreeRegressor in version 3.0.1 of IAI (<https://docs.interpretable.ai/stable/>). We were given a free license for this software. Source code was not available.

EvolutionaryTree (Evtree):

We used the CRAN package of evtree, version 1.0-8 (<https://cran.r-project.org/web/packages/evtree/index.html>).

Classification and Regression Tree (CART):

The Python implementation from Sci-Kit Learn 1.1.1.

Generalized,Unbiased,Interaction Detection and Estimation (GUIDE):

We used the executable file(version 40.2) from (<https://pages.stat.wisc.edu/~loh/guide.html>). Source code was not available.

D Experiment: Loss vs. Sparsity

Collection and Setup: We ran this experiment on 8 datasets: *airfoil*, *airquality*, *real-estate*, *seoul-bike*, *servo*, *sync*, *yacht*, *insurance*. We trained models on the entire dataset to measure time to convergence/optimality. All runs that exceeded the time limit of 5 minutes were discarded (for depths of 7-9, time-outs occurred for OSRT and evtree, whereas for depth 6 or less, most runs are within the 5-minute time limits, please see Section E for run time statistics). For each dataset, we ran algorithms with different configurations:

- **CART:** We ran this algorithm with 8 different configurations: depth limit, d , ranging from 2 to 9, and a corresponding maximum leaf limit 2^d . All other parameters were set to the default.
- **GUIDE:** We ran this algorithm with 8 different configurations: depth limit, d , ranging from 2 to 9, and a corresponding maximum leaf limit 2^d . The minimum leaf node size was set to 2. All other parameters were set to the default.
- **IAI:** We ran this algorithm with 8×20 different configurations: depth limits ranging from 2 to 9, and 20 different regularization coefficients (0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.003, … 0.009, 0.01, 0.035, 0.055, 0.08, 0.1, 0.105, 0.2, 0.5). The random seed was 1. All other parameters were set to the default.
- **Evtree:** We ran this algorithm with 8×20 different configurations: depth limits ranging from 2 to 9, and 20 different regularization coefficients (0.1, 0.2, 0.3, …, 0.9, 1, 1.1, 1.2 … 2). The minimum leaf node size was set to 1, and the minimum internal node size was set to 2. The random seed was set to 666. All other parameters were set to the default.
- **OSRT (our method):** We ran this algorithm with 8×20 different configurations: depth limits ranging from 2 to 9, and 20 different regularization coefficients (0.0001, 0.0002, 0.0005, 0.001, 0.002, 0.003, … 0.009, 0.01, 0.035, 0.055, 0.08, 0.1, 0.105, 0.2, 0.5).

Note: OSRT and IAI have the exact same objective function while Evtree has a slightly different regularization term: the regularization coefficient depends on the number of samples, N . Therefore we used a different scale of coefficients for Evtree.

Calculations: We drew one plot (training loss vs. number of leaves) for each combination of dataset and depth limit. Under the same depth limit, runs of one algorithm with different regularization coefficients may generate trees with the same number of leaves. In this case, we plotted the median loss of those trees and showed the best and worst loss among these trees as lower and upper error values respectively. These plots do not display trees where the number of leaves exceeds 30, as these tend to be uninterpretable and overfitted.

Results: Figures 8-15 show that OSRT consistently produces optimal trees that minimize the objective defined in Equation 2, while IAI and Evtree lose optimality when the depth limit increases and regularization coefficient decreases. OSRT, which provides provably optimal trees, defines a frontier between training loss and the number of leaves. These plots also show how far away the CART and GUIDE objectives are from the optimal solution. *It is not possible to determine whether the solutions of any method are optimal or close to optimal, without comparing to OSRT, because OSRT is the only method with an optimality guarantee.* IAI and Evtree are likely to find optimal trees under shallow depth constraints (2 and 3), but often fail once the depth constraint is greater than 3. Note that, often, all methods do achieve an optimal solution, which means there is no room for improvement on that problem.

We also observed high variance in the loss of evtrees (and sometimes IAI trees) under the same depth limit, even with a fixed random seed (see Figure 8 at depths 5 and 6, Figure 9 at depths 5, 6, 7, 8 and 9, Figure 10 at depths 5, 6 and 8, and Figure 11 at depths 7 and 8). Given a dataset and depth limit, if two regularization coefficients result in trees with the same number of leaves, those should have the same loss, otherwise at least one of them must be sub-optimal. We observe that sometimes they can be very far from optimal. We discuss the uncontrollability of IAI and Evtree in Section F.

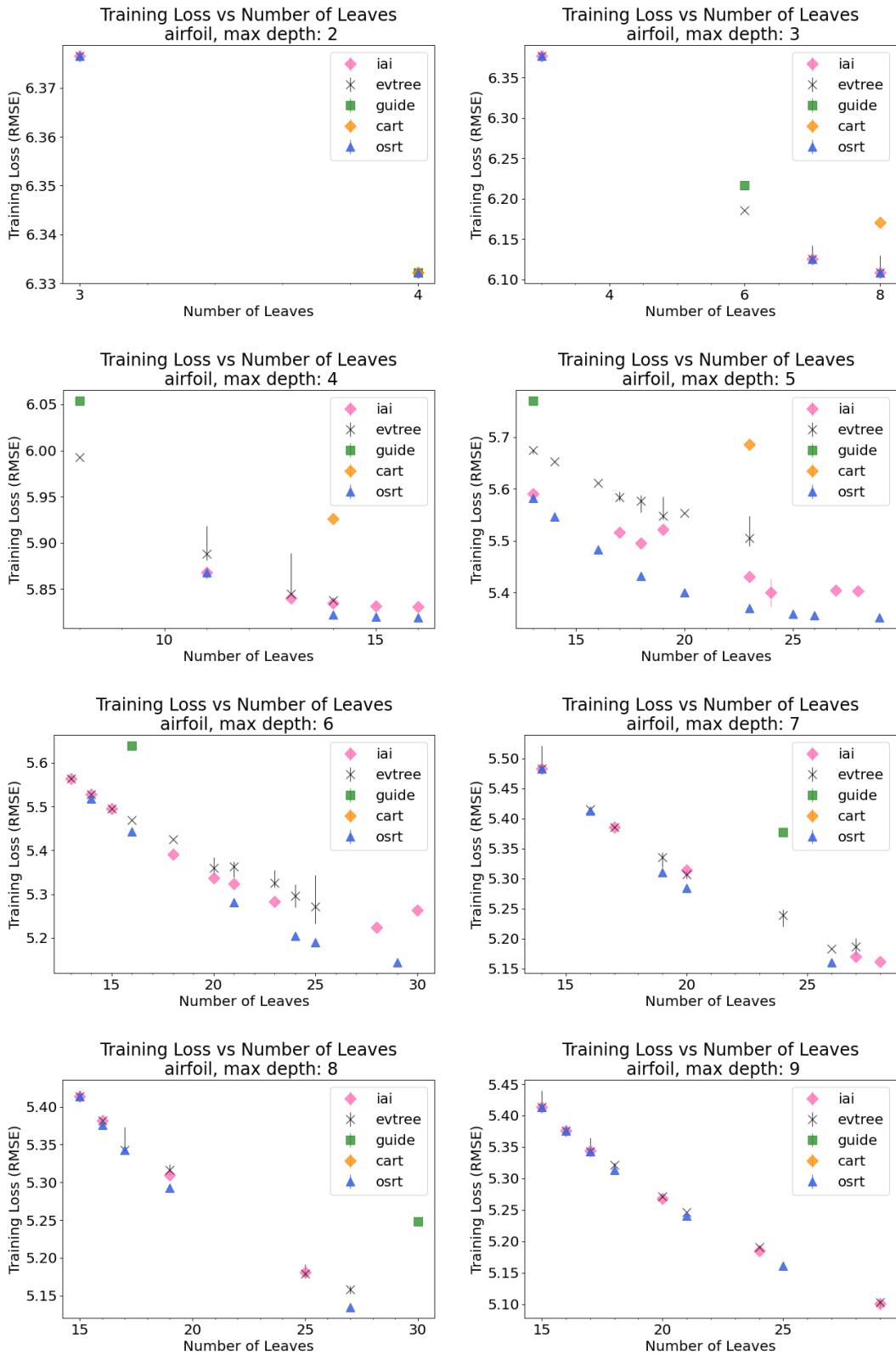


Figure 8: Training loss achieved by IAI, Evtree, GUIDE, CART and OSRT as a function of number of leaves on dataset: airfoil.

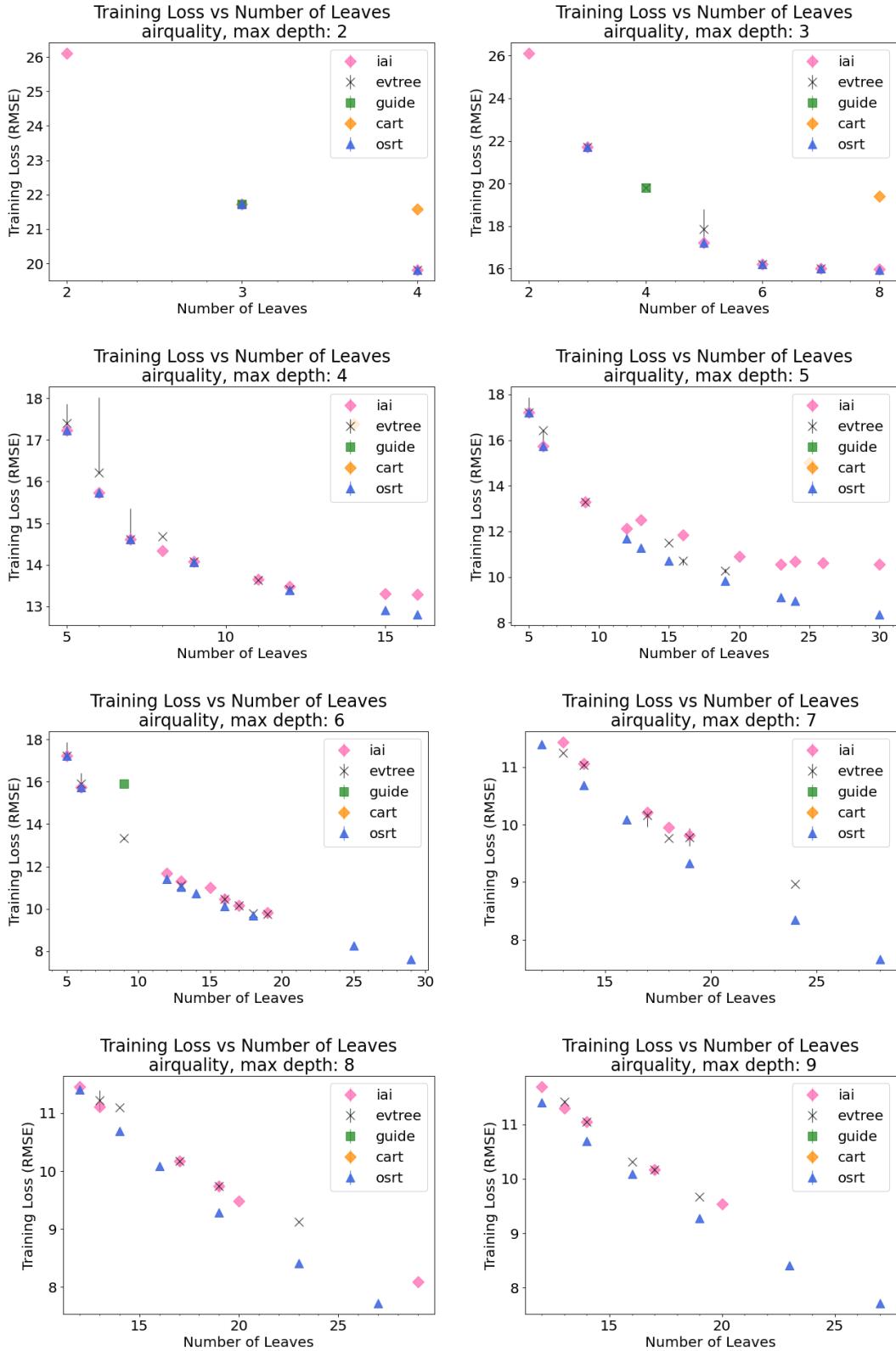


Figure 9: Training loss achieved by IAI, Evtree, GUIDE, CART and OSRT as a function of number of leaves on dataset: airquality.

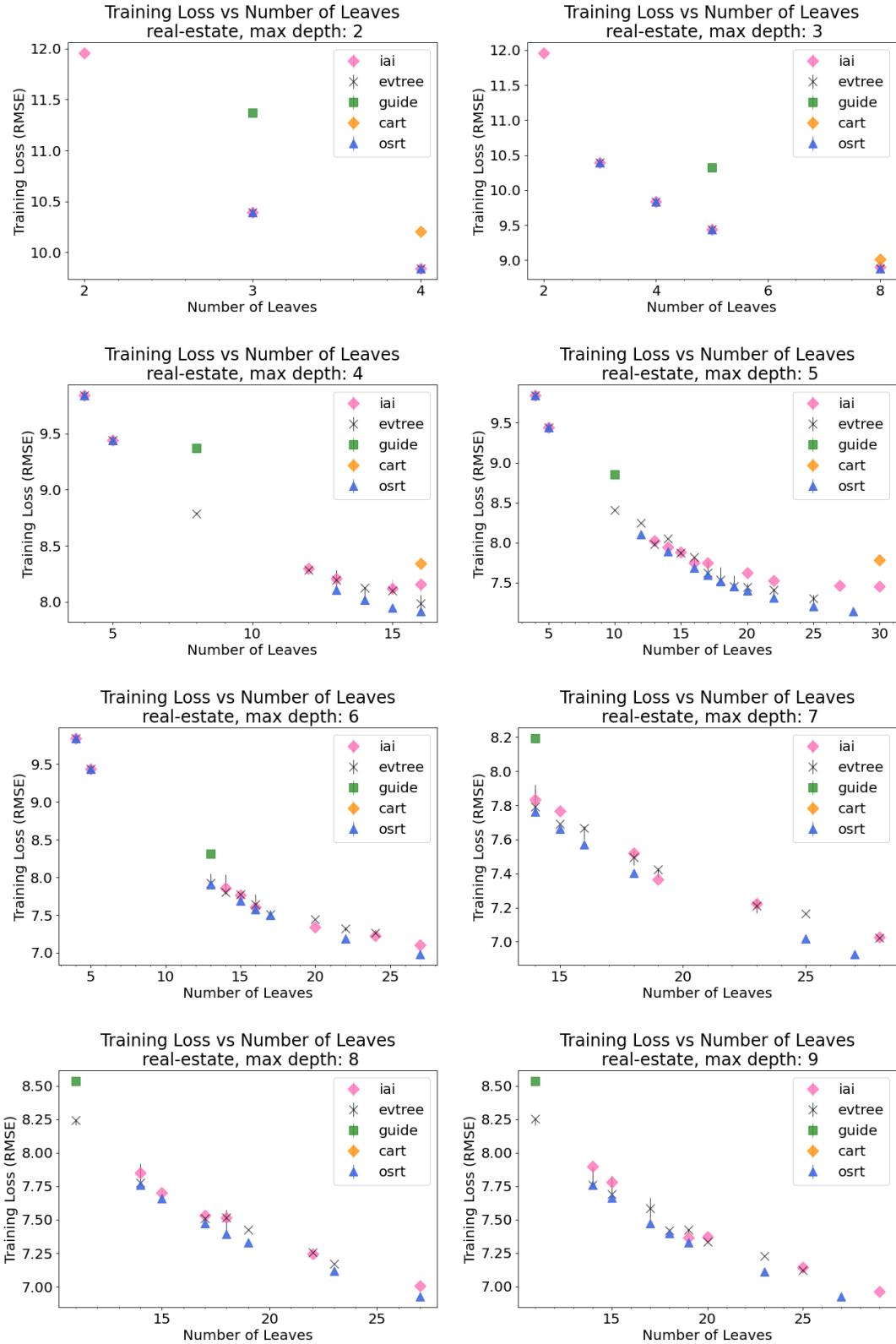


Figure 10: Training loss achieved by IAI, Evtree, GUIDE, CART and OSRT as a function of number of leaves on dataset: real-estate.

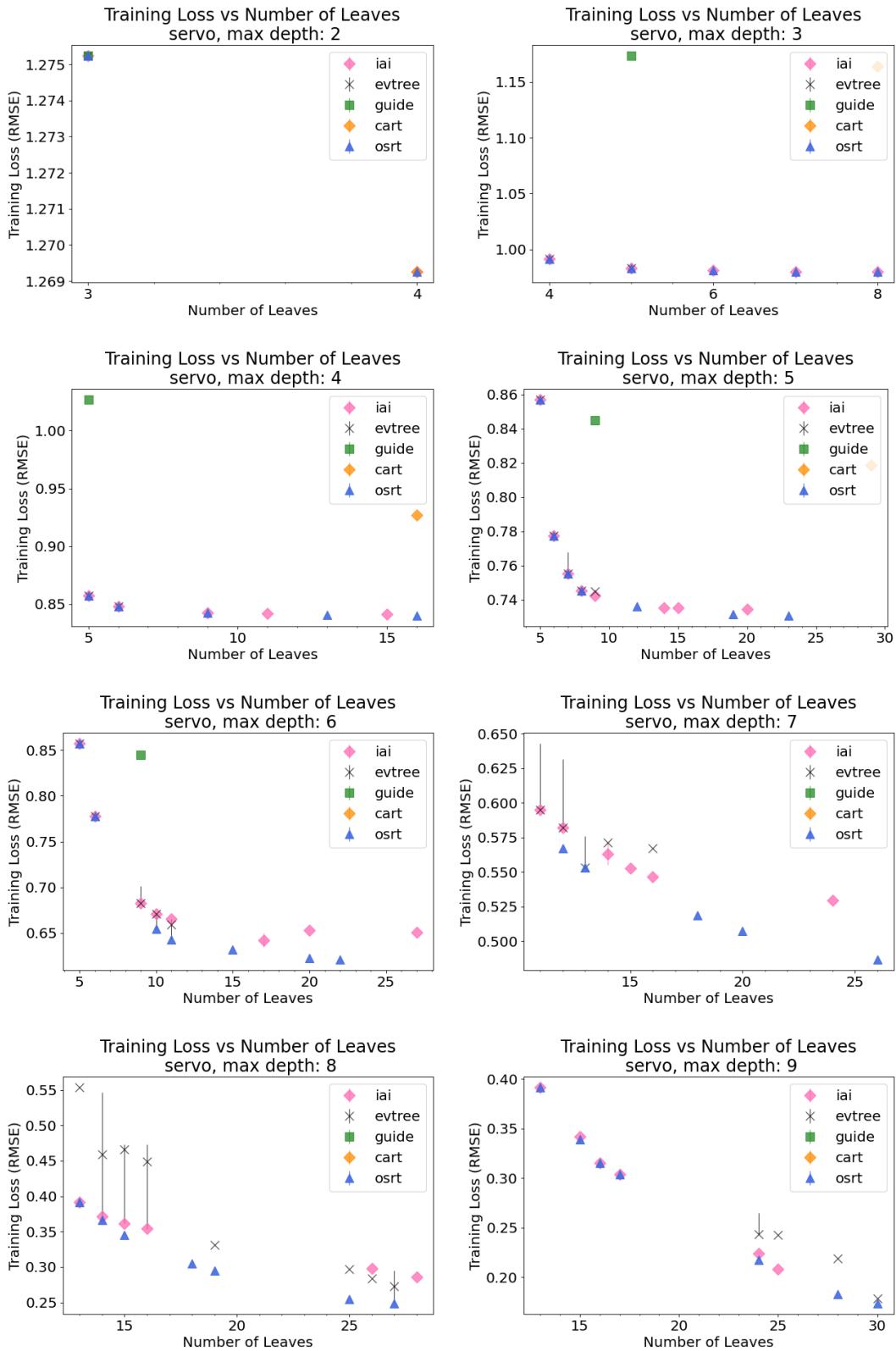


Figure 11: Training loss achieved by IAI, Evtree, GUIDE, CART and OSRT as a function of number of leaves on dataset: servo.

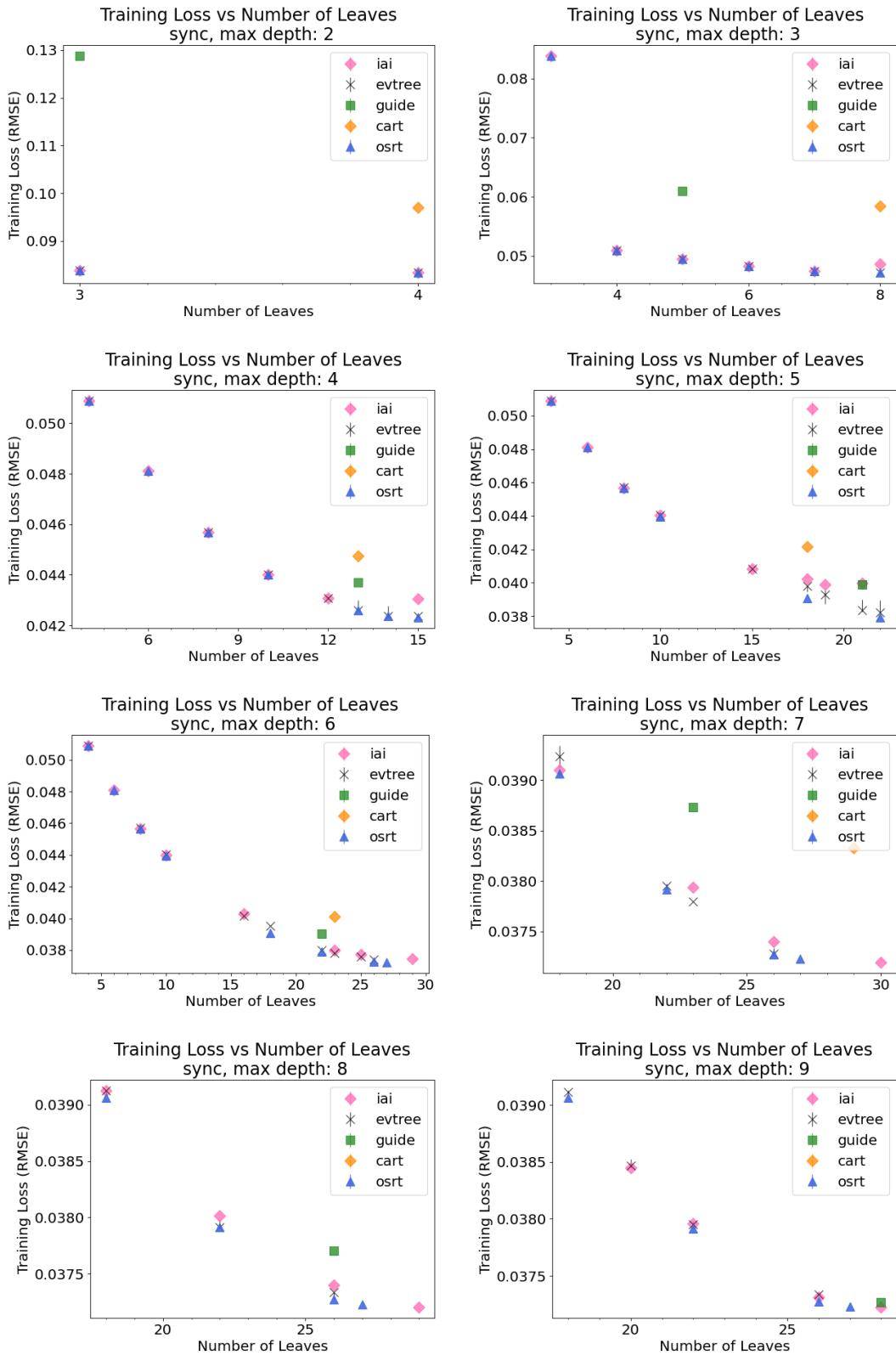


Figure 12: Training loss achieved by IAI, Evtree, GUIDE, CART and OSRT as a function of number of leaves on dataset: sync.

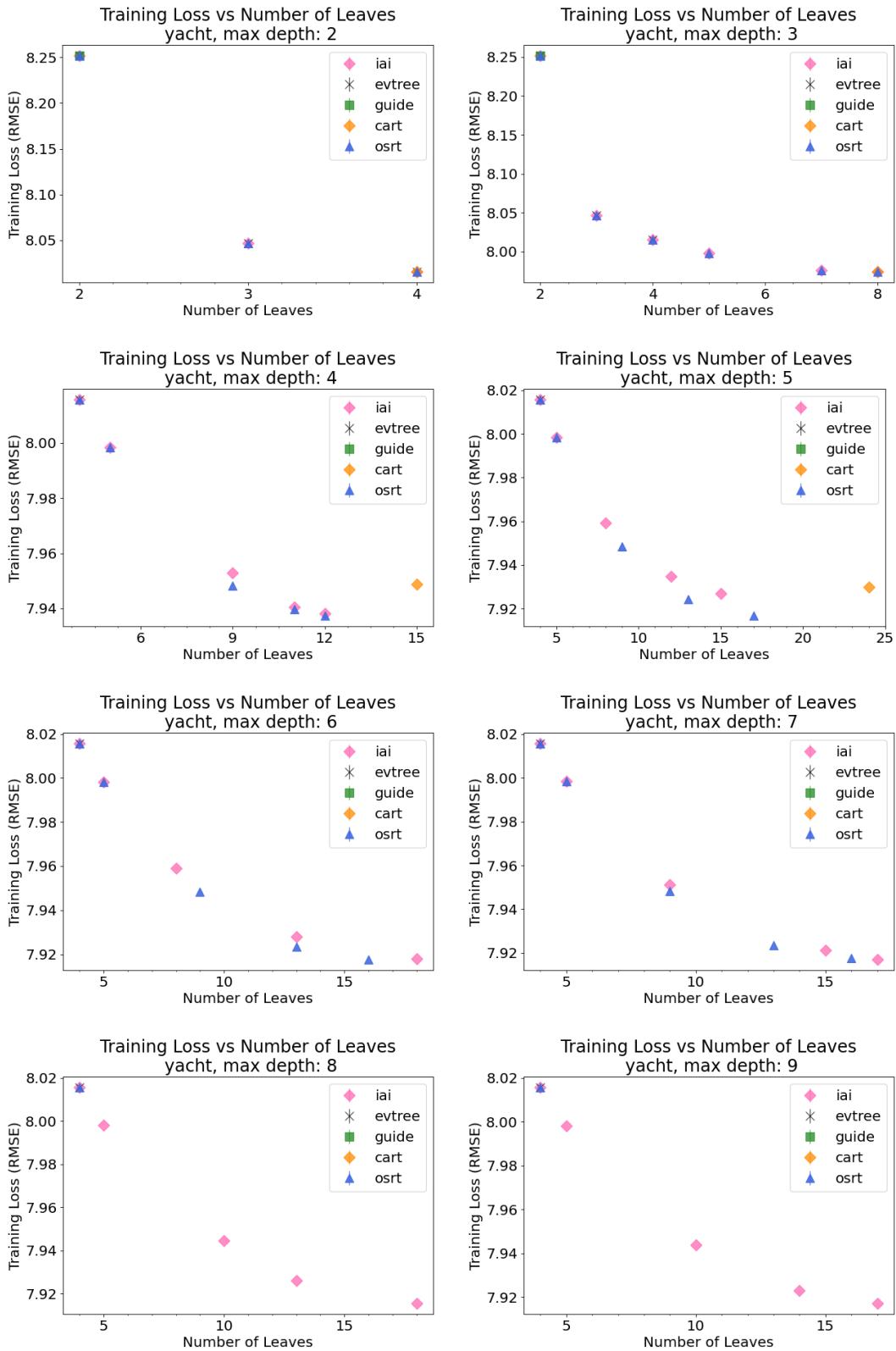


Figure 13: Training loss achieved by IAI, Evtree, GUIDE, CART and OSRT as a function of number of leaves on dataset: yacht.

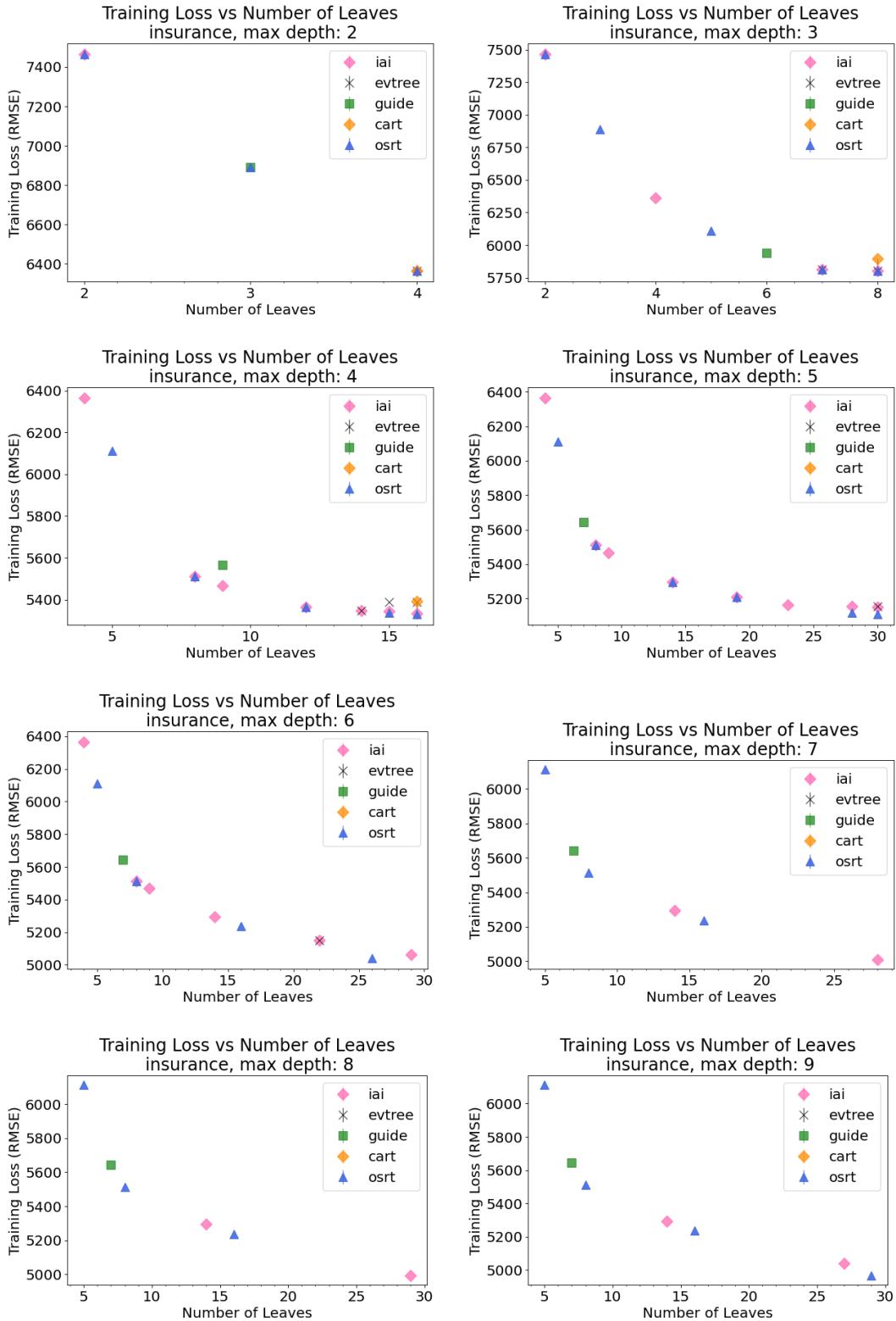


Figure 14: Training loss achieved by IAI, Evtree, GUIDE, CART and OSRT as a function of number of leaves on dataset: insurance.

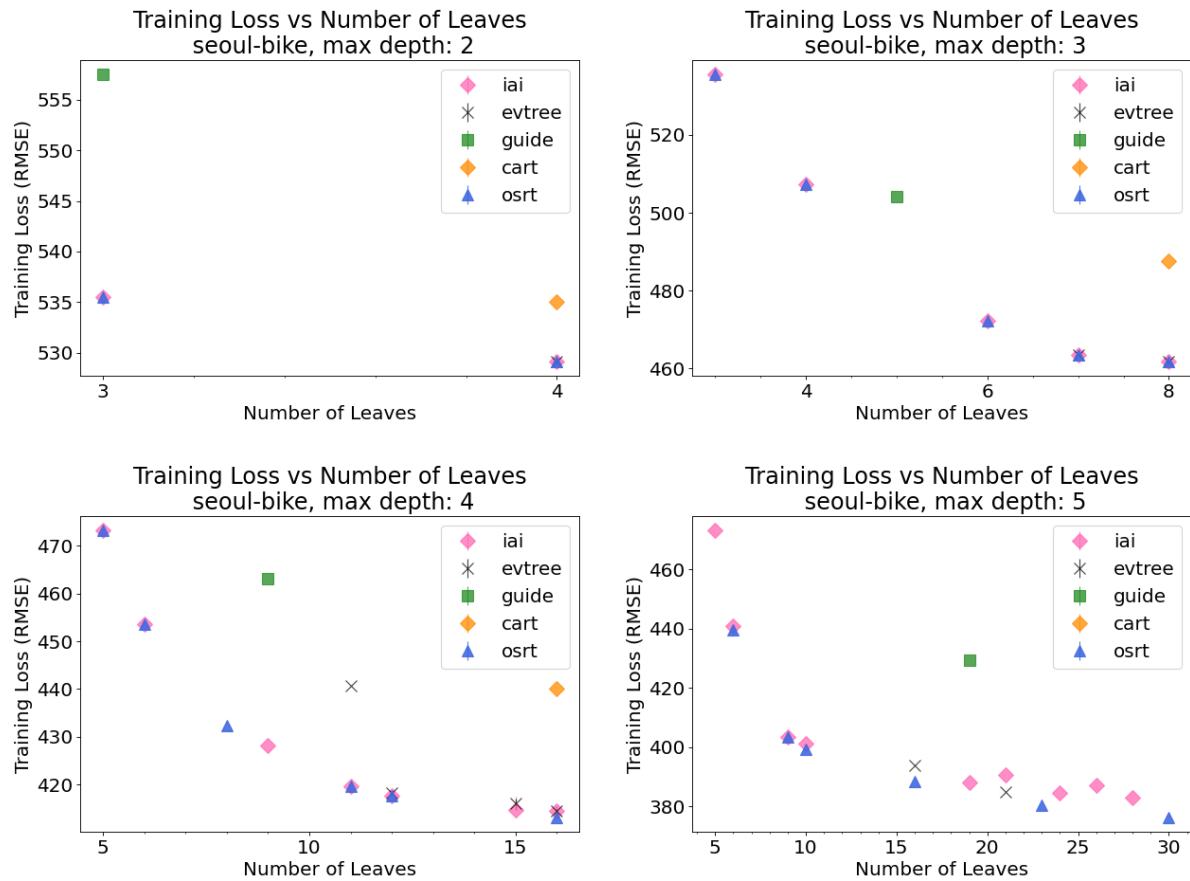


Figure 15: Training loss achieved by IAI, Evtree, GUIDE, CART and OSRT as a function of number of leaves on dataset: seoul-bike, depths 2 – 5. Depths 6 – 9 are omitted since Evtree and OSRT timed out.

E Timeout Statistics

We observed timeouts in some of the experiments in Section D. Evtree timed out on dataset *insurance* and *seoul-bike*, and OSRT timed out on dataset *seoul-bike*. *Evtree tends to time out more often than OSRT*. Tables 3 and 4 show the number of timed out runs.

Table 3: Number of Timeout on Dataset *insurance*

Depth	#Timeout/total Evtree runs	#Timeout/total OSRT runs
4	0	0
5	0	0
6	10%	0
7	60%	0
8	70%	0
9	80%	0
Total	27.5%	0

Table 4: Number of Timeout on Dataset *seoul-bike*

Depth	#Timeout/total Evtree runs	#Timeout/total OSRT runs
4	10%	0
5	40%	30%
6	90%	45%
7	100%	45%
8	100%	45%
9	100%	45%
Total	55%	26.25%

F Experiment: Controllability

Collection and Setup: We ran this experiment on 2 datasets: *real-estate*, *servo*. For each dataset, we gave IAI, Evtree and OSRT the same depth limit and regularization coefficient. (Recall that Evtree has different scale of regularization coefficients, we gave it coefficient that produce trees with a similar number of leaves.) For each dataset, we used the configurations described below:

- *real-estate*: Set depth limit 4 – 9 for all three algorithms, regularization coefficients 0.0005, 0.001, 0.005, 0.01, 0.05 for IAI and OSRT, 0.05, 0.08, 0.1, 0.5, 1 for Evtree.
- *servo*: Set depth limit 4 – 9 for all three algorithms, regularization coefficients 0.0005, 0.001, 0.005, 0.01, 0.05 for IAI and OSRT, 0.05, 0.08, 0.1, 0.5, 1 for Evtree.

For each combination of dataset, depth limit, regularization coefficient and algorithm, we ran 10 times with 10 different random seeds.

Calculations: For each combination of dataset, regularization coefficient and algorithm, we produced a set of up to 10 trees, depending on if the runs exceeded the time limit. We summarized the measurements of training loss and number of leaves across the set of up to 10 trees by plotting the median of training loss and number of leaves; we showed the minimum and maximum number of leaves, and the best and worst training error in the set as the lower and upper error values respectively.

Results: Figures 16 and 17 show that for a given dataset, depth constraint and regularization coefficient, *OSRT consistently found the same optimal tree, while IAI and Evtree tended to depend on the random seeds*. Trees generated by IAI and Evtree could vary a lot in terms of prediction accuracy and sparsity. Poor sparsity can produce models that are uninterpretable and overfitted; IAI and Evtree are unable to avoid these phenomena.

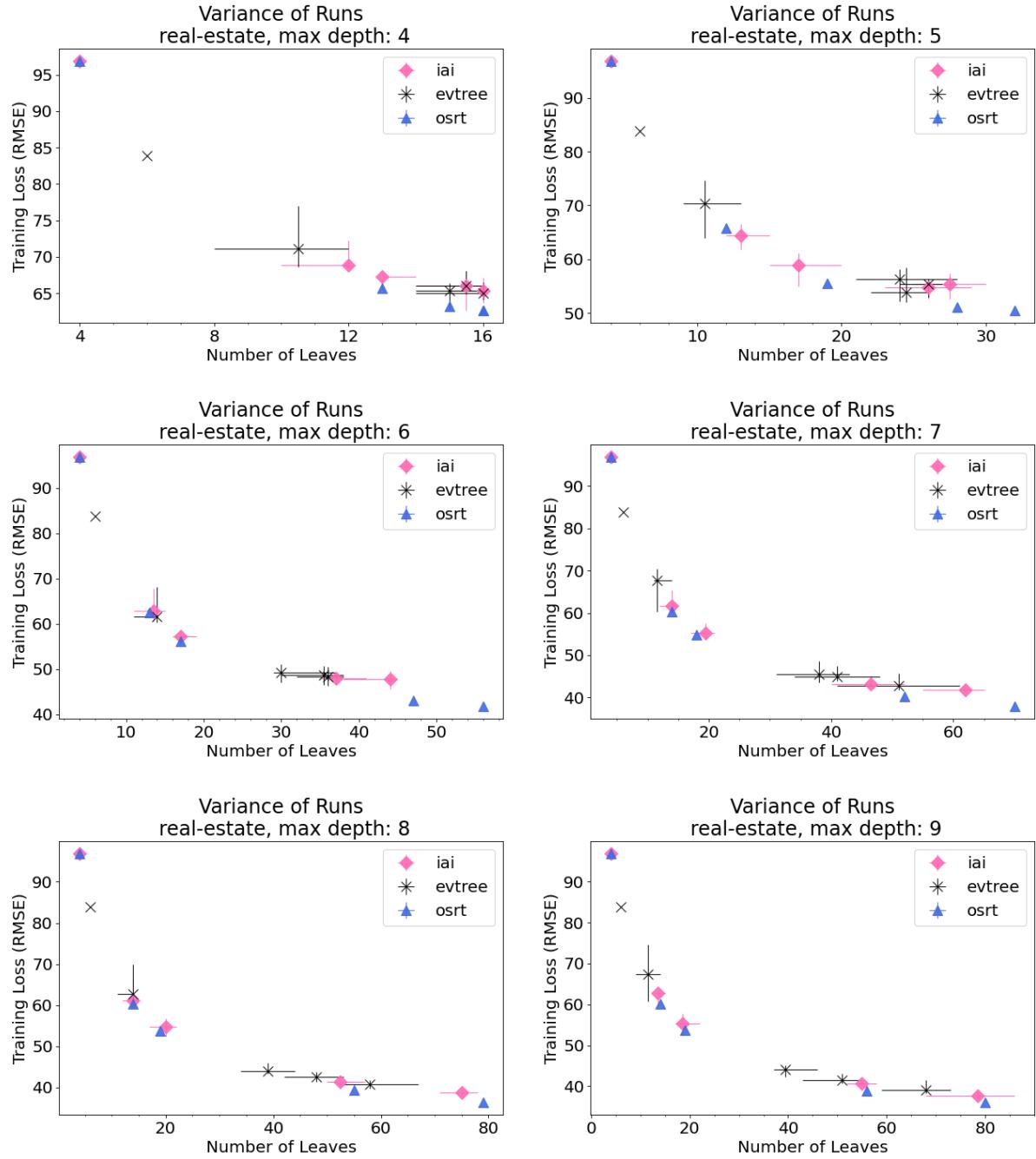


Figure 16: Variance (horizontal and vertical lines) of trees generated by IAI, Evtree and OSRT on dataset: *real-estate*, when different random seeds were used. Note that in cases like the max depth 5 plot for approximately 11 leaves in a tree, while evtree's loss lower bound drops below OSRT's blue triangle (which is optimal), it is not because the blue triangle is suboptimal; it is instead because evtree produced a larger size tree to get that low error, which is why there is horizontal variance in addition to the vertical variance.

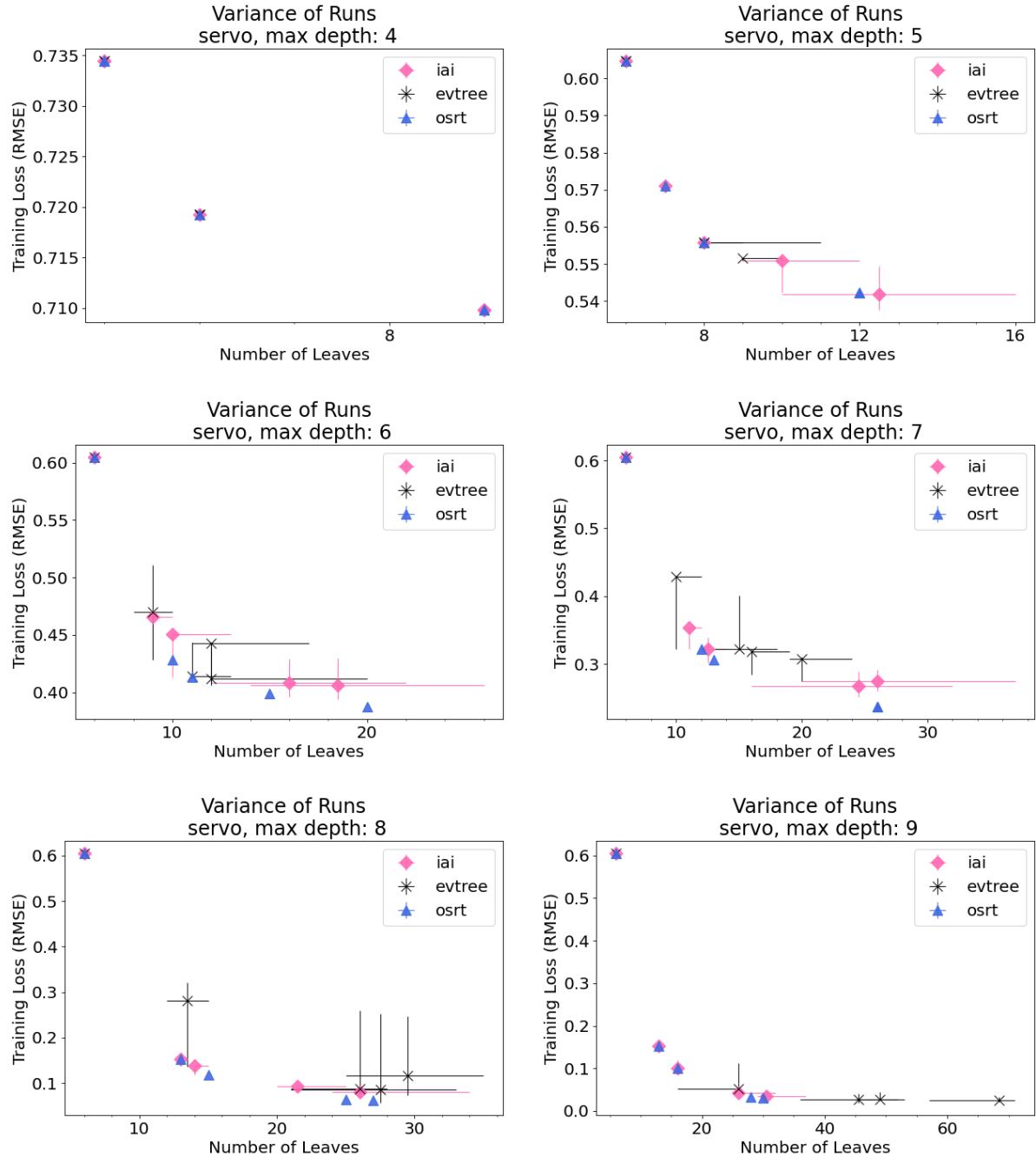


Figure 17: Variance (horizontal and vertical lines) of trees generated by IAI, Evtree and OSRT on dataset: servo, when different random seeds were used. Again, in cases like the max depth 6 figure for 10 leaves, IAI's loss lower bound drops below OSRT's blue triangle, which is optimal. This is because IAI produced a larger model to achieve this low loss, as indicated by the horizontal variance for IAI at 10 leaves.

G Experiment: Time vs. Sparsity

Collection and Setup: We compared the running time of different methods that produce trees of similar size, by running this experiment on 6 datasets: *airfoil*, *real-estate*, *servo*, *sync*, *yacht*, *insurance*. We used the same configurations in Section D, except we set the time limit to 30 minutes in this experiment and set regularization coefficients (0.10, 0.11, 0.12 ..., 0.99, 1.00, 1.1, 1.2, ..., 1.9, 2.0) for Evtree.

Calculations: We created one plot (time vs number of leaves) for each combination of dataset and depth limit. We again excluded trees with more than 30 leaves. Given a dataset and depth limit, one algorithm might produce multiple trees with the same number of leaves, so we plotted the median of these training times and show the best and worst time as lower and upper error values respectively.

Results: Figures 18-23 show that *OSRT* is generally faster than *Evtree*. Recall that trees of IAI and Evtree become sub-optimal once the depth limit is greater than 3, and OSRT is only slightly slower than IAI and much faster than Evtree, which means our method is *both accurate and fast*. Evtree tends to produce large and uninterpretable trees when the depth limit is greater than 7, therefore there are only few evtree points in some plots. OSRT slows down when the optimal trees for given depth and regularization are overfitted (due to correlated variables), which is shown in Section H. OSRT typically performs well for interpretable and sparse trees.

Evtree tends to converge to sub-optimal trees early if the depth is large and regularization is small, when the difference of the objective between trees is also relatively small, since Evtree will terminate when its best 5% population of trees stabilizes.

We observe “abort” issues when we run IAI on datasets that need more training time (e.g., *auction*, *seoul-bike* and *optical*). Error messages (see Listing 1) in calls of the local search function indicate IAI is using a local search approach, which may explain why its running time stays relatively unchanged with increasing tree depth. Since IAI is proprietary, we cannot confirm its algorithmic approach.

Listing 1: IAI Error Code

```

1 RuntimeError: <PyCall.jlwrap (in a Julia function called from Python)
2 JULIA: obj mismatch: before -2.384185791015625e-7 after 0.0
3 Stacktrace:
4 [1] error(s::String)
5 [2] greedy_search!(tree::IAITrees.Tree{IAIBase.RegressionTask, IAITrees.Node{IAIBase.
    RegressionTask, IAIBase.RegressionFit}}, gs::OptimalTrees.LocalSearcher{IAIBase.
    RegressionTask, OptimalTrees.RegressionEvaluatorMSEConstant, IAIBase.
    RegressionTarget})
6 [3] run_worker_iteration!(rep::Int64, show_progress::Bool, ls::OptimalTrees.
    LocalSearcher{IAIBase.RegressionTask, OptimalTrees.RegressionEvaluatorMSEConstant,
    IAIBase.RegressionTarget}, gs::OptimalTrees.LocalSearcher{IAIBase.RegressionTask,
    OptimalTrees.RegressionEvaluatorMSEConstant, IAIBase.RegressionTarget}, rng_gen::
    IAIBase.RandomStreams.MRG32k3aGen)
7 [4] run_task(tid::Int64, f_run::Function, f_setup::Function, job_channel::Channel{Int64
}, results_channel::Channel{Any}, progress::IAIBase.Progress, uses_subprogress::Bool
, obj::Tuple{OptimalTrees.LocalSearcher{IAIBase.RegressionTask, OptimalTrees.
    RegressionEvaluatorMSEConstant, IAIBase.RegressionTarget}, OptimalTrees.
    LocalSearcher{IAIBase.RegressionTask, OptimalTrees.RegressionEvaluatorMSEConstant,
    IAIBase.RegressionTarget}})
8 [5] spawn_tasks(::Int64, ::Int64, ::Function, ::Vararg{Any})
9 [6] run_distributed!(f_consume::Function, obj::Tuple{OptimalTrees.LocalSearcher{IAIBase.
    RegressionTask, OptimalTrees.RegressionEvaluatorMSEConstant, IAIBase.
    RegressionTarget}, OptimalTrees.LocalSearcher{IAIBase.RegressionTask, OptimalTrees.
    RegressionEvaluatorMSEConstant, IAIBase.RegressionTarget}}, f_run::Function, f_setup
    ::Function, n_jobs::Int64, procs::Vector{Int64}, n_threads::Int64, progress::IAIBase.
    Progress, uses_subprogress::Bool)
10 [7] run_distributed(show_progress::Bool, procs::Vector{Int64}, n_threads::Int64, n_jobs
    ::Int64, message::String, iter_func::Function, iter_input::Tuple{OptimalTrees.
    LocalSearcher{IAIBase.RegressionTask, OptimalTrees.RegressionEvaluatorMSEConstant,
    IAIBase.RegressionTarget}, OptimalTrees.LocalSearcher{IAIBase.RegressionTask,
    OptimalTrees.RegressionEvaluatorMSEConstant, IAIBase.RegressionTarget}}, iter_setup
    ::Function, iter_uses_subprogress::Bool, consume_func::Function)
11 [8] (::IAIBase.var"#run_distributed##kw") (::NamedTuple{(:iter_input, :iter_setup,
    :iter_func, :iter_uses_subprogress, :consume_func), Tuple{Tuple{OptimalTrees.
    LocalSearcher{IAIBase.RegressionTask, OptimalTrees.RegressionEvaluatorMSEConstant,
    IAIBase.RegressionTarget}, OptimalTrees.LocalSearcher{IAIBase.RegressionTask,
    OptimalTrees.RegressionEvaluatorMSEConstant, IAIBase.RegressionTarget}}, typeof(
    OptimalTrees.task_local_copy), typeof(OptimalTrees.run_worker_iteration!), Bool,
    OptimalTrees.var"#95#97{OptimalTrees.OptimalTreeRegressor}}})
```

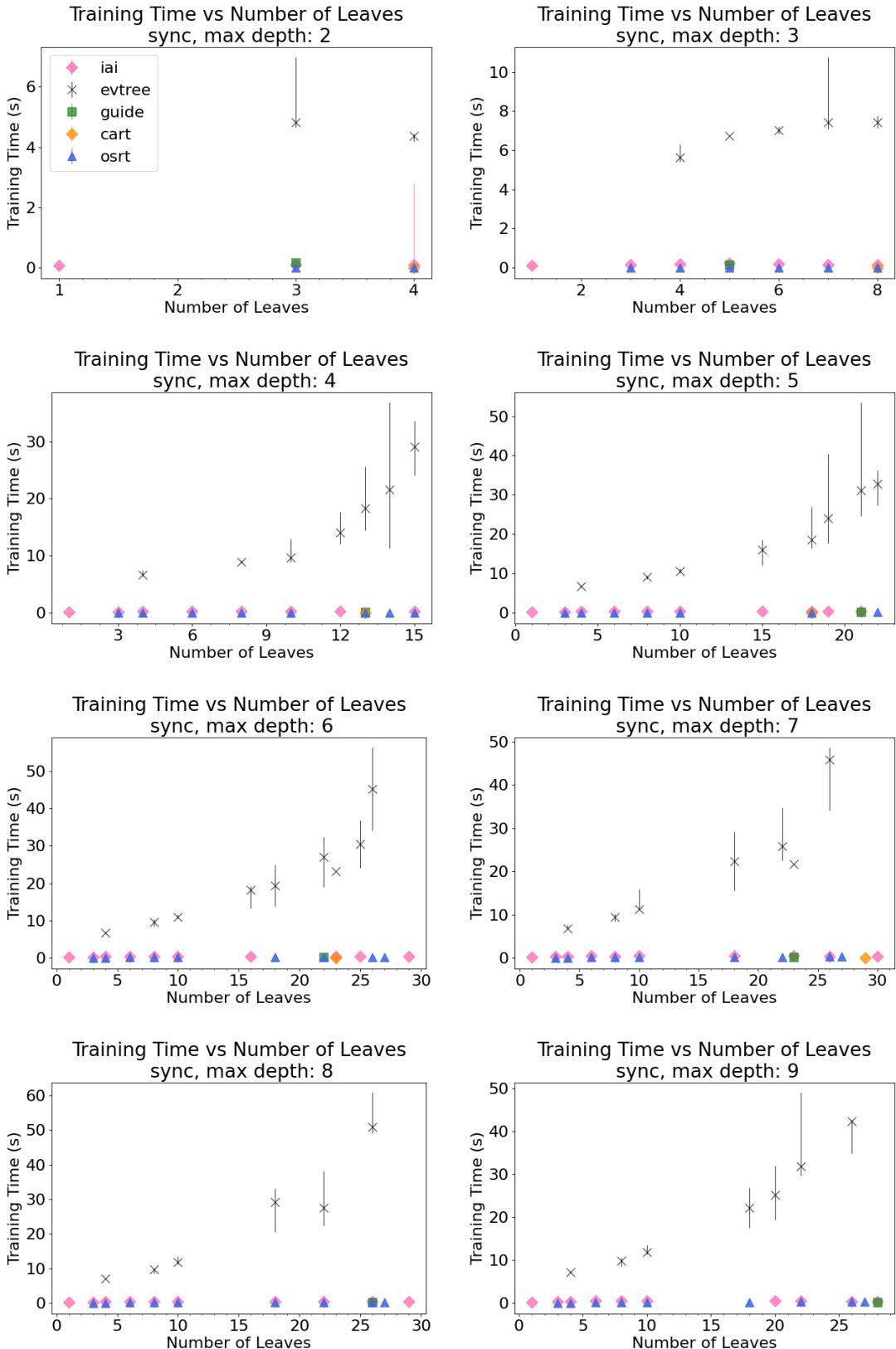


Figure 18: Training time of OSRT, IAI, Evtree, CART, GUIDE, as a function of number of leaves, on dataset: sync

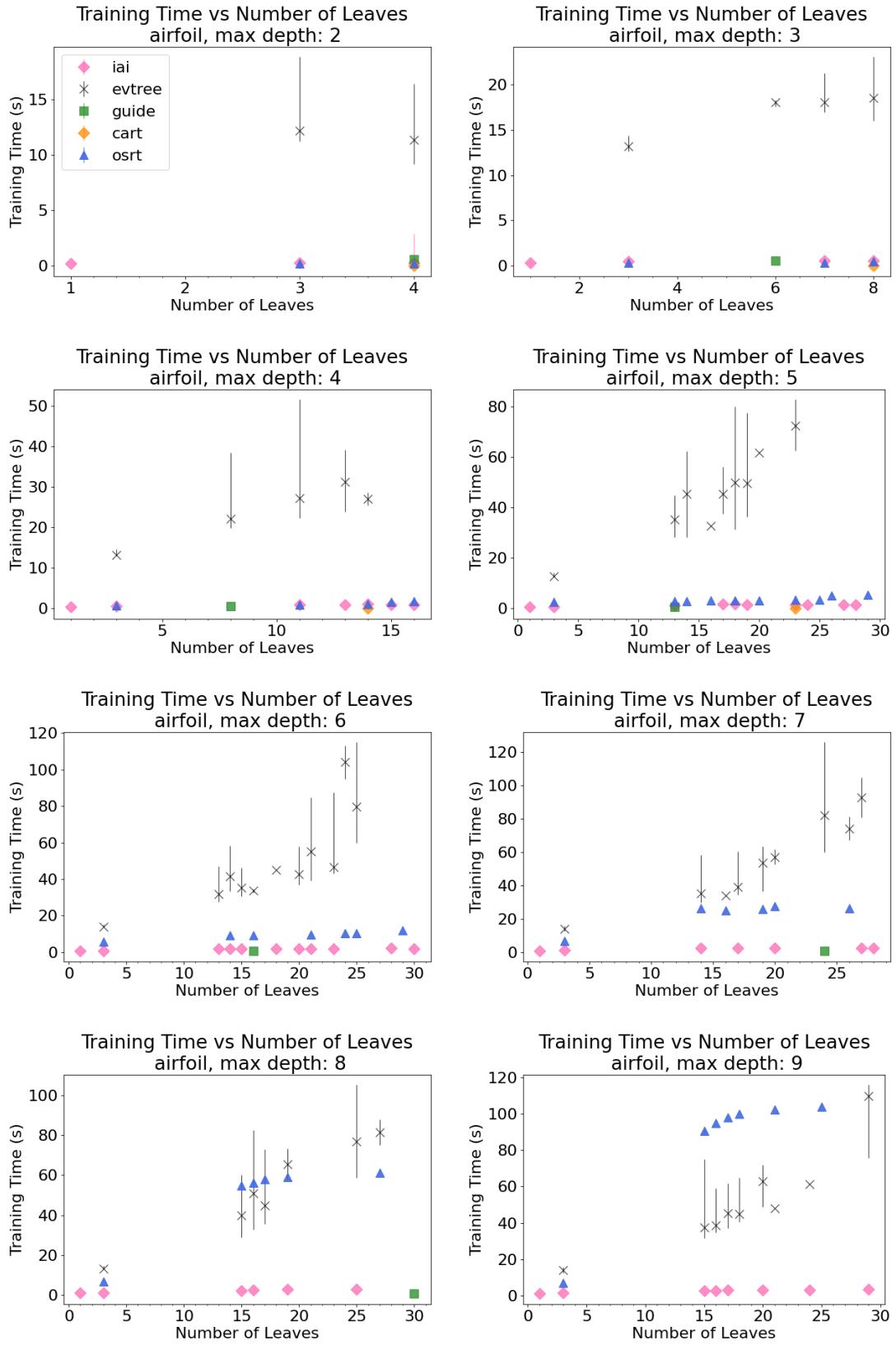


Figure 19: Training time of OSRT, IAI, Evtree, CART, GUIDE as a function of number of leaves on dataset: airfoil

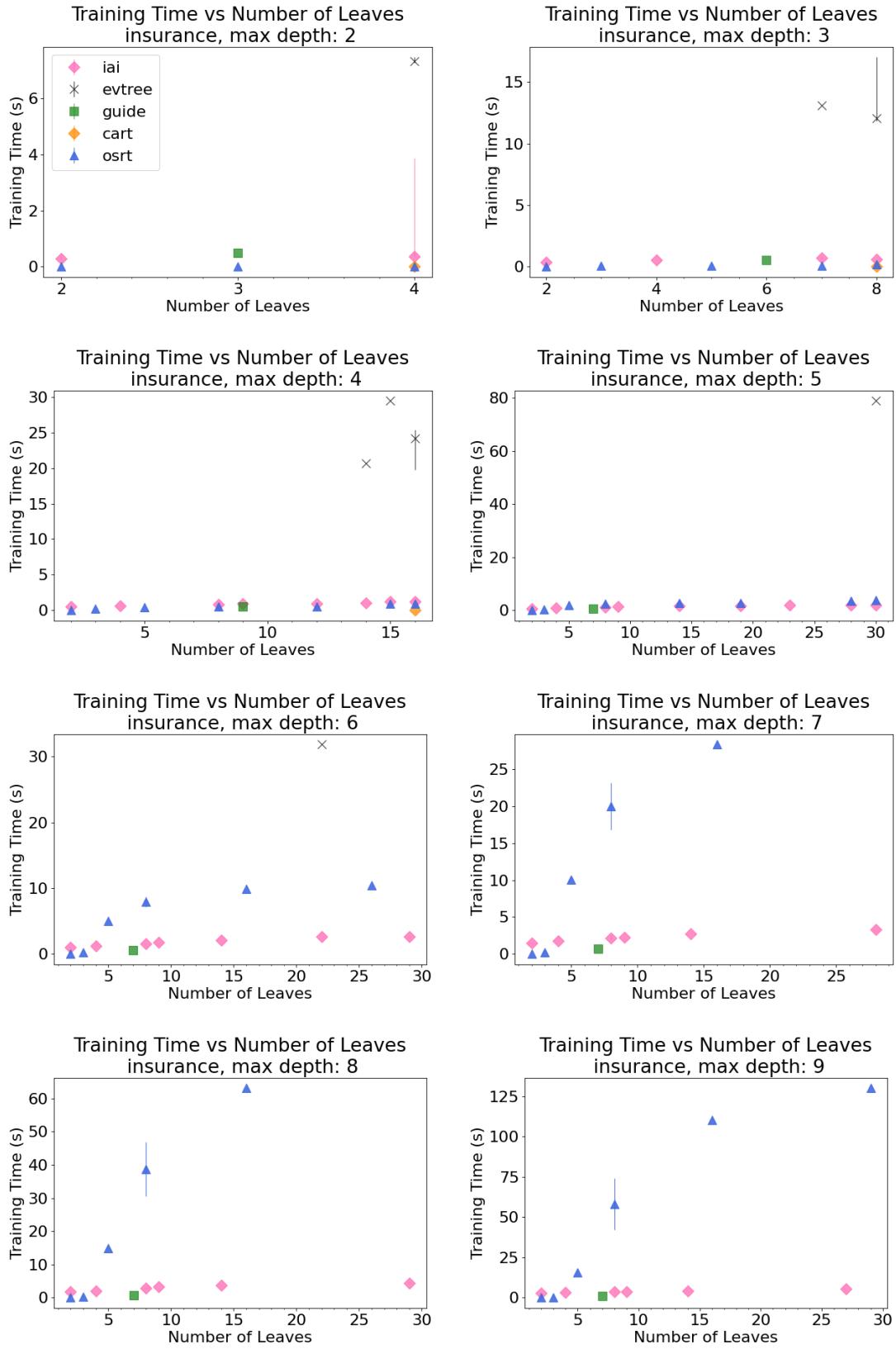


Figure 20: Training time of OSRT, IAI, Evtree, CART, GUIDE, as a function of number of leaves, on dataset: insurance

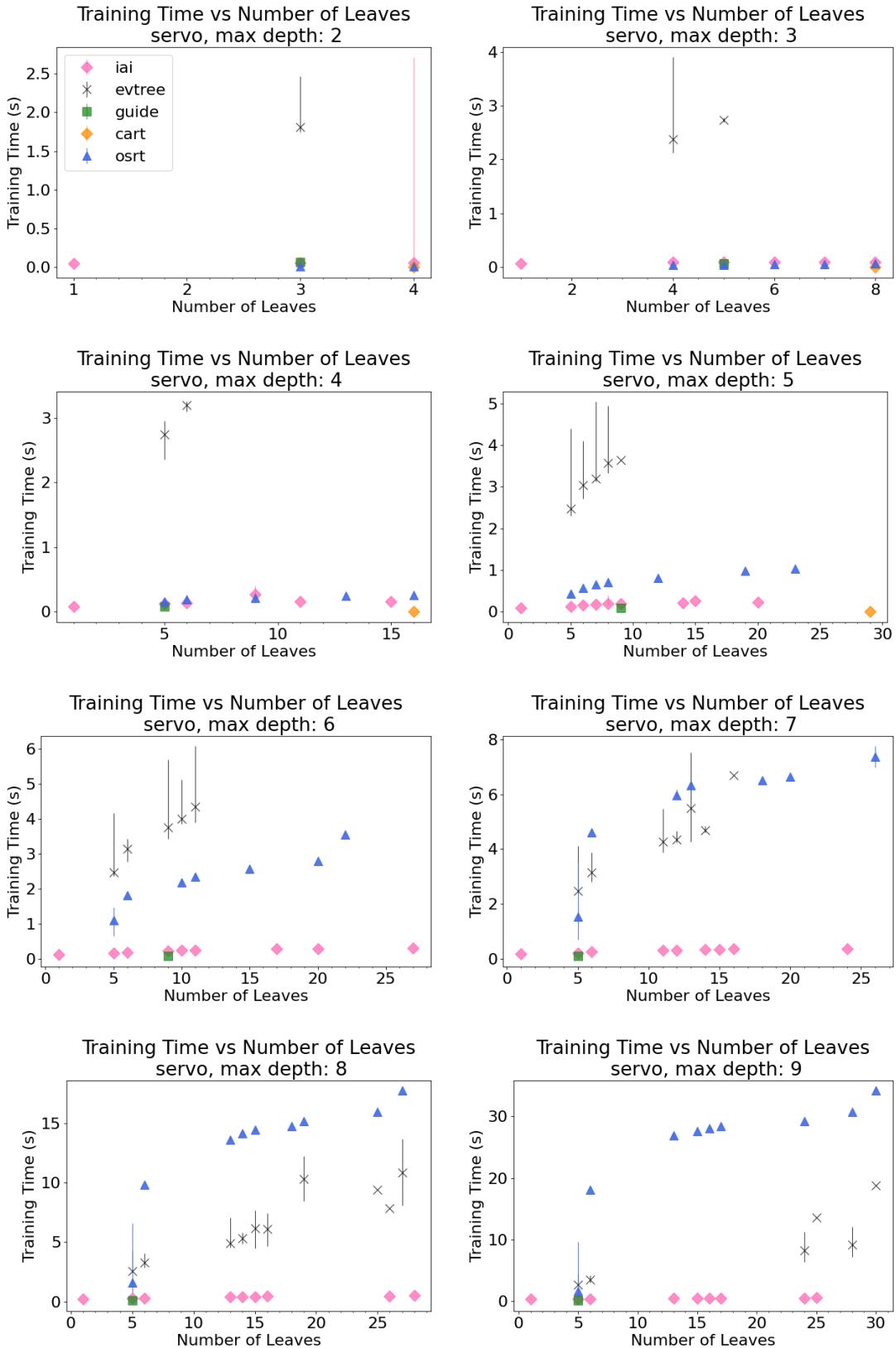


Figure 21: Training time of OSRT, IAI, Evtree, as a function of number of leaves, on dataset: servo

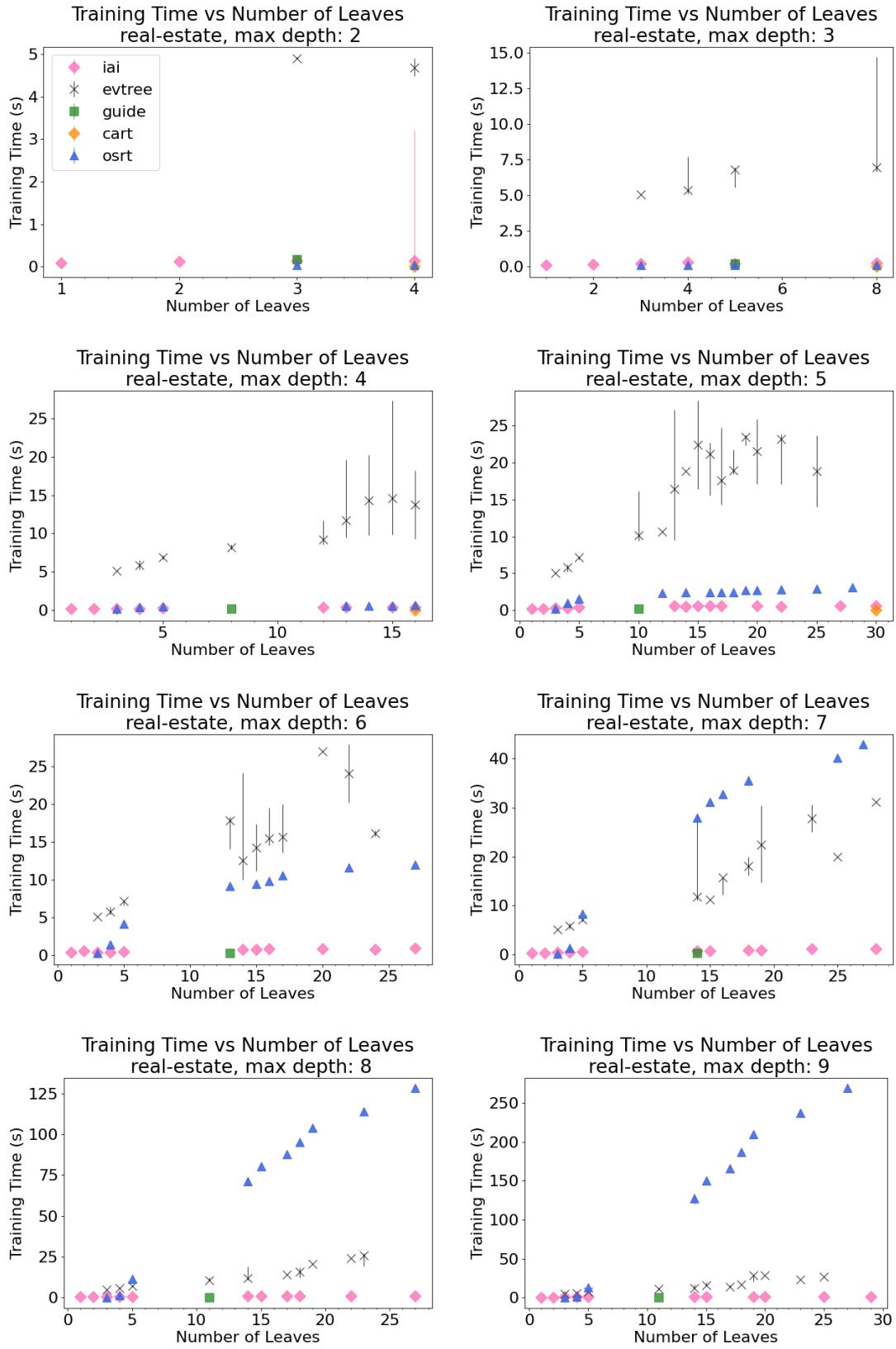


Figure 22: Training time of OSRT, IAI, Evtree, CART, GUIDE, as a function of number of leaves, on dataset: real-estate

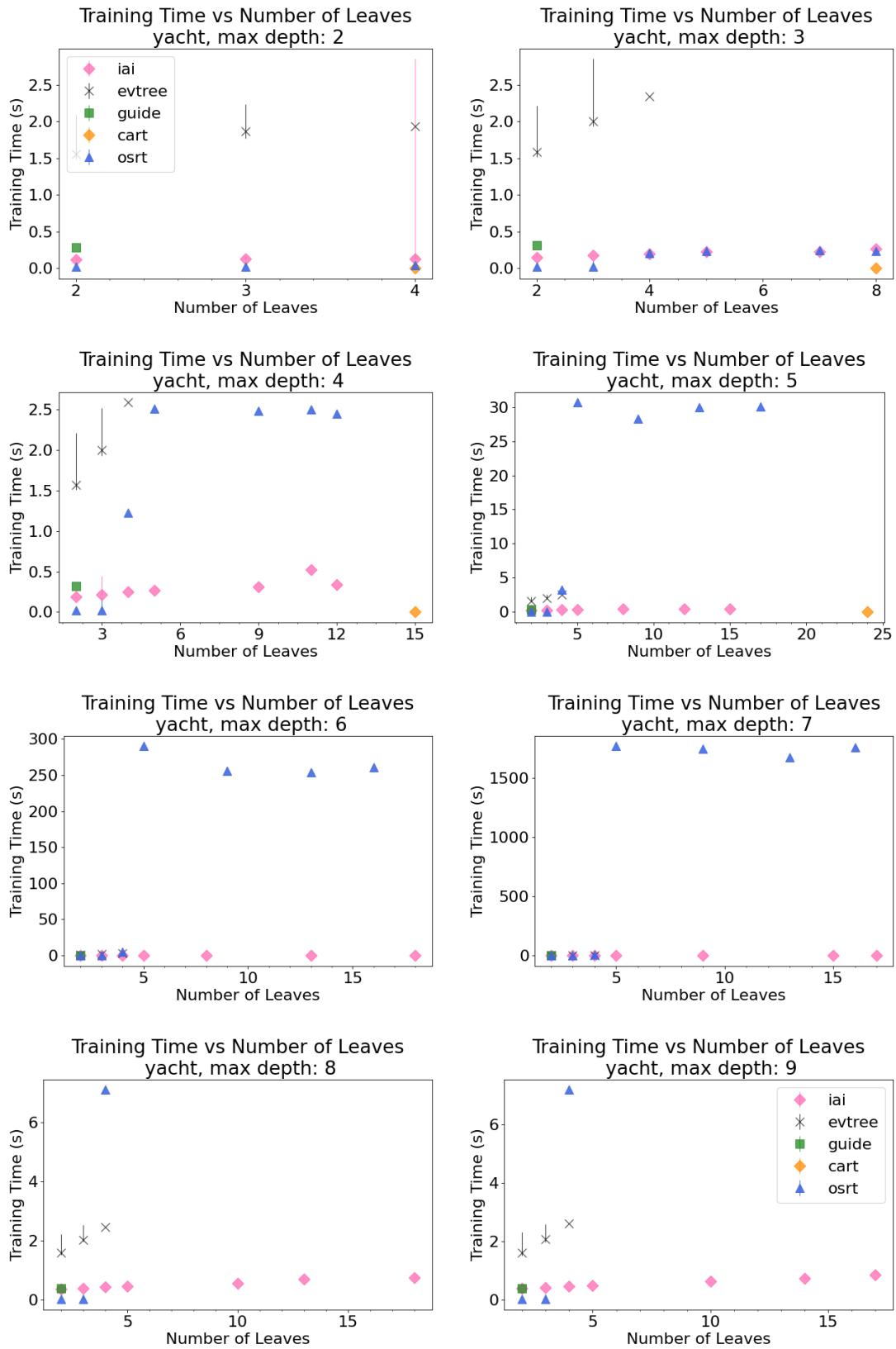


Figure 23: Training time of OSRT, IAI, Evtree, CART, GUIDE, as a function of number of leaves, on dataset: yacht

H Experiment: Cross Validation

Collection and Setup: We ran 5-fold cross-validation on the 5 datasets: *airfoil*, *sync*, *servo*, *seoul-bike*, *insurance*. The time limit was set to 5 minutes. For each dataset, we ran algorithms with different configurations:

- **CART:** We ran this algorithm with 4 different configurations: depth limit, d , ranging from 2 to 5, and a corresponding maximum leaf limit 2^d . All other parameters were set to their default.
- **GUIDE:** We ran this algorithm with 4 different configurations: depth limit, d , ranging from 2 to 5, and a corresponding maximum leaf limit 2^d . The minimum leaf node size was set to 2. All other parameters were set to the default.
- **IAI:** We ran this algorithm with 4×10 different configurations: depth limits ranging from 2 to 5, and 10 different regularization coefficients (0.1, 0.05, 0.025, 0.01, 0.0075, 0.005, 0.0025, 0.001, 0.0005, 0.0001).
- **Evtree:** We ran this algorithm with 4×20 different configurations: depth limits ranging from 2 to 5, and 20 different regularization coefficients (0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2). The minimum leaf node size was 1, minimum internal node size was 2. All other parameters were set to the default.
- **OSRT:** We ran this algorithm with 4×10 different configurations: depth limits ranging from 2 to 5, and 10 different regularization coefficients (0.1, 0.05, 0.025, 0.01, 0.0075, 0.005, 0.0025, 0.001, 0.0005, 0.0001).

Calculations: We drew one plot per dataset and depth. For each combination of regularization coefficient and algorithm in the same plot, we produced a set of up to 5 trees, depending on if the runs exceeded the time limit. We summarized the measurements of training loss, testing loss and number of leaves across the set of up to 5 trees by plotting the median, showing the minimum and maximum number of leaves, the best and worst training/testing error in the set as the lower and upper error values respectively.

Results: Figure 24 to 28 show that OSRT trees produce the lowest testing loss among all the regression trees. We noticed that the generalization performance of GUIDE is much worse than that of the other four methods. (GUIDE trees are not shown, because their testing loss is over four times higher than that of other methods). If an optimal tree significantly outperforms other sub-optimal trees in terms of training performance, it is also outperformed in testing (e.g., *airfoil* depth 5), otherwise the difference in testing loss becomes insignificant due to generalization error. Note that large trees start overfitting when depth is greater than 4 or 5, and sparse trees tend to have better generalization.

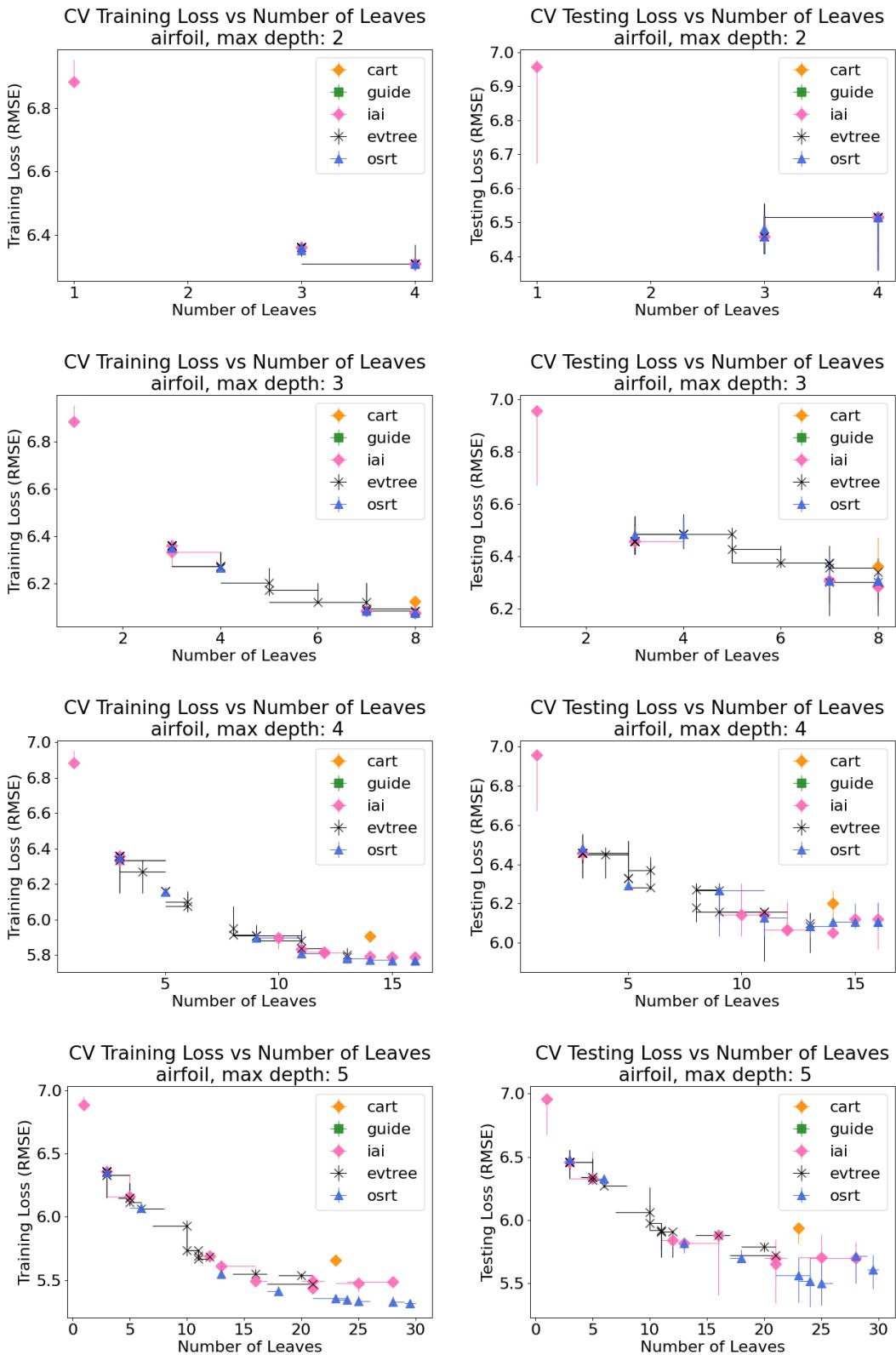


Figure 24: 5-fold CV of OSRT, IAI, Evtree, CART, GUIDE as a function of number of leaves on dataset: airfoil

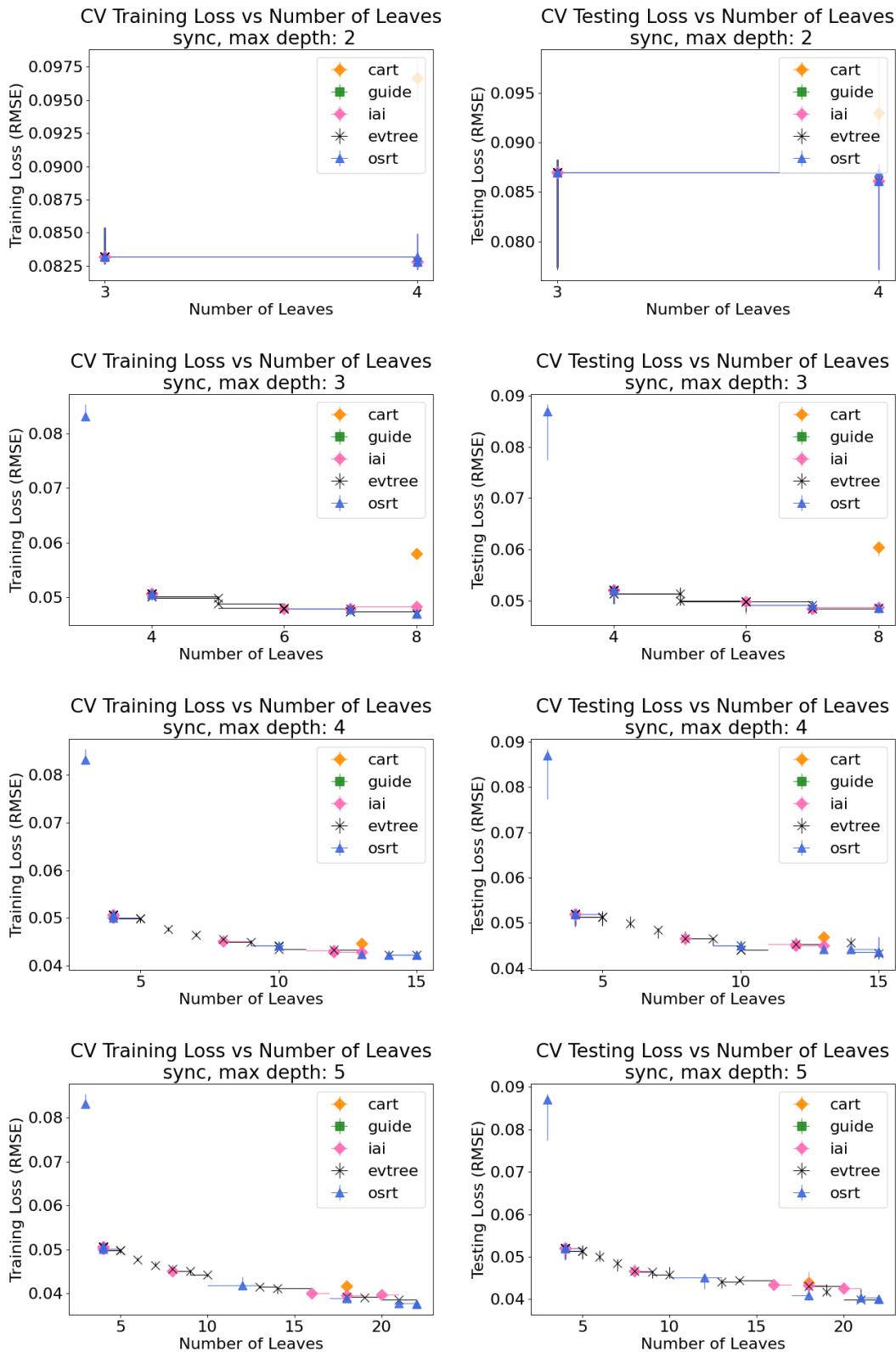


Figure 25: 5-fold CV of OSRT, IAI, Evtree, CART, GUIDE as a function of number of leaves on dataset: sync

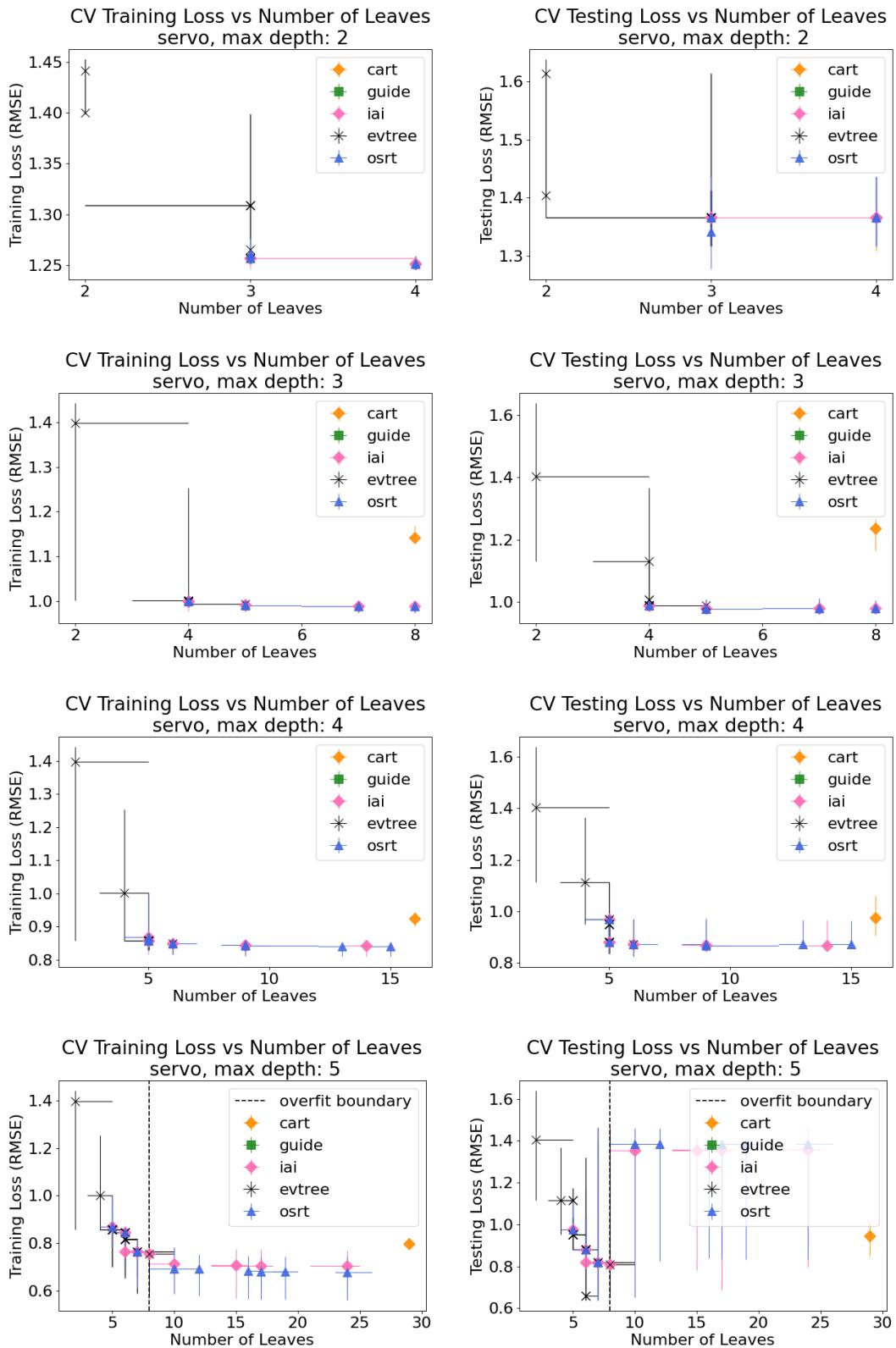


Figure 26: 5-fold CV of OSRT, IAI, Evtree, CART, GUIDE as a function of number of leaves on dataset: servo

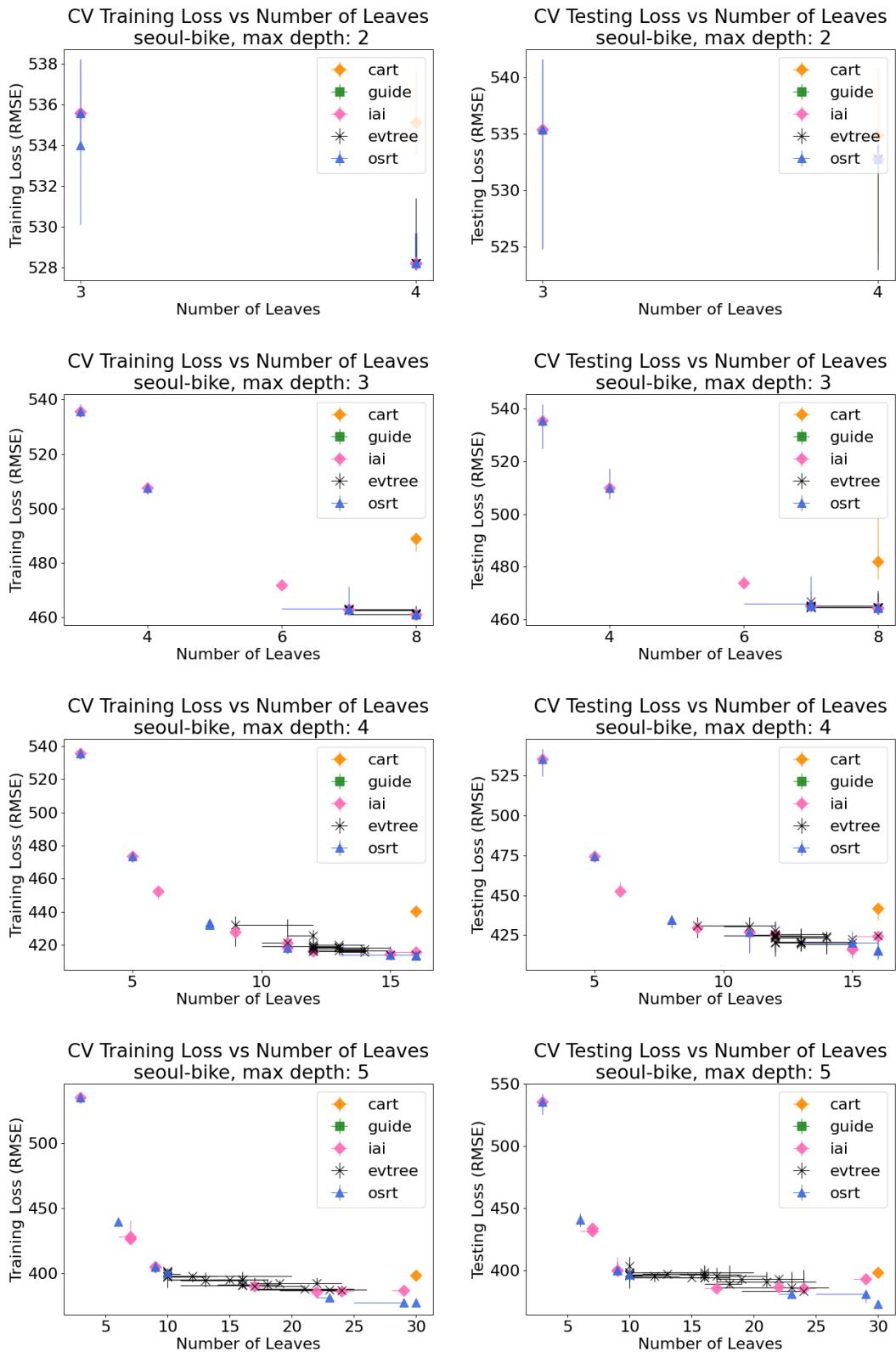


Figure 27: 5-fold CV of OSRT, IAI, Evtree, CART, GUIDE as a function of number of leaves on dataset: seoul-bike

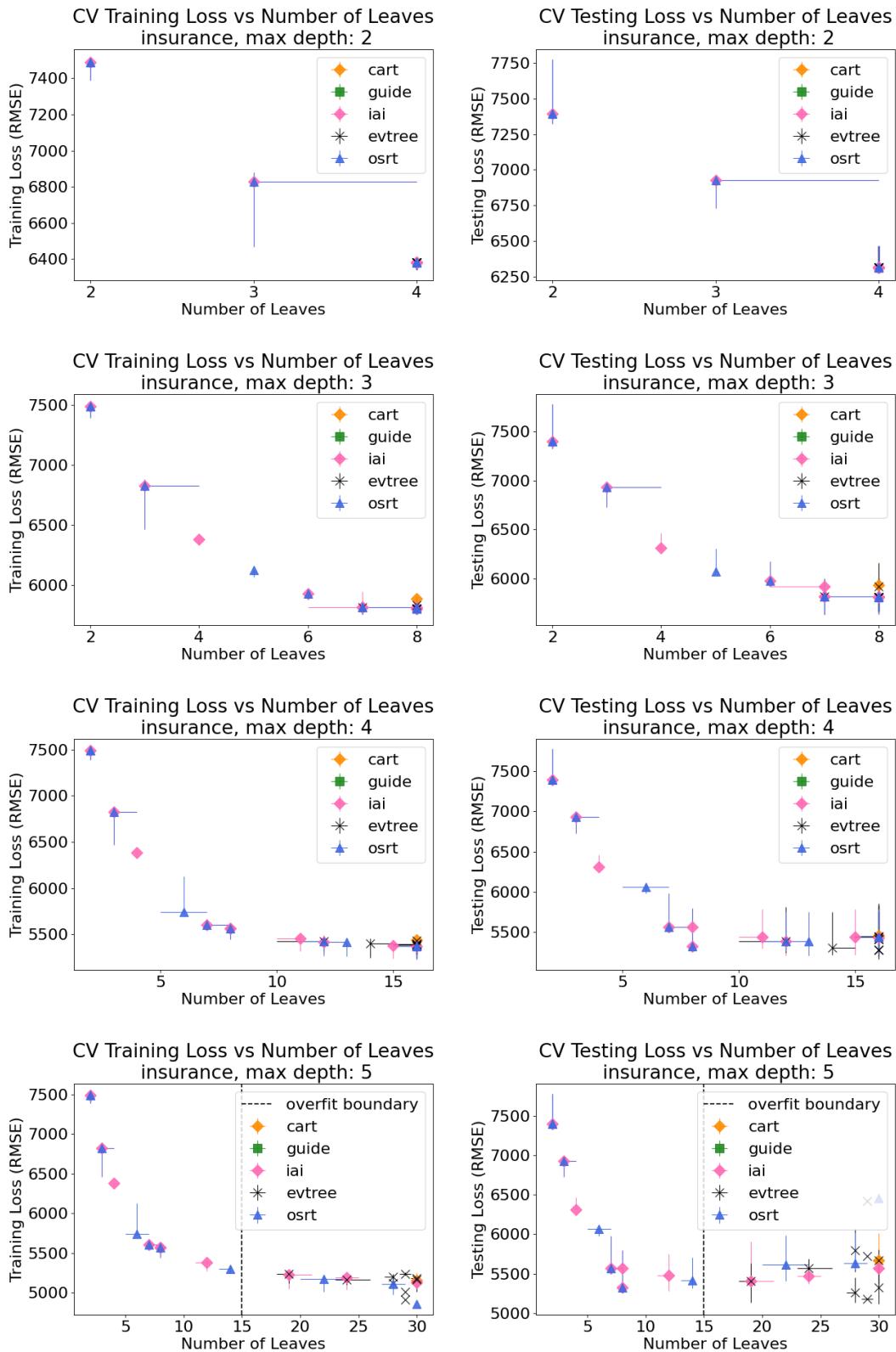


Figure 28: 5-fold CV of OSRT, IAI, Evtree, CART, GUIDE as a function of number of leaves on dataset: insurance

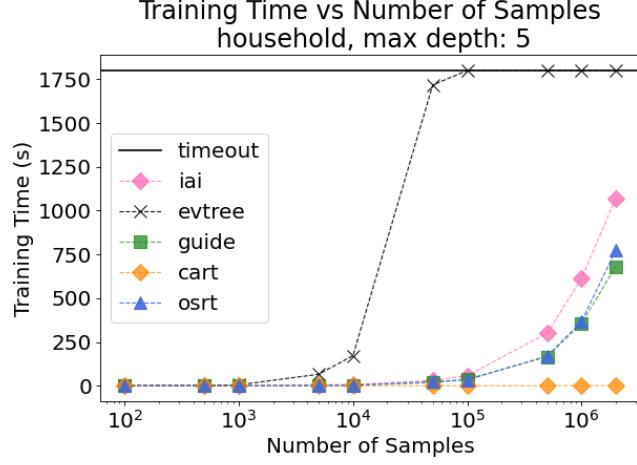


Figure 29: Training time of CART, GUIDE, IAI, Evtree and OSRT as a function of sample size on dataset: house-hold, $d = 5$, $\lambda = 0.035$. (30-minute time limit).

I Experiment: Scalability

Collection and Setup: We ran this experiment on the dataset *household* for CART, IAI, Evtree and OSRT. We subsampled 100, 500, 1000, 5000, 10,000, 50,000, 100,000, 500,000, 1000,000 samples from it to form 10 datasets (including the original dataset). The depth limit was set to 5 (for all methods) and regularization coefficient to 0.035 for IAI and OSRT and 0.2 for Evtree. We set the time limit of each run to 30 minutes in this experiment.

Results: Figure 29 shows that the scalability of our method is significantly better than Evtree, and it generally performs better than IAI. Evtree timed out when the sample size was larger than 50,000. Figure 30 is zoomed in to the regime of low training times, showing that our method is faster than all but CART when sample size is less than 10,000.

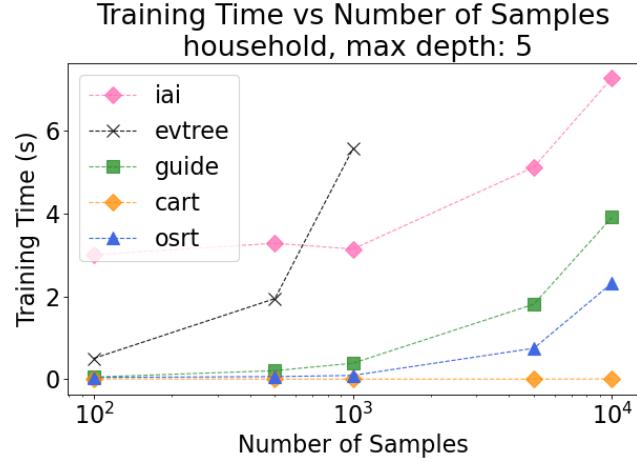


Figure 30: (Zoomed in) Training time of CART, GUIDE, IAI, Evtree and OSRT as a function of sample size on dataset: house-hold, $d = 5$, $\lambda = 0.035$. 30-minute time limit; the plot omits evtree results after 5000 samples (65.69 seconds for 5000 samples, 172.143 seconds for 10,000 samples, 1718.138 seconds for 50,000 samples, time out for the rest).

J Experiment: Ablation

J.1 Value of k-Means Lower Bound

We explored how much our new *k-Means* lower bound contributes to speeding up the optimization. Recall that our method reaches optimality when the current best objective score (upper bound) converges with the objective lower bound.

Collection and Setup: We ran this experiment on 5 datasets (*airquality*, *enb-cool*, *enb-heat*, *sync*, *yacht*) for variations of OSRT. We set depth limit to 6 and regularization coefficient to 0.005 for all datasets. For each dataset, we ran OSRT twice,

once using the *k-Means lower bound* and once using the *equivalent point lower bound*. We set the time limit to 30 minutes in this experiment.

Calculations: We recorded the elapsed time, iterations, and size of dependency graph when each run of OSRT converged or timed out.

Results: Figure 31 shows that our novel *k-Means lower bound* is significantly faster than the *equivalent points lower bound*. We noticed substantially better running time when the k-Means lower bound is much tighter than the equivalent points lower bound. It typically reduces runtime by a factor of two and sometimes more. The iteration number and graph size in our optimization framework were also reduced when using the k-Means lower bound, which means less memory was used.

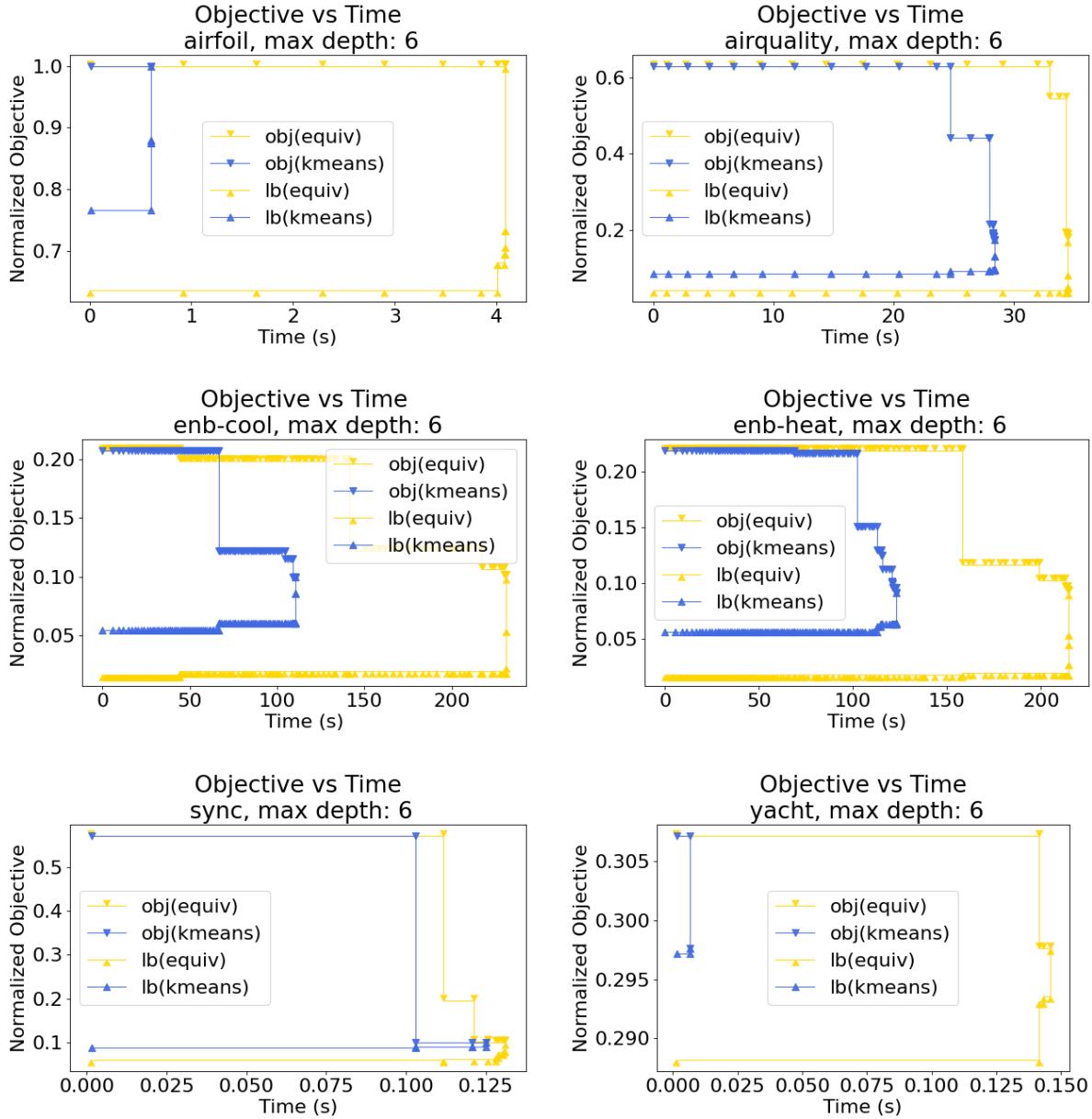


Figure 31: Convergence Time of OSRT variations: we verified *k-Means lower bound* has fewer iterations to completion than the *equivalent points lower bound* on Datasets *sync*, *yacht*.

J.2 Depth Constraint

Recall that our method can optimize the objective score *without depth constraints* while other methods cannot. Adding a depth constraint significantly reduces the search space, so being able to work without a depth constraint, OSRT can solve *much harder* problems than other methods. For most datasets we used in previous sections, a depth constraint substantially reduced OSRT searching time; but we also observed cases where, given a relatively small regularization coefficient and a large depth limit on some datasets (e.g., *sync*, *airquality*), OSRT took longer to reach optimality compared to optimizing with the regularization penalty only. For the *airquality* dataset, we used $\lambda = 0.001$ and ran OSRT with depth limit 7 and with no depth limit: OSRT without the depth limit took 115 seconds to reach optimality while the other run took 203 seconds. This is because the hard depth constraint prevents the algorithm from re-using the bounds it has calculated.

K Experiment: Execution Trace

Recall that OSRT keeps tracking the lower and upper bound of root problem objective, and optimal trees are found when two bounds converge. **Collection and Setup:** We ran this experiment on 4 datasets (*airfoil*, *airquality*, *optical*, *insurance*) only for OSRT. We used two configurations: $(d = 4, \lambda = 0.05)$ and $(d = 7, \lambda = 0.001)$. For each combination of dataset and configuration, we ran OSRT and recorded the lower and upper bounds of the objective score during the algorithm execution. We set the time limit as 30 minutes in this experiment.

Results: Figure 32 shows the optimality gap of the objective score during execution of our method. If our method does not reach optimality (e.g., the *optical* dataset) before the time limit, we can still gain insight by examining the difference between the current best objective and the current lower bound while IAI and Evtree do not provide such information.

L Optimal Regression Trees

We now visually compare the optimal trees found by OSRT and trees. Figure 33 contains two 6-leaf trees generated by OSRT and Evtree respectively. The OSRT tree has training mean squared error (MSE) of 247.50, $R^2 : 77.44\%$ while Evtree has training MSE of 324.61, $R^2 : 70.41\%$. OSRT explains 7% more training data variance than Evtree with the same number of leaves. Figures 34 and 35 show two 13-leaf trees generated by OSRT and IAI respectively. The OSRT tree has training MSE of 127.46, $R^2 : 88.38\%$ while Evtree has training MSE of 156.59, $R^2 : 85.71\%$. OSRT explains 2.66% more training data variance than Evtree with the same number of leaves. Figures 36 and 37 show two 10-leaf trees generated by OSRT, IAI and Evtree (IAI and Evtree found the same tree). The OSRT tree has training MSE of 0.42838, $R^2 : 82.28\%$ while IAI and Evtree have training MSE of 0.45055, $R^2 : 81.36\%$. We noticed that these two trees have some identical leaves.

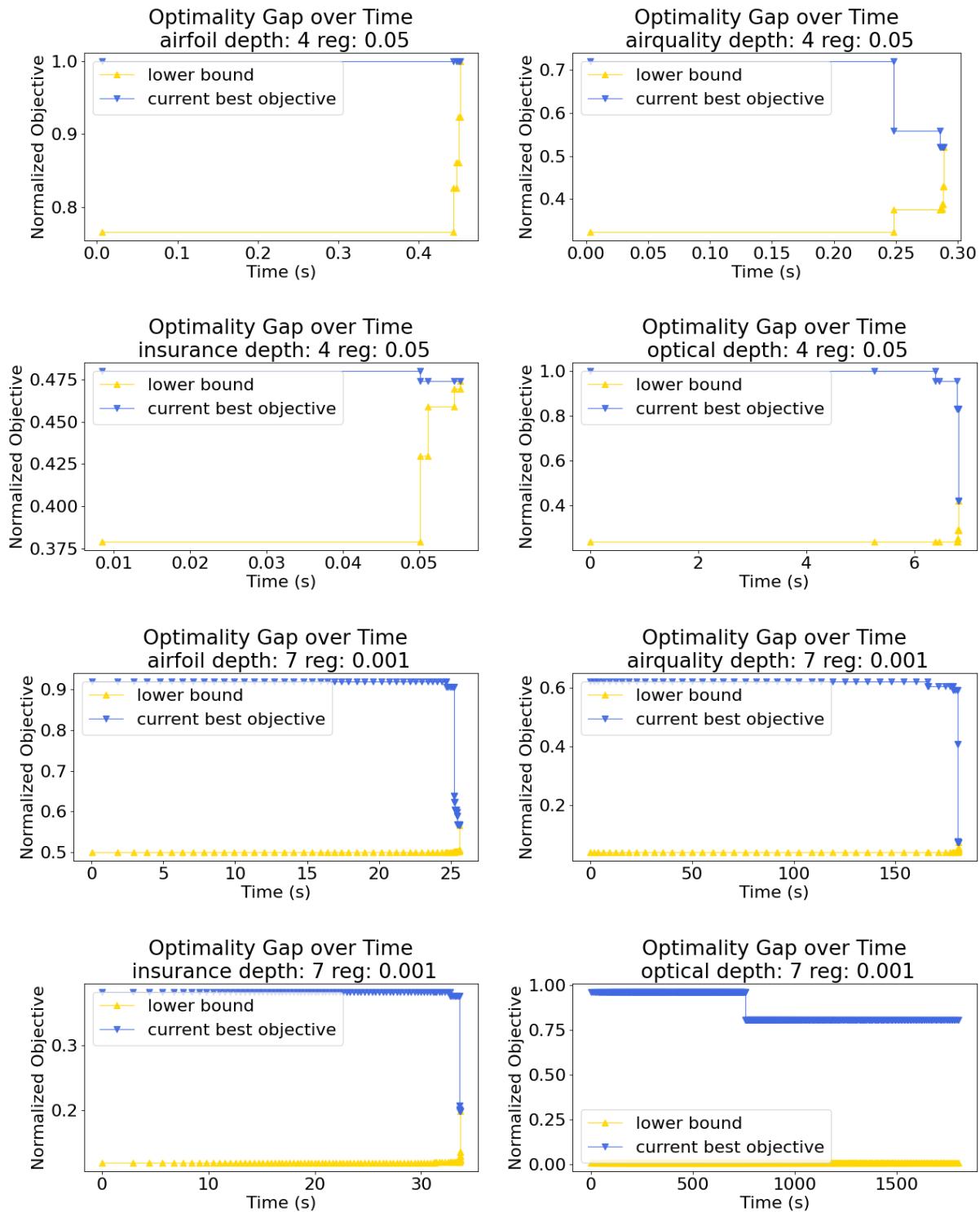
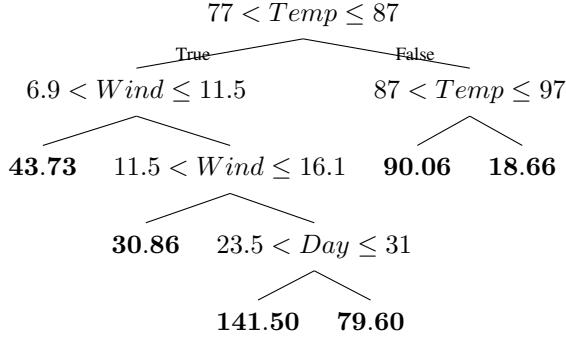
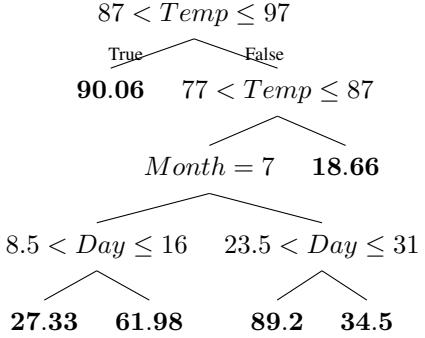


Figure 32: Execution Trace of OSRT over time, timed out on dataset *optical* with small regularization.



(a) OSRT, Training loss: 247.5, R^2 : 77.44%



(b) Evtree, Training loss: 324.609, R^2 : 70.41%

Figure 33: Regression trees produced by OSRT and Evtree for **airquality** dataset with 6 leaves.

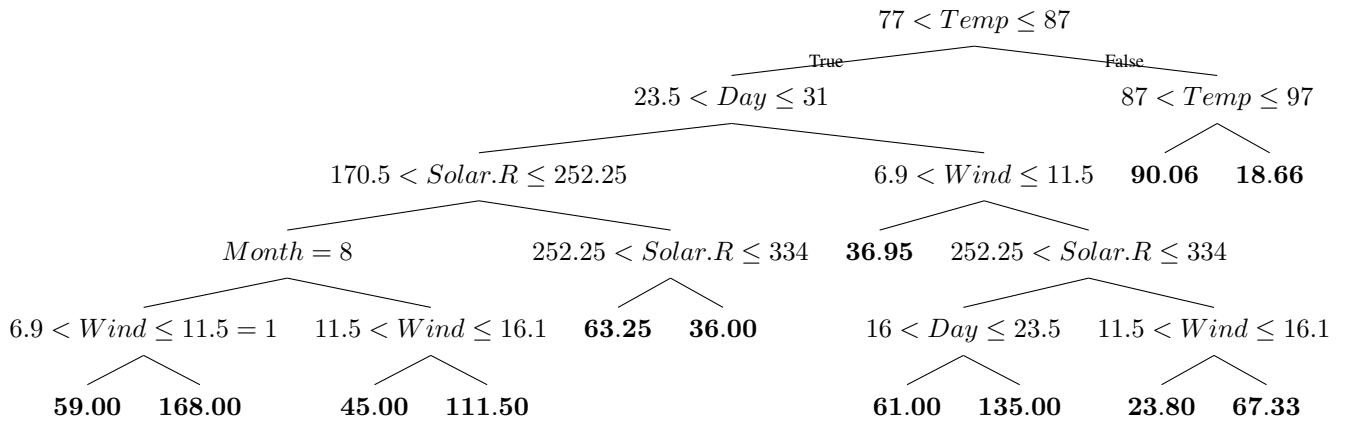


Figure 34: Optimal regression tree produced by OSRT for **airquality** dataset with 13 leaves. Training loss: 127.46, R^2 : 88.38%

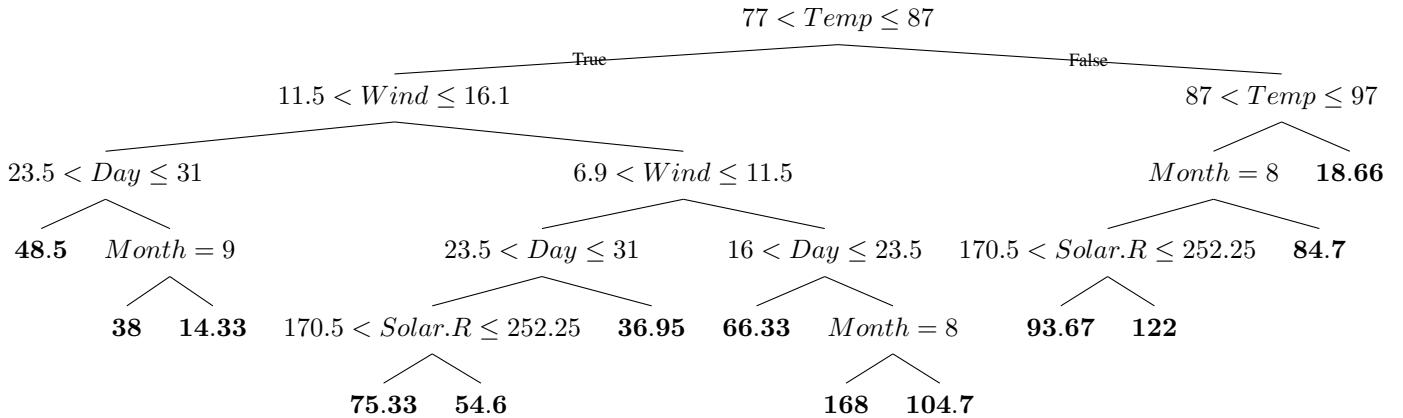


Figure 35: Sub-optimal tree produced by IAI for **airquality** dataset with 13 leaves. Training loss: 156.59, R^2 : 85.72%

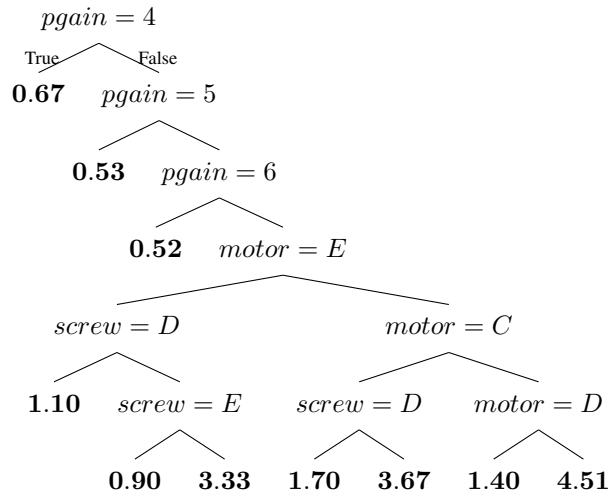


Figure 36: Optimal regression tree produced by OSRT for **servo** dataset with 10 leaves. Training loss: 0.42838, $R^2 : 82.28\%$

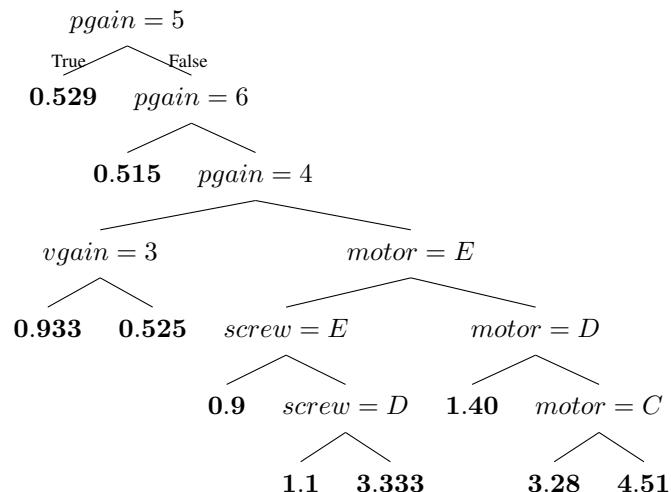


Figure 37: Sub-optimal tree produced by IAI and Evtree for **servo** dataset with 10 leaves. Training loss: 0.45055, $R^2 : 81.36\%$