Optimal Regression Trees via Dynamic Programming

Optimization techniques for learning Regression Trees

Mim van den Bos[1]

Supervisor(s): Dr. E. Demirović[1]**, Ir. J.G.M. van der Linden**[1]

[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 20, 2023

Name of the student: Mim van den Bos
Final project course: CSE3000 Research Project
Thesis committee: Emir Demirović, Koos van der Linden, Burcu Özkan

Abstract

Decision trees make decisions in a way interpretable to humans, this is important when machines are increasingly used to aid in making high-stakes and socially sensitive decisions. While heuristics have been used for a long time to find decision trees with reasonable accuracy, recent approaches find fully optimal trees. Due to the computational hardness of finding fully optimal decision trees, it is only practically possible to find shallow trees for a limited dataset size. However, continuous algorithmic improvements keep pushing the scale of feasible solutions. Dynamic Programming approaches promise to find scalable optimal decision trees but need to be adapted for different objectives, such as regression. We combine and adapt the algorithmic techniques of two Dynamic Programming methods, creating a new method that improves the scalability of optimal regression trees. This new method often achieves an order of magnitude speed improvement over a previous state-of-the-art method.

1 Introduction

Decision trees are a machine learning (ML) model that is naturally interpretable by humans. It is possible to follow the branches of the tree to the leaf that makes the decision. Interpretable models are important when making decisions in high-stakes areas such as healthcare and criminal justice, and unlike popular belief are not necessarily worse than black-box models that cannot be fully explained [1].

Classically, decision trees are constructed greedily by iteratively finding branches, based on a heuristic metric such as gini impurity or information gain [2]. A heuristic optimizes the metric locally when finding the next branch but does not optimize a metric globally for the whole tree.

A different way of finding a decision tree is by optimizing a global metric, thereby creating an optimal decision tree. Optimal decision trees fit the dataset better and generalize better to new data [3]. Optimality can also be especially important to guarantee fairness over the data [4]. Finding an optimal decision tree is NP-hard [5], but with novel methods to optimize this process and advances in computing power, it becomes possible for increasingly large datasets. The use of larger datasets allows the application of optimal decision trees to a wider range of problems.

We can use a decision tree to assign a numerical value to an instance of a problem. In that case, we speak of a regression problem. A global metric a decision tree can optimize for regression is the Mean Squared Error (MSE). When the MSE is optimal for a decision tree it is an *optimal regression tree*. This paper focuses on finding these. A different problem that we can solve with decision trees is classification. A metric we can use for this is the number of misclassifications, and when it is optimal they are *optimal classification trees*.

MurTree [6] is a recent algorithm showing optimal classification trees can be found faster using dynamic programming, caching, and counting. MurTree is a scalable approach for binary classification trees but only discusses how to generalize this approach to other models of decision trees like multi-classification, regression, anytime behavior, and more. They do not make these generalizations and do not show that these generalizations would perform as well as or better than the state-of-the-art in those areas.

Van der Linden et al. [7] generalize the optimization techniques presented in MurTree by providing a framework they call STreeD. This framework can optimize any *separable* objective. An objective is separable when that objective can be optimized for subtrees independent of the whole tree. They present necessary and sufficient conditions to use a cost function in the framework. Furthermore, they provide an implementation for several objectives and compare these to other methods in that domain. However, while they do provide a reference implementation of the MSE cost function, they do not explicitly show that it is separable, and they do not show how their method compares to the state-of-the-art in regression trees.

In this paper, we show that regression is a separable objective, introduce algorithmic techniques taking advantage of the regression problem, implement regression and algorithmic techniques from [8] in the STreeD framework, and compare the implementation to the Optimal Sparse Regression Trees (OSRT) method [8]. This comparison shows that our method often finds optimal regression trees an order of magnitude faster than [8].

The remaining part of this paper is structured as follows: First, sections two to four define the optimal regression tree problem, consider related work, and define terminology. Then, in the fifth and sixth sections, the main contribution of this paper is laid out and the results of experiments are examined. In the seventh section, there is a reflection on the reproducibility of this research. And finally, the paper is concluded in the eighth section.

2 Problem definition

Decision trees are a popular machine-learning model for regression and classification problems. A decision tree can be seen as a function where the input is a set of features, and the output is the label of the leaf node obtained by following the branches of the tree. The quality of a decision tree can be determined by comparing the input and output of the tree to a large dataset of known inputs and outputs. This paper considers regression trees where branching nodes split on a single binary feature, as in [6][7][8], and where the leaf nodes have some real number as their label, as in [8][3][9]. Note that binary decisions of non-binary features can be represented by preprocessing the data. A regression tree is *optimal* when the Mean Squared Error (MSE), combined with some regularization term that favors simple trees, cannot be lower for any other tree.

The problem of finding an optimal regression tree is as follows. Given a maximum tree depth and a dataset $\mathcal{D}$ of instances (x, k) with x some binary vector of features and its label k. An optimal regression tree is one where the result of the following expression is minimal:

$$\frac{1}{|\mathcal{D}|}\sum_{(x,k)\in\mathcal{D}}(x-l(u))^2+\lambda N \tag{1}$$

Where $l(u)$ is the label of the leaf node that classifies the instance, N is a measure of complexity for the tree such as the number of nodes, and λ is a scalar parameter describing the importance of the simplicity of the tree. Note that the division by the size of the dataset may be omitted since it is the same for all trees. In section 4 the definition of a decision tree and its notation is further formalized.

3 Related work

Because finding optimal decision trees is NP-hard [5], heuristic methods such as CART [2] (1984) have been the state-of-the-art method to generate classification and regression trees for a long time. Many papers still compare their out-of-sample performance against trees generated using CART [8][3][9]. However, it provides no optimality guarantees.

More recently, following advances in algorithmic optimizations and hardware, methods that guarantee optimality can be practically computed for limited depth and dataset size. Many of these methods focus on classification trees and not regression trees [6][10][11], but are often followed by an adaptation to optimize other objectives like regression [7][8][3].

Bertsimas et al. [3] expand their Mixed Integer Programming (MIP) model for classification trees [11] to generate optimal regression trees. They optimize directly for real-valued features and optionally branch on multiple features at once. Their method is applied to a practical problem and they claim a 2% increase in out-of-sample performance compared to CART [2]. However, they only consider trees up to depth two.

Verwer and Zhang [9] introduce a MIP model for classification and regression trees that scales better for larger datasets than their previous method [12], and also show that for shallow depth the out-of-sample accuracy is higher compared to CART. However, their runtime is on the order of minutes for problems that recent dynamic programming methods can solve in seconds.

Van der Linden et al. [7] generalize the MurTree [6] dynamic programming method to solve optimal decision trees for any objective that satisfies a set of requirements. Regression satisfies these requirements as described later in this paper. They show that the scalability of these algorithms is good for some objectives, but do not compare to the state-of-the-art in optimal regression trees. They argue that dynamic programming works well because it can take advantage of the structure of the decision tree, in particular, that subtrees can be found independently, and that the order in which features are branched on does not matter.

Zhang et al. [8] adapted GOSDT [10] to optimize regression trees and discovered a novel lower bound using two facts. First is that instances with equivalent features cannot be classified in different leaf nodes, this gives a minimum error for a leaf that includes it. Second, if there is a tree with k leaf nodes, there is no lower error possible than if each leaf node would include those instances whose labels are nearest to each other. Summing the minimum error obtained by both gives the k-Means Equivalent Points Bound, for which they present an efficient algorithm using the one-dimensional k-Means algorithm from [13] and [14]. However, they do not make use of the optimizations presented in [6], such as the specialized depth two algorithm, and the similarity-based lower bound.

Finally, we briefly note that recent works consider adjustments to regression trees intending to improve out-of-sample accuracy, such as linear or polynomial functions in the leaves [15] or making a decision by considering the sum of the leaves weighted by the probability an instance belongs to it [16]. As these are different problems they are not further considered in this paper.

4 Preliminaries

This section introduces the notation adapted from [7] to define the problem and shows how to use that notation to formulate a dynamic programming (DP) solution to the optimal regression tree problem.

Let $\mathcal{F}$ be a set of features and $\mathcal{D}$ be a dataset of instances (x,k), where $x\in\{0,1\}^{|\mathcal{F}|}$, where a one denotes the binary feature being present and zero not, and $k\in\mathbb{R}$. For a feature $f\in\mathcal{F}$ and instance (x,k) let x_f if f is present in x, and $x_{\bar{f}}$ if not. Also, let $\mathcal{D}_f=\{(x,k)\in\mathcal{D}\mid x_f\}$.

Let $\tau=(B,L,b,l)$ be a binary tree with B and L the set of branching and leaf nodes respectively, and $b:B\to\mathcal{F}$ and $l:L\to\mathbb{R}$ the assignment of nodes to features for branching nodes and labels for leaf nodes respectively. The set of all nodes $B\cup L$ are the decision variables of the tree. A branching node u has two children u^- and u^+ representing a negative and positive outcome of a feature test respectively.

Optimizing for the MSE plus a regularization term, define the cost function $g:(\mathcal{D}\times(\mathcal{F}\cup\mathbb{R}))\to\mathbb{R}$ as $g(\mathcal{D},\hat{k})=\sum_{(x,k)\in\mathcal{D}}(k-\hat{k})^2$ for making a leaf node decision and $g(\mathcal{D},f)=\lambda$ for making a branching decision, where λ is a parameter for penalizing more complex trees. The total cost C of a regression tree can then be computed as follows:

$$C(\mathcal{D},u)=\begin{cases}g(\mathcal{D},l(u)) & \text{if } u\in L\\ C(\mathcal{D}_{b(u)},u^+)+C(\mathcal{D}_{\overline{b(u)}},u^-)+\lambda & \text{if } u\in B\end{cases} \tag{2}$$

The minimum cost can be computed more efficiently with DP by caching solutions to subproblems. Adapting [7], the minimum MSE cost for a given maximum depth d can be computed with the following DP formulation:

$$T(\mathcal{D},d)=\begin{cases}\sum_{(x,k)\in\mathcal{D}}(k-\bar{k})^2 & \text{if } d=0\\ \min_{f\in\mathcal{F}}\{T(\mathcal{D}_f,d-1)+ T(\mathcal{D}_{\bar{f}},d-1)+\lambda\} & \text{if } d>0\end{cases} \tag{3}$$

Where $\bar{k}$ denotes the mean of the labels in $\mathcal{D}$.

When a lower bound $LB(\mathcal{D},d)\le T(\mathcal{D},d)$ for the cost of a subtree is known, it can be used to prune a part of the search space. We use the similarity-based lower bound as described in detail in [6] and generalized in [7]. The similarity bound uses a lower bound $LB(\mathcal{D}, d)$ to compute a lower bound $LB(\mathcal{D}', d)$ by comparing $\mathcal{D}$ and $\mathcal{D}'$ and looking at the per-instance contribution to the cost of the subtree. Any instances that are present in both datasets do not change the error. Any instances that are present in $\mathcal{D}'$ but not in $\mathcal{D}$ can only worsen the error, and therefore do not change the lower bound. And finally, any instance that was in $\mathcal{D}$ but is not in $\mathcal{D}'$ can potentially improve the error, and therefore decreases the lower bound by the contribution of that instance. Finding the exact contribution per instance is only possible by computing the cost of the tree, but an upper bound can be formulated depending on the objective. The conditions needed for an optimization task to use a similarity lower bound are that it has a subtraction operator, does not have constraints, does not depend on parent nodes' branching decisions and it is per-instance additive [7].

Finally, we introduce the notion of an additive objective, as defined by [17], and repeat the claim from [7] that any objective that is additive is separable, and can therefore be used in the STreeD framework. An objective O for a decision tree τ is additive when it can be written as follows:

$$O(\mathcal{D}, \tau) = \sum_{u \in L} f_l(\mathcal{D}_u, u) + \sum_{u \in B} f_b(\mathcal{D}_u, u) \tag{4}$$

5 Main contribution

First, we show that the regression objective satisfies the constraint for separable objectives shown in [7]. Then, we show that the specialized algorithm for trees of depth two presented in [6] can be used because the MSE is additive per-instance [7]. Finally, we present a bound on the worst error that a single instance can introduce. This bound can be used with the similarity-based lower bound presented in [6].

5.1 Regression as a separable objective

We show that regression is an additive objective to prove that the dynamic programming formulation in section 4 finds an optimal regression tree. Recall that section 4 defines an additive objective as one that can be written as:

$$O(\mathcal{D}, \tau) = \sum_{u \in L} f_l(\mathcal{D}_u, u) + \sum_{u \in B} f_b(\mathcal{D}_u, u) \tag{5}$$

Now, letting $f_l(\mathcal{D}, u) = g(\mathcal{D}, l(u))$ and $f_b(\mathcal{D}, u) = \lambda$ allows us to express the regression objective in an additive form. Therefore, the regression objective is additive and separable, and we can use it in the STreeD framework.

5.2 Specialized algorithm for trees of depth two

In [6] a specialized algorithm is used to compute trees of maximum depth two with a frequency counting approach. When there is some way to deconstruct the cost into the contribution per instance, we do not need to compute all combinations explicitly. For example, when we know the total cost as a sum of all the per-instance costs, and the cost for all instances where a single feature is present, we can subtract the second from the first to get the cost for all instances where that feature is not present. Similar formulations can be made for all combinations of solutions with depth two, only needing to compute the cost for all pairs of features where these features are both present. Finally, we check all combinations to find the best solutions for one, two, and three branching nodes. These are cached so that similar subtrees can reuse the solution. For full details, we refer to the original paper.

To use this specialized algorithm, [7] argues that a problem must be deconstructed so that the depth two solver can pre-compute the per-instance costs. We cannot use the MSE directly when the label to be assigned is not yet known. However, we can deconstruct it in the following way:

$$\begin{aligned}
C(\mathcal{D}) &= \sum_{(x,k) \in \mathcal{D}} (k - \bar{k})^2 && (6)\\
&= \sum_{(x,k) \in \mathcal{D}} (k^2 - 2k\bar{k} + \bar{k}^2) && (7)\\
&= \sum_{(x,k) \in \mathcal{D}} k^2 - 2\bar{k} \sum_{(x,k) \in \mathcal{D}} k + |\mathcal{D}|\bar{k}^2 && (8)\\
&= \sum_{(x,k) \in \mathcal{D}} k^2 - 2|\mathcal{D}|\bar{k}\bar{k} + |\mathcal{D}|\bar{k}^2 && (9)\\
&= \sum_{(x,k) \in \mathcal{D}} k^2 - |\mathcal{D}|\bar{k}^2 && (10)\\
&= \sum_{(x,k) \in \mathcal{D}} k^2 - \frac{(\sum_{(x,k) \in \mathcal{D}} k)^2}{|\mathcal{D}|} && (11)
\end{aligned}$$

The per-instance cost is then the three-tuple $(k, k^2, 1)$, and the combining operator element-wise addition. We write the summed per-instance cost C_i for a dataset as $C_i(\mathcal{D}) = (\sum_{(x,k) \in \mathcal{D}} k, \sum_{(x,k) \in \mathcal{D}} k^2, |\mathcal{D}|)$, and the function to go from a summed per-instance cost solution $(\sum k, \sum k^2, N)$ to the MSE cost as $C_{\text{MSE}}(\sum k, \sum k^2, x) = \sum k^2 - \frac{(\sum k)^2}{N}$, as derived in equation 11. For example, to compute the MSE cost $C(\mathcal{D}_{\bar{f}})$ for some feature f, we first compute the summed per-instance cost for the whole dataset $C_i(\mathcal{D})$ and when the feature is present $C_i(\mathcal{D}_f)$, and then element-wise subtract the second from the first to get the MSE cost for when the feature is not present:

$$C(\mathcal{D}_{\bar{f}}) = C_{\text{MSE}}(C_i(\mathcal{D}) - C_i(\mathcal{D}_f)) \tag{12}$$

As such, we can compute $C(\mathcal{D}_{\bar{f}})$ quickly from the summed per-instance cost of other datasets, and there is no need to loop over all the instances again.

5.3 Similarity bound

As mentioned in section 4 and described in detail in [6], we can find a lower bound on the error by comparing the dataset to previously computed datasets and subtracting the worst contribution the removed instances could have had on the cost. We show that regression satisfies the requirements to use a similarity bound and present a per-instance upper bound specific to the regression objective.

The conditions needed for an optimization task to use a similarity lower bound are that it has a subtraction operator, does not have constraints, does not depend on parent nodes' branching decisions and it is per-instance additive [7]. For regression, the subtraction operator is $-$ and it does not have constraints. While the task has a branching component in the cost, it does not depend on which branching decisions were made. Finally, regression is directly per-instance additive as we write the cost function as a sum over the instances.

For regression, we find an upper bound on the contribution to the error by comparing the instance to the two extremes of the dataset. The label for any given leaf node cannot be above the instance with the highest label nor below the instance with the lowest label. If it were, then the MSE would be higher for all instances. This means for any leaf node u: $\min_{(x,k)\in\mathcal{D}}(k) <= l(u) <= \max_{(x,k)\in\mathcal{D}}(k)$. This directly gives a bound on the maximum error that any single instance (x', k') can have in a leaf:

$$\max\{(k' - \min_{(x,k)\in\mathcal{D}}(k))^2, (k' - \max_{(x,k)\in\mathcal{D}}(k))^2\} \tag{13}$$

Any other label within those bounds would be closer to the label of the instance. For simplicity, we compute the minimum and maximum labels over the whole dataset, but a tighter bound could be found by only using the instances considered for that subtree.

Table 1: The datasets used to run the experiments, $|\mathcal{D}|$ is the number of instances, and $|\mathcal{F}|$ the number of binary features

Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{F}\rvert$
airfoil	1503	17
airquality	111	17
enb-cool	768	27
enb-heat	768	27
household	2049280	15
insurance	1338	48
optical	640	29
real-estate	414	18
seoul-bike	8760	32
servo	167	15
sync	557	12
yacht	308	35

6 Experimental Setup and Results

We perform two experiments to show the effectiveness of the method presented in this paper. First, to show its scalability, we measure the runtime of the algorithm, both for increasing depth and increasing size of the dataset, and compare it to OSRT [8]. It shows that, in most cases, our method with all lower bounds enabled performs best. Then, to show the pruning potential of various lower bounds, we measure the number of calls to the depth two solver and compare it to a baseline where the bounds are disabled. This shows that all lower bounds combined have a larger pruning potential than using a single lower bound, and indicates that improving the performance of calculating the bounds will increase the performance of the algorithm as a whole.

We use the same twelve datasets and binarization for our experiments as used in the experiments in [8]. These are listed in table 1. All experiments were run on the DelftBlue [18] supercomputer using an Intel XEON E5-6248R 24C 3.0GHz. The experiments used a timeout of 100 seconds with a memory limit of 8GB of RAM, except for the experiment with increasing dataset size, which used a timeout of 30 minutes and a memory limit of 100GB. We repeated all experiments three times for an error margin, and used multiple regularization terms λ for each dataset and depth. The regularization term λ, which gives a penalty for each branching node in the tree, is normalized by the variance and size of the dataset: $\lambda = W \sum_{(x,k)\in\mathcal{D}} (k - \bar{k})^2$ with $W \in \{0.0001, 0.001, 0.005, 0.01, 0.1\}$.

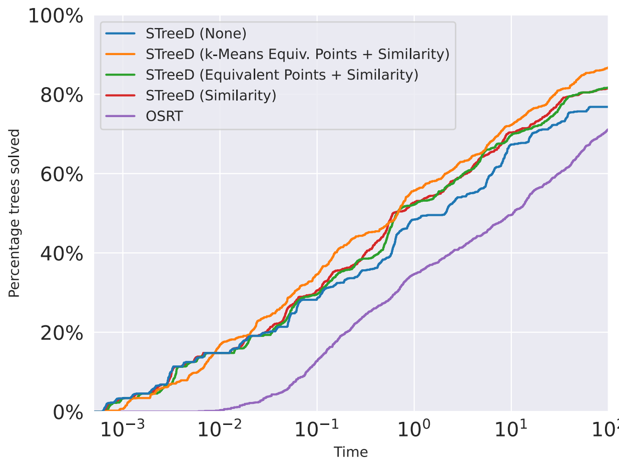

Figure 1: Percentage of trees computed after a certain time, for all datasets except household

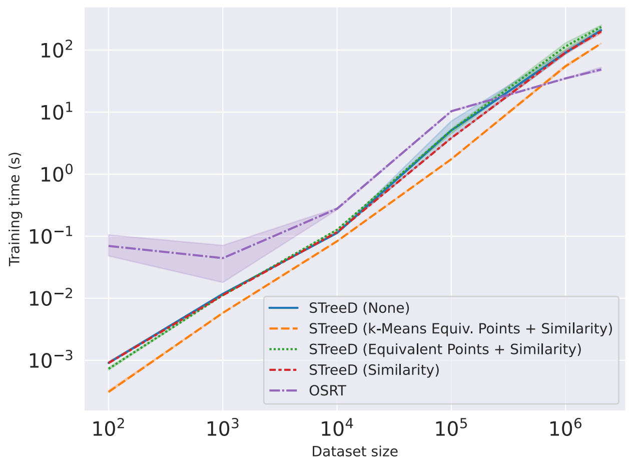

Figure 2: Training time for increasing subset size of dataset household, with d=5 and W=0.035, and a confidence interval of 95%

6.1 Scalability

For scalability, we performed two experiments. First, to show the scalability of the maximum depth, we use all datasets except household for finding trees of varying depth. Then, we use the household dataset to show the scalability in the number of instances.

For the depth scalability experiment, we used a maximum depth of three to ten inclusive to find trees. This experiment focuses on showing the potential scalability of the algorithm, a very high depth of decision tree is often not desired, as it becomes less interpretable. Figure 1 shows the percentage of trees solved within the amount of time on the x-axis. Note that the x-axis has a logarithmic scale. All methods follow a roughly logarithmic trend in the number of trees solved within some amount of time, which is expected for an NP-hard problem. However, as seen from the graph, STreeD with all bounds enabled is about an order of magnitude of time ahead of OSRT.

Figure 2 shows scalability in the number of instances used to find the optimal regression tree. This experiment uses splits of the household dataset. We ran experiments for subsets of $\mathcal{D}$ with the first 10^i instances for every $i >= 2$ until it contains the whole dataset. For this experiment, the maximum depth is five, and the regularization weight is 0.035. The figure shows that STreeD performs well for subsets containing up to about 10^5 instances, and OSRT outperforms STreeD for any subset larger than this. We have not investigated the cause for this due to time constraints and suggest it for future work.

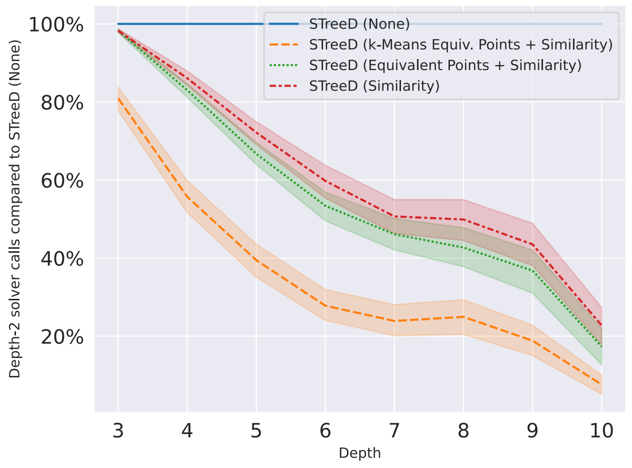

Figure 3: Difference in depth-2 solver calls for different lower bounds for all datasets except household, with a confidence interval of 95%

6.2 Pruning potential

Figure 3 shows the ratio of calls to the depth two solver made by each lower bound compared to the baseline of STreeD without any lower bounds. A lower percentage of calls indicates a higher pruning potential. This shows the potential of the lower bounds without binding them to the performance of this implementation.

The k-Means Equivalent Points lower bound generally has the highest amount of pruning. With very high depth we see its performance fall off compared to other methods. An explanation for this is that the implementation has a hard limit of $k = 50$, which means the lower bound becomes relatively less performant after exceeding that threshold.

The performance of the equivalent points bound depends heavily on the presence of instances with equivalent features but different labels. While the bound does provide an improvement over using only the similarity bound, its pruning potential is far lesser when not combined with the k-Means bound.

Finally, we observe that while the lower bounds in some cases have only 10% of the depth two calls, the time after which the approach without bounds catches up to the percentage of trees solved is not an order of ten removed in figure 1. This means improving the lower bound computation has the potential to make the algorithm faster.

7 Responsible Research

This paper adheres to the FAIR principles to reinforce the reusability of this research [19]. The four principles are listed below along with a description of how they were followed.

Findable The metadata and content of this work are discoverable in the TU Delft education repository, and the implementation of both the experiments[1] and adapted STreeD[2] is available publicly. Publishing this paper and code allows others to find them.

Accessible The repositories for both the paper and code require no authentication, and are freely available to anyone with an internet connection. This makes the research accessible to the general public.

Interoperable Both the datasets used in the paper and the results of the experiments use the common CSV file format. This allows others to use them in a system of their own choice. Furthermore, the main code is written in C++ and the experiment code is written in Python. Both of these are executable on many platforms.

Reusable The code repositories contain instructions to build and run the regression tree solver and experiment setup. The experiments can be run on a supercomputer like in this paper, but also on a regular computer albeit with limited size. This allows anyone to reproduce the results obtained in this paper.

By following these guidelines, we hope that the results presented in this paper are reproducible and more broadly usable.

8 Conclusions and Future Work

The paper aims to scale the maximum feasible depth and dataset size for optimal regression trees. We combine previous optimal decision tree methods, [7] and [8], and introduce a novel upper bound on the contribution of an instance,

[1]https://github.com/mimvdb/regression-murtree

[2]The source code will be made available when the source code for STreeD is made public.

used with the similarity lower bound from [6]. We show with extensive experiments that this method is often an order of magnitude faster than [8], a state-of-the-art method of finding optimal regression trees. Reducing the time needed to find optimal regression trees increases the depth and dataset size that can feasibly be used, thereby increasing the scalability of the method.

This new method is often an order of magnitude faster than [8], but there is room for improvement in future work. In our experiment, OSRT [8] performs better than our method for dataset sizes one million and above. Future work can investigate the cause of this performance difference and propose a way to improve it. In addition, we suggest three optimizations for future work. First, when computing the bound on the per-instance contribution, we can use the minimum and maximum of only the currently relevant subset of instances instead of the minimum and maximum of the whole dataset. Secondly, adding the k-Means Equivalent Points bound to the cache used for similarity lower bound computation would allow it to calculate a tighter bound when computing a bound for a similar dataset. Finally, two versions of the dataset could be maintained, one sorted by features to compute equivalent points, and one sorted by the label to compute k-Means in $O(k|\mathcal{D}|)$ time. This might be an improvement to the current approach that sorts the data in $O(|\mathcal{D}| \log |\mathcal{D}|)$ time for every k-Means lower bound computation.

References

[1] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, May 2019. DOI: 10.1038/s42256-019-0048-x.

[2] L. Breiman, J. Friedman, C. J. Stone, and R. Olshen, *Classification and Regression Trees* (Wadsworth). Chapman and Hall/CRC, 1984.

[3] D. Bertsimas, J. Dunn, and A. Paschalidis, "Regression and classification using optimal decision trees," in *2017 IEEE MIT Undergraduate Research Technology Conference (URTC)*, 2017, pp. 1–4. DOI: 10.1109/URTC.2017.8284195.

[4] S. Aghaei, M. Azizi, and P. Vayanos, "Learning Optimal and Fair Decision Trees for Non-Discriminative Decision-Making," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 1418–1426, Jul. 2019. DOI: 10.1609/aaai.v33i01.33011418.

[5] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is np-complete," *Information Processing Letters*, vol. 5, no. 1, pp. 15–17, 1976. DOI: 10.1016/0020-0190(76)90095-8.

[6] E. Demirović, A. Lukina, E. Hebrard, *et al.*, "Murtree: Optimal decision trees via dynamic programming and search," *Journal of Machine Learning Research*, vol. 23, no. 26, pp. 1–47, 2022.

[7] J. G. M. van der Linden, M. M. de Weerdt, and E. Demirović, *Optimal decision trees for separable objectives: Pushing the limits of dynamic programming*, 2023. arXiv: 2305.19706 [cs.LG].

[8] R. Zhang, R. Xin, M. Seltzer, and C. Rudin, *Optimal sparse regression trees*, 2023. arXiv: 2211.14980 [cs.LG].

[9] S. Verwer and Y. Zhang, "Learning optimal classification trees using a binary linear program formulation," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 1625–1632, Jul. 2019. DOI: 10.1609/aaai.v33i01.33011624.

[10] J. Lin, C. Zhong, D. Hu, C. Rudin, and M. Seltzer, "Generalized and scalable optimal sparse decision trees," in *Proceedings of the 37th International Conference on Machine Learning*, ser. ICML'20, JMLR.org, 2020.

[11] D. Bertsimas and J. Dunn, "Optimal classification trees," *Machine Learning*, vol. 106, no. 7, pp. 1039–1082, Jul. 2017. DOI: 10.1007/s10994-017-5633-9.

[12] S. Verwer and Y. Zhang, "Learning decision trees with flexible constraints and objectives using integer optimization," in *Integration of AI and OR Techniques in Constraint Programming*, D. Salvagnin and M. Lombardi, Eds., Cham: Springer International Publishing, 2017, pp. 94–103, ISBN: 978-3-319-59776-8. DOI: 10.1007/978-3-319-59776-8_8.

[13] H. Wang and M. Song, "Ckmeans.1d.dp: Optimal k-means clustering in one dimension by dynamic programming," *The R Journal*, vol. 3, no. 2, pp. 29–33, 2011. DOI: 10.32614/RJ-2011-015.

[14] M. Song and H. Zhong, "Efficient weighted univariate clustering maps outstanding dysregulated genomic zones in human cancers.," *Bioinformatics*, vol. 36, no. 20, pp. 5027–5036, 2020. DOI: 10.1093/bioinformatics/btaa613.

[15] D. Bertsimas, J. Dunn, and Y. Wang, "Near-optimal nonlinear regression trees," *Operations Research Letters*, vol. 49, no. 2, pp. 201–206, 2021, ISSN: 0167-6377. DOI: 10.1016/j.orl.2021.01.002.

[16] R. Blanquero, E. Carrizosa, C. Molero-Río, and D. R. Morales, "On sparse optimal regression trees," *European Journal of Operational Research*, vol. 299, no. 3, pp. 1045–1054, 2022, ISSN: 0377-2217. DOI: 10.1016/j.ejor.2021.12.022.

[17] S. Nijssen and E. Fromont, "Optimal constraint-based decision tree induction from itemset lattices," *Data Mining and Knowledge Discovery*, vol. 21, no. 1, pp. 9–51, Jul. 2010. DOI: 10.1007/s10618-010-0174-x.

[18] DHPC, *DelftBlue Supercomputer (Phase 1)*, https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase1, 2022.

[19] M. Wilkinson, M. Dumontier, I. J. Aalbersberg, *et al.*, "The FAIR Guiding Principles for scientific data management and stewardship," *Scientific Data*, vol. 3, no. 1, Mar. 2016. DOI: 10.1038/sdata.2016.18.