Finding Robust Optimal Regression Trees using exhaustive search

And why this is not trivial

Jesse Vleeschdraager[1]
Supervisors: Emir Demirović[1]**, Koos van der Linden**[1]**, Daniël Vos**[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 21, 2025

Name of the student: Jesse Vleeschdraager
Final project course: CSE3000 Research Project
Thesis committee: Emir Demirović, Koos van der Linden, Daniël Vos, Jasmijn Baaijens

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Abstract

Decision trees are accurate and interpretable models that can predict classes or values based on features of a data point, but they are vulnerable to small changes to the data that greatly affect the predictions. Previous work has resulted in methods that can find robust optimal classification trees or robust regression trees, but not robust optimal regression trees. Computing optimal decision trees is already NP-hard, and continuous prediction values make robust regression trees more complex than robust classification trees. We analyze in further detail why this is a difficult problem to solve. We introduce ForTree, a method that uses exhaustive search with pruning to find robust optimal regression trees out of a set of trees with limited prediction values. It evaluates all possible configurations of trees with this limitation and stores the tree with the highest adversarial accuracy. We also introduce the partial accuracy as a lower bound for pruning, which we use to speed up the runtime of ForTree. Our experiments show that ForTree achieves up to 1.23 times the adversarial accuracy of state-of-the-art robust regression tree methods, but that the runtime of ForTree is greatly sensitive to the amount of unique feature values in a dataset. We propose multiple approaches to find robust regression trees with higher adversarial accuracy or within less runtime in the future.

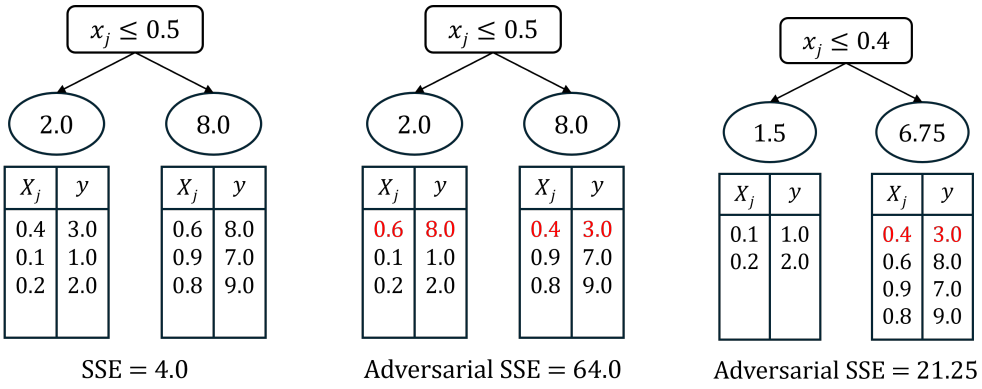
1 Introduction

Decision tree learning is a popular approach in machine learning because of its effectiveness in solving classification and regression problems and how interpretable the resulting models are for humans. Classification and regression trees respectively predict the class or value of data points by splitting based on the values of its features. This structure is easily interpretable and can be used to clearly explain predictions, unlike when working with black box methods. Figure 1a contains a small example of a regression tree.

Robustness is a specific problem within decision tree learning that has recently gotten more attention (Demirović et al., 2022; Van den Bos et al., 2024; Briţa et al., 2025). Because decision trees make splits based on certain thresholds, small perturbations to the input data can easily change the result of these splits and therefore have a big impact on the accuracy of the model as demonstrated in Figure 1b. These perturbations can come for example from adversarial attacks (Szegedy et al., 2014), where an adversary slightly changes the data to greatly increase the error, which is why the accuracy after these changes is denoted as adversarial accuracy. Robust decision trees aim to minimize the potential effect of such perturbations.

Several robust decision tree methods have been developed in recent years. This started with greedy methods such as one by Chen et al. (2019) and TREANT (Calzavara et al., 2020) that can find both robust classification and robust regression trees. A more recent method is ROCT (Vos & Verwer, 2022), which can find robust optimal classification trees. All these methods try to solve the min-max problem of minimizing some error while the adversary tries to maximize this error by perturbing the data points. Figure 1c shows the adversarial accuracy of a robust tree compared to that of a regular tree in Figure 1b.

However, most of this previous work on robust optimal decision trees has focused on classification trees instead of regression trees. Regression trees are more complicated to work with than classification because they predict continuous values instead of predicting a class. For regular regression trees, the optimal prediction value is easily found by taking the mean of all relevant values, but this mean is not necessarily optimal after perturbations to the data. Because of all the possible perturbations, the optimal prediction values cannot



(a) Optimal regression tree (b) Adversarial attack ($\epsilon = 0.2$) (c) Robust optimal regression tree

Figure 1: An example of an optimal regression tree, the effects of an adversarial attack, and an example of a robust optimal regression tree to minimize the effects of that adversarial attack. Each leaf node shows the data points that get mapped to it after splitting on feature X_j. The SSE is the sum of squared errors for y and the prediction values in the leaf nodes. Red data points have their value of X_j perturbed by at most ϵ to cause the data point to be mapped to another leaf node.

easily be calculated. This is why previous works have only resulted in greedy methods for robust regression trees that do not produce robust optimal regression trees.

We propose ForTree[1], a method to find robust optimal regression trees. ForTree uses exhaustive search to evaluate a wide set of regression trees and then takes the one with the highest adversarial accuracy. Exhaustive search is a simple yet effective approach for this problem, because it can evaluate all instances of a set and choose the best one. It is however not possible to exhaustively search all possible regression trees, because the continuous prediction values mean that there are infinitely many trees. To solve this, ForTree first makes a selection of prediction values that accurately reflect the target values in the training set, and then searches all possible trees limited to those prediction values. This approach lets us find the optimal tree out of the set of trees with the chosen prediction values, and by making a smart selection of prediction values, this can get close to a robust regression tree that is truly optimal. This approach is made feasible because ForTree uses pruning to ignore trees that can be proven to be not optimal before fully evaluating them. ForTree uses the regular accuracy as a lower bound to prune, and we introduce partial accuracy pruning, which calculates the accuracy while only considering a subset of the leaf nodes, and uses that as a lower bound to prune as well.

We compare the adversarial accuracy achieved by ForTree on multiple datasets to that achieved by the state-of-the-art greedy robust regression tree methods, the method by Chen et al. (2019) and TREANT (Calzavara et al., 2020). Our experiments show that ForTree achieves an adversarial accuracy that is at most 1.23 times as high as the adversarial accuracy of state-of-the-art greedy robust regression tree methods, but ForTree runs much slower for datasets with many unique feature values.

To summarise, ForTree is the first method to produce robust optimal regression trees for a given set of prediction values, which is done with exhaustive search and pruning. While ForTree does not find robust optimal regression trees out of the set of all possible regression

[1] https://gitlab.tudelft.nl/brp-25q4/fortree

trees, it does on average achieve higher adversarial accuracy than the previous state-of-the-art greedy robust regression tree methods. We discuss what causes finding robust optimal regression trees to be a difficult problem develop a method for, and we make multiple suggestions to potentially solve this problem or to get closer to solving it.

The remainder of the paper is structured as follows. Section 2 discusses what research has been done regarding decision trees and the progress that has been made in the recent years. Section 3 describes the notation that is used throughout this paper and defines the problem that ForTree aims to solve. Section 4 goes into detail about why finding robust optimal regression trees is a difficult problem. Then, Section 5 discusses the implementation of ForTree, our proposal to find robust optimal regression trees for a limited set of prediction values. Section 6 shows the evaluation of ForTree and what the results of these experiments mean. Section 7 describes our suggestions for how a better method could be developed. Section 8 shortly reflects on the ethical aspects on this research, and finally Section 9 concludes on this research and discusses possible future improvements on ForTree.

2 Related Work

This section goes over the previous research done related to greedy decision trees in Section 2.1, optimal decision trees in Section 2.2 and robust decision trees in Section 2.3.

2.1 Greedy decision trees

Because computing optimal decision trees is an NP-hard problem (Hyafil & Rivest, 1976), a common solution for computing decision trees is using greedy heuristics to determine locally optimal splits for partitioning the data. The first usage of these greedy heuristics was AID (Morgan & Sonquist, 1963), which uses the sum of squared errors (SSE) as its heuristic for regression trees. One of the currently most popular greedy models is CART (Breiman et al., 1984), which uses residual reduction for regression trees. These greedy models have short training times because they find the best split for each decision node and then set the prediction value of each leaf to the mean of the values that fall into that leaf, but this also results in overall less accurate trees (van der Linden et al., 2024).

2.2 Optimal decision trees

An approach to find trees with global optimality (instead of the local optimality that comes from greedy heuristics) is Mixed-Integer Programming (MIP). This started with a model proposed by Bertsimas and Dunn (2017), and was improved many times. Most of this work was focused on optimal classification trees, but there are also a few MIP models that can find optimal regression trees such as a later method by Dunn (2018). This uses Mixed-Integer Quadratic Programming (MIQP) instead of Mixed-Integer Linear Programming (MILP) in order to use the SSE as an objective function. MILP already lacks scalability, and MIQP's increased complexity with quadratic constraints makes it generally much slower.

Similarly, models that use constraint programming (Verhaeghe et al., 2020), SAT (Narodytska et al., 2018; Janota & Morgado, 2020) or use MaxSAT (H. Hu et al., 2020) run into the same issues. This causes these models to struggle when the amount of data points go into the thousands.

The current state-of-the-art of scalable models that can find optimal decision trees mainly use dynamic programming. After the introduction of DL8 (Nijssen & Fromont, 2007), this

has been further improved on by the additions of bound-based pruning (Aglin et al., 2020), better lower bounds (X. Hu et al., 2019; Lin et al., 2020), and a quicker sub-procedure for finding trees of depth two (Demirović et al., 2022). Lower bound pruning uses relaxations of the problem to get a lower bound score of the optimal tree, so trees that already score worse than that lower bound can be ignored to speed up the algorithm. There are also models that use dynamic programming to find optimal regression trees, such as SRT (Van den Bos et al., 2024). Dynamic programming however operates by solving subtrees as independent subproblems, and because an adversary can make it so data points can fall into either subtree of a decision node, this is not the case for robust decision trees.

2.3 Robust decision trees

There has also been previous research into robust decision trees, but the resulting methods produce either classification or greedy trees. This started with a model that finds potential misclassifications in classification trees to then make them more robust (Kantchelian et al., 2016), but this uses MILP, so extending this to regression trees would require making it a MIQP problem, where we would run into issues with the scalability. Following that, Chen et al. (2019) take the greedy approach of finding locally optimal splits and makes it more robust by instead solving a min-max problem for each split, considering that attackers try to maximize the SSE while each split aims to minimize it. Later methods both improve on this greedy algorithm and allow users to specify the attacker's capabilities, for example using axis-aligned rules in TREANT (Calzavara et al., 2020) or using box-shaped perturbation limits in GROOT (Vos & Verwer, 2021).

There is currently one method that can produce decision trees that are both optimal and robust against adversarial attacks: ROCT (Vos & Verwer, 2022). ROCT consists of different formulations of the min-max problem for robustness that can be solved by either MILP or MaxSAT. However, ROCT can only produce robust optimal classification trees, and as previously mentioned, extending this to find robust optimal regression trees would require MIQP. Dunn (2018) previously managed to use MIQP to find optimal regression trees because this only required finding optimal splits, but robust optimal regression trees also require finding optimal prediction values, which makes MIQP less feasible because of its poor scalability.

To summarize, most research into optimal decision trees has been focused on classification trees, and finding robust optimal regression trees requires a different approach than the current solution for finding robust optimal classification trees. Because of this, ForTree uses exhaustive search to find robust optimal regression trees, and we improve its scalability by using both old and new pruning techniques.

3 Preliminaries

This section first describes the notation used throughout this paper in Section 3.1. Then, Section 3.2 defines the problem that ForTree aims to solve.

3.1 Notation

The notation used in this paper is summarized in Table 1. This section further explains this notation.

T denotes a regression tree which takes in any data point x, maps it to leaf node $m = T(x)$, and assigns c_m as the prediction for the data point. T consists of decision nodes T_D with a split feature j_m and a threshold value $b_m \in V_{j_m}$, and leaf nodes T_L with a prediction value $c_m \in C$. We denote the total amount of unique threshold values as $|V_u|$.

Training a regression tree requires data points X with n rows consisting of p features, where X_{ij} is the value for feature j of data point i, and target values y where y_i is the target value for data point i.

An adversary can change values in X within the perturbation limit ϵ, resulting in possible perturbed values $\tilde{X}_{ij}$. $\tilde{X}_i$ cannot always be mapped to a single leaf node anymore and instead results in the set of leaf nodes $T_L^{\tilde{X}_i}$ that $\tilde{X}_i$ can be mapped to. The goal of the adversary is to maximize the prediction errors e_i.

Symbol	Definition
j_m	Decision node m's split feature
b_m	Decision node m's threshold value
c_m	Leaf node m's predicted value
e_i	Prediction error for data point i
X_{ij}	Value of data point i in feature j
y_i	Target value of data point i
ϵ	Perturbation limit
n	Number of data points
p	Number of features
$\|V_u\|$	Number of unique threshold values
B_j	Possible split values for feature j
C	Possible prediction values
T_D	All decision nodes
T_L	All leaf nodes
$\tilde{X}_{ij}$	Possible perturbed values of data point i in feature j
$T_L^{\tilde{X}_i}$	Possible mapped leaf nodes for data point i after perturbations

Table 1: Summary of the notation used throughout the paper.

3.2 Problem definition

The aim of ForTree is to find robust optimal regression trees. This subsection defines the problem of finding a robust optimal regression tree.

A regression tree is optimal iff there exists no other regression tree with a lower prediction error $L(T)$ for some loss function. This is often limited to all possible regression trees in some set $\mathcal{T}$, for example, all regression trees with a maximum depth of two $\mathcal{T}_{d=2}$. We can say that a regression tree T^* is optimal iff the following equation holds.

$$L(T^*) = \min_{T \in \mathcal{T}} L(T) \tag{1}$$

There are many metrics for this prediction error. In our method, we minimize the sum of squared errors (SSE) because it is a commonly used metric that more heavily penalizes bigger errors, which are likely to be caused by an adversary. This results in the following equation, where $e_i = (c_m - y_i)^2$.

$$\sum_{i=0}^{n-1} e_i = \min_{T \in \mathcal{T}} \sum_{i=0}^{n-1} e_i \tag{2}$$

We can see that this holds for the regression tree in Figure 1a. Any other possible regression tree (of depth one) will result in an equal or higher SSE, therefore the equation holds and the regression tree is optimal.

Finding a robust optimal regression tree turns this minimization problem into a min-max problem. Instead of minimizing the SSE, we aim to minimize the adversarial SSE, which is the maximum SSE achievable by perturbing the data within the specified limit. Based on that, we can construct the following equation that has to hold for T^* to be a robust optimal regression tree.

$$\max_{X \in \tilde{X}} \sum_{i=0}^{n-1} e_i = \min_{T \in \mathcal{T}} \left(\max_{X \in \tilde{X}} \sum_{i=0}^{n-1} e_i \right) \tag{3}$$

Figure 1c shows an example of a robust regression tree where this equation holds. No other regression tree results in a lower adversarial SSE, so the tree in that example is a robust optimal regression tree.

In conclusion, ForTree aims to find a regression tree with the minimum possible sum of squared errors, assuming that an adversary tries to maximize this SSE by perturbing the data.

4 Why Robust Optimal Regression Trees are not trivial to find

This section explains the major challenge of finding robust optimal regression trees. Normally you only need to determine the optimal splits, since the optimal prediction value is just the mean, but because of the adversary you also need to determine optimal prediction values. The one current robust optimal decision tree method, ROCT (Vos & Verwer, 2022), deals with binary classification trees, which only have two possible classes for each leaf. Perturbing one data point so that it falls into a different leaf does not by default change the total amount of misclassifications, but for regression trees, there are infinite options for the continuous prediction values, and any shift in the data points has impact on the SSE.

To demonstrate the increased complexity of robust optimal regression trees, let us look at an example. Figure 2 shows a dummy regression tree with split features j_m, split thresholds b_m and prediction values c_m. For this small example, we take a dataset with $p = 3$ with for each feature $|B_j| = 10$ and C with $|C| = 10$. There are three splits to be made, resulting in $(p * |B_j|)^3 = 27,000$ possible combinations of splits, ignoring impossible splits. If this were a binary classification problem, there would be two possible classes per leaf for a total of $(p * |B_j|)^3 * 2^4 = 432,000$ possible trees. For a regression problem with $|C| = 10$, that becomes $(p * |B_j|)^3 * |C|^4 = 270,000,000$ possible trees. We can generalize this to the following equation for the amount of trees in a set.

$$|\mathcal{T}| = \left(\sum_{j=0}^{p-1} B_j \right)^{|T_D|} * |C|^{|T_L|} \tag{4}$$

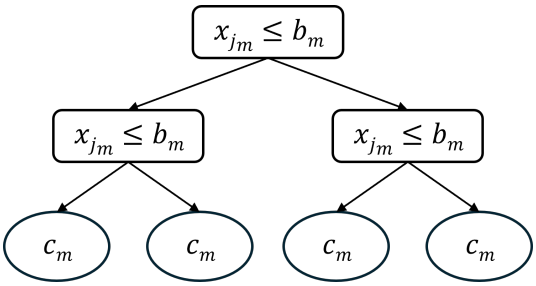


Figure 2: Dummy regression tree of depth two with split features j_m, split thresholds b_m and prediction values c_m.

Each decision node has possibilities equal to the total amount split thresholds across all features, each leaf node has possibilities equal to the amount of possible prediction values, and we get the total amount of possible trees by taking their powers equal to the amount of nodes. This shows that having a set of prediction values of some size, instead of only two classes like ROCT, adds another polynomial term to this equation.

This demonstrates the problem of increased complexity due to prediction values, which most attempts to extend current regression tree methods into robust regression tree methods would run into. As Dunn (2018) showed, it is possible to turn the MILP problem of finding optimal classification trees into a MIQP problem to find optimal regression trees, but extending that to find robust optimal regression trees, including the prediction values, would increase the already increased runtime from MIQP compared to MILP. Most other methods run into the same issue, and although it is also present in exhaustive search, we can more easily limit our search to a certain set of prediction values.

5 ForTree: Robust Optimal Regression Trees

This section describes ForTree, our proposed method to find robust optimal regression trees. Section 5.1 discusses the underlying exhaustive search algorithm of ForTree. Then, Section 5.2 goes into detail about the pruning methods used to speed up that algorithm.

5.1 Exhaustive Search

The underlying approach of ForTree is a depth first exhaustive search through all trees with the possible combinations of split features, split thresholds, and prediction values. The basic structure of this is detailed in Algorithm 1.

The main issue with exhaustive search is that we need to define the set of trees $\mathcal{T}$ to search through. To do this, we have to determine a set of prediction values C. We denote the set of trees limited to prediction values from C as $\mathcal{T}_{c_m \in C}$. We use percentile values of the target values as prediction values, since this results in a small set of values that accurately represents the distribution of the target values. We use 30 percentile values by default, but a different amount can be specified as a hyperparameter. We can write this as follows, with k as the hyperparameter.

$$C = \left\{ \text{percentile}(y, \frac{i}{k-1}) \;\middle|\; i = 0, \ldots, k \right\} \tag{5}$$

Algorithm 1 Underlying exhaustive search algorithm of ForTree

```
minSSE ← ∞
for m ∈ T_D
    for j_m ∈ {0, ..., p − 1}
        for b_m ∈ B_j
            for m ∈ T_L
                for c_m ∈ C
                    sse ← computeAdversarialSSE()
                    if sse < minSSE
                        bestTree ← current tree configuration
                        minSSE ← sse
return bestTree
```

We also have to determine the possible split thresholds for each split feature, but this is easier to do and does not reduce the set of trees. We take the unique feature values for a feature, and add ϵ to the splits to make them robust. This results in minimal sets of split thresholds that can still produce every unique possible split per feature, so it does not reduce the set of regression trees.

$$B_j = \{X_{ij} + \epsilon \mid \text{unique}(X_{ij})\} \tag{6}$$

ForTree effectively searches through all $T \in \mathcal{T}_{c_m \in C}$ and chooses a T^* such that no other $T \in \mathcal{T}_{c_m \in C}$ has a lower adversarial SSE. Therefore, Equation 3 holds for the set $\mathcal{T}_{c_m \in C}$, and ForTree finds a robust optimal regression tree for this set.

5.2 Pruning

The goal of exhaustive search is to evaluate all possible trees, but pruning can prevent needing to evaluate each possible tree individually. If an early evaluation shows that a group of possible trees cannot possibly be optimal, there is no need to further explore and evaluate them, and time can be saved by not doing so. ForTree uses two main pruning methods to speed up the exhaustive search algorithm this way. Algorithm 2 shows the structure of ForTree with pruning. Section 5.2.1 discusses regular SSE pruning, and Section 5.2.2 goes into detail about partial SSE pruning.

5.2.1 Regular SSE Pruning

ForTree uses the regular SSE as a lower bound for pruning. The regular SSE can easily be computed once the split features and split thresholds have been set for all decision nodes. This is done by setting the prediction value for each leaf to the mean of the values of all data points that fall into each leaf, as is also done by for example Breiman et al. (1984) and Dunn (2018) to get prediction values with the minimum SSE for these decision nodes. Recall from Equation 3 that we want to minimize $\max_{X \in \tilde{X}} \sum_{i=0}^{n-1} e_i$. The lower bound is $\sum_{i=0}^{n-1} e_i$ for X without perturbations, which is still part of $\tilde{X}$, so the following always holds.

$$\max_{X \in \tilde{X}} \sum_{i=0}^{n-1} e_i \geq \sum_{i=0}^{n-1} e_i \tag{7}$$

Algorithm 2 ForTree algorithm with pruning

```
minSSE ← ∞
for m ∈ T_D
    for j_m ∈ {0, ..., p − 1}
      for b_m ∈ B_j
        sse ← computeRegularSSE()
        if sse > minSSE
          move on to next threshold value
        for m ∈ T_L
          for c_m ∈ C
            if m is not last node
              sse ← computePartialAdversarialSSE()
              if sse > minSSE
                move on to next prediction value
            else
              sse ← computeAdversarialSSE()
              if sse < minSSE
                bestTree ← current tree configuration
                minSSE ← sse
return bestTree
```

If the lower bound for the trees with these decision nodes is greater than the minimum adversarial SSE found so far, then the adversarial SSE for all these trees will also be greater than the minimum adversarial SSE, so we can stop searching through these trees and move on to the next configuration of decision nodes. Using regular SSE when running exhaustive search on the example from Section 4 reduces the amount of fully evaluated trees from 270,000,000 to 1,900,000.

5.2.2 Partial SSE Pruning

Most of the runtime of the algorithm is spent in the leaf nodes. This is because of the high amount of possible combinations of prediction values, and having to compute the adversarial SSE for each one of these. We reduce the runtime spent on this by computing a partial adversarial SSE per leaf node instead of computing the entire adversarial SSE for each combination. Let us denote the set of all leaf nodes considered up until this point as T_{L_m} The formula for the partial adversarial SSE is as follows.

$$\sum_{i=0}^{n-1} \begin{cases} \max\limits_{m \in T_{L_m}^{\tilde{X}_i}} (c_m - y_i)^2 & \text{if } \left| T_{L_m}^{\tilde{X}_i} \right| > 0 \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

We evaluate the partial adversarial SSE by summing the maximum squared errors for each data point while only considering the leaf nodes that have been considered so far, and using zero if the data point does not fall into any of those leaf nodes. Because the adversary maximizes the SSE, considering additional leaf nodes will never result in a lower partial adversarial SSE. When considering all leaf nodes, the partial adversarial SSE simply computes $\sum_{i=0}^{n-1} \max_{m \in T_L^{\tilde{X}_i}} (c_m - y_i)^2$, which is equivalent to $\max_{X \in \tilde{X}} \sum_{i=0}^{n-1} e_i$, which is the adversarial SSE that we try to minimize.

In conclusion, if the partial adversarial SSE for the trees with a set of defined decision nodes and some defined leaf nodes is already greater than the minimum adversarial SSE found so far, the adversarial SSE for these trees will also be greater than the minimum adversarial SSE, and we can once again stop searching and move on. This reduces the fully evaluated trees from 270,000,000 to 7,423,790, and combining the two main pruning methods reduces it to 175,120.

6 Experiments

In our experiments, we compare the adversarial accuracy of ForTree to that of several other regression tree methods, and we evaluate the scalability of ForTree. Our experiments show that on average, ForTree achieves a higher adversarial training accuracy than the previous state-of-the-art methods. They also show that the runtime of ForTree mainly depends on the amount of features and the amount of unique feature values. First, Section 6.1 discusses the setup that was used for our experiments. Then, Section 6.2 describes the results regarding adversarial accuracy, and Section 6.3 details the results regarding scalability.

6.1 Experimental Setup

The setup for fetching and cleaning datasets for our experiments is taken from Van den Bos et al. (2024). Table 2 lists all the datasets that are used. These are all fetched from the UCI Repository[2], and then cleaned to turn usable categorical variables such as season into numerical variables or to remove unusable features such as car name. Each dataset gets randomly split into five folds to get a more accurate average adversarial accuracy.

We compare ForTree to three other methods, which are also listed in Table 2. We import an implementation of CART from the sklearn library[3], import TREANT from GROOT (Vos & Verwer, 2021), and use our own implementation of the heuristic introduced by Chen et al. (2019). We implemented ForTree itself in C++ and we use pybind to run it in our experiments, which are written in Python.

We compare the results of these methods with the adversarial R^2 score. This is similar to the standard R^2 score, which is defined as $1 - SSE/TSS$, but instead uses the adversarial SSE. This standardizes the SSE to better compare the results for different datasets. Both the training (for robust methods) and scoring happens with $\epsilon = 0.2$, and all trees are trained with a maximum depth of two. We limit our experiments to trees of depth two, since this lets our experiments run within a reasonable amount of time.

All our experiments were run with one thread on a HP ZBook Power G7 with 2.6GHz processing power. Each experiment was run with a timeout of 15 minutes. If ForTree reaches the timeout, it returns the best tree found so far.

6.2 Adversarial accuracy

Table 2 contains the results from our accuracy comparison. We see that ForTree outperforms the other methods for one out of the four datasets where ForTree found an optimal tree within the timeout limit. ForTree achieves 1.23 times as high adversarial accuracy as the second best method for the AutoMPG dataset, TREANT. In one case, ForTree, Chen and TREANT et al. have the same adversarial R^2 score when rounding to two decimals, and in

[2]http://archive.ics.uci.edu/ml
[3]https://pypi.org/project/scikit-learn/

two cases, TREANT slightly outperforms ForTree. For the datasets where ForTree has to return a suboptimal tree because of the timeout limit, the accuracy is always very similar to that of Chen et al. and TREANT. To accurately compare the overall performance of ForTree, we take the geometric mean of the ratios of the SSE achieved by Fortree and by Chen et al. or TREANT. ForTree achieves on average 1.39 times as high adversarial accuracy as Chen et al., and 1.07 times as high adversarial accuracy as TREANT.

Dataset	n	p	p_c	$\lvert V_u \rvert$	**CART**	**Chen et al.**	**TREANT**	**ForTree**
Airfoil	1503	5	5	163	-2.05	-0.16	**0.01**	0.00*
Auction	2043	7	7	54	-0.56	**0.45**	0.08	0.36
AutoMPG	392	8	7	636	-0.48	-1.59	0.36	**0.48**
Household	2049280	8	6	3631	-3.69	timeout	timeout	**-0.01***
OpticalNet.	640	9	9	2566	-60.75	**0.04**	**0.04**	-0.01*
SeoulBike	8760	12	11	3535	-0.65	-0.08	timeout	**0.02***
Servo	167	4	4	19	-2.02	0.10	**0.14**	0.13
Synch.	557	4	4	409	0.16	**0.25**	**0.25**	**0.25***
Yacht	308	6	6	64	-0.81	-0.88	**0.29**	0.27

Table 2: Adversarial R^2 from training data for the regression tree methods. The highest adversarial accuracy is for each dataset is marked bold. Timeouts are indicated by 'timeout', or in the case of ForTree by * after the result. p_c is the number of features of the cleaned dataset.

The results show that ForTree finds trees with on average higher adversarial training accuracy than the previous state-of-the-art methods, but it also shows that our robust optimal regression trees can sometimes have lower adversarial training accuracy than the robust greedy regression trees. Greedy methods can still sometimes outperform our method because we only get the optimal tree out of a set of trees with limited prediction values.

6.3 Scalability

Table 2 shows that ForTree does not finish the full exhaustive search for five out of the nine datasets. We further test the scalability of ForTree by running experiments with varying values for n, p, ϵ and $|V_u|$. Figure 3 contains the results of these experiments.

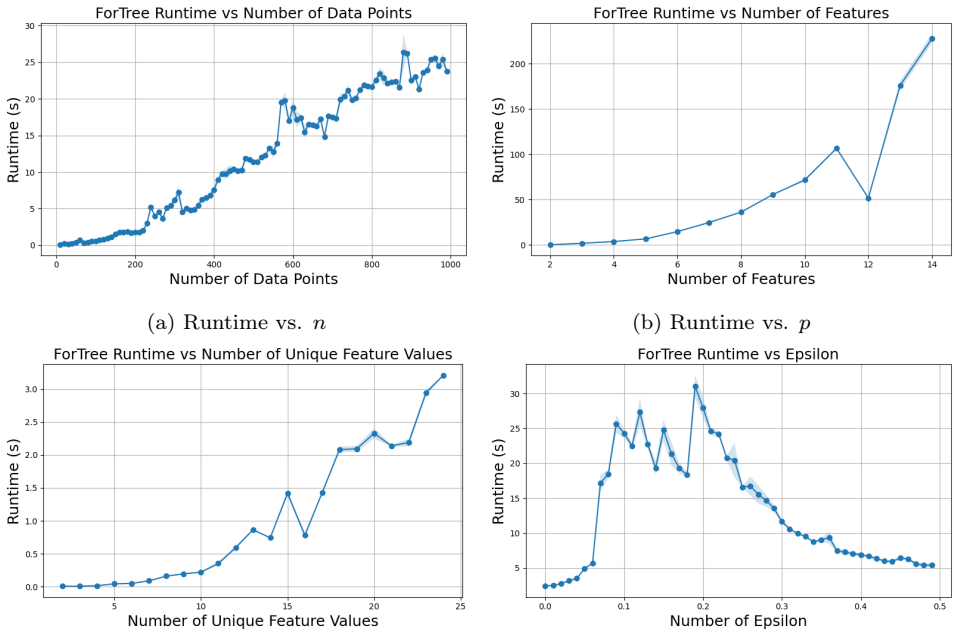
Figure 3a shows the runtime of ForTree compared to n, the amount of data points in the training set. For this experiment, we run ForTree 100 times on the SeoulBike dataset with n ranging from 10 to 1000 with steps of 10. All these experiments use 10 prediction values. We see an approximately linear relationship between the amount of data points and the runtime of ForTree.

Figure 3b contains a comparison of the runtime of ForTree and p, the amount of features in the training set. This experiment uses uniformly generated data to create small datasets with $n = 100$ and p in the range [2, 14]. The target values y are uniformly generated values in the range [0, 10] and the features values X are uniformly generated integer values in the range [0, 10], normalized to [0, 1] with steps of 0.1. This makes it so the amount of unique feature value stays at the same low value for all instances of this experiment. In these results we see an exponential relationship between the amount of features and the runtime of ForTree, with one random outlier.

These previous two experiments show an increase in runtime when the amount of data points or features increases. However, the results from Table 2 show that ForTree does not finish the full exhaustive search for the Synch. dataset which has both less data points

and less features than the Auction dataset, for which our method does successfully finish running within the time limit. Instead, we look to the amount of unique feature values for an explanation. Figure 3c shows the runtime of ForTree compared to the amount of unique feature values. This experiment uses uniformly generated datasets with $n = 100$ and $p = 2$, with the amounts of unique feature values in the range [2, 24]. We see an approximately exponential relationship between the amount of unique feature values and the runtime of ForTree, which mostly explains what datasets did or did not finish the full exhaustive search within the timeout limit.

Lastly, Figure 3d shows the runtime of ForTree compared to the value of ϵ. This experiment also uses uniformly generated random data, where $n = 25$ and $p = 4$ and the feature values are normalized to [0, 1]. We see that the runtime is the highest from ϵ values between 0.1 and 0.2. Lower values make it so the adversary has less impact, and there are on average less leaf nodes that a data point can be mapped to, which causes a lower runtime. On the other side, higher values cause data points to be mapped to a majority of the possible leaf nodes, which makes it so the individual splits and prediction values matter less, also reducing the runtime.



(a) Runtime vs. n

(b) Runtime vs. p

(c) Runtime vs. $|V_u|$

(d) Runtime vs. ϵ

Figure 3: Runtime comparison across different input dimensions.

7 Future Improvements

This section discusses what can be done to develop a better method for finding robust optimal regression trees, either by improving ForTree or by developing completely new methods.

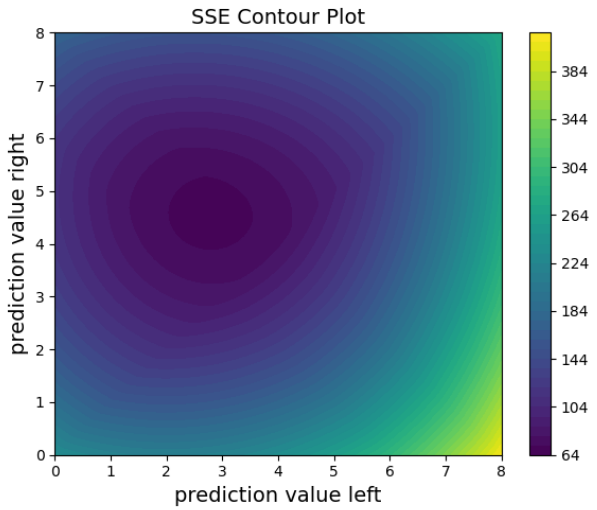


Figure 4: A contour plot of the SSE of an arbitrary depth one tree with the prediction values of the leaf nodes as parameters.

There are two main ways in which ForTree can be improved: more (effective) pruning, and searching more prediction values. We could simply increase the hyperparameter for the amount of prediction values, but this also increases our problem with runtime. Adding more pruning methods would of course counteract some of that problem, but we could also look at other ways of searching more prediction values.

A potential improvement would be optimizing the best combination of prediction values by using gradient descent. Once ForTree has found an optimal tree for a set of prediction values, these prediction values could be fine tuned in very small steps to find a locally optimal combination of prediction values that is not limited to the original set of prediction values anymore. This does not guarantee finding a tree with a globally minimal SSE if there are multiple local minima across all possible combinations of prediction values, but it would still improve the best tree found by ForTree. Additionally, Figure 4 shows that the function for the SSE of an arbitrary depth one tree with the two prediction values as parameters is a concave function with only one minimum. Future research could prove whether the function for the SSE of a depth two tree is also concave and whether gradient descent can be used to find globally optimal robust regression trees.

The other potential improvement that we suggest is using Mixed Integer Programming (MIP). We previously discussed that although Dunn (2018) used MIQP to find optimal regression trees, adding robustness to this problem would greatly improve the runtime, which is why our method uses exhaustive search instead. There are however still ways to potentially use MIP for finding globally optimal robust regression trees. A way to speed up this approach could be combining the exhaustive search from ForTree to search through all possible splits and MIQP to then find the optimal combination of prediction values. Another solution would be to not use quadratic constraints at all, which would prevent using SSE as our metric for the prediction error, but there are alternatives such as the Sum of Absolute Errors (SAE). The SSE of a tree with a globally optimal SAE could also be used as a lower bound for further pruning on other methods. We hope that these suggestions can allow

further research to find more accurate robust regression trees.

8 Responsible Research

This research follows the FAIR principles, which are a set of guidelines that focus on the usability of data and software. The acronym FAIR stands for the following four principles:

- Findable: All the data and the code used for this paper are available in a public repository.
- Accessible: There is no authentication or authorization needed to access the data and the code.
- Interoperable: All data used is stored in the CSV format.
- Reusable: The repository contains instructions on how to use ForTree.

Because this research follows these principles, it can be used a base to further explore the possibilities of using exhaustive search algorithms to find optimal robust regression trees.

It should also be mentioned that ChatGPT[4] was used in the development of ForTree. It was used for catching errors in the code, setting up the environment for testing, and generating basic code snippets. ChatGPT was also used for the generation of the algorithm visualizations in this paper.

9 Conclusions

In this paper, we introduce ForTree, our method to find robust optimal regression trees given a set of possible prediction values, using an exhaustive search approach with pruning. We introduce the partial adversary accuracy as a lower bound for pruning. We compare ForTree to state-of-the-art robust regression tree methods and show that ForTree finds regression trees with up to 1.23 times higher accuracy than the regression trees from the previous state-of-the-art methods, but this does require a longer runtime for datasets with high amounts of unique feature values. We also discuss why finding robust optimal regression trees is such a difficult problem to develop an effective method for, and we suggest multiple ideas to tackle this problem in the future. ForTree can further be improved by reducing the runtime with more (effective) pruning methods, and by increasing the accuracy with searching more prediction values.

References

Aglin, G., Nijssen, S., & Schaus, P. (2020). Learning optimal decision trees using caching branch-and-bound search. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(4), 3146–3153. https://doi.org/10.1609/aaai.v34i04.5711

Bertsimas, D., & Dunn, J. (2017). Optimal classification trees. *Machine Learning*, *106*(7), 1039–1082. https://doi.org/10.1007/s10994-017-5633-9

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Wadsworth.

[4]https://chat.openai.com/

Briţa, C. E., Van der Linden, J. G. M., & Demirović, E. (2025). Optimal classification trees for continuous feature data using dynamic programming with branch-and-bound. *Proceedings of the AAAI Conference on Artificial Intelligence*, *39*(11), 11131–11139. https://doi.org/10.1609/aaai.v39i11.33210

Calzavara, S., Lucchese, C., Tolomei, G., Abebe, S. A., & Orlando, S. (2020). Treant: Training evasion-aware decision trees. *Data Mining and Knowledge Discovery*, *34*(5), 1390–1420. https://doi.org/10.1007/s10618-020-00694-9

Chen, H., Zhang, H., Boning, D., & Hsieh, C.-J. (2019). *Robust decision trees against adversarial examples*. arXiv: 1902.10660 [cs]. https://doi.org/10.48550/arXiv.1902.10660

Demirović, E., Lukina, A., Hébrard, E., Chan, J., Bailey, J., Leckie, C., Ramamohanarao, K., & Stuckey, P. J. (2022). Murtree: Optimal decision trees via dynamic programming and search. *Journal of Machine Learning Research*, *23*(26), 1–47. http://jmlr.org/papers/v23/20-520.html

Dunn, J. W. (2018). *Optimal trees for prediction and prescription* [PhD thesis]. Massachusetts Institute of Technology. http://hdl.handle.net/1721.1/119280

Hu, H., Siala, M., Hébrard, E., & Huguet, M.-J. (2020). Learning optimal decision trees with maxsat and its integration in adaboost. *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)*, 1170–1176. https://doi.org/10.24963/IJCAI.2020/163

Hu, X., Rudin, C., & Seltzer, M. (2019). Optimal sparse decision trees. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). https://proceedings.neurips.cc/paper_files/paper/2019/file/ac52c626afc10d4075708ac4c778ddfc-Paper.pdf

Hyafil, L., & Rivest, R. L. (1976). Constructing optimal binary decision trees is np-complete. *Information Processing Letters*, *5*(1), 15–17. https://doi.org/10.1016/0020-0190(76)90095-8

Janota, M., & Morgado, A. (2020). Sat-based encodings for optimal decision trees with explicit paths. In L. Pulina & M. Seidl (Eds.), *Theory and applications of satisfiability testing â sat 2020* (pp. 501–518, Vol. 12178). Springer International Publishing. https://doi.org/10.1007/978-3-030-51825-7_35

Kantchelian, A., Tygar, J. D., & Joseph, A. (2016, June). Evasion and hardening of tree ensemble classifiers. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning* (pp. 2387–2396, Vol. 48). https://proceedings.mlr.press/v48/kantchelian16.html

Lin, J., Zhong, C., Hu, D., Rudin, C., & Seltzer, M. (2020). Generalized and scalable optimal sparse decision trees. In H. Daumé III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (pp. 6150–6160, Vol. 119). https://proceedings.mlr.press/v119/lin20g.html

Morgan, J. N., & Sonquist, J. A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, *58*(302), 415–434. https://doi.org/10.1080/01621459.1963.10500855

Narodytska, N., Ignatiev, A., Pereira, F., & Marques-Silva, J. (2018). Learning optimal decision trees with sat. *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI-18)*, 1362–1368. https://doi.org/10.24963/ijcai.2018/189

Nijssen, S., & Fromont, E. (2007). Mining optimal decision trees from itemset lattices. *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 530–539. https://doi.org/10.1145/1281192.1281250

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., & Fergus, R. (2014). Intriguing properties of neural networks. *2nd International Conference on Learning Representations (ICLR 2014), Conference Track Proceedings*. http://arxiv.org/abs/1312.6199

Van den Bos, M., Van der Linden, J. G. M., & Demirović, E. (2024). Piecewise constant and linear regression trees: An optimal dynamic programming approach. In R. Salakhutdinov, Z. Kolter, K. Heller, A. Weller, N. Oliver, J. Scarlett, & F. Berkenkamp (Eds.), *Proceedings of the 41st international conference on machine learning* (pp. 48994–49007, Vol. 235). https://proceedings.mlr.press/v235/van-den-bos24a.html

van der Linden, J. G. M., Vos, D., de Weerdt, M. M., Verwer, S., & Demirović, E. (2024). Optimal or greedy decision trees? revisiting their objectives, tuning, and performance. *CoRR*, *abs/2409.12788*. https://doi.org/10.48550/ARXIV.2409.12788

Verhaeghe, H., Nijssen, S., Pesant, G., Quimper, C.-G., & Schaus, P. (2020). Learning optimal decision trees using constraint programming. *Constraints*, *25*(3–4), 226–250. https://doi.org/10.1007/s10601-020-09312-3

Vos, D., & Verwer, S. (2021). Efficient training of robust decision trees against adversarial examples. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (pp. 10586–10595, Vol. 139). https://proceedings.mlr.press/v139/vos21a.html

Vos, D., & Verwer, S. (2022). Robust optimal classification trees against adversarial examples. *Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI-22)*, 8520–8528. https://doi.org/10.1609/aaai.v36i8.20829